CORE JAVA INTERVIEW QUESTIONS

Basic Interview Questions

1)What do you know about Java?
Ans: Java is a high-level programming language which was originally developed by Sun Microsystems and was released in 1995. Java runs on a variety of platforms, such as Windows, Mac OS and various versions of UNIX.

2)Which platforms are supported by Java Programming Language?
Ans: Java runs on a variety of platforms, such as Windows, Mac OS and various versions of UNIX/ Linux like HP-Unix, Sun Solaris, Redhat Linux, Ubuntu, CentOS, etc.

3)Why is Java Architectural Neutral?
Ans: The Java compiler generates an architecture-neutral object file format, which makes the compiled code to be executable on many processors, with the presence of Java runtime system.

4)How does Java enables High Performance?
Ans: Java uses Just-In-Time(JIT) compiler to enable high performance. Just-In-Time(JIT) compiler is a program that turns Java bytecode, which is a program containing instructions that can be sent directly to the processor.

5)Why is Java considered dynamic?
Ans: Java is designed to adapt to an evolving environment. Java programs can carry out extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time

6)List two Java IDEs.
Ans: Netbeans and Eclipse are two Java IDEs.

7)List the features of Java Programming language?
Ans: There are the following features in Java Programming Language.
Simple: Java is easy to learn. The syntax of Java is based on C++ which makes easier to write the program in it.
Object-Oriented: Java follows the object-oriented paradigm which allows us to maintain our code as the combination of different type of objects that incorporates both data and behavior.
Portable: Java supports read-once-write-anywhere approach. We can execute the Java program on every machine. Java program (.java) is converted to bytecode (.class) which can be easily run on every machine.

Platform Independent: Java is a platform independent programming language. It is different from other programming languages like C and C++ which needs a platform to be executed. Java comes with its platform on which its code is executed. Java doesn't depend upon the operating system to be executed.

Secured: Java is secured because it doesn't use explicit pointers. Java also provides the concept of ByteCode and Exception handling which makes it more secured.

Robust: Java is a strong programming language as it uses strong memory management. The concepts like Automatic garbage collection, Exception handling, etc. make it more robust.

Architecture Neutral: Java is architectural neutral as it is not dependent on the architecture. In C, the size of data types may vary according to the architecture (32 bit or 64 bit) which doesn't exist in Java.

Interpreted: Java uses the Just-in-time (JIT) interpreter along with the compiler for the program execution.

High Performance: Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++).

Multithreaded: We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.

Distributed: Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

Dynamic: Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand.

8)What do you understand by Java virtual machine?

Ans: It is a virtual machine that enables the computer to run the Java program. JVM acts like a run-time engine which calls the main method present in the Java code. JVM is the specification which must be implemented in the computer system. The Java code is compiled by JVM to be a Bytecode which is machine independent and close to the native code.

9)Define JRE (Java Runtime Environment).

Ans: Java Runtime Environment(JRE) is an implementation of the Java Virtual Machine(JVM) which executes Java programs. It provides the minimum requirements for executing a Java application.

10)What is a WAR file?

Ans: WAR is a Web Archive File and is used to store XML, Java classes and JavaServer pages which is used to distribute a collection of JavaServer Pages, Java Servlets, Java classes, XML files, static Web pages etc.

11)What is a JAR file?

Ans: A JAR file is Java Archive file and it aggregates many files into one. It holds Java classes in a library. JAR files are built on ZIP file format and have .jar file extension.

12(What is the difference between JDK, JRE, and JVM?

Ans: JVM

JVM is an acronym for Java Virtual Machine; it is an abstract machine which provides the runtime environment in which Java bytecode can be executed. It is a specification which specifies the working of Java Virtual Machine. Its implementation has been provided by Oracle and other companies. Its implementation is known as JRE.

JRE

JRE stands for Java Runtime Environment. It is the implementation of JVM. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.

JDK

JDK is an acronym for Java Development Kit. It is a software development environment which is used to develop Java applications and applets. It physically exists. It contains JRE + development tools. JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:

- Standard Edition Java Platform
- Enterprise Edition Java Platform
- Micro Edition Java Platform

13)What is JIT compiler?

Ans: Just-In-Time(JIT) compiler: It is used to improve the performance. JIT compiles parts of the bytecode that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here the term “compiler” refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

14)What gives Java its 'write once and run anywhere' nature?

Ans: The bytecode. Java compiler converts the Java programs into the class file (Byte Code) which is the intermediate language between source code and machine code. This byte-code is not platform specific and can be executed on any computer.

15)When is a 'byte' datatype used?

Ans: This datatype is used to save space in large arrays, mainly in place of integers, since a 'byte' is four times smaller than an 'int'.

16)What is the default value of 'byte' datatype in Java?

Ans: Default value of 'byte' datatype is 0.

17)What is the default value of 'float' and 'double' datatype in Java?

Ans: Default value of 'float' and 'double' datatype is different as compared to that in C/C++. For float its 0.0f and for double it is 0.0d

18)How many bits are used to represent Unicode, ASCII, UTF-16 and UTF-8 characters?
Ans: Unicode requires 16 bits and ASCII requires 7 bits. Although the ASCII character set uses only 7 bits, it is usually represented as 8 bits. UTF-8 represents characters using 8, 16, and 18 bit patterns. UTF-16 uses 16-bit and larger bit patterns.

19)According to Java Operator precedence, which operator is considered to be of highest precedence value?
Ans: Postfix operators i.e ( ) [ ] is considered to be of the highest precedence value

20)What is 'Public Static Void Main' in Java?
Ans: In Java programs, the point from where the program starts its execution or simply the entry point of Java programs is the **main()** method. Hence, it is one of the most important methods of Java

- **1. Public**

- It is an *Access modifier*, which specifies from where and who can access the method. Making the *main()* method public makes it globally available. It is made public so that JVM can invoke it from outside the class as it is not present in the current class.

- **2. Static**

- It is a *keyword* that is when associated with a method, making it a class-related method. The *main()* method is static so that JVM can invoke it without instantiating the class. This also saves the unnecessary wastage of memory which would have been used by the object declared only for calling the *main()* method by the JVM.

- **3. Void**

- It is a keyword and is used to specify that a method doesn't return anything. As the *main()* method doesn't return anything, its return type is *void*. As soon as the *main()* method terminates, the java program terminates too. Hence, it doesn't make any sense to return from the *main()* method as JVM can't do anything with the return value of it.

- **4. main**

- It is the name of the Java main method. It is the identifier that the JVM looks for as the starting point of the java program. It's not a keyword

- **5. String[] args**

- It stores Java command-line arguments and is an array of type *java.lang.String* class. Here, the name of the String array is *args* but it is not fixed and the user can use any name in place of it.

21)Why does Java not support pointers?
Ans: The pointer is a variable that refers to the memory address. They are not used in Java because they are unsafe(unsecured) and complex to understand.

22)If I don't provide any arguments on the command line, then what will the value stored in the String array passed into the main() method, empty or NULL?
Ans: It is empty, but not null

23)When is parseInt() method used?
Ans: The parseInt() method is used to get the primitive datatype of a certain string.

24)What is the difference between object oriented programming(OOP) language and object based programming language?
Ans: Object based programming languages follow all the features of OOPs except 'Inheritance'. JavaScript is an example of object based programming languages.

25)Which one is faster in java ?

for(int i = 100000; i > 0; i--) { }

for(int i = 1; i < 100001; i++) { }
Ans: a)

```
for(int i = 100000; i > 0; i--)
 {
 }
```

26)MyClass.java and empty files are valid source files. True or false?
Ans: True.

28)Which one of these primitive types are unsigned?
a) int
b) long
c) char
d) double
e) float
Ans: c) char (All numeric data types are signed. 'char' is the only unsigned integer type.)

27)Does File class have any method to read or write content in a file?
Ans: No.

28)Which one is faster in java ?

Math.max(a,b);

(a>b)?a:b
Ans: b) (a>b)?a:b

29)Which one of these statements are valid?

1. Char \u0061r a =’a’;

2. Char \u0062 = ’b’;

3. Char c =’\u0063’;

Options : a) 1. b) 2. c) 3. d) ALL. e) NONE.
Ans: d) ALL.

30)Is this statement correct: char ch = 'd'; if(ch < 32.00){ }
Ans: Yes the above statement is correct.

31)What will be output from the following statements :

System.out.println(1+2+”3”);

System.out.println(“1”+2+3);
Ans:
33
123

32)public main(int number) { } is a valid method or not?
Ans: No it is not a valid method.

33)Is this method valid or not? public static final main(String[] args){ }
Ans: Yes it is a valid method.

34)Why is main() method static?
Ans: To access a static method class object is not needed. The method can be accessed directly with the help of ClassName. So when a program is started the jvm search for the class with main method and calls it without creating an object of the class.

35)What is the difference between static and instance methods?
Ans:

- Instance method belongs to the instance of a class therefore it requires an instance before it can be invoked, whereas static method belongs to the class itself and not to any class instance so it doesn’t need an instance to be invoked.
- Instance methods use dynamic (late) binding, whereas static methods use static (early) binding. When the JVM invokes a class instance method, it selects the method to invoke based on the type of the object reference, which is always known at run-time.
- On the other hand, when the JVM invokes a static method, it selects the method to invoke based on the actual class of the object, which may only be known at compile time.

1)What is the full form of JDK?

Ans: The full form of JDK is Java Development Kit.

2)Define JDK.

Ans: JDK is a superset of JRE and it contains everything that is in the JRE as well as tools such as the compilers and debuggers necessary for developing applets and applications.

3)What is the meaning of an OpenJDK?

Ans: OpenJDK is an open-source implementation of the Java SE 7 JSR (JSR 336). There is almost no difference between the Oracle JDK and the OpenJDK. Last year, Oracle took this decision of moving to OpenJDK as the official Java SE 7 Reference Implementation.

4)What is the difference between a JDK and a JVM?

- It is very close - our build process for Oracle JDK releases builds on OpenJDK 7 by adding just a couple of pieces, like the deployment code, which includes Oracle's implementation of the Java Plugin and Java WebStart, as well as some closed source third party components like a graphics rasterizer.

Ans: JDK is Java Development Kit which is for development purpose and it includes execution environment also.

But JVM is purely a run time environment and hence you will not be able to compile your source files using a JVM.

5)Explain what is JDK

Ans:

- Java Development Kit (JDK): Java Development Kit is the core component of Java Environment and provides all the tools, executables and binaries required to compile, debug and execute a Java Program.
- JDK is a platform specific software where we have separate installers for Windows, Mac and Unix systems.
- We can say that JDK is superset of JRE since it contains JRE with Java compiler, debugger and core classes.
- Current version of JDK is 1.7 also known as Java 7.

6)Define JDK, JRE and JVM.

Ans:

- JDK is for development purpose whereas JRE is for running the java programs.
- JDK and JRE both contains JVM so that we can run our java program.
- JVM is the heart of java programming language and provides platform independence.

7)Java supports pass by value or pass by reference?

Ans: Java supports only pass by value. The arguments passed as a parameter to a method is mainly primitive data types or objects. For the data type the actual value is passed.

Java passes the references by value just like any other parameter. The pointer to the object is passed as value. Thus, method manipulation will alter the objects, since the references point to the original object but will not initialize the new object.

Consider the example : The method successfully alters the value of pnt1, even though it is passed by value; however, a swap of pnt1 and pnt2 fails! This is the major source of confusion. In the main() method, pnt1 and pnt2 are nothing more than object references. When you pass pnt1 and pnt2 to the tricky() method, Java passes the references by value just like any other parameter. This means the references passed to the method are actually copies of the original references.

```
public void tricky(Point arg1, Point arg2)
{
   arg1.x = 100;
   arg1.y = 100;
   Point temp = arg1;
   arg1 = arg2;
   arg2 = temp;
}
public static void main(String [] args)
{
   Point pnt1 = new Point(0,0);
   Point pnt2 = new Point(0,0);
   System.out.println("X: " + pnt1.x + " Y: " +pnt1.y);
   System.out.println("X: " + pnt2.x + " Y: " +pnt2.y);
   System.out.println(" ");
   tricky(pnt1,pnt2);
   System.out.println("X: " + pnt1.x + " Y:" + pnt1.y);
   System.out.println("X: " + pnt2.x + " Y: " +pnt2.y);
}
```

Output:

```
X: 0     Y: 0
X: 0     Y: 0
X: 100   Y: 100
X: 0     Y: 0
```

8)What is the difference between equals() and ==?

Ans: == operator is used to compare the references of the objects. public boolean equals(Object o) is the method provided by the Object class.

The default implementation uses == operator to compare two objects. But since the method can be overridden like for String class. equals() method can be used to compare the values of two objects.

```
String str1 = "MyName";
```

```
String str2 = new String("MyName");
String str3 = str2;
if(str1 == str2)
{
   System.out.println("Objects are equal")
}
else
{
   System.out.println("Objects are not equal")
}
if(str1.equals(str2))
{
   System.out.println("Objects are equal")
}
else
{
   System.out.println("Objects are not equal")
}
```

Output:

Objects are not equal
Objects are equal

```
String str2 = "MyName";
String str3 = str2;
if(str2 == str3)
{
   System.out.println("Objects are equal")
}
else
{
   System.out.println("Objects are not equal")
}
if(str3.equals(str2))
{
   System.out.println("Objects are equal")
}
else
{
   System.out.println("Objects are not equal")
}
```

Output:

Objects are equal
Objects are equal

9)How does Java allocate stack and heap memory?

Ans:

- Each time an object is created in Java it goes into the part of memory known as heap.
- The primitive variables like int and double are allocated in the stack, if they are local method variables and in the heap if they are member variables (i.e. fields of a class). In Java methods local variables are pushed into stack.
- When a method is invoked and stack pointer is decremented when a method call is completed. In a multi-threaded application each thread will have its own stack but will share the same heap. This is why care should be taken in your code to avoid any concurrent access issues in the heap space.
- The stack is threadsafe (each thread will have its own stack) but the heap is not threadsafe unless guarded with synchronization through your code.

JVM Interview Question

1)JVM means? What is JVM?

Ans:

- A Java Virtual Machine is a runtime environment required for execution of a Java application.
- Every Java application runs inside a runtime instance of some concrete implementation of abstract specifications of JVM.
- It is JVM which is crux of 'platform independent' nature of the language

2)How is your Java program executed inside JVM?

Ans:

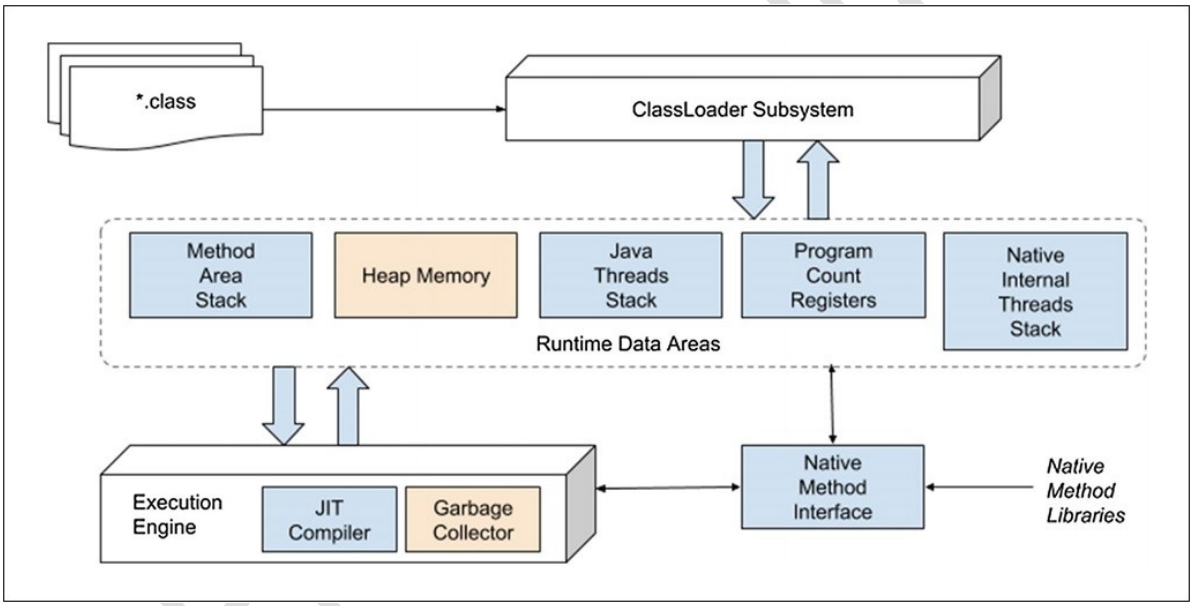
- When JVM executes a Java application, a runtime instance of JVM is born. This runtime instance invoke main() method of Java application. The main() method of an application serves as the starting point for that application's initial thread.
- The initial thread can in turn fire off other threads. This thread has a program counter (PC) and Java stack. Whenever main() method is invoked, a stack frame is pushed onto the stack, this then becomes the active tack frame.
- The program counter in the new Java stack frame will point to the beginning of the method. If there are more method invocations within main() method then this process of pushing new stack frame onto the stack for each method call is repeated as and when they are invoked.
- When a method returns, the active frame is popped from the stack and the one below becomes the active stack frame. The PC is set to the instruction after the method call and the method continues.
- There is only one heap corresponding to an instance of JVM and all objects created are stored here. This heap is shared by all threads created in an application. Inside the Java virtual machine, threads come in two flavors: daemon and non- daemon.

- A daemon thread is ordinarily a thread used by the virtual machine itself, such as a thread that performs garbage collection. The application, however, can mark any threads it creates as daemon threads.
- The initial thread of an application--the one that begins at main()--is a non- daemon thread. A Java application continues to execute (the virtual machine instance continues to live) as long as any non-daemon threads are still running.
- When all non-daemon threads of a Java application terminate, the virtual machine instance will exit. If permitted by the security manager, the application can also cause its own demise by invoking the exit() method of class Runtime or System. When main() returns, it terminates the application's only non-daemon thread, which causes the virtual machine instance to exit.
-

3)What is a JVM consisted of?

Ans: Each time a Java Application is executed then an instance of JVM, responsible for its running, is created. A JVM instance is described in terms of subsystems, memory areas, data types, and instructions.

The block diagram given below depicts a view of Internal Architecture of JVM:



4)What is Java class file's magic number?

Ans:

- A Magic Number of a class file is a unique identifier for tools to quickly differentiate class files from non-class files.
- The first four bytes of each Java class file has the magic value as 0xCAFEBABE.
- And the answer to why this number, I do not actually know but there may be very few sensible and acceptable options possible constructed from letters A-F which can surely not be 'CAFEFACE' or 'FADECAFE'..

5)What is heap and stack?

Ans:

- The heap is the part of memory of JVM where all objects reside.
- The stack is consisted of stack frames.
- When a thread invokes a method, the JVM pushes a new frame onto that thread's Java stack. Each stack frame is consisted of operand stack and the local variable array.
- All arguments, local variables, intermediate computations and return values if any are kept in these stack corresponding to the method invoked. The stack frame on the top of the stack is called the active stack frame, which is the current place of execution. When the method completes, the virtual machine pops and discards the frame for that method.

6)How JVM performs Thread Synchronization?

Ans:

- JVM associates a lock with an object or a class to achieve multithreading. A lock is like a token or privilege that only one thread can "possess" at any one time. When a thread wants to lock a particular object or class, it asks the JVM.
- JVM responds to thread with a lock maybe very soon, maybe later, or never. When the thread no longer needs the lock, it returns it to the JVM. If another thread has requested the same lock, the JVM passes the lock to that thread. If a thread has a lock, no other thread can access the locked data until the thread that owns the lock releases it.
- The JVM uses locks in conjunction with monitors. A monitor is basically a guardian in that it watches over a sequence of code, making sure only one thread at a time executes the code. Each monitor is associated with an object reference. It is the responsibility of monitor to watch an arriving thread must obtain a lock on the referenced object. When the thread leaves the block, it releases the lock on the associated object.
- A single thread is allowed to lock the same object multiple times. JVM maintains a count of the number of times the object has been locked. An unlocked object has a count of zero. When a thread acquires the lock for the first time, the count is incremented to one. Each time the thread acquires a lock on the same object, a count is incremented. Each time the thread releases the lock, the count is decremented. When the count reaches zero, the lock is released and made available to other threads. In Java language terminology, the coordination of multiple threads that must access shared data is called synchronization.
- The language provides two built-in ways to synchronize access to data: with synchronized statements or synchronized methods. The JVM does not use any special opcodes to invoke or return from synchronized methods.
- When the JVM resolves the symbolic reference to a method, it determines whether the method is synchronized. If it is, the JVM acquires a lock before invoking the method. For an instance method, the JVM acquires the lock associated with the object upon which the method is being invoked. For a class method, it acquires the lock associated with the class to which the method belongs.
- After a synchronized method completes, whether it completes by returning or by throwing an exception, the lock is released. Two opcodes, monitorenter and monitorexit are used by JVM for accomplishing this task. When monitorenter is encountered by the Java virtual

machine, it acquires the lock for the object referred to by objectref on the stack. If the thread already owns the lock for that object, a count is incremented. Each time monitorexit is executed for the thread on the object, the count is decremented. When the count reaches zero, the monitor is released.

7)What is a class loader and what is its responsibilities?

Ans: The Class loader is a subsystem of a JVM which is responsible, predominantly for loading classes and interfaces in the system. Apart from this, a class loader is responsible for the following activities:-

- Verification of imported types (classes and interfaces)-Allocating memory for class variables and initializing them to default values. Static fields for a class are created and these are set to standard default values but they are not explicitly initialized.
- The method tables are constructed for the class.-Resolving symbolic references from type to direct references The class loaders can be of two types: a bootstrap or primordial class loader and user defined class loaderEach JVM has a bootstrap class loader which loads trusted classes, including classes from Java API.
- JVM specs do not tell how to locate these classes and is left to implementation designers. A Java application with user defined class loader objects can customize class loading. These load untrustworthy classes and not an intrinsic part of JVM. They are written in Java, converted to class files and loaded into the JVM and installed like any other objects.

8)How to profile heap usage?

Ans:

- Try using - Xaprof to get a profile of the allocations (objects and sizes) of your application.
- Also try - agentlib:hprof=heap=all (or other option, try - agentlib:hprof=help for a list).

9)How JVM performs Garbage Collection?

Ans:

- Whenever a reference to an object on heap lies dangling or no longer in use then it becomes eligible for being garbage collected by JVM.
- JVM specifications do not force any specific kind of garbage collection algorithm though there are several algorithms like reference counting, tracing, compacting, copying, generational etc. in place.
- It is very important that garbage collection should be efficient and non-interfering in execution of Java programs.
- There is a trade off between ease of implementation versus better performance while implementing garbage collection feature for a JVM.

10)An application has a lot of threads and is running out of memory, why?

Ans:

- You may be running into a problem with the default stack size for threads. In Java SE 6, the default on Sparc is 512k in the 32-bit VM, and 1024k in the 64-bit VM.

- On x86 Solaris/Linux it is 320k in the 32-bit VM and 1024k in the 64-bit VM. On Windows, the default thread stack size is read from the binary (java.exe). As of Java SE 6, this value is 320k in the 32-bit VM and 1024k in the 64-bit VM. You can reduce your stack size by running with the -Xss option.
- For example: java -server -Xss64kNote that on some versions of Windows, the OS may round up thread stack sizes using very coarse granularity. If the requested size is less than the default size by 1K or more, the stack size is rounded up to the default; otherwise, the stack size is rounded up to a multiple of 1 MB.64k is the least amount of stack space allowed per thread.

11)If your program is I/O bound or running in native methods, do these activities engage JVM?

Ans: Programs that spend significant portions of their time in those native code libraries will not see their performance on HotSpot improved as much as programs that spend most of their time executing byte codes.

14)Why Java based GUI intensive program has performance issues?

The overall performance of a Java application depends on four factors :

- The design of the application
- The speed at which the virtual machine executes the Java bytecodes
- The speed at which the libraries that perform basic functional tasks execute (in native code)
- The speed of the underlying hardware and operating system

The virtual machine is responsible for byte code execution, storage allocation, thread synchronization, etc.

Running with the virtual machine are native code libraries that handle input and output through the operating system, especially graphics operations through the window system.

Programs that spend significant portions of their time in those native code libraries will not see their performance on HotSpot improved as much as programs that spend most of their time executing byte codes.

12)Different types of memory used by JVM ?

Ans:

- Class.
- Heap.
- Stack.
- Register.

Native Method Stack

13)What is 64 bit Java ?

Ans: GUI intensive Java application mostly run underlying OS specific native libraries which is time and more CPU cycles consuming.

- A 64-bit version of Java has been available to Solaris SPARC users since the 1.4.0 release of J2SE.
- A 64-bit capable J2SE is an implementation of the Java SDK (and the JRE along with it) that runs in the 64-bit environment of a 64-bit OS on a 64-bit processor. The primary advantage of running Java in a 64-bit environment is the larger address space. This allows for a much larger Java heap size and an increased maximum number of Java Threads, which is needed for certain kinds of large or long-running applications.
- The primary complication in doing such a port is that the sizes of some native data types are changed. Not surprisingly the size of pointers is increased to 64 bits.
- On Solaris and most Unix platforms, the size of the C language long is also increased to 64 bits.
- Any native code in the 32-bit SDK implementation that relied on the old sizes of these data types is likely to require updating. Within the parts of the SDK written in Java things are simpler, since Java specifies the sizes of its primitive data types precisely. However even some Java code needs updating, such as when a Java int is used to store a value passed to it from a part of the implementation written in C.
-

14)What is the difference between JVM and JRE?

Ans:

- A Java Runtime Environment (JRE) is a prerequisite for running Java applications on any computer.
- A JRE contains a Java Virtual Machine (JVM), all standard, core java classes and runtime libraries. It does not contain any development tools such as compiler, debugger, etc.
  JDK (Java Development Kit) is a whole package required to Java Development which essentially contains JRE+JVM, and tools required to compile and debug, execute Java applications

15)Which memory segment loads the java code ?

Ans: Code segment.

16)When are static variables loaded in memory ?

Ans: They are loaded at runtime when the respective Class is loaded.

17)What is a String Pool ?

Ans: String pool (String intern pool) is a special storage area in Java heap. When a string is created and if the string already exists in the pool, the reference of the existing string will be returned, instead of creating a new object and returning its reference.

18)Which are the different segments of memory ?

Ans:

- **Stack Segment -**It contains local variables and Reference variables(variables that hold the address of an object in the heap).
- **Heap Segment -**It contains all created objects in runtime, objects only plus their object attributes (instance variables).
- **Code Segment -** The segment where the actual compiled Java bytecodes resides when loaded.

19)Describe what happens when an object is created in Java ?

Ans:

1. Memory is allocated from heap to hold all instance variables and implementation-specific data of the object and its superclasses. implementation-specific data includes pointers to class and method data.
2. The instance variables of the objects are initialized to their default values.
3. The constructor for the most derived class is invoked. The first thing a constructor does is call the constructor for its superclasses. This process continues until the constructor for java.lang.Object is called, as java.lang.Object is the base class for all objects in java.
4. Before the body of the constructor is executed, all instance variable initializers and initialization blocks are executed. Then the body of the constructor is executed. Thus, the constructor for the base class completes first and constructor for the most derived class completes last.

20)How many objects are created with this code :- String s = new String("abc");

Ans: Two objects will be created here. One object creates memory in heap with new operator and second in stack constant pool with "abc".

21)Describe, in general, how java's garbage collector works ?

Ans:

- The Java runtime environment deletes objects when it determines that they are no longer being used. This process is known as garbage collection.
- The Java runtime environment supports a garbage collector that periodically frees the memory used by objects that are no longer needed.
- The Java garbage collector is a mark-sweep garbage collector that scans Java's dynamic memory areas for objects, marking those that are referenced. After all possible paths to objects are investigated, those objects that are not marked (i.e. are not referenced) are known to be garbage and are collected.

22)Can I import same package/class twice? Will the JVM load the package twice at runtime?

Ans: One can import the same package or same class multiple times. Neither compiler nor JVM complains wil complain about it. And the JVM will internally load the class only once no matter how many times you import the same class.

23)What is a class loader ? What are the different class loaders used by JVM ?

Ans: Class loader is part of JVM which is used to load classes and interfaces.
Class loaders used by JVM are:

- Bootstrap
- Extension
- System

24)Is JVM, a compiler or interpreter ?
Ans: Its an interpretor.

25)Explain java.lang.OutOfMemoryError ?
Ans: This Error is thrown when the Java Virtual Machine cannot allocate an object because it is out of memory, and no more memory could be made available by the garbage collector.

26)Should we override finalize method ?
Ans: Finalize is used by Java for Garbage collection. It should not be done as we should leave the Garbage Collection to Java itself.

27)Difference between loadClass and Class.forName ?
Ans: loadClass only loads the class but doesn't initialize the object whereas Class.forName initialize the object after loading it.
28)Which kind of memory is used for storing object member variables and function local variables ?
Ans: Local variables are stored in stack whereas object variables are stored in heap.

29)Why do member variables have default values whereas local variables don't have any default value ?
Ans: Member variable are loaded into heap, so they are initialized with default values when an instance of a class is created. In case of local variables, they are stored in stack until they are being used.

30)What are various types of Class loaders used by JVM ?
Ans:

- **Bootstrap -** Loads JDK internal classes, java.* packages.
- **Extensions -** Loads jar files from JDK extensions directory - usually lib/ext directory of the JRE
- **System -** Loads classes from system classpath.

31)How are classes loaded by JVM ?
Ans: Class loaders are hierarchical. The very first class is specially loaded with the help of static main() method declared in your class. All the subsequently loaded classes are loaded by the classes, which are already loaded and running.

32)Which memory areas does instance and static variables use ?
Ans: Instance variables are stored on stack whereas static variables are stored on heap.

33)What is metaspace ?
Ans: The Permanent Generation (PermGen) space has completely been removed and is kind of replaced by a new space called Metaspace.
The consequences of the PermGen removal is that obviously the PermSize and MaxPermSize JVM arguments are ignored and you will never get a java.lang.OutOfMemoryError: PermGen error.

34)Why Java don't use pointers ?
Ans: Pointers are vulnerable and slight carelessness in their use may result in memory problems and hence Java intrinsically manage their use.

35)Difference between static vs dynamic class loading?
Ans: Static loading - Classes are statically loaded with Java's "new" operator.
Dynamic class loading - Dynamic loading is a technique for programmatically invoking the functions of a class loader at run time.
Class.forName (Test className);

36)What are the disadvantages of using arrays ?
Ans:

- Arrays are of fixed size and have to reserve memory prior to use. Hence if we don't know size in advance arrays are not recommended to use.
- Arrays can store only homogeneous elements.
- Arrays store its values in contentious memory location. Not suitable if the content is too large and needs to be distributed in memory.
- There is no underlying data structure for arrays and no ready made method support for arrays, for every requirement we need to code explicitly.
-

37)Can we call the garbage collector explicitly ?
Ans: Yes, We can call garbage collector of JVM to delete any unused variables and unreferenced objects from memory using gc( ) method.
This gc( ) method appears in both Runtime and System classes of java.lang package.

38)What are different ways to create String Object? Explain.
Ans: When we create a String using double quotes, JVM looks in the String pool to find if any other String is stored with same value. If found, it just returns the reference to that String object else it creates a new String object with given value and stores it in the String pool.
When we use new operator, JVM creates the String object but don't store it into the String Pool. We can use intern() method to store the String object into String pool or return the reference if there is already a String with equal value present in the pool.
String str = new String("abc");

String str1 = "abc";

39)How substring() method of String class create memory leaks?

Ans: Substring method would build a new String object keeping a reference to the whole char array, to avoid copying it. Hence you can inadvertently keep a reference to a very big character array with just a one character string.

40)What are strong, soft, weak and phantom references in Java ?

Ans:

Garbage Collector won't remove a **strong** reference.

- A **soft** reference will only get removed if memory is low.
- A **weak** reference will get removed on the next garbage collection cycle.
- A **phantom** reference will be finalized but the memory will not be reclaimed. Can be useful when you want to be notified that an object is about to be collected.

41)How Java provide high Performance ?

Ans: Java uses Just-In-Time compiler to enable high performance. Just-In-Time compiler is a program that turns Java bytecode into instructions that can be sent directly to the processor.

42)Why is Java considered Portable Language ?

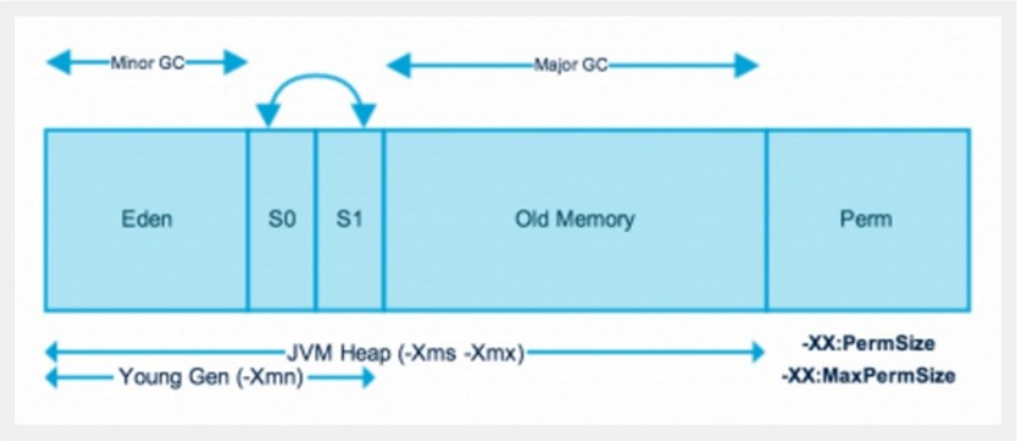
Ans: Java is a portable-language because without any modification we can use Java byte-code in any platform(which supports Java). So this byte-code is portable and we can use in any other major platforms.

43)How to find if JVM is 32 or 64 bit from Java program ?

Ans: You can find JVM - 32 bit or 64 bit by using System.getProperty() from Java program.

44)What is Java (JVM) Memory Model?

Ans:



As you can see in the above image, JVM memory is divided into separate parts. At broad level, JVM Heap memory is physically divided into two parts – Young Generation and Old Generation.

45)What is Young Generation of Java (JVM) Memory Model?

Ans: Young generation is the place where all the new objects are created. When young generation is filled, garbage collection is performed. This garbage collection is called Minor GC. Young Generation is divided into three parts – Eden Memory and two Survivor Memory spaces.

Important Points about Young Generation Spaces:

- Most of the newly created objects are located in the Eden memory space.
- When Eden space is filled with objects, Minor GC is performed and all the survivor objects are moved to one of the survivor spaces.
- Minor GC also checks the survivor objects and move them to the other survivor space. So at a time, one of the survivor space is always empty.
- Objects that are survived after many cycles of GC, are moved to the Old generation memory space. Usually it's done by setting a threshold for the age of the young generation objects before they become eligible to promote to Old generation.

46)Java Heap Memory Switches.

Ans: Java provides a lot of memory switches that we can use to set the memory sizes and their ratios. Some of the commonly used memory switches are:

VM SWITCH	VM SWITCH DESCRIPTION
-Xms	For setting the initial heap size when JVM starts
-Xmx	For setting the maximum heap size.
-Xmn	For setting the size of the Young Generation, rest of the space goes for Old Generation.
-XX:PermGen	For setting the initial size of the Permanent Generation memory
-XX:MaxPermGen	For setting the maximum size of Perm Gen
-XX:SurvivorRatio	For providing ratio of Eden space and Survivor Space, for example if Young Generation size is 10m and VM switch is -XX:SurvivorRatio=2 then 5m will be reserved for Eden Space and 2.5m each for both the Survivor spaces. The default value is 8.
-XX:NewRatio	For providing ratio of old/new generation sizes. The default value is 2.

47)What is Java Stack Memory?

Ans: Java Stack memory is used for execution of a thread. They contain method specific values that are short-lived and references to other objects in the heap that are getting referred from the method.

48)What is Method Area?

Ans: Method Area is part of space in the Perm Gen and used to store class structure (runtime constants and static variables) and code for methods and constructors.

49)What is Runtime Constant Pool?

Ans: Runtime constant pool is per-class runtime representation of constant pool in a class. It contains class runtime constants and static methods. Runtime constant pool is the part of method area.

50)How to change the heap size of a JVM?

Ans: The old generation's default heap size can be overridden by using the -Xms and -Xmx switches to specify the initial and maximum sizes respectively : java -Xms -Xmx program
For example : java -Xms64m -Xmx128m program

51)What is Permanent Generation?

Ans:

Permanent Generation or "Perm Gen" contains the application metadata required by the JVM to describe the classes and methods used in the application. Note that Perm Gen is not part of Java Heap memory.

Perm Gen is populated by JVM at runtime based on the classes used by the application. Perm Gen also contains Java SE library classes and methods. Perm Gen objects are garbage collected in a full garbage collection.

52)What is memory leak?

Ans: A memory leak is where an unreferenced object that will never be used again still hangs around in memory and doesn't get garbage collected.

53)A class without a method can be run by JVM if its ancestor class has main. True/False?

Ans: True.

Access Specifier Interview Question

1)What is an Access Specifier ?

Ans: It is used to explicitly mention the way how the data (variables and methods of a class) will be available outside the scope. An access specifier is something which mentions the way how the member of a class will be made available to anything outside the class. It cannot be used with the local variables(i.e. present inside method/scope). Access specifiers are the keywords using which we can control the accessibility of the members of a class.

2) What are the various access specifiers in Java?

Ans: In Java, access specifiers are the keywords which are used to define the access scope of the method, class, or a variable. In Java, there are four access specifiers given below.

1:Public The classes, methods, or variables which are defined as public, can be accessed by any class or method.

2:Protected Protected can be accessed by the class of the same package, or by the sub-class of this class, or within the same class.

3:Default Default are accessible within the package only. By default, all the classes, methods, and variables are of default scope.
4:Private The private class, methods, or variables defined as private can be accessed within the class only.

3)Which Access Specifier can be used with a class ?
Ans: For an outer class we can only use 'public' and 'default' access specifiers. We can use 'private' access specifier for an inner class.

4)Can we reduce the accessibility of an inherited or overridden method ?
Ans: No, we cannot reduce the accessibility of an inherited or overridden method.

5)What is the difference between public, private, default and protected access specifiers ?
Ans:

- **Private -** The code defined within a 'private' access specifier is not accessible outside the object scope.
- **Public -** The code defined within a 'public' access specifier is accessible from anywhere.
- **Default -** The code defined within a 'default' access specifier is accessible from anywhere within the same package.
- **Protected -** The code defined within a 'default' access specifier is accessible by object and the sub class objects.
-

6)What will happen if we make the constructor 'private' ?
Ans: We won't be able to create the objects directly by invoking a new operator by making a constructor private.

7)Can we instantiate the object of derived class if parent constructor is 'protected' ?
Ans: No, we cannot instantiate the object of derived class if parent constructor is protected.

8)Can we declare an abstract method 'private' ?
Ans: No. An abstract method can only be declared 'protected' or 'public'.

9)Can a private variable or method of a class can be accessed?
Ans: Yes its possible using reflection.

Classes & Objects Interview Question

1)What do you mean by an Object?
Ans: Object is a runtime entity whose state is stored in fields and behavior is shown via methods. Methods operate on an object's internal state and serve as the primary mechanism for object-to-object communication.

2)An object is resurrected by making other object refer to the dying object in finalize method. Will this object be ever garbage collected?

Ans: Resurrection can happen in 'finalize' method which will prevent GC to reclaim the object memory. However this could be done only once. Next time GC will not invoke 'finalize' method before garbage collection.

3)Define class.

Ans: A class is a blue print from which individual objects are created. A class can contain fields and methods to describe the behavior of an object.

4)What is classloader?

Ans: Classloader is a subsystem of JVM which is used to load class files. Whenever we run the java program, it is loaded first by the classloader. There are three built-in classloaders in Java.

- **Bootstrap ClassLoader**: This is the first classloader which is the superclass of Extension classloader. It loads the rt.jar file which contains all class files of Java Standard Edition like java.lang package classes, java.net package classes, java.util package classes, java.io package classes, java.sql package classes, etc.
- **Extension ClassLoader**: This is the child classloader of Bootstrap and parent classloader of System classloader. It loads the jar files located inside $JAVA_HOME/jre/lib/ext directory.
- **System/Application ClassLoader**: This is the child classloader of Extension classloader. It loads the class files from the classpath. By default, the classpath is set to the current directory.
-

5)What kind of variables a class can consist of?

Ans: A class consists of a Local variables, instance variables and class variables.

6)What is a Local Variable?

Ans: Variables defined inside 'methods', 'constructors' or 'blocks' are called 'local variables'. The variable will be declared and initialized within the method and it will be destroyed when the method gets completed.

7)What are the different types of references in java?

Ans:

Java has a more expressive system of reference than most other garbage-collected programming languages, which allows for special behavior for garbage collection. A normal reference in Java is known as a strong reference. The java.lang.ref package defines three other types of references—soft, weak and phantom references. Each type of reference is designed for a specific use.

- A **SoftReference** can be used to implement a cache. An object that is not reachable by a strong reference (that is, not strongly reachable) but is referenced by a soft reference is called softly reachable. A softly reachable object may be garbage collected at the discretion of the garbage collector. This generally means that softly reachable objects will only be

garbage collected when free memory is low, but again, it is at the discretion of the garbage collector. Semantically, a soft reference means "keep this object unless the memory is needed."

- A **WeakReference** is used to implement weak maps. An object that is not strongly or softly reachable, but is referenced by a weak reference is called weakly reachable. A weakly reachable object will be garbage collected during the next collection cycle. This behavior is used in the class java.util.WeakHashMap. A weak map allows the programmer to put key/value pairs in the map and not worry about the objects taking up memory when the key is no longer reachable.
- A **PhantomReference** is used to reference objects that have been marked for garbage collection and have been finalized, but have not yet been reclaimed. An object that is not strongly, softly or weakly reachable, but is referenced by a phantom reference is called phantom reachable. This allows for more flexible cleanup than is possible with the finalization mechanism alone. Semantically, a phantom reference means "this object is no longer needed and has been finalized in preparation for being collected."
-

8)What is an Instance Variable?

Ans: Instance variables are variables within a class but outside a method. These variables are instantiated when the class is loaded.

9)What is difference between instanceof and isInstance(Object obj)?

Ans: Differences are as follows :

1. instanceof is a reserved word of Java, but isInstance(Object obj) is a method of java.lang.Class.
2. instanceof is used of identify whether the object is type of a particular class or its subclass but isInstance(obj) is used to identify object of a particular class.
3.

10)What is a Class Variable?

Ans: Class variables are the variables that are declared within a class, outside any method and with the 'static' keyword

11)List the three steps that are performed for creating an Object for a class?

Ans: An Object is first declared, instantiated and then it is initialized.

12)What are the different types of inner classes?

Ans:

- An 'inner class' is part of the implementation of its enclosing class (or classes). As such, it has access to the private members of any enclosing class. Top-level nested classes are declared with 'static' keyword.
- Top level inner classes can be accessed / instantiated without an instance of the outer class. Can access only static members of outer class. Cannot access instance variables or methods of the enclosing class.

- Non-static inner classes which are declared without 'static' keyword cannot exist without enclosing class. Can access all the features (even private) of the enclosing outer class.
- Local classes are defined inside a block (could be a method, a constructor, a local block, a static initializer or an instance initializer). Cannot be specified with 'static' modifier.

A class cannot have non-static inner interface. All inner classes except anonymous can be 'abstract' or 'final'.

13)What is difference between instanceof and isInstance(Object obj)?

Ans: Differences are as follows :

1. instanceof is a reserved word of Java, but isInstance(Object obj) is a method of java.lang.Class.
2. instanceof is used of identify whether the object is type of a particular class or its subclass but isInstance(obj) is used to identify object of a particular class.
3. Abstract
4.

14)When an obj is passed through a function, one can set the properties but cannot set a new memory location?

Ans: It is because when u pass an object the address value is passed and stored in some new address like if address 1234 is passed, it is stored in 4567 location.

So if u change in the value of an object it will take the address from 4567 and do 1234.setXXX(). If you set the object to null it will set 4567 = null.

Constructor Interview Question

1)Define a constructor.

Ans: Constructor is a special method provided in OOP language for creating and initializing an object.In java, the role of a constructor is only to initialize an object and new key role is creating an object.

2)How many types of constructors are used in Java?

Ans: Based on the parameters passed in the constructors, there are two types of constructors in Java.

- **Default Constructor:** default constructor is the one which does not accept any value. The default constructor is mainly used to initialize the instance variable with the default values. It can also be used for performing some useful task on object creation. A default constructor is invoked implicitly by the compiler if there is no constructor defined in the class.
- **Parameterized Constructor:** The parameterized constructor is the one which can initialize the instance variables with the given values. In other words, we can say that the constructors which can accept the arguments are called parameterized constructors.
-

3) What is the purpose of a default constructor?

Ans: The purpose of the default constructor is to assign the default value to the objects. The java compiler creates a default constructor implicitly if there is no constructor in the class.

4) Is constructor inherited?

Ans: No, The constructor is not inherited.

5) What do you understand by copy constructor in Java?

Ans: There is no copy constructor in java. However, we can copy the values from one object to another like copy constructor in C++.

There are many ways to copy the values of one object into another in java. They are:

- By constructor
- By assigning the values of one object into another
- By clone() method of Object class

6)Is it mandatory to define a constructor in a class?

Ans: No, it is optional. If we do not define a constructor, compiler will automatically consider it as a default constructor.

7)Can we define a method with same name as that of the class?

Ans: Yes, it is allowed to define a method with same name as that of a class. No compile time error or runtime error will occur, but this practice is not recommended as per coding standards.

8)How can a compiler and a JVM differentiate between constructor and method definitions if both have same name as the class has?

Ans: A compiler and a JVM can differentiate between the definitions of the the constructor and method with the help of return type. If there is a return type then it is considered as a method else it is a constructor.

9)What are the rules for defining a constructor?

Ans:

- Constructor name should be same as class name.
- It should not contain return type.
- It should not contain non Access Modifiers such as final ,static, abstract, synchronized etc.
- A logic return statement with a value is not allowed.
- It can have all four accessibility modifiers i.e. private , public, protected and default.
- It can have parameters.
- It can have a 'throws' clause which means we can throw an exception from a constructor.
- It can have a logic and as a part of logic it can have all java legal statements except the return statement with value.
- We cannot place 'return' in constructor.

10)Why a return type is not allowed for constructor?

Ans: As there is a possibility of a method having the same name as class name, return type is not allowed in a constructor to differentiate constructor block from method block.

11)Can we declare constructor as 'private'?

Ans:

- Yes we can declare a constructor as private.
- All four access modifiers are allowed for a constructor.
- We can declare a constructor as private if we do not want to allow a user to create an object from outside that class.
- Basically we declare private constructor in Singleton design pattern.

12)Why a compiler given constructor is called as default constructor?

Ans: A constructor defined/given by a class is called as a default constructor because it obtains all its default properties from its class.

They are as mentioned below:

- Its accessibility modifier is same as its class accessibility modifier.
- Its name is same as class name.
- It does not have parameters and logic.

13)Why a constructor name is same as class name?

Ans: Every class object is created using the same 'new' keyword, so it must have information about the class to which it must create object. For this reason constructor name should be same as class name.

14)What is default accessibility modifier of default constructor?

Ans: The default accessibility modifier of default constructor is assigned from its class.

15)In which situation it is mandatory for the developer to provide constructor explicitly?

Ans: A developer needs to provide constructor explicitly when one needs to execute some logic at the time of object creation. This logic might be object initialized logic or some other useful logic.

16)When does a compiler provide default constructor?

Ans: Only if there is no explicit constructor defined by developer then in such situation a compiler will provide a default constructor.

17)If class has an explicit constructor then will it have a default constructor?

Ans: No. Compiler places default constructor only if there is no explicit constructor.

18)What is constructor chaining?

Ans: Constructor Chaining is a technique of calling another constructor from one constructor. 'this()' is used to call same class constructor where as 'super()' is used to call super class constructor.

```
class SuperClass{
   public SuperClass(int i){
```

```
        System.out.println("Super Class Constructor");
    }
}
class SubClass extends SuperClass{
    public SubClass (){
        this(10); //Calling same class constructor
    }
    public SubClass(int i){
        super(i); //Calling super class constructor
    }
}
```

19)Can we call sub class constructor from super class constructor?

Ans: No. There is no way to call sub class constructor from a super class constructor in Java.

20)What is No-arg constructor?

Ans: Constructor without any arguments is called no-arg constructor. Default constructor in java is always known as no-arg constructor.

```
class MyClass{
    public MyClass()
        // No-arg constructor
    }
}
```

21)What is the use of private constructor?

Ans: Private constructors are used to restrict the instantiation of a class. When a class needs to prevent other classes from creating it's objects then private constructors are suitable for that. Objects to the class which has only private constructors can be created within the class. A very good use of private constructor is in singleton pattern. This ensures that only one instance of a class exists at any point of time. Here is an example of singleton pattern using private constructor.

```
class MyClass{
    private static MyClass object = null;
    private MyClass(){
        // private constructor
    }
    public MyClass getObject(){
    if(object == null){
        object = new MyClass(); // Creating object using private constructor
    }
    return object;
    }
}
```

22)Can we overload constructors?
Ans: Yes, we can overload constructors.

23)Does a constructor create the object?
Ans: 'New' operator in Java creates the objects. Constructor comes in the later stage in object creation. Constructor's job is to initialize the members after the object has reserved memory for itself.

24)If we place a return type in a constructor prototype will it throw an Error?
Ans: No, because compiler and JVM considers it as a method.

25)Can you make a constructor final?
Ans: No, the constructor can't be final.

26)Does constructor return any value?
Ans: yes, The constructor implicitly returns the current instance of the class (You can't use an explicit return type with the constructor)

27)What are the differences between the constructors and methods?

Java Constructor	Java Method
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type. The constructor is invoked implicitly.	A method must have a return type. The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be same as the class name.	The method name may or may not be same as class name.

Arrays Interview Question

1)What is ArrayStoreException in java? When will this exception occur ?
Ans: ArrayStoreException is a runtime exception which occurs when you try to store non-compatible element in an array object. These type of the elements must be compatible with the type of array object. Say for instance, you can store only string elements in an array of strings. If we try to insert an integer element in an array of strings, we will get ArrayStoreException at runtime.

```
public class DemoClass{
   public static void main(String[] args){
       Object[] stringArray = new String[5]; // No compile time error : String[] is auto-upcasted to Object[]
      stringArray[1] = "JAVA";
         stringArray[2] = 100; // No compile time error will occur, but this statement will throw java.lang.ArrayStoreException at runtime
      //This is because we are inserting an integer element into an array of string
   }
```

2)What are the drawbacks of the arrays in java?

Ans: The main drawback of the arrays is that arrays are of fixed size. We cannot change the size of an array once it has been created. Therefore, we must know how many elements we want in an array before creating it. You cannot insert or delete the elements once an array has been created. Only the value of the elements can be changed.

3)What is an anonymous array? Give example.

Ans: Remember that any class that implements an interface must implement the method headings that are declared in that interface. If that particular interface extends from other interfaces, then the implementing class must also implement the methods in the interfaces that are being extended or derived from. As shown in the example above, if we have a class that implements the Car interface, then that class must define the method headings in both the 'Car' interface and the 'Vehicle' interface. Anonymous array is an array without a reference. For example:

```
public class MainClass{
   public static void main(String[] args){
      //Creating anonymous arrays
      System.out.println(new int[]{1, 2, 3, 4, 5}.length); //Output : 5
      System.out.println(new int[]{21, 14, 65, 24, 21}[1]); //Output : 14
   }
}
```

4)Can we pass a negative number as an array size ?

Ans: No. We cannot pass a negative integer as an array size. If we do so, there will be no compile time error but NegativeArraySizeException will be prompted at runtime.

```
public class MainClass{
   public static void main(String[] args){
      int[] array = new int[-5]; //No compile time error
        //but you will get java.lang.NegativeArraySizeException at run time
   }
}
```

5)Can you change the size of an array once it has been defined? OR Can you insert or delete the elements after creating an array?

Ans: No. We cannot change the size of an array once it has been defined. We cannot insert or delete the elements after creating an array. Only the value of the elements can be changed.

6)What is the difference between int[] a and int a[] ?

Ans: Both are the legal methods to declare the arrays in java.

7)"int a[] = new int[3]{1, 2, 3}" – is it a legal way of defining the arrays in java?

Ans: No. We cannot mention the size of an array when we are providing the array contents.

8)There are two array objects of int type. One contains 100 elements and another one contains 10 elements. Can you assign array of 100 elements to an array of 10 elements?

Ans: Yes, you can assign array of 100 elements to an array of 10 elements provided they should be of same type. While assigning, compiler only checks the type of the array and not the size.

```
public class MainClass{
   public static void main(String[] args){
      int[] a = new int[10];
      int[] b = new int[100];
      a = b; //Compiler checks only type, not the size
   }
}
```

9)Differenciate between Array and ArrayList in java.

Ans:

Array	ArrayList
Arrays are of fixed length.	ArrayList is of variable length.
You cannot change the size of an array once it has been created.	Size of the ArrayList grows and shrinks as you add or remove the elements.
Arrays are of fixed length.	ArrayList is of variable length.
You can use arrays to store both primitive types as well as reference types.	You can store only reference types in an ArrayList.

10)How do you check the equality of two arrays in java? OR How can we compare the two arrays in java?

Ans: You can use Arrays.equals() method to compare one dimensional arrays and to compare multi-dimensional arrays, use Arrays.deepEquals() method.

11)What are the different ways of copying an array into another array?

Ans: There are four methods available in java to copy an array.

- Using 'for' loop.
- Using System.arraycopy() method.

- Using Arrays.copyOf() method.
- Using clone() method.

12)What is ArrayIndexOutOfBoundsException in java and when does it occur?

Ans: ArrayIndexOutOfBoundsException is a runtime exception which occurs when your program tries to access invalid index of an array i.e negative index or index higher than the size of the array.

13)How can you sort the array elements?

Ans: You can sort the array elements using Arrays.sort() method. This method internally uses quick sort algorithm to sort the array elements.

```
import java.util.Arrays;
public class MainClass{
   public static void main(String[] args){
      int[] a = new int[]{45, 12, 78, 34, 89, 21};
      Arrays.sort(a);
      System.out.println(Arrays.toString(a));
   }
}
```

14)Java supports both multi-dimensional and nested arrays. True/False?

Ans: False [Java does not support multi-dimensional arrays. It only supports nested arrays.]

15)How can you find the intersection of two arrays in java ?

Ans: This is one of the most common java interview question asked to freshers as well as to experienced java professionals of 1 or 2 years. We will discuss couple of methods to find common elements between two arrays.

- Using iterative Method: In this method, we iterate both the given arrays and compare each element of one array with elements of other array. If the elements are found to be equal, we will add that element into HashSet. This method also works for those arrays which contain duplicate elements.
- Using retainAll() method : This is one of the easiest methods to find the common elements from two arrays. In this method, we create two HashSets using given two arrays and then use reatinAll() method of HashSet to retain only common elements from the two sets.

16)How do you find duplicate elements in an array ?

Ans: This is one of the most asked java interview program for freshers. We have discussed two methods of finding duplicates in an array.

- **Using Brute Force Method :**In this method, we compare each element of an array with other elements. The performance of this method is very low if an array contains lots of elements. Therefore, this method is not recommended in real time.
- **Using HashSet :**HashSet contains only unique elements. HashSet never allows duplicate elements. We use this property of HashSet to find duplicates in an array. We try to add each

element of an array into HashSet using add() method. This method will return true if an element is added successfully otherwise it returns false.

Here is the java program which uses both the methods to find duplicate elements in an array.

17)What are the different ways of declaring multidimensional arrays in java?

Ans:

- dataType[][] arrayRefVar;
- dataType [][]arrayRefVar;
- dataType arrayRefVar[][];
- dataType []arrayRefVar[];

18)While creating the multidimensional arrays, can you specify an array dimension after an empty dimension?

Ans:

No. You cannot specify an array dimension after an empty dimension while creating multi-dimensional arrays. It gives compile time error.

```
int[][][] a = new int[][5][]; // Compile time error
int[][][] b = new int[5][][5]; //Compile time error
int[][][] c = new int[][5][5]; // Compile time error
```

19)How do you search an array for a specific element ?

Ans: You can search an array to check whether it contains the given element or not using Arrays.binarySearch() method. This method internally uses binary search algorithm to search for an element in an array.

20)What value does array elements get, if they are not initialized ?

Ans: They get default values.

21)What are the different ways to iterate over an array in java ?

Ans: 1) Using normal 'for' loop

```
public class MainClass{
   public static void main(String[] args){
      int[] a = new int[]{48, 21, 87, 43, 87, 22};
      // Iterating over an array using normal for loop
      for (int i = 0; i < a.length; i++){
         System.out.println(a[i]);
      }
   }
}
```

2) Using extended 'for' loop

```
public class MainClass{
   public static void main(String[] args){
```

```
        int[] a = new int[]{48, 21, 87, 43, 87, 22};
        // Iterating over an array using extended for loop
        for (int i : a){
            System.out.println(i);
        }
    }
}
```

22)How do you find second largest element in an array of integers ?

Ans: This is also one of the java interview program asked many times to java freshers to check the candidate's logical ability and understanding of the language's fundamentals. While writing the program you should not use any sorting methods or any collection types. You should find the second largest number in the given array by iterating the array only once. These are the likely conditions interviewer may ask you to follow.

23)How do you find all pairs of elements in an array whose sum is equal to a given number?

Ans: Given an array of integers, you have to find all pairs of elements in this array such that whose sum must be equal to a given number. For instance if {4, 5, 7, 11, 9, 13, 8, 12} is an array and 20 is the given number, then you have to find all pairs of elements in this array whose sum must be 20. In this example, (9, 11), (7, 13) and (8, 12) are such pairs whose sum is 20.

24)How do you separate zeroes from non-zeroes in an integer array ?

Ans: Given an integer array, you have to separate all zero elements from non-zero elements. You have to move zeroes either to end of the array or bring them to beginning of the array. For example, if {14, 0, 5, 2, 0, 3, 0} is the given array, then moving zeros to end of the array will result {14, 5, 2, 3, 0, 0, 0} and bringing zeroes to front will result {0, 0, 0, 14, 5, 2, 3}. In this post, we will see both as how to move zeros to end and front of an array.

25)How do you find continuous sub array whose sum is equal to a given number?

Ans: You have been given an integer array and a number. You need to find the continuous sub array of the given array whose sum is equal to given number. For example, If {12, 5, 31, 9, 21, 8} is the given array and 45 is the given number, then you have to find continuous sub array in this array such that whose elements add up to 45. In this case, {5, 31, 9} is such sub array whose elements add up to 45.

26)Are array operations faster or Vector operations?

Ans: Array operations are faster.

27)Arrays are always initialized whether it is local or of class level. True/False?

Ans: True.

Immutable Class Interview Question

1)What is an immutable class?

Ans: Immutable class is a class which when once created, it's contents cannot be changed under any circumstances. Immutable objects are the objects whose state cannot be changed once constructed. Say for instance 'String class'.

2)How to create an immutable class?

Ans: To create an immutable class one needs to follow the following steps as mentioned below:

1. Create a final class.
2. Set the values of properties using constructor only.
3. Make the properties of the class final and private.
4. Do not provide any setters for these properties.
5. If the instance fields include references to mutable objects, don't allow those objects to be changed
6. Don't provide methods that modify the mutable objects.
7. Don't share references to the mutable objects. Never store references to external, mutable objects that are passed to the constructor; if necessary, create copies and store references to the copies. Similarly, create copies of your internal mutable objects when it is necessary to avoid returning the originals in your methods.

```
public final class FinalPerson {
   private final String name;
   private final int age;
   public FinalPerson(final String name, final int age) {
      this.name = name;
      this.age = age;
   }
   public int getAge() {
      return age;
   }
   public String getName() {
      return name;
   }
}
```

3)Immutable objects are automatically thread-safe. True or false?

Ans: True. Since the state of the immutable objects cannot be changed once they are created they are automatically synchronized and are thread-safe

4)What is an immutable object? Can you write an immutable object ?

Ans: Immutable classes are Java classes whose objects cannot be modified once they are created. Any modification in immutable object will result into new object. For example 'isString' is immutable in Java. Mostly immutable are also final in Java in order to prevent sub class from overriding methods in Java which can compromise on immutability. You can achieve same functionality by making a member which is not final but private and cannot be modified except in constructor.

5)What is the difference between creating String as new() and literal?

Ans: When we create a string with new() operator, it's created in heap and not added into string pool whereas string created using literal is added in the string pool itself which exists in PermGen area of heap.

String s = new String("Profound edutech");

The above shown example will not put the object in string pool and so we need to call String.intern() method which is used to put them into string pool explicitly. This will only happen when you make a

string object as string literal e.g. String s = "Javabykiran_Test". Java will automatically add this into the string pool.

6)Write a code in order to find the first non-repeating character in the String.

Ans: This is another good question related to Java asked in interviews. This kind of question is mainly asked by Amazon and other such equivalent companies.

7)Which classes in java are immutable?

Ans: All wrapper classes in java.lang are immutable which includes String, Integer, Boolean, Character, Byte, Short, Long, Float, Double, BigDecimal, BigInteger.

8)What are the advantages of immutability?

Ans:

- Immutable objects are thread-safe by default and apart from this, the overhead caused due to the use of synchronization is avoided.
- Once created the state of the immutable object cannot be changed so there is no possibility of them getting into an inconsistent state.
- The references to the immutable objects can be easily shared or cached without any need to copy or clone them as there state can never be changed after its construction.
- The immutable objects can be best used as the keys of a map.
-

9)Immutable Class Example in Java.

Ans: Here is complete code which gives an example of writing immutable class in Java. We have followed simplest approach and all rules for making a class immutable, which includes making class final to avoid putting immutability at risk due to Inheritance and Polymorphism.

```
public final class Contacts{
   private final String name;
```

```
    private final String mobile;
    public Contacts(String name, String mobile) {
        this.name = name;
        this.mobile = mobile;
    }
    public String getName() {
        return name;
    }
    public String getMobile() {
        return mobile;
    }
}
```

10)Give an example of Immutable Class.

Ans:

- This Java class is immutable because it's state cannot be changed once it has been created. One can see that all of it's fields are final.
- This is one of the most simple way of creating immutable class in Java, where all fields of class also remains immutable like String in above case.
- Sometimes one may need to write immutable class which includes mutable classes like java.util.Date.Despite storing 'Date' into a 'final' field it can be modified internally if internal date is returned to the client.
- In order to preserve immutability in such cases,its advised to return copy of original object, which is also one of the Java's best practice.

Here is another example of making a class immutable in Java, which includes mutable member variable.

```
public final class Immutable {
    private final Date remindingDate;
    public Immutable(Date remindingDate) {
        if(remindingDate.getTime() < System.currentTimeMillis()) {
        throw new IllegalArgumentException("Can not set reminder” +
          “ for past time: " + remindingDate);
        }
        this.remindingDate = new Date(remindingDate.getTime());
    }
    public Date getRemindingDate() {
        return (Date) remindingDate.clone();
    }
}
```

11)What are the advantages of using Immutable class?

Ans: Some of the advantages of using immutable class are as follows:

- **Thread safe -** Immutable classes are thread safe, they will never create race condition.
- **Key in HashMap -** Immutable classes can be used as key in Map (HashMap etc.)
- **HashCode is cached -** JVM caches the HashCode of Immutable classes used in application.This will avoid JVM from calculating hashcode again. Hence, performance of application is improved significantly.
- **Immutable class throws Exception -** If Immutable class throws Exception, they are never left in undesirable state.

12)Benefits of Immutable Classes in Java.

Ans: As mentioned earlier Immutable classes offers several benefits, here are some of them as mentioned below:

- Immutable objects are by default thread safe and can be shared without synchronization in concurrent environment.
- Immutable object simplifies development because its easier to share between multiple threads without external synchronization.
- Immutable objects boost performance of Java applications by reducing synchronization in code.
- Another important benefit of Immutable objects is its reusability. One can cache Immutable objects and reuse them just like String literals and Integers. One can use static factory methods to provide methods like valueOf(), which can return an existing Immutable object from cache, instead of creating a new one.

13)What is the difference when a String gets created using literal and new() operator ?

Ans: When we create string with new() operator it gets created in heap and is not added into string pool while String created using literal operator are added in the String pool itself which exists in Perm area of heap.

Java Keywords Interview Question

1)What is the use of final keyword?

Ans: The final keyword can be assigned to :

1. Class level variable
2. Method
3. Class
4. Objects

If 'final' keyword is assigned to a variable, the variable behaves as a constant. It means that the value of variable once set cannot be changed.

```
final int i = 1;
i = 5; // error
// If 'final' keyword is assigned to a method
// then it cannot be overridden in its child class.
class Parent {
```

```
final void print() {
        System.out.println("Inside");
    }
}
class Child extends Parent {
    public final void print() {
        // error cannot override final method
        System.out.println("Inside");
    }
}
final Map map = new HashMap();
map.put("key";,"value");
map = new HashMap(); // error
```

If a class is declared as final, then no other class can extend it and make it as parent class.
Example : String Class

2)What is the use of 'synchronized' keyword?
Ans:

- This keyword is used to prevent concurrency. 'Synchronized' keyword can be applied to static/non-static methods or a block of code.
- Only one thread at a time can access synchronized methods and if there are multiple threads trying to access the same method then other threads have to wait for the execution of method to get completed by the previous thread.
- 'Synchronized' keyword provides a lock on the object and thus prevents race condition.

Example :

```
public void synchronized method(){}
public void synchronized staticmethod(){}
public void myMethod() {
    synchronized (this){
        // synchronized keyword on block of code
    }
}
```

3)Can this keyword be assigned null value?
Ans: No.

4)What will be the value of Point p after methods in a and b if the value before method call is (700, 800)?

i)
```
static void changePoint ( Point p) {
p.x = 100; p.y=200;
}
```

ii)
```
static void changePoint(Point p) {
```

```
    p = new Point(100,200);
    }
```

Ans:

1. (100,200)
2. (700,800)

[Note : Primitive type is passed by value in method parameter and objects are passed by value of the reference. In a method if the object values are changed , it will reflect, but if we try to change the reference itself its original reference/object will not change, as only copy of the reference is changed.]

5)Local variables cannot be declared static, final or transient. True/False?

Ans: True.

6)What is a transient variable?

Ans: If some of the properties of a class are not required to be serialized then the varaibles are marked as transient. When an object is deserialized the transient variables retains the default value depending on the type of variable declared and hence lost its original value.

7)What is a static variable?

Ans: Static keyword can be used with the variables and methods but not with the class. Anything declared as static is related to class and not objects.

Static variable : Multiples objects of a class shares the same instance of a static variable.

Consider the example :

```
public class Counter {
   private static int count=0;
   private int nonStaticcount=0;
   public void incrementCounter() {
      count++;
      nonStaticcount++;
   }
   public int getCount() {
      return count;
   }
   public int getNonStaticcount() {
      return nonStaticcount;
   }
   public static void main(String args[]) {
      Counter countObj1 = new Counter();
      Counter countObj2 = new Counter();
      countObj1.incrementCounter();
      countObj2.incrementCounter();
      System.out.println("Static count for Obj1: "+countObj1.getCount());
```

```
        System.out.println("NonStatic count for Obj1: "+countObj1.getNonStaticcount());
        System.out.println("Static count for Obj2: "+countObj2.getCount());
        System.out.println("NonStatic count for Obj2: "+countObj2.getNonStaticcount());
    }
}
```

Output:

```
Static count for Obj1    : 2
NonStatic count for Obj1 : 1
Static count for Obj2    : 2
NonStatic count for Obj2 : 1
```

In the above program obj1 and obj2 share the same instance of static variable count hence if the value is incremented by one object, the incremented value will be reflected across the other objects.

8)What is a static method?

Ans: A method defined as static is called static method. A static method can be accessed without creating the objects. Just by using the Class name the method can be accessed. Static method can only access static variables and not local or global non-static variables.

Example :

```
public class Test {
    public static void printMe() {
        System.out.println("Hello World");
    }
}

public class MainClass {
    public static void main(String args[]) {
        Test.printMe()
    }
}
```

Output:

```
Hello World
```

Also static method can call only static methods and not non static methods. But non-static methods can call static methods.

9)Why static methods cannot access non static variables or methods?

Ans: A static method cannot access non static variables or methods because static methods can be accessed without instantiating the class, so if the class is not instantiated the variables are not intialized and thus cannot be accessed from a static method.

10)What are the restrictions that are applied to the Java static methods?

Ans: Two main restrictions are applied to the static methods.

1:The static method can not use non-static data member or call the non-static method directly.
2:This and super cannot be used in static context as they are non-static.

11)What is static class ?

Ans: A class cannot be declared static. But a class can be said a static class if all the variables and methods of the class are static and the constructor is private. Making the constructor private will prevent the class to be instantiated. So the only possibility to access is using Class name only

12)What is throw keyword?

Ans: Throw keyword is used to throw the exception manually. It is mainly used when the program fails to satisfy the given condition and it wants to warn the application.The exception thrown should be subclass of Throwable.

```
public void parent() {
   try {
      child();
   }
   catch(MyCustomException e){ }
}
   public void child {
   String iAmMandatory = null;
   if(iAmMandatory == null) {
      throw (new MyCustomException("exception using throw keyword");
   }
}
```

13)What is use of throws keyword?

Ans: Throws clause is used to throw the exception from a method to the calling method which could decide to handle exception or throw to its calling method in a class.

```
public void parent() {
   try {
      child();
   }
   catch(MyCustomException e){ }
}
public void child throws MyCustomException {
   //put some logic so that the exception occurs.
}
```

14)Can we override the static methods?

Ans: No, we can't override static methods.

15)What is the static block?

Ans: Static block is used to initialize the static data member. It is executed before the main method, at the time of classloading.

16)What if the static modifier is removed from the signature of the main method?
Ans: Program compiles. However, at runtime, It throws an error "NoSuchMethodError."

17)Can we make constructors static?
Ans: As we know that the static context (method, block, or variable) belongs to the class, not the object. Since Constructors are invoked only when the object is created, there is no sense to make the constructors static. However, if you try to do so, the compiler will show the compiler error.

18)What is this keyword in java?
Ans: The this keyword is a reference variable that refers to the current object. There are the various uses of this keyword in Java. It can be used to refer to current class properties such as instance methods, variable, constructors, etc. It can also be passed as an argument into the methods or constructors.

19)Can you access non-static variable in static context?
Ans: You cannot access static variable in non-static context in Java. You can refer further as to why you cannot access non-static variable from static method to learn more about this tricky Java question

20)What is super in java?
Ans: The super keyword in Java is a reference variable that is used to refer to the immediate parent class object. Whenever you create the instance of the subclass, an instance of the parent class is created implicitly which is referred by super reference variable. The super() is called in the class constructor implicitly by the compiler if there is no super or this.

21)What are the main uses of the super keyword?
Ans: There are the following uses of super keyword.
Super can be used to refer to the immediate parent class instance variable.
Super can be used to invoke the immediate parent class method
super() can be used to invoke immediate parent class constructor.

22)Can you use this() and super() both in a constructor?
Ans: No, because this() and super() must be the first statement in the class constructor

23)What is the final variable?
Ans: In Java, the final variable is used to restrict the user from updating it. If we initialize the final variable, we can't change its value. Tthe final variable once assigned to a value, can never be changed after that. The final variable which is not assigned to any value can only be assigned through the class constructor.

24)What is the final method?

Ans: If we change any method to a final method, we can't override it

25)What is the final class?

Ans: If we make any class final, we can't inherit it into any of the subclasses.

26)What is the difference between static (class) method and instance method?

Ans:

static or class method	instance method
1)A method that is declared as static is known as the static method.	A method that is not declared as static is known as the instance method.
2)We don't need to create the objects to call the static methods.	The object is required to call the instance methods.
3)Non-static (instance) members cannot be accessed in the static context (static method, static block, and static nested class) directly.	Static and non-static variables both can be accessed in instance methods.

27)What are the differences between this and super keyword?

Ans: There are the following differences between this and super keyword.

The super keyword always points to the parent class contexts whereas this keyword always points to the current class context.

The super keyword is primarily used for initializing the base class variables within the derived class constructor whereas this keyword primarily used to differentiate between local and instance variables when passed in the class constructor

The super and this must be the first statement inside constructor otherwise the compiler will throw an error.

28)What is the use of final keyword in java?

Ans: By using final keyword we can make,

- Final class.
- Final method.
- Final variables.
- If we declare any class as final we can not extend that class.
- If we declare any method as final it can not be overridden in sub class.
- If we declare any variable as final its value unchangeable once assigned.

29)What is the main difference between abstract method and final method?

Ans: Abstract methods must be overridden in sub class where as final methods can not be overridden in sub class.

30)What is the actual use of final class in java?

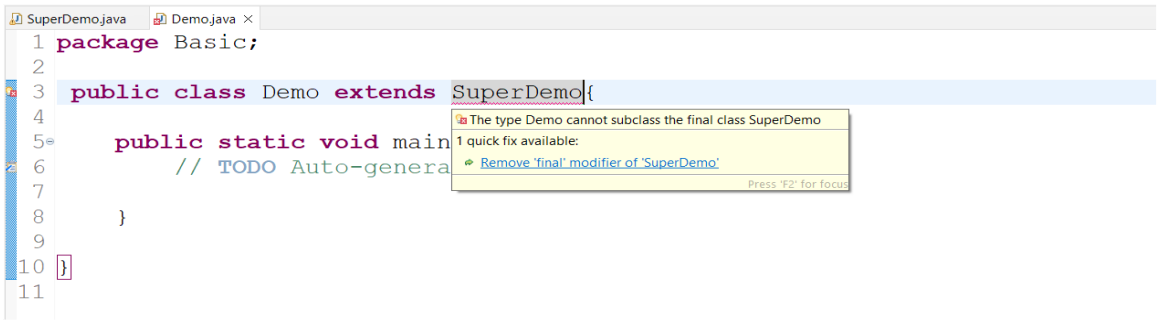
Ans:

- If a class needs some security and it should not participate in inheritance in this scenario we need to use final class.
- We can not extend final class.

31)What will happen if we try to extend final class in java?

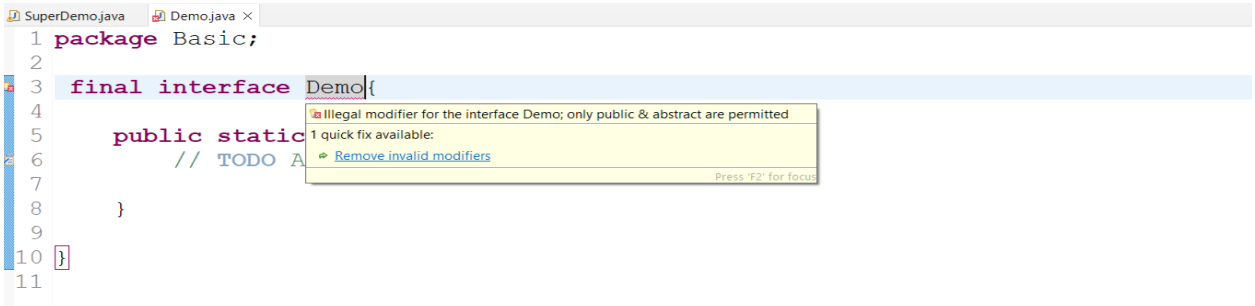
Ans:

- Package Basic;
  final class SuperDemo{
  final int a;
  }
- Package Basic;
  Public class Demo extends SuperDemo {
  //The type Demo cannot subclass the final class SuperDemo
  }



32)Can we declare interface as final?

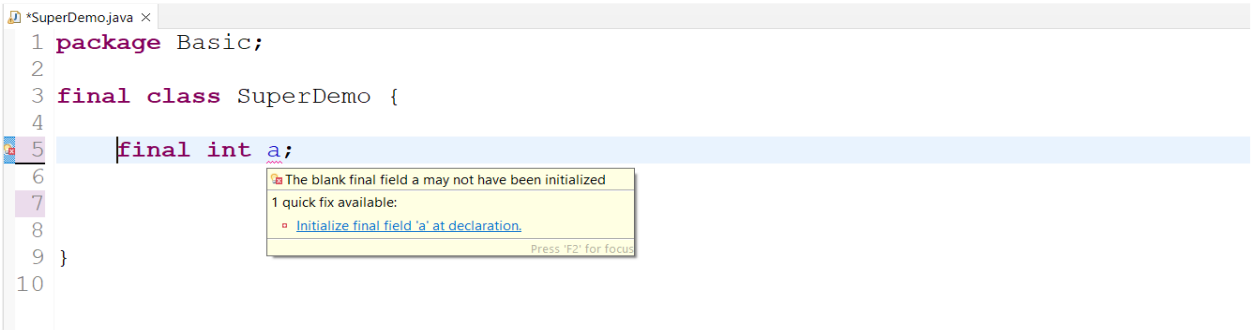
Ans: No, we cannot declare interface as final because interface should be implemented by some class so its not possible to declare interface as final.

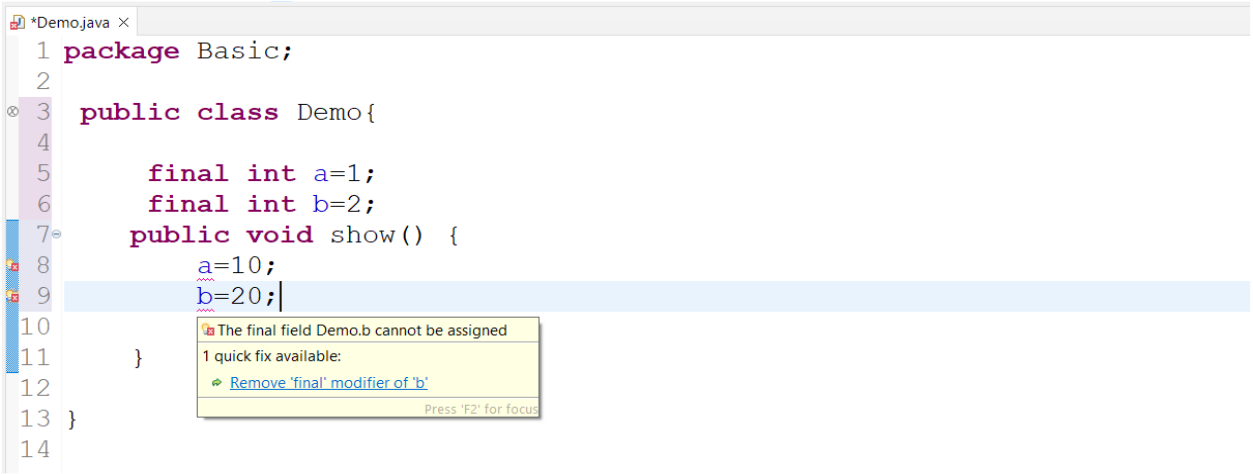


33)Is it possible to declare final variables without initialization?

Ans:

- No. It's not possible to declare a final variable without initial value assigned.
- While declaring itself we need to initialize some value and that value cannot be change at any time.





34)What will happen if we try to override final methods in sub classes?

Ans: Compile time error will come : Cannot override the final method from Super class.

35)What is the most common predefined final class object you used in your code?

Ans: String (for example).

36)What is the use of final keyword in java?

Ans: Final keyword in java is used to make any class or a method or a field as unchangeable.You can't extend a final class, you can't override a final method and you can't change the value of a final field.Final keyword is used to achieve high level of security while coding

37)What is the blank final field?

Ans: Uninitialized final field is called blank final field.

38)When do you override hashcode and equals() ?

Ans: Whenever necessary especially if you want to do equality check or want to use your object as key in HashMap

39)Can we change the state of an object to which a final reference variable is pointing?
Ans: Yes, we can change the state of an object to which a final reference variable is pointing, but we can't re-assign a new object to this final reference variable.

40)What is the main difference between abstract methods and final methods?
Ans: Abstract methods must be overridden in the sub classes and final methods are not at all eligible for overriding.

41)What is the use of final class?
Ans: A final class is very useful when you want a high level of security in your application. If you don't want inheritance of a particular class, due to security reasons, then you can declare that class as a final.

42)Can we change the value of an interface field? If not, why?
Ans: No, we can't change the value of an interface field. Because interface fields, by default, are final and static. They remain constant for whole execution of a program.

43)Where all we can initialize a final non-static global variable if it is not initialized at the time of declaration?
Ans: In all constructors or in any one of instance initialization blocks.

44)What are final class, final method and final variable?
Ans:

- final class —> cannot be extended.
- final method —> cannot be overridden in the sub class.
- final variable —> cannot change it's value once it is initialized.

45)Where all we can initialize a final static global variable if it is not initialized at the time of declaration?
Ans: In any one of static initialization blocks.

46)Can we use non-final local variables inside a local inner class?
Ans: No. Only final local variables can be used inside a local inner class.

47)Can we declare constructors as final?
Ans: No, constructors cannot be final.

48)List some Java keywords(like C, C++ keywords).
Ans: Some Java keywords are import, super, finally, etc

49)Can we use this() and super() in a method?

Ans: No, we can't use this() and super() in a method.

```
class SuperClass{
   public SuperClass(){
      System.out.println("Super Class  Constructor");
   }
}
class SubClass extends SuperClass{
   public SubClass(){
      System.out.println("Sub Class  Constructor");

   }
void method(){
      this(); // Compile time error
      super(); // Compile time error
   }
}
```

50)What are the common uses of "this" keyword in java ?

Ans: "this" keyword is a reference to the current object and can be used for the following -

1. Passing itself to another method.
2. Referring to the instance variable when local variable has the same name.
3. Calling another constructor in constructor chaining

51)Can we create object for final class?

Ans: Yes we can create object for final class.

52)What is dynamic binding and static binding?

Ans: Method invocation The Java programming language provides two basic kinds of methods: instance methods and class (or static) methods.

The differences are :

- Instance methods require an instance before they can be invoked, whereas class methods do not.
- Instance methods use dynamic (late) binding, whereas class methods use static (early) binding.

When the Java virtual machine invokes a class method, it selects the method to invoke based on the type of the object reference, which is always known at compile-time.

On the other hand, when the virtual machine invokes an instance method, it selects the method to invoke based on the actual class of the object, which may only be known at run time.

Abstract

53)What is difference between static block and the init block?

Ans: The static block is loaded when the class is loaded by the JVM for the 1st time only whereas init {} block is loaded every time class is loaded. Also first the static block is loaded then the init block.

```
public class LoadingBlocks
{
   static
   {
      System.out.println("Inside static");
   }
   {
      System.out.println("Inside init");
   }
   public static void main(String args[])
   {
      new LoadingBlocks();
      new LoadingBlocks();
      new LoadingBlocks();
   }
}
```

Output:

```
Inside static
Inside init
Inside init
Inside init
```

54)Why static methods cannot access non static variables or methods?

Ans: A static method cannot access non static variables or methods because static methods doesn't need the object to be accessed. So if a static method has non static variables or non static methods which has instantiated variables they will no be initialized since the object is not created and this could result in an error.

55)Final variables declared without initialization can be initialized in static initializer (static final var) or in constructor(final var). True/False?

Ans: True, but the most once and not more than that.

Singleton Class Interview Question

1)What is a Singleton class? Have you used Singleton class before?

Ans: Singleton is a class which has only one instance in whole application and provides a getInstance() method to access the singleton instance. There are many classes in JDK which is

implemented using Singleton pattern like java.lang.Runtime which provides getRuntime() method to get access and is used to get free memory and total memory in Java.

2)Which classes are candidates of Singleton? Which kind of class do you make Singleton in Java?

Ans:

- Any class which you want it to make available to whole application and only one instance is viable can be made as a Singleton class.
- One example of this is Runtime class as on whole java application only one runtime environment is possible and so making Runtime as singleton can be considered as right decision.
- Another example is a utility class say for instance 'Popup' in 'GUI application' where you can show popup with a message. Here you can have one PopUp class on whole GUI application where you just need to get instance anytime and call 'show()' method with message

3)What is lazy and early loading of Singleton and how to implement it?

Ans:

- There are a number of ways to implement Singleton class like using double checked locking or Singleton class with static final instance that can be initialized during class loading.
- The former is called 'lazy loading' because Singleton instance is created only when client calls 'getInstance()' method while later is called 'early loading' because Singleton instance is created when class is loaded into memory.

4)Give some examples of Singleton pattern from Java Development Kit (JDK).

Ans: There are many classes in Java Development Kit which is written using singleton pattern, some of which are as mentioned below:

1. Java.lang.Runtime with getRuntime() method
2. Java.awt.Toolkit with getDefaultToolkit()
3. Java.awt.Desktop with getDesktop()

5)What is double checked locking in Singleton?

Ans: Double checked locking' in singleton is a technique to prevent the creation of another instance of Singleton when a call is made to the 'getInstance()' method in multi-threading environment. In 'Double checked locking pattern' as shown in below example, singleton instance is checked two times before initialization.

```
public static Singleton getInstance()
{
   if(_INSTANCE == null)
   {
      synchronized(Singleton.class)
```

```
    {
      // double checked locking -
      // because second check of Singleton instance with lock
      if(_INSTANCE == null)
      {
        _INSTANCE = new Singleton();
      }
    }
  }
  return _INSTANCE;
}
```

'Double checked locking' should only be used when you have requirement for 'lazy initialization' otherwise use 'Enum' to implement 'singleton' or 'simple static final variable'.

6)How do you prevent from creating another instance of Singleton using clone() method?

Ans: It is advised not to implement 'Cloneable' interface by creating 'clone()' method Singleton. One can do this using throw Exception from clone() method as “Cannot create clone of Singleton class”.

7)How do you prevent from creating another instance of Singleton using reflection?

Ans: Since constructor of Singleton class is supposed to be private, it prevents creating instance of Singleton from outside but Reflection can access private fields and methods, which opens a threat of another instance. This can be avoided by throwing Exception from constructor as “Singleton already initialized”

8)How do you prevent from creating another instance of Singleton during serialization?

Ans: You can prevent the creation of another instance of singleton during serialization using 'readResolve()' method, since during serialization 'readObject()' method is used to create instance and it returns new instance every time but by using 'readResolve()' method you can replace it with original Singleton instance.

9)Why one should avoid the singleton anti-pattern and replace it with DI?

Ans: Singleton Dependency Injection: every class that needs access to a singleton gets the object through its constructors or with a DI-container.

10)Why Singleton is anti pattern

Ans: With more and more classes calling 'getInstance()' method the code gets more and more tightly coupled, monolithic, not testable and hard to change as well as hard to reuse because of non-configurable and hidden dependencies. Also, there would be no need for this clumsy double checked locking if you call 'getInstance()' method less often (i.e. once a while).

11)In how many ways you can write Singleton Class in Java?

Ans: One can implement Singleton pattern in Java in four ways:

1. Singleton by synchronizing 'getInstance()' method.
2. Singleton with 'public static final' field initialized during class loading.
3. Singleton generated by 'static' nested class, also referred as 'Singleton' holder pattern.
4. From Java 5 onwards, Singleton class in Java can be written using Enum.

12)How to write thread-safe Singleton in Java?

- Thread safe Singleton usually refers to writing thread safe code which creates one and only one instance of Singleton if called by 'multiple' threads at the same time.
- There are many ways to achieve this say for instance by using 'double checked locking' technique as shown above and by using 'Enum' or 'Singleton' initialized by class loader

OOPs Interview Question

1)What are different oops concept in java?

Ans: OOPs stands for Object Oriented Programming. The concepts in oops are similar to any other programming languages. Basically, it is program agnostic. The different OOps concepts are :

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism
- Aggregation
- Composition
- Association

2)What is an abstraction ?

Ans: Abstraction is a way of converting real world objects in terms of class. Its a concept of defining an idea in terms of classes or interface. For example creating a class Vehicle and injecting properties into it. E.g

```
public class Vehicle
{
public String colour;
   public String model;
}
```

3)What is Encapsulation?

Ans: The encapsulation is achieved by combining the methods and attribute into a class. The class acts like a container encapsulating the properties. The users are exposed mainly public methods.The idea behind is to hide how things work and just exposing the requests a user can do.

4)What is inheritance?

Ans: Inheritance allows a Child class to inherit properties from its parent class. In Java this is achieved by using extends keyword. Only properties with access modifier public and protected can be accessed in child class.

```
public class Parent {
   public String parentName;
   public String familyName;
   protected void printMyName()
   {
      System.out.println(“ My name is “+ chidName+” “ +familyName);
   }
}
public class Child extends Parent {
   public String childName;
   public int childAge; //inheritance
   protected void printMyName()
   {
      System.out.println(“ My child name is “+ chidName+” “ +familyName);
   }
}
```

5)What is polymorphism?

Ans: The ability to define a function in multiple forms is called Polymorphism. In java, c++ there are two types of polymorphism: Compile time polymorphism (overloading) and Runtime polymorphism (overriding).

Mehtod overriding : Overriding occurs when a class method has the same name and signature as a method in parent class. When you override methods, JVM determines the proper methods to call at the program’s run time, not at the compile time.

Overloading : Overloading is determined at the compile time. It occurs when several methods have same names with:

- Different method signature and different number or type of parameters.
- Same method signature but different number of parameters.
- Same method signature and same number of parameters but of different type.

Example of Overloading :

```
int add(int a,int b)
float add(float a,int b)
float add(int a ,float b)
void add(float a)
int add(int a)
void add(int a) //error conflict with the method int add(int a)

class BookDetails
{
```

```
   String title;
   setBook(String title){}
}
class ScienceBook extends BookDetails
{
   setBook(String title){} //overriding
   setBook(String title, String publisher,float price){} //overloading
}
```

6)What is Aggregation?

Ans: Aggregation is a specialize form of Association where all object have their own lifecycle but there is ownership and child object can not belongs to another parent object. Let's take an example of Department and teacher. A single teacher can not belongs to multiple departments, but if we delete the department teacher object will not destroy. We can think about "has-a" relationship.

7)What is Composition ?

Ans: Composition is again specialize form of Aggregation and we can call this as a "death" relationship. It is a strong type of Aggregation. Child object dose not have their lifecycle and if parent object deletes all child object will also be deleted. Let's take again an example of relationship between House and rooms. House can contain multiple rooms there is no independent life of room and any room can not belongs to two different house if we delete the house room will automatically delete.

8)What is Association?

Ans: Association is a relationship where all object have their own lifecycle and there is no owner. Let's take an example of Teacher and Student. Multiple students can associate with single teacher and single student can associate with multiple teachers but there is no ownership between the objects and both have their own lifecycle. Both can create and delete independently.

Encapsulation Interview Question

1)What is the primary benefit of encapsulation?

Ans: The main benefit of encapsulation is the ability to modify the implemented code without breaking the code of others who use our code. It also provides us with maintainability, flexibility and extensibility to our code.

2)What is the difference between encapsulation and abstraction?

Ans:

- Abstraction solves the problem at design level while encapsulation solves the problem at implementation level.

- Abstraction is used for hiding the unwanted data and provide only the required data. On the other hand encapsulation means hiding the code and data into a single unit to protect the data from outside world.
- **4)Does garbage collection guarantee that a program will not run out of memory?**
- **Ans:** Garbage collection does not guarantee that a program will not run out of memory. It is possible for programs to use up memory resources faster than they are garbage collected. It is also possible for programs to create objects that are not subject to garbage collection.Abstraction lets you focus on what the object does instead of how it does it while Encapsulation means hiding the internal details or mechanics of how an object does something.
- For example: Outer Look of a Television i.e. it has a display screen and channel buttons to change channel explains 'abstraction' but inner implementation detail of a television i.e. how CRT and display screen are connected with each other using different circuits explains 'encapsulation'.

3)What are the features of encapsulation?

Ans:

- Encapsulation means combining the data of our application and its manipulation at one place.
- Encapsulation allows the state of an object to be accessed and modified through behaviour.
- It reduces the coupling of modules and increases the cohesion inside them.

4)Explain in detail encapsulation in Java?

Ans:

- Encapsulation is nothing but protecting anything which is prone to change. Rational behind encapsulation is that if any functionality which is well encapsulated in code i.e maintained in just one place and not scattered around code is easy to change.
- This can be better explained with a simple example of encapsulation in Java. We all know that constructor is used to create object in Java and constructor can accept argument. Suppose we have a class 'Loan' which has a constructor and in various classes we have created instance of 'loan' using this constructor. Now requirements will change and you need to include 'age of borrower' as well while taking loan.
- Since this code is not well encapsulated i.e. not confined in one place you need to change at every place where you are calling this constructor i.e. for one change you need to modify several files instead of just one file which is more error prone and tedious. Though it can be done with refactoring feature of advanced IDE it would prove better if we only need to make change at one place.
- This is possible if we encapsulate 'Loan' creation logic in one method say 'createLoan()'. The code written for client will call this method and this method internally creates 'Loan' object. in this case you only need to modify this method instead of the whole client code.

5)Give an example of Encapsulation in Java.

Ans:

```
Class Bank {
   private int duration; //private variables examples of encapsulation
   private String loan;
   private String borrower;
   private String salary;

   //public constructor can break encapsulation instead use factory method
   private Bank(int duration, String loan, String borrower, String salary){
      this.duration = duration;
      this.loan = loan;
      this.borrower = borrower;
      this.salary = salary;
   }
//no argument constructor omitted here
// create loan can encapsulate loan creation logic
   public Bank createLoan(String loanType){
      //processing based on loan type and than returning loan object
      return loan;
   }
}
```

6)What are the advantages of using encapsulation in Java and OOPS?

Ans: Below mentioned are few advantages of using encapsulation while writing code in Java or any Object oriented programming language:

- Encapsulated Code is more flexible and easy to change with inception of new requirements.
- Encapsulation in Java makes unit testing easy.
- Encapsulation in Java allows you to control who can access what.
- Encapsulation also helps to write immutable class in Java which is a good choice in multi-threading environment.
- Encapsulation reduces coupling of modules and increases cohesion inside a module because all the pieces of one thing are encapsulated in one place.
- Encapsulation allows you to change one part of code without affecting other part of code.
- What should you encapsulate in code?

Anything which can be changed or which is more likely to be changed in near future is candidate of encapsulation. This also helps to write more specific and cohesive code. For instance object creation code, code which can be improved in future like sorting and searching logic.

7)Mention some important points about encapsulation in Java.

Ans:

- "Whatever changes encapsulate it" is a famous design principle.
- Encapsulation helps in loose coupling and high cohesion of code.

- Encapsulation in Java is achieved using access modifiers private, protected and public.
- 'Factory pattern' and 'singleton pattern' in Java makes good use of encapsulation.

8)Explain design pattern based on encapsulation in Java.

Ans:

- Many design patterns in Java uses encapsulation concept, one of them is 'factory pattern' which is used to create objects.
- 'Factory pattern' is a better choice then 'new' operator for creating object of those classes whose creation logic can vary and also for creating different implementations of same interface.
- 'BorderFactory class' of JDK is a good example of encapsulation in Java which creates different types of 'border' and encapsulates creation logic of border.
- 'Singleton pattern' in Java also encapsulates how you create instance by providing getInstance() method.
- Since object is created inside one class and not from any other place in code you can easily change how you create object without effecting other part of code.

9)What are the benefits of encapsulation?

Ans:

- The fields of a class can be made read-only or write-only.
- A class can have total control over what is stored in its fields.
- The users of a class do not know how the class stores its data. A class can change the datatype of a field and users of the class do not need to make any changes to their code.

10)Give an example of how to achieve encapsulation in Java?

Ans: To achieve encapsulation in Java:

- Declare the variables of a class as 'private'.
- Provide public setter and getter methods to modify and view the variable's values.

Below given is an example that demonstrates how to achieve Encapsulation in Java:

```
public class Encap{
   private String name;
   private String idNum;
   private int age;
public int getAge(){
      return age;
   }
public String getName(){
      return name;
   }
   public String getIdNum(){
      return idNum;
   }
```

```
   public void setAge( int newAge){
      age = newAge;
   }
   public void setName(String newName){
      name = newName;
   }
   public void setIdNum( String newId){
      idNum = newId;
   }
}
```

Inheritance Interview Question

1)What is the Inheritance?

Ans: Inheritance is a mechanism by which one object acquires all the properties and behavior of another object of another class. It is used for Code Reusability and Method Overriding. The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also. Inheritance represents the IS-A relationship which is also known as a parent-child relationship.

2)How many types of inheritance is supported in java?

Ans: There are 3 types of inheritance in Java.

- Single-level inheritance
- Multi-level inheritance
- Hierarchical Inheritance

3)Why we need to use Inheritance?

Ans:

- For Code Reusability.
- For Method Overriding

4)Which of these keyword must be used to inherit a class?

a) super
b) this
c) extent
d) extends

Ans: d)extends

5)What is syntax of inheritance?

Ans:

```
public class subclass extends superclass {
   // all methods and variables declare here
}
```

6)Which class is the superclass for all the classes?

Ans: The object class is the superclass of all other classes in Java.

7)What is the output of this program?

```
class A {
   int i;
   void display() {
      System.out.println(i);
   }
}
class B extends A {
   int j;
   void display() {
      System.out.println(j);
   }
}
Class demo {
    public static void main(String args[]){
      B obj = new B();
      obj.i=1;
      obj.j=2;
      obj.display();
   }
}
```

Options :

a) 0
b) 1
c) 2
d) Compilation Error

Ans: c)2

Explanation : class A & class B contain display() method, class B inherits class A, when display() method is called by object of class B, display() method of class B is executed rather than that of Class A.

Output:

```
$ javac demo.java
$ java demo 2
```

8)How Inheritance can be implemented in java?

Ans: Inheritance can be implemented in JAVA using below two keywords :

- extends
- implements
- **extends** is used for developing inheritance between two classes and two interfaces.
- **implements** keyword is used to developed inheritance between interface and class.

9)What is multilevel inheritance?

Ans: Multilevel inheritance getting the properties from one class object to another class object level wise with different priorities.

10)Does Java support Multiple Inheritance ?

Ans: No, Java doesn't support multiple inheritance. Interfaces doesn't facilitate inheritance and hence implementation of multiple interfaces doesn't make multiple inheritance.

11)What is Multiple inheritance? Why Java doesn't support multiple inheritance?

- The concept of getting the properties from multiple class objects to sub class object with same priorities is known as multiple inheritance.
- In multiple inheritance there is every chance of multiple properties of multiple objects with the same name available to the sub class object with same priorities leads for the ambiguity also known as diamond problem. One class extending two super classes.
- Because of multiple inheritance there is chance of the root object getting created more than once.
- Always the root object i.e object of object class hast to be created only once.
- Because of above mentioned reasons multiple inheritance would not be supported by java.
- Thus in java a class cannot extend more than one class simultaneously. At most a class can extend only one class.

12)How do you restrict a member of a class from inheriting to it’s sub classes?

Ans: By declaring that member as a private. Because, private members are not inherited to sub classes.

13)Which of these keywords is used to refer to member of base class from a sub class?

a) upper
b) super
c) this
d) None of the mentioned.

Ans: b)super

Explanation : Whenever a subclass needs to refer to its immediate superclass, it can do so by use of the keyword super.

14)How do you implement multiple inheritance in java?

Ans: Using interfaces java can support multiple inheritance concept in java. In java we cannot extend more than one classes, but a class can implement more than one interfaces.

Program:

```
interface A {
}

interface B {
}

class C implements A,B {
}
```

15)What is the output of this program?

```
class A {
   int i;
}
class B extends A {
   int j;
   void display() {
      super.i = j + 1;
      System.out.println(j + " " + i);
   }
}
class demo {
    public static void main(String args[]){
      B obj = new B();
      obj.i=1;
      obj.j=2;
      obj.display();
   }
}
```

Options :

a) 2 2
b) 3 3
c) 2 3
d) 3 2

Ans : c)2 3

Output:

```
$ javac demo.java
$ java dmeo 2 3
```

16)Can a class extend itself?

Ans: No, A class can't extend itself.

17)What happens if super class and sub class having same field name?
Ans: Super class field will be hidden in the sub class. You can access hidden super class field in sub class using super keyword.

18)Are constructors inherited? Can a subclass call the parent's class constructor? When?
Ans: You cannot inherit a constructor. That is, you cannot create a instance of a subclass using a constructor of one of it's superclasses.One of the main reasons is because you probably don't want to override the superclasses constructor, which would be possible if they were inherited.By giving the developer the ability to override a superclasses constructor you would erode the encapsulation abilities of the language.

19)Can we declare derived class first and then base class in Java?
Ans: Yes, we can declare derived class first and then base class in Java.

20)You have reference variable of parent class type and you assign a child class object to that variable and invoke static method. Which method will be invoked, parent or child?
Ans: Parent method will be invoked first.

21)Can we cast two derived classes for each other, both having same parent class?
Ans: No.

22)What is the output of this program?

```
class A {
   public int i;
   private int j;
}
class B extends A {
   void display() {
      super.j = super.i + 1;
      System.out.println(super.i + " " + super.j);
   }
}
class demo {
   public static void main(String args[]){
      B obj = new B();
      obj.i=1;
      obj.j=2;
      obj.display();
   }
}
```

Options :

a) 2 2

b) 3 3

c) Runtime Error

d) Compilation Error

Ans : d)Compilation Error.

Explanation : class contains a private member variable j, this cannot be inherited by subclass B and does not have access to it.

Output:

$ javac inheritance.java Exception in thread “main” java.lang.Error:
Unresolved compilation problem: The field A.j is not visible

23)You know that all classes in java are inherited from java.lang.Object class. Are interfaces also inherited from Object class?

Ans : No, only classes in java are inherited from Object class. Interfaces in java are not inherited from Object class. But, classes which implement interfaces are inherited from Object class.

24)What is the output of this program?

```
class A {
   public int i;
   public int j;
   A() {
      i = 1;
      j = 2;
   }
}
class B extends A {
   int a;
   B() {
      super();
   }
}
class Demo {
   public static void main(String args[]){
      B obj = new B();
      System.out.println(obj.i + " " + obj.j)
   }
}
```

Options :

a) 1 2

b) 2 1

c) Runtime Error

d) Compilation Error

Ans : a)1 2

Explanation : Keyword super is used to call constructor of class A by constructor of class B. Constructor of a initializes i & j to 1 & 2 respectively.

Output:

```
$ javac Demo.java
$ java Demo
1 2
```

25)Can we reduce the visibility of the inherited or overridden method ?

Ans : No.

26)A class member declared protected becomes member of subclass of which type?

Options :

a) public member

b) private member

c) protected member

d) static member

Ans : b)private member

Explanation : A class member declared protected becomes private member of subclass.

27)Which of the following is tightly bound ? Inheritance or Composition ?

Ans : Inheritance.

28)What is the output of this program?

```
class A {
   public int i;
   protected int j;
}
class B extends A {
   int j;
   void display() {
      super.j = 3;
      System.out.println(i + " " + j);
   }
}
class Demo {
   public static void main(String args[]){
      B obj = new B();
      obj.i=1;
      obj.j=2;
```

```
        obj.display();
    }
}
```

Options :

a) 12
b) 21
c) 13
d) 31

Ans: a)12

Explanation : Both class A & B have member with same name that is j, member of class B will be called by default if no specifier is used. I contains 1 & j contains 2, printing 1 2.

Output:

```
$ javac Demo.java
$ java Demo
1 2
```

29)What will happen if class implement two interface having common method?

Ans : That would not be a problem as both are specifying the contract that implement class has to follow. If class C_JBK implement interface A_JBK & interface B_JBK then Class C_JBK thing I need to implement print() because of interface A_JBK then again Class think I need to implement print() again because of interface B_JBK, it sees that there is already a method called test() implemented so it's satisfied.

30)Does a class inherit the constructor of its super class?

Ans : No.

31)Which of these is correct way of inheriting class A by class B?

Options :

a) class B + class A { }
b) class B inherits class A { }
c) class B extends A { }
d) class B extends class { }

Ans: c)class B extends A{ }

32)Are constructors and initializers also inherited to sub classes?

Ans: No, Constructors and initializers(Static initializers and instance initializers) are not inherited to sub classes. But, they are executed while instantiating a sub class.

33)How do you implement multiple inheritance in java?

Ans: Through interfaces, we can implement multiple inheritance in java. As classes in java can not extend more than one classes, but a class can implement more than one interfaces.

```
interface A {
```

```
}
interface B {
}
class C implements A,B {
   // Class implementing two interfaces.
}
```

34)What happens if both, super class and sub class, have a field with same name?

Ans: Super class field will be hidden in the sub class. You can access hidden super class field in sub class using super keyword.

35)Are static members inherited to sub classes?

Ans: Yes, Static members are also inherited to sub classes.

```
class A {
   static int i = 10;
   static void method() {
      System.out.println("Static Method");
   }
}

class B extends A {
}
public class StaticInitializers {
   public static void main(String[] args) {
      B.method(); // Calling inherited static method
      System.out.println(B.i); // printing inherited static field.
   }
}
```

36)Which of the following statements are incorrect?

Options :

a) public members of class can be accessed by any code in the program.

b) private members of class can only be accessed by other members of the class.

c) private members of class can be inherited by a sub class, and become protected members in sub class.

d) protected members of a class can be inherited by a sub class, and become private members of the sub class.

Ans: c) private members of class can be inherited by a sub class, and become protected members in sub class.

Explanation : private members of a class cannot be inherited by a sub class.

Polymorphism Interview Question

1)What is polymorphism in Java ?

Ans:

- Polymorphism is an 'OOPS' concept which advice use of common interface instead of concrete implementation while writing code.
- When we program for interface, our code is capable of handling any new requirement or enhancement that may arise in near future due to new implementation of our common interface.
- If we don't use common interface and rely on concrete implementation, we always need to change and duplicate most of our code to support new implementation.
- Its not only Java but other object oriented languages like C++ that supports polymorphism and its a fundamental along with other OOPS concepts like Encapsulation, Abstraction and Inheritance.

2)Member variables are resolved at compile time or runtime?

Ans: Member variables are resolved at compile time.

3)What is function overloading?

Ans: Method overloading is the polymorphism technique which allows us to create multiple methods with the same name but different signature. We can achieve method overloading in two ways.

- By Changing the number of arguments
- By Changing the data type of arguments

Method overloading increases the readability of the program. Method overloading is performed to figure out the program quickly.

4)What is method overriding?

Ans: If a subclass provides a specific implementation of a method that is already provided by its parent class, it is known as Method Overriding. It is used for runtime polymorphism and to implement the interface methods.

Rules for Method overriding:

The method must have the same name as in the parent class.

The method must have the same signature as in the parent class.

Two classes must have an IS-A relationship between them.

5)What is the difference between 'Overloading' and 'Overriding'?

Ans: Method overloading increases the readability of the program. On the other hand, method overriding provides the specific implementation of the method that is already provided by its super class.Parameter must be different in case of overloading whereas it must be same in case of overriding.

6)What is Function Overriding and Overloading in Java ?

Ans:

- Overloading in Java occurs when two or more methods in the same class have the same name, but different parameters.
- On the other hand, in method overriding a child class redefines the same method as a parent class. Overridden methods must have the same name, argument list and return type. The overriding method may not limit the access of the method it overrides.

7)Can we overload the main() method?

Ans: Yes, we can have any number of main methods in a Java program by using method overloading

8)Why can we not override static method?

Ans: It is because the static method is the part of the class, and it is bound with class whereas instance method is bound with the object, and static gets memory in class area, and instance gets memory in a heap.

9)Why can we not override static method?

Ans: It is because the static method is the part of the class, and it is bound with class whereas instance method is bound with the object, and static gets memory in class area, and instance gets memory in a heap.

10)What is covariant return type?

Ans: Now, since java5, it is possible to override any method by changing the return type if the return type of the subclass overriding method is subclass type. It is known as covariant return type. The covariant return type specifies that the return type may vary in the same direction as the subclass.

11)Difference between method Overloading and Overriding?

Ans:

Method Overloading	**Method Overriding**
1) Method overloading increases the readability of the program.	Method overriding provides the specific implementation of the method that is already provided by its superclass.
2) Method overloading occurs within the class.	Method overriding occurs in two classes that have IS-A relationship between them.
3) In this case, the parameters must be different.	In this case, the parameters must be the same.

12)What is the difference between compile-time polymorphism and runtime polymorphism?
Ans:

Sno	Compile Time Polymorphism	Run Time Polymorphism
1	Compile time polymorphism means binding is occuring at compile time	Run time polymorphism where at run time we came to know which method is going to invoke
2	It can be achieved through static binding	It can be achieved through dynamic binding
3	Inheritance is not involved	Inheritance is involved
4	Method overloading is an example of compile time polymorphism	Method overriding is an example of runtime polymorphism

13)What are points to consider in terms of access modifier when we are overriding any method?
Ans :

- Overriding method cannot be more restrictive than the overridden method.
- Reason : In case of polymorphism , at object creation jvm look for actual runtime object. jvm does not look for reference type and while calling methods it look for overridden method.
- If by means subclass were allowed to change the access modifier on the overriding method, then suddenly at runtime—when the JVM invokes the true object's version of the method rather than the reference type's version then it will be problematic.
- In case of subclass and superclass define in different package, we can override only those method which have public or protected access.
- We cannot override any private method because private methods can not be inherited and if method cannot be inherited then method can not be overridden.

14)How compiler handles the exceptions in overriding ?
Ans :

- The overriding methods can throw any runtime Exception , here in the case of runtime exception overriding method (subclass method) should not worry about exception being thrown by superclass method.
- If superclass method does not throw any exception then while overriding, the subclass method can not throw any new checked exception but it can throw any runtime exception.
- Different exceptions in java follow some hierarchy tree(inheritance). In this case, if superclass method throws any checked exception , then while overriding the method in subclass we can not throw any new checked exception or any checked exception which are higher in hierarchy than the exception thrown in superclass method.

15)What is the rule regarding overriding methods throwing exceptions?

Ans: Overriding methods cannot throw more generic exception than base method.

16)Member variables are resolved at compile time or runtime?
Ans: Member variables are resolved at compile time.

17)Write the nearest equivalent of size operator in C. (Hint: Use Runtime class)
[Note: Since GC can't be enforced in java the result is not always predictable.]
Ans: Static Runtime runtime=Runtime.getRuntime();
long start,end;
Object obj;
runtime.gc();
start=runtime.freememory();
obj=new object();
end= Runtime.freememory();
System.out.println('size of obj' + (start-end)+ 'bytes');

18)Can we override variables?
Ans:
```
class S1Demo{
   public string S= " S1";
   public string gets() {
      returns S;
   }
}
class S2Demo extend S1Demo {
   public string S = " S2";
   public string gets() {
      return S;
   }
}
public class Shadow{
   public static void main(String S[]) {
      S1Demo s1 = new S1Demo();
      S2Demo s2= new S2Demo();
      System.out.println("Print S1 " + s1.s);
      System.out.println("Print S1 " + s2.s);
      s1=s2;
      System.out.println("Print S1 now " + s1.S) ;
      System.out.println( "Print s1.gets() now " + s1.gets());
   }
}
```
Yes, we can override variables but variables when overridden shadows the super class variable. Print S1 S1 Print S1 S2 Print S1 now S1 Print S1.gets() now S2.

19)If an overridden method calls super class method which accesses class member variable, which variable will be used for base class or super class?

Ans:

```
class S1Demo {
   string S= “ S1”;
   public string gets (){
      return S;
   }
   void display () {
      System.out.println("Display in S1 " + S);
   }
}
class S2Demo extends S1Demo{
   string S= “ S2”;
   void display(){
      super.display();
      System.out.println("Display in S2 " +S);
   }
}
public class Shadow2 {
   string s =” base”;
   public static void main(String s[]) {
      S2Demo s2=new S2Demo();
      S2.display ();
      S1Demo s1=new S1Demo();
      System.out.println("Print S1 " + s1.gets());
      System.out.println("Print S2 " + s2.gets());
   }
}
```

Methods access variables only in the context of the class they belong to will be used for base class or super class. If subclass calls super class method, it will access super class variable. Display in S1 S1 Display in S2 S2 Print S1 S1 Print S2 S1

20)Can you override private or static method in Java ?

Ans: This is another common tricky Java question. As mentioned before method overriding is a good topic to ask tricky questions in Java. One cannot override private or static method in Java. If you create a similar method with same return type and same method arguments then it is called method hiding.

Abstraction Interview Question

1)What is abstraction ?

Ans: Abstraction refers to the ability to make a class abstract in OOP. It helps to reduce the complexity and also improves the maintainability of the system.

2)What is an abstract class ?

Ans: These classes cannot be instantiated and are either partially implemented or not at all implemented. This class contains one or more abstract methods which are simply method declarations without a body.

3)When is an abstract method used ?

Ans: An abstract method is declared in the parent class when we want a class which contains a particular method but on the other hand we want its implementation to be determined by child class.

4)Can an interface be extended by another interface in Java ?

Ans: An interface can be extended by another interface in Java. The code for the same would be like as shown below :

This interface extends from the Body interface :

```
public interface Car extends Vehicle {
   public void Speed();
}
```

5)What is fully abstract class?

Ans: An abstract class which has all methods as abstract and all fields are public static final.

6)Can an abstract class have a constructor?

Ans: Yes an abstract class has a default and parameterized constructors

7)Can an abstract class have a static method?

Ans: Yes an abstract class has a static method and it can be accessed by any other class (even not a concrete class).

8)In which kind of situation would an interface be extended by another interface ?

Ans: Remember that any class that implements an interface must implement the method headings that are declared in that interface. If that particular interface extends from other interfaces, then the implementing class must also implement the methods in the interfaces that are being extended or derived from. As shown in the example above, if we have a class that implements the Car interface, then that class must define the method headings in both the 'Car' interface and the 'Vehicle' interface.

9)Differentiate an Interface and an Abstract class.

Ans: An abstract class may have many instance methods which sport default behavior. On the other hand, an interface cannot implement any default behaviour. However, it can declare different constants and instance methods.
Whereas an interface has all the public members, an abstract class contains only class members like private, protected and so on.

10)How to define an abstract class ?
Ans: A class containing abstract method is called an abstract class. An Abstract class cannot be instantiated.
Example of Abstract class :

```
abstract class AbstractClass{
   protected String S;
   public String getS(){
      return S;
   }
   public abstract string AbstractFunction();
}
```

11)How can we define an interface?
Ans: In Java an interface just defines the methods and not implement them. Interface can include constants. A class that implements the interfaces is bound to implement all the methods defined in an interface.
Example of Interface :

```
public interface myInterface {
   public void one();
   public long CONSTANT_ONE = 1000;
}
```

12)What is the difference between an interface and an abstract class? Also discuss their similarities.
Ans:

- An abstract class is a class which contains one or more abstract methods and has to be implemented by sub classes.
- An interface is a Java Object which contains method declaration but does not contain its implementation.
- The classes which implement the interfaces must provide the method definition for all the methods.
- An abstract class has a class prefix with an 'abstract' keyword followed by 'class definition'. On the other hand interface starts with an 'interface' keyword.
- An abstract class may contain one or more abstract methods whereas an interface contains all abstract methods and final declarations. Abstract classes are useful in a situation where

general methods need to be implemented while specialization behavior should be implemented by child classes.

- Interfaces are useful in a situation where all properties need to be implemented.
- Difference between an Interface and an Abstract class is as follows:
- Interfaces provide a kind of multiple inheritance. A class can extend only one class.
- Interfaces are limited to public methods and constants with no implementation of methods. Abstract classes can have a partial implementation, protected methods, static methods, etc.
- A class may implement several interfaces. But in case of an abstract class, a class may extend only one abstract class.
- Interfaces are slow as it requires extra direction to find corresponding method in the actual class whereas abstract classes are fast.
- Similarities:
- Neither abstract classes nor interfaces can be instantiated.

Interface Interview Question

1)Why use Java interface?

Ans: There are mainly three reasons to use interface. They are given below:

- It is used to achieve fully abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.
- In other words, Interface fields are public, static and final bydefault, and methods are public and abstract.

*Printable.java

```
1 package Basic;
2
3 interface Printable {
4     int MIN=5;
5     void print();
6
7 }
8
```

*Printable.java

```java
package Basic;

interface Printable {
    public static final int MIN=5;
    public abstract void print();

}
```

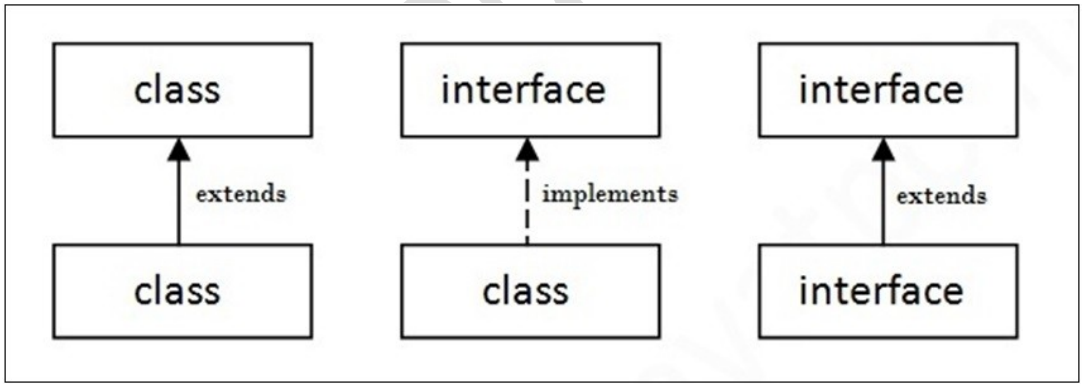
2)What is Interface in Java?

Ans:

- An interface in java is a blueprint of a class. It has static constants and abstract methods only.
- The interface in java is a mechanism to achieve fully abstraction. There can be only abstract methods in the java interface not method body. It is used to achieve fully abstraction and multiple inheritance in Java.
- Java Interface also represents IS-A relationship.
- It cannot be instantiated just like abstract class

3)What is the Understanding relationship between classes and interfaces?

Ans: As shown in the figure given below, a class extends another class, an interface extends another interface but a class implements an interface



4)Give the Simple example of Java interface.

Ans: In this example, Printable interface have only one method, its implementation is provided in the A class.

```
interface printable{
   void print();
}
```

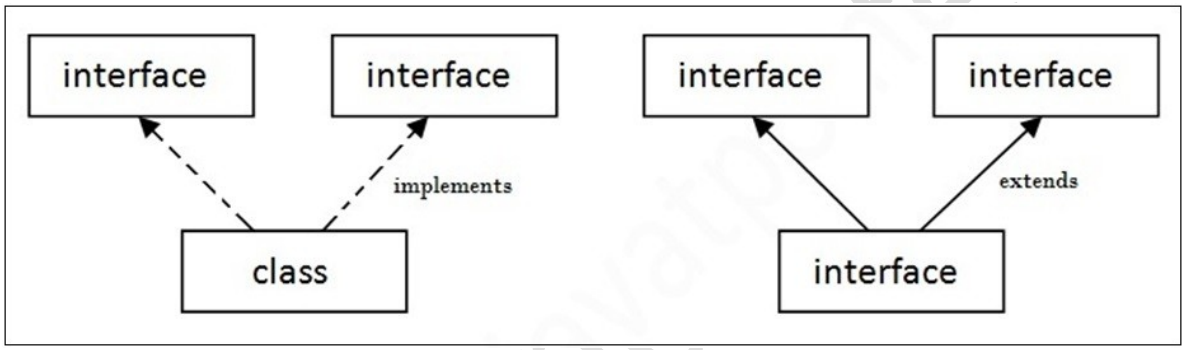
```
class A implements printable{
   public void print() {
      System.out.println("Hello");
   }

   public static void main(String args[]){
      A obj = new A();
      obj.print();
   }
}
```

```
Output:
 Hello
```

5)What is the Multiple inheritance in Java by interface?

Ans: If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance.




```
interface Printable{
   void print();
}
interface Showable{
   void show();
}

class A implements Printable, Showable{
   public void print()
   {
    System.out.println("Hello");
   }
   public void show()
   {
    System.out.println("Welcome");
   }
   public static void main(String args[])
   {
    A obj = new A();
    obj.print();
    obj.show();
   }
```

```
}
     Output:
      Hello
      Welcome
```

6)Multiple inheritance is not supported through class in java but it is possible by interface, why?

Ans: As we have explained in the inheritance chapter, multiple inheritance is not supported in case of class. But it is supported in case of interface because there is no ambiguity as implementation is provided by the implementation class.

Example :

```
interface Printable {
   void print();
}

interface Showable{
   void print();
}

class Interface1 implements Printable, Showable {
    public void print()
    {
       System.out.println("Hello");
    }
    public static void main(String args[])
    {
 Interface1 obj = new Interface1 ();
      obj.print();
    }
 }
```

Output:

```
 Hello
```

As you can see in the above example, Printable and Showable interface have same methods but its implementation is provided by class TestTnterface1, so there is no ambiguity.

7)What is Interface inheritance ?

Ans: A class implements interface but one interface extends another interface.

```
interface Printable {
   void print();
}

interface Showable extends Printable {
   void show();
```

```
}

class Interface2 implements Showable {
   public void print()
   {
      System.out.println("Hello");
   }
   public void show()
   {
      System.out.println("Welcome");
   }
   public static void main(String args[])
   {
      Interface2 obj = new Interface2 ();
      obj.print();
      obj.show();
   }
}
```

```
Output:
 Hello
 Welcome
```

8)What is Nested Interface in Java?

Ans: An interface can have another interface i.e. known as nested interface. We will learn it in detail in the nested classes chapter.

Example :

```
interface printable {
   void print();
   interface MessagePrintable {
      void msg();
   }
}
```

9)Can a class implement two interfaces with same variable names?

Ans: If both the interface have same variable and the variable is not declared in implementing class, the compiler will throw an ambiguous error.

10)Can a class implement two interfaces with same variable names?

Ans: If both the interface have same variable and the variable is not declared in implementing class, the compiler will throw an ambiguous error.

11)Which one of these statements are valid?

Char \u0061r a =’a’;

Char \u0062 = ’b’;

Char c =’\u0063’;

Options : a) 1. b) 2. c) 3. d) ALL. e) NONE.
Ans: d) ALL.

12)Which one is faster in java ?

Math.max(a,b);

(a>b)?a:b
Ans: b) (a>b)?a:b

13)Is this statement correct: char ch = 'd'; if(ch < 32.00){ }
Ans: Yes, the mentioned statement is correct.

14)What will be the output of the following statements:

System.out.println(1+2+”3”);

System.out.println (“1”+2+3);
Ans:
a) 33
b) 123

15)Can an interface be final?
Ans: Interface cannot be declared 'final' as they are implicitly 'abstract'.

16)Does the 'finalize' method in subclass invoke 'finalize' method in super class?
Ans: 'Finalize' is not implicitly chained. A 'finalize' method in sub-class should call 'finalize' in super class explicitly as its last action for proper functioning. Compilers does not enforce this check.

17)Can we have static method in interface?
Ans: All methods in an interface are implicitly 'public', 'abstract' but never 'static'.

18)Can an interface have variables? Can these variables be transient?
Ans: All variables in an interface are implicitly static , public and final. They cannot be transient or volatile. A class can shadow the interface variable with its variable while implementing

19)What is the use of transient variable? Can a transiant variable be static?

Ans: Transient variables are not stored as object's persistence state and is not serialized for security. Transient variables may not be final or static. Compilers do not give any errors as static variables and anyways they are not serialized.

20)Nested classes can extend only the enclosing class and cannot implement any interface. True/False?
Ans: False. [Nested class can extend any class or implement any interface.]

21)Can a class implement two interfaces which has got methods with same name and signatures?
Ans: Yes.

22)Can an interface be final?
Ans: No, interface cannot be declared final as they are implicitly abstract.

23)Can we have static method in interface?
Ans: All methods in an interface are implicitly public, abstract but never static.

Package Interview Question

1)What are the advantages of a java package?
Ans: Java packages helps to resolve naming conflicts when different packages have classes with the same names. This also helps one organize files within the project.
For example, java.io package does operations related to I/O while java.net package is to do with network and so on. If we tend to put all .java files into a single package, it becomes tough to handle this package as the project gets bigger.

2)Define Packages in Java.
Ans: A 'package' can be defined as a group of related types (classes, interfaces, enumerations and annotations) providing access protection and name space management

3)Why are the packages used?
Ans: Packages are used in Java in order to prevent naming conflicts, to control the access, to perform search operation and to make the use of classes, interfaces, enumerations, annotations, etc quite easier.

4)Can we import same package/class twice? Will the JVM load the package twice at runtime?
Ans: One can import the same package or same class multiple times. Neither compiler nor JVM complains anything about it. JVM will internally load the class only once no matter how many times one imports the same class.

5)Explain the usage of Java packages.

Ans:

- A Java package is a naming context for classes and interfaces.
- A package is used to create a separate name space for groups of classes and interfaces.
- Packages are also used to organize related classes and interfaces into a single API unit and to control accessibility to these classes and interfaces.
-

6)What is the base class of all classes?

Ans: 'java.lang.Object' is the base class of all classes.

7)What do you think is the logic behind having a single base class for all classes?

Ans: Casting, hierarchical and object oriented structure is the main logic behind having a single base class for all classes.

8)Why most of the thread functionalities are specified in Object Class?

Ans: Most of the thread functionalities are specified in object class for the interthread communication.

9)What is a package in Java? Explain in detail.

Ans:

- A package in Java is a way of organizing related functionalities in a single place. In Java, File System package represents a directory where Java source file is stored before compilation and class files are stored after compilation.
- For example, if you create a class 'HelloWorld' in a package called 'com.profound.welcome;' then it will reside under directory 'com/profound/welcome' in source tree and you can view that in your IDE like 'Eclipse' or 'Netbeans' or even by navigating to file system.
- Once you compile your Java program either by using your IDE, Ant build Script or maven compile plugin, it will create class files under same package structure.
- For example maven will create 'target/classes/com/profound/welcome' directory and place 'HelloWorld.class' inside that. Its mandatory that class files reside on same package or directory as declared in there source file using 'package' keyword. If failed to do this, it will result in 'java.lang.NoClassDefFoundError' in Java.
- In short we can say that 'package' is a keyword in Java which is used to specify directory structure for a particular class file.

10)How to create a package in Java?

Ans:

- If you are using IDE like 'Eclipse' for developing your Java program then you don't need to do anything.
- Just click on new-->package and Eclipse will ask you name of the package, put name of the package and you are good to go.

- Now if you want to create Java class on that package, just select the package in package explorer and create new-->Java Class.
- If you are not using any IDE than you manually need to create directories corresponding to package in Java.

11)Why do we need to use package in Java?

Ans:

- Package provides encapsulation in Java program.
- Default access modifier for any variable or class is package-private i.e. they are only visible into package, on which they are declared.
- By using package you Encapsulate whole functionality which allows you to change the functionality, include new functionality or just change the implementation without breaking whole application
- Though package is not the highest degree of Encapsulation in Java which is achieved using 'private' keyword, it is still the second best option and a must in order to encapsulate whole functionality rather than just a class.
-

12)Which package is always imported by default?

Ans: The 'java.lang' package is always imported by default.

13)How to use a package in Java?

Ans: Using a package in Java is very simple. Just use the 'package' keyword along with name of package at top of your Java source file to declare package in Java. Package declaration must be first line in Java source file even before import statement. Here is an example of how to use package in Java.

In this example we have a directory 'blog/java67/welcome' which is a package and 'Hello.java' class is stored under it. When Java compiler will compile this class, it will verify whether this particular class is stored as '/blog/java67/welcome/Hello.java', which is relative to classpath.

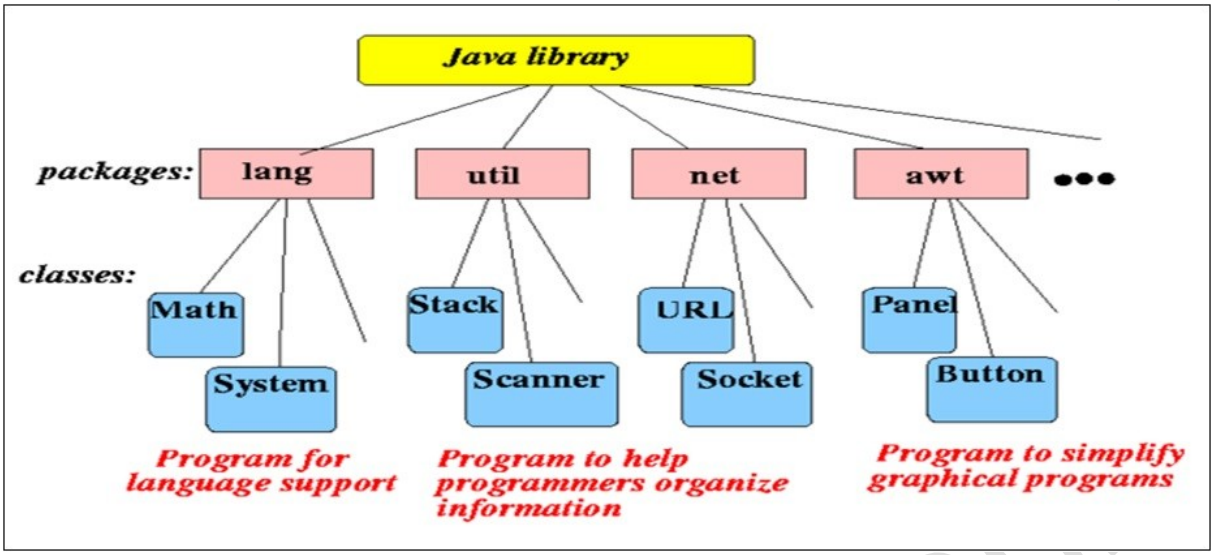
```
package com.profound.welcome
public class Hello {
   public static void main(String args[]) {
      System.out.println("An Example of using package com.profound.welcome in Java");
   }
}
```

14)Explain what is a predefined package in Java.

Ans: Predefined packages are those which are developed by SUN micro systems and are supplied as a part of JDK (Java Development Kit) to simplify the task of java programmer.

15)What is a predefined package in Java?

Ans:



Java classes are structured into different packages based upon there functionality

- Say for instance 'java.lang' package contains classes which are essential to Java programming language e.g. Thread, Exception Error, Object etc.
- On the other hand package like 'java.util' contains all utility classes e.g. Collection classes, Scanner and other utility.
- 'java.io' package contains Java classes related to Input and Output functionality.
- 'java.util.concurrent' also known as the sub package of 'java.util' contains concurrent utility classes like CountDownLatch, CyclicBarrier, Semaphore etc.

16)Which are the important things one need to keep in mind about package in Java?

Ans: Here are some of the key details to remember about package in Java programming language :

1. 'java.lang' package is automatically imported in every Java class.
2. You can import all classes and its sub packages from a package by using wildcard '*'. Say for e.g. import com.abc.* will import all classes of package com.abc as well as classes of sub packages if any.
3. Package must be the first statement, even before import statement in Java source file.

This is all one should know about packages in Java. We have learned from basic like what is package in Java, why should we use package in Java and how to use package in Java to some of the best practices while using package in Java application.

Good knowledge of package features is important to structure complex Java application whereas clever use of package-private encapsulation can lead to highly flexible and maintainable software.

One example of clever use of package is 'java.util.EnumSet' class, which is abstract and both of its implementation 'JumboEnumSet' and 'RegularEnumSet' are package-private. Since instance of EnumSet is created via factory methods and these classes are package-private, there is no way client can use them directly, allowing you to ship a new implementation in future without affecting any client.

17)Which package is always imported by default?

Ans: No. It is by default loaded internally by the JVM. The java.lang package is always imported by default.

18)Does importing a package imports the sub packages as well? E.g. Does importing com.bob.* also import com.bob.code.*?

Ans: No. We will have to import the sub packages explicitly. Importing 'com.bob.*' will import classes in the package 'bob' only. It will not import any class in any of its sub package's.

Exception Handling Interview Question

1)How many types of exception can occur in a Java program?

Ans: There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:

- Checked Exception: Checked exceptions are the one which are checked at compile-time. For example, SQLException, ClassNotFoundException, etc.
- Unchecked Exception: Unchecked exceptions are the one which are handled at runtime because they can not be checked at compile-time. For example, ArithmaticException, NullPointerException, ArrayIndexOutOfBoundsException, etc.
- Error: Error cause the program to exit since they are not recoverable. For Example, OutOfMemoryError, AssertionError, etc.

2)What is Exception Handling?

Ans: Exception Handling is a mechanism that is used to handle runtime errors. It is used primarily to handle checked exceptions. Exception handling maintains the normal flow of the program. There are mainly two types of exceptions: checked and unchecked. Here, the error is considered as the unchecked exception.

3)How can you handle error condition while writing stored procedure or accessing stored procedure from java ?

Ans: This is one of the toughest questions of Java usually asked in an interview which is open for all. A friend of mine was unaware of the answer to this question and also did not hesitate in sharing this with me. According to me stored procedure returns an error code only if some operation fails but if stored procedure itself fails than catching SQLException is the only choice left.

4)What is the difference between Checked Exception and Unchecked Exception?

Ans: 1) Checked Exception

The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions, e.g., IOException, SQLException, etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes that extend RuntimeException are known as unchecked exceptions, e.g., ArithmeticException, NullPointerException, etc. Unchecked exceptions are not checked at compile-time.

5)What is the base class for Error and Exception?
Ans: The Throwable class is the base class for Error and Exception

6)Is it necessary that each try block must be followed by a catch block?
Ans: It is not necessary that each try block must be followed by a catch block. It should be followed by either a catch block OR a finally block

7)What will happen if you call return statement or System.exit on try or catch block ? Will finally block execute?
Ans: This is a very common tricky Java question and its considered tricky because many programmers think that 'finally' block always gets executed. This question challenges the concept by putting return statement in 'try' or 'catch' block or calling System.exit from 'try' or 'catch' block. The answer to this question in Java is that 'finally' block will execute even if you put 'return' statement in 'try' block or 'catch' block but 'finally' block won't run if you call System.exit form 'try' or 'catch'

8)What is finally block?
Ans: The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not. In other words, we can say that finally block is the block which is always executed. Finally block follows try or catch block. If you don't handle the exception, before terminating the program, JVM runs finally block, (if any). The finally block is mainly used to place the cleanup code such as closing a file or closing a connection.

9)Can finally block be used without a catch?
Ans: Yes, According to the definition of finally block, it must be followed by a try or catch block, therefore, we can use try block instead of catch

10)Can an exception be rethrown?
Ans: Yes.

11)If a method throws NullPointerException in super class then can we override it with a method which throws RuntimeException?
Ans: One can throw super class of RuntimeException in overridden method but you cannot do the same if its checked Exception.

12)What is the difference between throw and throws?
Ans:

throw keyword	**throws keyword**

1)The **throw** keyword is used to throw an exception explicitly.	The **hrows** keyword is used to declare an exception.
2) The**throw** keyword is followed by an instance.	The **throws** keyword is followed by class.
3)The **throw k**eyword is used within the method.	The **throws** keyword is used with the method declaration.
4)You cannot throw multiple exceptions.	You can declare multiple exceptions, e.g., public void method()throws IOException, SQLException.

13)How are the exceptions handled in Java?

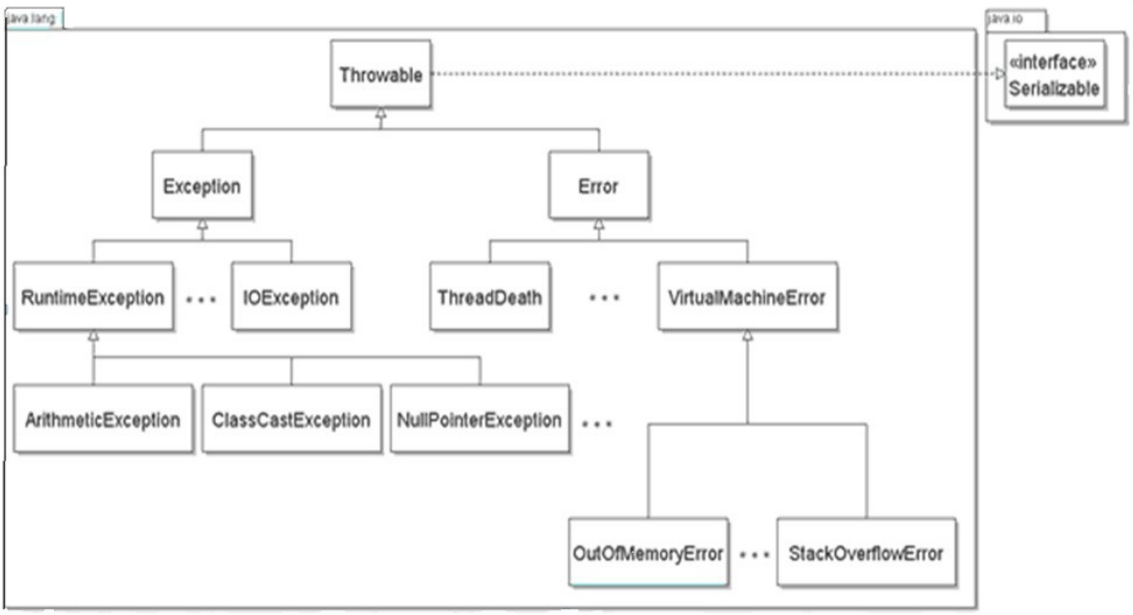
Ans: Whenever an exception occurs the process of execution of the program is transferred to an appropriate exception handler. The try-catch-finally block is used to handle the exceptions. The code in which the exception may occur is enclosed in a try block, also called as a guarded region. The catch clause matches a specific exception to a block of code which handles that exception and the clean up code which needs to be executed no matter the exception occurs or not is put inside the finally block

14)Exceptions are defined in which java package?

Ans: All the exceptions are subclasses of java.lang.Exception.

15)Explain the exception hierarchy in java.

Ans: Throwable is a parent class of all exception classes. There are two types of exceptions: 'Checked exceptions' and 'unchecked exceptions' or 'RunTimeExceptions'. Both type of exceptions extends 'exception' class.



16)What is Runtime Exception or unchecked exception?

Ans: Runtime exception represents the problems that occurs because of a programming problem. Such problems include some of the following :

- Arithmetic exceptions : eg. Dividing by zero.
- Pointer exceptions : eg. Trying to access an object through a null reference.
- Indexing exceptions : eg. Attempting to access an array element through an index that is too large or too small.

Runtime exceptions need not be explicitly caught in try catch block as it can occur anywhere in a program and in a typical one there can be numerous. By adding runtime exceptions in every method declaration a program's clarity will get reduced. Thus, the compiler does not require one to catch or specify runtime exceptions (although one can). The solution is to rectify the programming logic wherever the exception has occurred or provide a check.

17)What is a checked exception?

Ans: Checked exceptions forces the programmer to catch them explicitly in try-catch block. It is a subclass of Exception. Example : IOException

18)What is difference between an error and exception?

Ans: An error is an irrecoverable condition occurring at runtime say for instance 'OutOfMemory' error. These JVM errors cannot be repaired at runtime. Though error can be caught in catch block still the execution of application will come to a halt and is not recoverable. On the other hand exceptions are conditions that occur because of bad input or human error. For e.g. 'FileNotFoundException' will be thrown if the specified file does not exist or a 'NullPointerException' will take place if one tries using a null reference. In most of the cases it is possible to recover from an exception (probably by giving user a feedback for entering proper values etc.)

19)What is the use of 'throws' keyword?

Ans: If the function is not capable of handling the exception then it can ask the calling method to handle it by simply putting the 'throws' clause at the function declaration.

```
public void parent() {
try {
      child();
   }
   catch(MyCustomException e){ }
}
public void child throws MyCustomException {
   // put some logic so that the exception occurs.
}
```

20)If there is an exception in finalize method, will the object be garbage collected?

Ans: Yes.

21)Can constructor throw exception?
Ans: Yes. [Constructor can throw exception.]

22)What is the rule regarding overriding methods throwing exceptions?
Ans: Overriding methods cannot throw more generic exceptions than base method.

23)Can finalize method be overloaded?
Ans: Yes, finalize method can be overloaded but only the below given version is called by garbage collector : protected void finalize() throws Throwable { };

24)Can static block throw exception?
Ans:

- Yes, static block can throw only Runtime exception or can use a try-catch block to catch checked exception.
- Typically scenario will be if JDBC connection is created in static block and it fails then exception can be caught, logged and application can exit.
- If System.exit() is not done, then application may continue and next time if the class is referred JVM will throw NoClassDefFounderror since the class was not loaded by the Classloader.
-

25)What is the difference between final, finally and finalize() in Java?
Ans:

- **final -** declared as constant. A final variable act as constant, a final class is immutable and a final method cannot be overridden.
- **finally -** handles exception. The finally block is optional and provides a mechanism to clean up regardless of what happens within the try block (except System.exit(0) call). Use the finally
- block to close files or to release other system resources like database connections, statements etc.
- **finalize() -** method helps in garbage collection. A method that is invoked before an object is discarded by the garbage collector, allowing it to clean up its state. Should not be used to release non-memory resources like file handles, sockets, database connections etc. because Java has only a finite number of these resources and you do not know when the garbage collection is going to kick in to release these non-memory resources through the finalize() method.

26)What is a 'throw' keyword?
Ans: 'Throw' keyword is used to throw the exception manually. It is mainly used when the program fails to satisfy the given condition and it wants to warn the application. The exception thrown should be subclass of 'Throwable'.

```
public void parent() {
   try {
```

```
        child();
    }
    catch(MyCustomException e) { }
}
public void child{
    String iAmMandatory=null;
    if(iAmMandatory == null) {
        throw (new MyCustomException("Throwing exception using throw keyword");
    }
}
```

27)What is difference between 'ClassNotFoundException' and 'NoClassDefFoundError'?

Ans: A 'ClassNotFoundException' is thrown when the reported class is not found by the 'ClassLoader' in the 'CLASSPATH'. It could also mean that the class in question is trying to be loaded from another class which was loaded in a parent classloader. Hence the class from the child classloader is not visible. Consider if 'NoClassDefFoundError' occurs then it would be something like this :

java.lang.NoClassDefFoundError src/com/TestClass But this does not mean that the 'TestClass' class is not in the CLASSPATH. It means that the class 'TestClass' was found by the 'ClassLoader'. However, when one tries to load the class, it runs into an error while reading the class definition. This typically happens when the class in question has static blocks or members which uses a Class that's not found by the ClassLoader. So, in order to find the culprit, view the source of the class in question (TestClass in this case) and look for code using static blocks or static members.

28)What are the possible combinations to write try, catch and finally block?

Ans:

```
try{
    //lines of code that may throw an exception
}
catch(Exception e){
    //lines of code to handle the exception thrown in try block
}
finally{
    //the clean code which is executed always no matter the exception occurs or not.
}
//2
try{}
finally{}
//3
try{
}
catch(Exception e){
```

//lines of code to handle the exception thrown in try block
}The try block must be always followed by the try block. If there are more than one catch blocks they all must follow each other without any other block in between. The finally block must follow the catch block if one is present. If the catch block is absent the finally block must follow the try block

29)When to make a custom checked Exception or custom unchecked Exception?
Ans: If an application can reasonably be expected to recover from an exception then it can be said as a checked exception. If an application cannot do anything to recover from the exception then it is said to be an unchecked exception.

30)How to create a custom Exception?
Ans: To create one's own exception extend the Exception class or any of its subclasses. Say for instance,

- class New1Exception extends Exception { } // this will create Checked Exception
- class NewException extends IOExcpetion { } // this will create Checked exception
- class NewException extends NullPonterExcpetion { } // this will create UnChecked exception

31)What is StackOverflowError?
Ans: The 'StackOverFlowError' is an error object thrown by the runtime system when it encounters that the application/code has ran out of the memory. It may occur in case of recursive methods or when a large amount of data is fetched from the server and stored in some object. This error is generated by JVM.

32)Once the control switches to the catch block does it return back to the try block to execute the balance code?
Ans: No. Once the control jumps to the catch block it never returns to the try block instead it goes to finally block (if present).

33)Where is the clean up code like release of resources put in try-catch-finally block and why?
Ans: The code is put in a finally block because irrespective of try or catch block execution, the control will flow to finally block. Typically, finally block contains release of connections, closing of result set etc

34)Is it valid to have a try block without catch or finally?
Ans: No. It is not possible to have a try block without catch or finally block as it will result in a compilation error. The try block must be followed by a catch or a finally block. It is acceptable to omit the either of the catch or the finally block but not both of them.

35)How do you get the descriptive information about the Exception occurred during the program execution?

Ans: All the exceptions inherit a method 'printStackTrace()' from the 'Throwable' class. This method prints the stack trace from where the exception occurred. It prints the most recently entered method first and continues down, printing the name of each method as it works its way down the call stack from the top.

36)Why it is not considered a good practice to write a single catch all handler instead of catch all the exceptions?

Ans: You can write a single catch block to handle all the exceptions thrown during the program. If you use the Superclass Exception in the catch block then you will not get the valuable information about each of the exception thrown during the execution, though you can find out the class of the exception that occurred. This will also reduce the readability of the code as the programmer will not understand what is the exact reason for putting the try-catch block.

37)Can a catch block throw the exception caught by itself ?

Ans: Yes, a catch block can throw the exception caught by itself which is called as rethrowing of the exception by catch block. e.g. the catch block below catches the FileNotFound exception and rethrows it again.

```
void checkEx() throws FileNotFoundException{
   try {
      //code that may throw the FileNotFoundException
   }
   catch(FileNotFound eFnf) {
      throw FileNotFound();
   }
}
```

38)What is exception matching?

Ans: Exception matching is the process by which the the jvm finds out the matching catch block for the exception thrown from the list of catch blocks. When an exception is thrown, Java will try to find by looking at the available catch clauses in the top down manner. If it doesn't find one, it will search for a handler for a supertype of the exception. If it does not find a catch clause that matches a supertype for the exception, then the exception is propagated down the call stack. This process is called as exception matching.

39)What happens if the handlers for the most specific exceptions is placed above the more general exceptions handler?

Ans: Compilation would fail in this case if the catch block for handling the most specific exceptions is not placed above the catch block written to handle the more general exceptions.

```
//The code below will not compile.
try {
```

```
    // code that can throw IOException or its subtypes
}
catch (IOException e) {
    // handles IOExceptions and its subtypes
}
catch (FileNotFoundException ex) {
    // handle FileNotFoundException only
}

// The code below will compile successfully
try {
    // code that can throw IOException or its subtypes
}
catch (FileNotFoundException ex) {
    // handles IOExceptions and its subtypes
}
catch (IOException e){
    // handle FileNotFoundException only
}
```

40)Is an empty catch block legal?

Ans: An empty catch block is considered legal by leaving the catch block without writing any actual code to handle the exception caught. e.g. The code below is legal but not appropriate, as in this case you will not get any information about the exception thrown.

41)Which one will throw an arithmetic exception:

1)int i = 100/0; 2)float f = 100.00/0.0

Ans: 2. [float f = 100.00/0.0. Float division by zero returns NAN (not a number) instead of exception.]

File Handling Interview Question

1)What do you understand by an IO stream?

Ans: The stream is a sequence of data that flows from source to destination. It is composed of bytes. In Java, three streams are created for us automatically.

- System.out: standard output stream
- System.in: standard input stream
- System.err: standard error stream

2)What is the difference between the Reader/Writer class hierarchy and the InputStream/OutputStream class hierarchy?

Ans: The Reader/Writer class hierarchy is character-oriented, and the InputStream/OutputStream class hierarchy is byte-oriented. The ByteStream classes are used to perform input-output of 8-bit bytes whereas the CharacterStream classes are used to perform the input/output for the 16-bit Unicode system. There are many classes in the ByteStream class hierarchy, but the most frequently used classes are FileInputStream and FileOutputStream. The most frequently used classes CharacterStream class hierarchy is FileReader and FileWriter

3)What are the super most classes for all the streams?

Ans: All the stream classes can be divided into two types of classes that are ByteStream classes and CharacterStream Classes. The ByteStream classes are further divided into InputStream classes and OutputStream classes. CharacterStream classes are also divided into Reader classes and Writer classes. The SuperMost classes for all the InputStream classes is java.io.InputStream and for all the output stream classes is java.io.OutPutStream. Similarly, for all the reader classes, the super-most class is java.io.Reader, and for all the writer classes, it is java.io.Writer.

4)What are the FileInputStream and FileOutputStream?

Ans: Java FileOutputStream is an output stream used for writing data to a file. If you have some primitive values to write into a file, use FileOutputStream class. You can write byte-oriented as well as character-oriented data through the FileOutputStream class. However, for character-oriented data, it is preferred to use FileWriter than FileOutputStream.

Java FileInputStream class obtains input bytes from a file. It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video, etc. You can also read character-stream data. However, for reading streams of characters, it is recommended to use FileReader class.

Consider the following example for reading bytes from a file.

5)What is the purpose of using BufferedInputStream and BufferedOutputStream classes?

Ans: Java BufferedOutputStream class is used for buffering an output stream. It internally uses a buffer to store data. It adds more efficiency than to write data directly into a stream. So, it makes the performance fast. Whereas, Java BufferedInputStream class is used to read information from the stream. It internally uses the buffer mechanism to make the performance fast.

6)What is serialization?

Ans: Serialization in Java is a mechanism of writing the state of an object into a byte stream. It is used primarily in Hibernate, RMI, JPA, EJB and JMS technologies. It is mainly used to travel object's state on the network (which is known as marshaling). Serializable interface is used to perform serialization. It is helpful when you require to save the state of a program to storage such as the file. At a later point of time, the content of this file can be restored using deserialization. It is also required to implement RMI(Remote Method Invocation). With the help of RMI, it is possible to invoke the method of a Java object on one machine to another machine.

7)How can you make a class serializable in Java?
Ans: A class can become serializable by implementing the Serializable interface.

8)What is Deserialization?
Ans: Deserialization is the process of reconstructing the object from the serialized state. It is the reverse operation of serialization. An ObjectInputStream deserializes objects and primitive data written using an ObjectOutputStream.

9)What is the transient keyword?
Ans: If you define any data member as transient, it will not be serialized. By determining transient keyword, the value of variable need not persist when it is restored.

10)Random access file extends from File. True/False?
Ans: False [Random access file descends from object and implements data input and data output.]

Serialization Interview Question

1)What is the need of serialization?
Ans: The serialization is used:

- To send state of one or more object's state over the network through a socket.
- To save the state of an object in a file.
- An object's state needs to be manipulated as a stream of bytes.

2)Other than serialization what are the other different approaches to make an object serializable?
Ans:
Besides the Serializable interface, at least three other alternate approaches can serialize Java objects:

- For object serialization, instead of implementing the Serializable interface, a developer can implement the 'Externalizable' interface, which extends Serializable. By implementing Externalizable, a developer is responsible for implementing the 'writeExternal()' and 'readExternal()' methods. As a result, a developer has sole control over reading and writing the serialized objects.
- 'XML' serialization is an often-used approach for data interchange. This approach lags runtime performance when compared with Java serialization, both in terms of the size of the object and the processing time. With a speedier XML parser, the performance gap with respect to the processing time narrows. Nonetheless, XML serialization provides a more malleable solution when faced with changes in the serializable object.
- Finally, consider a 'roll-your-own' serialization approach. You can write an object's content directly via either the 'ObjectOutputStream' or the 'DataOutputStream'. While this approach is more involved in its initial implementation, it offers the greatest flexibility and

extensibility. In addition, this approach provides a performance advantage over Java serialization.

3)Do we need to implement any method of 'Serializable' interface to make an object serializable?

Ans: No. Serializable is a Marker Interface. It does not have any methods.

4)What happens if the object to be serialized includes the references to other serializable objects?

Ans: If the object to be serialized includes references to the other objects, then all those object's state will also be saved as the part of the serialized state of the object in question. The whole object graph of the object to be serialized will be saved during serialization automatically provided all the objects included in the object's graph are serializable.

5)What happens if an object is serializable but it includes a reference to a non-serializable object?

Ans: If you try to serialize an object of a class which implements serializable, but the object includes a reference to a non-serializable class then a 'NotSerializableException' will be thrown at runtime.

```
public class NonSerial
{    //This is a non-serializable class
}
public class MyClass implements Serializable
{
   private static final long serialVersionUID = 1L;
   private NonSerial nonSerial;
   MyClass(NonSerial nonSerial)
   {
      this.nonSerial = nonSerial;
   }
   public static void main(String [] args)
   {
      NonSerial nonSer = new NonSerial();
      MyClass c = new MyClass(nonSer);
      try {
         FileOutputStream fs = new FileOutputStream("test1.ser");
         ObjectOutputStream os = new ObjectOutputStream(fs);
         os.writeObject(c);
         os.close();
      }
      catch (Exception e)
      {
         e.printStackTrace();
```

```
    } try {
       FileInputStream fis = new FileInputStream("test1.ser");
       ObjectInputStream ois = new ObjectInputStream(fis);
       c = (MyClass) ois.readObject();
       ois.close();
    }
    catch (Exception e)
    {
       e.printStackTrace();
    }
  }
}
```

On execution of above code following exception will be thrown:
java.io.NotSerializableException:NonSerialatjava.io.ObjectOutputStream.writeObject0(ObjectOutputStream.java)

6)Are the static variables saved as the part of serialization?

Ans: No. The static variables belonging to the class are not the part of the state of an object so they are not saved as the part of serialized object.

7)What will be the value of transient variable after de-serialization?

Ans: The value of transient variable after de-serialization will be its default value. Say for instance if the transient variable in question is an 'int', its value after deserialization will be zero.

```
public class TestTransientVal implements Serializable
{
  private static final long serialVersionUID = -22L;
  private String name;
  transient private int age;
  TestTransientVal(int age, String name)
  {
    this.age = age;
    this.name = name;
  }
  public static void main(String [] args)
  {
    TestTransientVal c = new TestTransientVal(1,"ONE");
    System.out.println("Before serialization:" + c.name + " "+ c.age);
    try {
      FileOutputStream fs =new FileOutputStream("testTransient.ser");
      ObjectOutputStream os = new ObjectOutputStream(fs);
      os.writeObject(c);
      os.close();
```

```
    }
    catch (Exception e)
    {
       e.printStackTrace();
    }
    try {
       FileInputStream fis =new FileInputStream("testTransient.ser");
       ObjectInputStream ois =new ObjectInputStream(fis);
       c = (TestTransientVal) ois.readObject();
       ois.close();
    } catch (Exception e)
    {
       e.printStackTrace();
    }
    System.out.println("After de-serialization:" + c.name +" "+ c.age);
  }
}
```

Result of the above piece of code is:

Output:

Before serialization : Value of non-transient variable ONE Value of transient variable 1
After de-serialization : Value of non-transient variable ONE Value of transient variable 0

Explanation for the same is as follows:
The transient variable is not saved as the part of the state of the serialized variable, it's value after de-serialization is it's 'default' value.

8)Does the order in which the value of the transient variables and the state of the object using the 'defaultWriteObject()' method saved during serialization matter?
Ans: Yes, while restoring the object's state the transient variables and the serializable variables that are stored must be restored in the same order in which they were saved.

Multithreading Interview Question

1)What is multithreading?
Ans: Multithreading is a process of executing multiple threads simultaneously. Multithreading is used to obtain the multitasking.

2)What is the thread?
Ans: A thread is a lightweight sub process, a smallest unit of processing. It is a
separate path of execution Threads are independent, if there occurs exception in one thread, it doesn't
affect other threads. It shares a common memory area.

3)What are the two ways of implementing thread in Java?

Ans: There are basically two ways of implementing thread in java as given below:

- Extending the Thread class
- Implementing Runnable interface in Java

4)What's the difference between thread and process?

Ans: Thread: It simply refers to the smallest units of the particular process. It has the ability to execute different parts (referred to as thread) of the program at the same time.
Process: It simply refers to a program that is in execution i.e., an active program. A process can be handled using PCB (Process Control Block).

5)What are the wait() and sleep() methods?

Ans: wait(): As the name suggests, it is a non-static method that causes the current thread to wait and go to sleep until some other threads call the notify () or notifyAll() method for the object's monitor (lock). It simply releases the lock and is mostly used for inter-thread communication. It is defined in the object class, and should only be called from a synchronized context.
sleep(): As the name suggests, it is a static method that pauses or stops the execution of the current thread for some specified period. It doesn't release the lock while waiting and is mostly used to introduce pause on execution. It is defined in thread class, and no need to call from a synchronized context.

6)What's the difference between notify() and notifyAll()?

Ans: notify(): It sends a notification and wakes up only a single thread instead of multiple threads that are waiting on the object's monitor.
notifyAll(): It sends notifications and wakes up all threads and allows them to compete for the object's monitor instead of a single thread.

7)What is the start() and run() method of Thread class?

Ans: start(): In simple words, the start() method is used to start or begin the execution of a newly created thread. When the start() method is called, a new thread is created and this newly created thread executes the task that is kept in the run() method. One can call the start() method only once.
run(): In simple words, the run() method is used to start or begin the execution of the same thread. When the run() method is called, no new thread is created as in the case of the start() method. This method is executed by the current thread. One can call the run() method multiple times.

8)What's the purpose of the join() method?

Ans: join() method is generally used to pause the execution of a current thread unless and until the specified thread on which join is called is dead or completed. To stop a thread from running until another thread gets ended, this method can be used. It joins the start of a thread execution to the end of another thread's execution. It is considered the final method of a thread class

9)Explain the meaning of the deadlock and when it can occur?

Ans: Deadlock, as the name suggests, is a situation where multiple threads are blocked forever. It generally occurs when multiple threads hold locks on different resources and are waiting for other resources to complete their task.

10)What happens when 'start()' method is called?

Ans: A new thread of execution with a new call stack starts when 'start()' method is called. The state of thread changes from 'new' to 'runnable'. When the thread gets chance to execute its target, 'run()' method starts to run.

11)What is the difference between StringBuffer and StringBuilder in Java ?

Ans: This is a classic Java question which some people find it quite tricky whereas some consider it very easy. StringBuilder in Java was introduced in Java 5 and the only difference between Stringbuffer and StringBuilder is that Stringbuffer methods are synchronized while the latter one is non-synchronized.

12)Is it better to synchronize critical section of getInstance() method or whole getInstance() method ?

Ans: Critical section is better because if we lock the whole method than every time when someone calls this method it will have to wait even though we are not creating any object.

13)How do threads communicate with each other?

Ans: Threads can communicate using three methods i.e., wait(), notify(), and notifyAll().

14)What is use of 'synchronized' keyword?

Ans: 'Synchronized' keyword can be applied to 'static/non-static' methods or a block of code. Only one thread at a time can access synchronized methods and if there are multiple threads trying to access the same method then other threads have to wait for the execution of method by one thread. 'Synchronized' keyword provides a lock on the object and thus prevents race condition. E.g.

```
public void synchronized method()
{
}
   public void synchronized staticmethod(){}
   public void myMethod(){
      synchronized (this){
      //synchronized keyword on block of code
   }
}
```

15)What is the synchronization process? Why use it?

Ans: Synchronization is basically a process in java that enables a simple strategy for avoiding thread interference and memory consistency errors. This process makes sure that resource will be

only used one thread at a time when one thread tries to access a shared resource. It can be achieved in three different ways as given below:

- By the synchronized method
- By synchronized block
- By static synchronization

16)What is synchronized method and synchronized block? Which one should be preferred?

Ans: Synchronized Method: In this method, the thread acquires a lock on the object when they enter the synchronized method and releases the lock either normally or by throwing an exception when they leave the method. No other thread can use the whole method unless and until the current thread finishes its execution and release the lock. It can be used when one wants to lock on the entire functionality of a particular method.

Synchronized Block: In this method, the thread acquires a lock on the object between parentheses after the synchronized keyword, and releases the lock when they leave the block. No other thread can acquire a lock on the locked object unless and until the synchronized block exists. It can be used when one wants to keep other parts of the programs accessible to other threads.

Synchronized blocks should be preferred more as it boosts the performance of a particular program. It only locks a certain part of the program (critical section) rather than the entire method and therefore leads to less contention.

17)What do you understand by thread-safety ? Why is it required and how can it be achieved in Java Applications ?

Ans:

- Java Memory Model defines the legal interaction of threads with the memory in a real computer system. In another way we can say that it describes which behaviors are legal in multi-threaded code. It determines when a thread can reliably see the write method done on variables by other threads. It defines semantics for volatile, final & synchronized which guarantees the visibility of memory operations across the Threads.
- Let's first discuss about Memory Barrier which is the base for the further discussions. There are two type of memory barrier instructions in JMM - read barriers and write barrier.
- A read barrier invalidates the local memory (cache, registers, etc) and then reads the contents from the main memory, so that changes made by other threads becomes visible to the current Thread.
- A write barrier flushes out the contents of the processor's local memory to the main memory, so that changes made by the current Thread becomes visible to the other threads.
- JMM semantics for synchronized
- When a thread acquires monitor of an object, by entering into a synchronized block of code, it performs a read barrier (invalidates the local memory and reads from the heap instead). Similarly exiting from a synchronized block as part of releasing the associated monitor, it performs a write barrier (flushes changes to the main memory) Thus modifications to a shared state using synchronized block by one Thread, is guaranteed to be visible to

subsequent synchronized reads by other threads. This guarantee is provided by JMM in presence of synchronized code block.

- JMM semantics for Volatile fields
- Read & write to volatile variables have same memory semantics as that of acquiring and releasing a monitor using synchronized code block. So the visibility of volatile field is guaranteed by the JMM. Moreover afterwards Java 1.5, volatile reads and writes are not reorderable with any other memory operations (volatile and non-volatile both). Thus when Thread A writes to a volatile variable V, and afterwards Thread B reads from variable V, any variable values that were visible to A at the time V was written are guaranteed now to be visible to B.

Let's try to understand the same using the following code
Data data = null;
volatile boolean flag = false;
Thread A
---
data = new Data();
flag = true; <!-- Writing to volatile will flush data as well as flag to main memory -->

Thread B
---
if(flag==true){ <!-- as="" barrier="" data.="" flag="" font="" for="" from="" perform="" read="" reading="" volatile="" well="" will="" -->
use data; <!-- Data is guaranteed to visible even though it is not declared volatile because of the JMM semantics of volatile flag. -->
}

18)What is the difference when the 'synchronized' keyword is applied to a 'static' method or to a 'non-static' method?

Ans: When a 'synchronized non-static' method is called a 'lock' is obtained on the object. When a 'synchronized static' method is called a 'lock' is obtained on the class and not on the object.
The lock on the object and the lock on the class don't interfere with each other. It means, if a thread accesses a synchronized non-static method, then the other thread can access the synchronized static method at the same time but can't access the synchronized non-static method.

19)Can you start a thread twice?

Ans: No, it's not at all possible to restart a thread once a thread gets started and completes its execution. Thread only runs once and if you try to run it for a second time, then it will throw a runtime exception i.e., java.lang.IllegalThreadStateException.

20)What do you mean by inter-thread communication?

Ans: Inter-thread communication, as the name suggests, is a process or mechanism using which multiple threads can communicate with each other. It is especially used to avoid thread polling in java and can be obtained using wait(), notify(), and notifyAll() methods.

21)What is Thread Scheduler?
Ans: Thread Scheduler: It is a component of JVM that is used to decide which thread will execute next if multiple threads are waiting to get the chance of execution. By looking at the priority assigned to each thread that is READY, the thread scheduler selects the next run to execute. To schedule the threads.

22)GC is a high priority thread. True/False?
Ans: False. [GC is a low priority thread.]

23)Explain Thread Group?
Ans: ThreadGroup is a class that is used to create multiple groups of threads in a single object. This group of threads is present in the form of three structures in which every thread group has a parent except the initial thread. Thread groups can contain other thread groups also. A thread is only allowed to have access to information about its own thread group, not other thread groups.

24)What will happen if we don't override the thread class run() method?
Ans: Nothing will happen as such if we don't override the run() method. The compiler will not show any error. It will execute the run() method of thread class and we will just don't get any output because the run() method is with an empty implementation.

25)If code running is a thread and it creates a new thread, then what will be the initial priority of the newly created thread?
Ans: When a code is running in a thread and it creates a new thread object, the priority of the new thread is set equal to the priority of the thread which has created it.

26)When jvm starts up, which thread will be started up first?
Ans: When jvm starts up the thread executing main method is started.

27)What is the lock interface?
Ans: Lock interface was introduced in Java 1.5 and is generally used as a synchronization mechanism to provide important operations for blocking.
Advantages of using Lock interface over Synchronization block:

- Methods of Lock interface i.e., Lock() and Unlock() can be called in different methods. It is the main advantage of a lock interface over a synchronized block because the synchronized block is fully contained in a single method.
- Lock interface is more flexible and makes sure that the longest waiting thread gets a fair chance for execution, unlike the synchronization block.

28)What is the difference between yield() and sleep()?

Ans:

- 'yield()' allows the current thread to release its lock from the object and scheduler gives the lock of the object to the other thread with same priority.
- 'sleep()' allows the thread to go to sleep state for x milliseconds. When a thread goes into sleep state it doesn't release the lock.

29)Is it possible to call the run() method directly to start a new thread?

Ans: No, it's not possible at all. You need to call the start method to create a new thread otherwise run method won't create a new thread. Instead, it will execute in the current thread.

30)Differentiate between Multithreading and Multiprocessing ?

Ans:

Multithreading	**Multiprocessing**
Thread is a main unit of Multithreading Program or process is the main unit of Multiprocessing.	Program or process is the main unit of Multiprocessing.
Multiple threads of a single process are executed at the same time	Multiple processes are executed at the same time
It is cost-effective	It is expensive
It is highly efficient.	It is less efficient.
It helps to develop an efficient application program.	It helps to develop an efficient operating system program.

31)What are the advantages or usage of threads?

Ans: Threads support concurrent operations. For example:

- Multiple requests by a client on a server can be handled as an individual client thread. Threads often result in simpler programs.
- In sequential programming, updating multiple displays normally requires a big while-loop that performs small parts of each display update. Threads provide a high degree of control.
- Imagine launching a complex computation that occasionally takes longer than is satisfactory. Threaded applications exploit parallelism.
- A computer with multiple CPUs can literally execute multiple threads on different functional units without having to simulate multi-tasking ("time sharing").

32)What are the two ways of creating thread?

Ans: There are two ways to create a new thread. Extend the 'Thread' class and override the 'run()' method in your class. Create an instance of the subclass and invoke the 'start()' method on it, which will create a new thread of execution.

```
public class NewThread extends Thread
{
   public void run()
   {
      // the code that has to be executed
      // in a separate new thread goes here
   }
   public static void main(String [] args)
   {
      NewThread c = new NewThread();
      c.start();
   }
}
```
This will implement the 'Runnable' interface. The class will have to implement the 'run()' method in the 'Runnable' interface. Create an instance of this class. Pass the reference of this instance to the 'Thread' constructor and a new thread of execution will be created.

```
public class NewThread implements Runnable
{
   public void run()
   {
      // the code that has to be executed
      // in a separate new thread goes here
   }
   public static void main(String [] args)
   {
      NewThread c = new NewThread();
      Thread t = new Thread(c);
      t.start();
   }
}
```

33)What are the different states of a thread's lifecycle?

Ans: The different states of threads are as follows:

- **New :** When a thread is instantiated it is in 'New' state until the 'start()' method is called on the 'thread' instance. In this state the thread is not considered to be alive.
- **Runnable :** The thread enters into this state after the 'start()' method is called in the thread instance. The thread may enter into the 'Runnable' state from 'Running' state. In this state the thread is considered to be alive.
- **Running :** When the thread scheduler picks up the thread from the 'Runnable' thread's pool, the thread starts running and the thread is said to be in 'Running' state.
- **Waiting/Blocked/Sleeping :** In these states the thread is said to be alive but not runnable. The thread switches to this state because of various reasons like when 'wait()' method is

called or 'sleep()' method has been called on the running thread or thread might be waiting for some input/output resource so blocked.

- **Dead :** When the thread finishes its execution i.e. the 'run()' method execution completes, it is said to be in 'dead' state. A 'dead' state cannot be started again. If a 'start()' method is invoked on a dead thread a runtime exception will occur.

34)What are the daemon threads?

- 'Daemon' threads are service provider threads which run in the background. These are not used to run the application code generally.
- When all user threads(non-daemon threads) complete their execution the jvm exit the application whatever may be the state of the daemon threads.
- Jvm does not wait for the daemon threads to complete their execution if all user threads have completed their execution. To create Daemon thread set the daemon value of Thread using 'setDaemon(boolean value)' method.
- By default all the threads created by user are user threads. To check whether a thread is a Daemon thread or a user thread use 'isDaemon()' method.
- Example of the Daemon thread is the 'Garbage' Collector run by jvm to reclaim the unused memory by the application. The 'Garbage' collector code runs in a Daemon thread which terminates as all the user threads are done with their execution.

35)Can the variables or classes be synchronized?

Ans: No. Only methods can be synchronized and not the variables or classes.

36)How many locks does an object have?

Ans: Each object has only one lock.

37)Can a class have both synchronized and non-synchronized methods?

Ans: Yes, a class can have both synchronized and non-synchronized methods.

38)If a class has a synchronized method and non-synchronized method, can multiple threads execute the non-synchronized methods?

Ans: Yes. If a class has a synchronized and non-synchronized methods, multiple threads can access the non-synchronized methods.

39)If a thread goes to sleep does it hold the lock?

Ans: Yes when a thread goes to sleep, invoked by 'Thread.sleep(xx)' it holds a lock.

40)Can a thread hold multiple locks at the same time?

Ans: Yes. A thread can hold multiple locks at the same time. Once a thread acquires a lock and enters into the synchronized method / block, it may call another synchronized method and acquire a lock on another object

41)Can a thread call multiple synchronized methods on the object of which it holds the lock?
Ans: Yes. Once a thread acquires a lock in some object, it may call any other synchronized method of that same object using the lock that it already holds. The locks are Reentrant.

42)Can static methods be synchronized?
Ans: Yes. Static methods are class methods and have only one copy of static data for the class. Only one lock for the entire class is required. Every class in java is represented by java.lang.Class instance. The lock on this instance is used to synchronize the static methods.

43)Can two threads call two different static synchronized methods of the same class?
Ans: No. The static synchronized methods of the same class always block each other as only one lock per class exists. So no two static synchronized methods can execute at the same time.

44)Does a static synchronized method block a non-static synchronized method?
Ans: No, the thread executing the static synchronized method holds a lock on the class and the thread executing the non-static synchronized method holds the lock on the object on which the method has been called. These two locks are different and these threads do not block each other.

45)Once a thread has been started can it be started again?
Ans: No. A thread can be started only once in its lifetime. If you try starting a thread which has been already started once an 'IllegalThreadStateException' is thrown, which is a runtime exception. A thread in runnable state or a dead thread cannot be restarted.

46)When does 'deadlock' occur and how to avoid it?
Ans: When a locked object tries to access a locked object which is trying to access the first locked object, the threads will wait for each other to release the lock on a particular object due to which deadlock occurs.

47)Which is a better way of creating multithreaded application, extending Thread class or implementing Runnable?
Ans: If a class is made to extend the thread class to have a multithreaded application then this subclass of Thread cannot extend any other class and the required application will have to be added to this class as it cannot be inherited from any other class. If a class is made to implement Runnable interface, then the class can extend other class or implement other interface.

48)Can the 'start()' method of the Thread class be overridden? If yes should it be overridden?
Ans: Yes, the 'start()' method can be overridden. But it should not be overridden as it's implementation in thread class has the code to create a new executable thread and is specialized.

49)Which methods of the thread class are used to schedule the threads?
Ans: The methods are as follows:

- public static void sleep(long millis) throws InterruptedException

- public static void yield()
- public final void join() throws InterruptedException
- public final void setPriority(int priority)
- public final void wait() throws InterruptedException
- public final void notify()
- public final void notifyAll()

50)Which thread related methods are available in Object class?

Ans: The methods are:

- public final void wait() throws Interrupted exception
- public final void notify()
- public final void notifyAll()

51)Which thread related methods are available in Thread class?

Ans: Methods which are mainly used are as follows:

- public static void sleep(long millis) throws Interrupted exception
- public static void yield() public final void join() throws Interrupted exception
- public final void setPriority(int priority)
- public void start()
- public void interrupt()
- public final void join()
- public void run()
- public void resume()

52)List the methods which when called the thread does not release the locks held?

Ans: Following are the methods:

- notify()
- join()
- sleep()
- yield()

53)List the methods which when called on the object the thread releases the locks held on that object?

Ans: wait().

54)Does each thread have its own thread stack?

Ans: Yes, each thread has its own call stack.

In the below example t1 and t3 will have the same stack and t2 will have its own independent stack.

```
Thread t1 = new Thread();
Thread t2 = new Thread();
Thread t3 = t1;
```

55)Explain re-entrant, recursive and idempotent methods/functions?

Ans: A method in stack is re-entrant allowing multiple concurrent invocations that do not interfere with each other.

56)What is thread starvation?

Ans: In a multi-threaded environment thread starvation occurs if a low priority thread is not able to run or get a lock on the resource because of presence of many high priority threads. This is mainly possible by setting thread priorities inappropriately.

Collection Framework Interview Question

1)What is the Collection framework in Java?

Ans: Collection Framework is a combination of classes and interface, which is used to store and manipulate the data in the form of objects. It provides various classes such as ArrayList, Vector, Stack, and HashSet, etc. and interfaces such as List, Queue, Set, etc. for this purpose.

2)Explain various interfaces used in Collection framework?

Ans: Collection framework implements various interfaces, Collection interface and Map interface (java.util.Map) are the mainly used interfaces of Java Collection Framework.

1. Collection interface: Collection (java.util.Collection) is the primary interface, and every collection must implement this interface.
2. List interface: List interface extends the Collection interface, and it is an ordered collection of objects. It contains duplicate elements. It also allows random access of elements.
3. Set interface: Set (java.util.Set) interface is a collection which cannot contain duplicate elements. It can only include inherited methods of Collection interface
4. Queue interface: Queue (java.util.Queue) interface defines queue data structure, which stores the elements in the form FIFO (first in first out).
5. Map interface: A Map (java.util.Map) represents a key, value pair storage of elements. Map interface does not implement the Collection interface. It can only contain a unique key but can have duplicate elements. There are two interfaces which implement Map in java that are Map interface and Sorted Map

3)Which are the two methods that you need to implement for key Object in HashMap ?

Ans: You need to override equals() and hashcode() methods of a class whose objects you want to use as Key in a hashmap.

4)What is the difference between an ArrayList and a Vector ?

Ans:

- Difference between Vector and Arraylist is one of the most common Core Java Interview questions you will come across in Collection. This question is mostly asked at an initial stage by the Interviewers before testing the deep knowledge regarding Collection.

- Vectors are synchronized. Any method that touches the Vector's contents is thread safe. ArrayList, on the other hand, is unsynchronized, making them, therefore, not thread safe.
- Vector and ArrayList classes are implemented using dynamically resizable array that enables fast random access and list traversal similar to that while using an ordinary array.
- ArrayList supports dynamic arrays in which changes can be made as per requirement that is ArrayList can be dynamically increased or decreased in size.

5)Map implements collection. True/False?
Ans: Map does not implement collection.

6)Can you write code for iterating over hashmap in Java 4 and Java 5 ?
Ans: It is a bit tricky but you can perform this task using the 'while' loop and 'for' loop.

7)When do you override hashcode and equals() ?
Ans: We can override hashcode whenever necessary and especially when we want to do an equality check or when we want to use our object as key in HashMap.

8)Which kind of problem you will encounter if you don't override hashcode() method ?
Ans: You will not be able to recover your object from HashMap if that is used as key in HashMap.

9)What are the main differences between array and collection?
Ans: Array and Collection are somewhat similar regarding storing the references of objects and manipulating the data, but they differ in many ways. The main differences between the array and Collection are defined below:
•Arrays are always of fixed size, i.e., a user can not increase or decrease the length of the array according to their requirement or at runtime, but In Collection, size can be changed dynamically as per need.
•Arrays can only store homogeneous or similar type objects, but in Collection, heterogeneous objects can be stored.
•Arrays cannot provide the ?ready-made? methods for user requirements as sorting, searching, etc. but Collection includes readymade methods to use.

10)What is the difference between List and Set?
Ans: The List can contain duplicate elements whereas Set includes unique items.
•The List is an ordered collection which maintains the insertion order whereas Set is an unordered collection which does not preserve the insertion order.
•The List interface contains a single legacy class which is Vector class whereas Set interface does not have any legacy class.
•The List interface can allow n number of null values whereas Set interface only allows a single null value.

11)What is the difference between HashSet and TreeSet?

Ans: HashSet maintains no order whereas TreeSet maintainsascending order.

- HashSet impended by hash table whereas TreeSet implemented by a Tree structure.
- HashSet performs faster than TreeSet.

12)What is the difference between Set and Map?

Ans: Set contains values only whereas Map contains key and values both.

- Set contains unique values whereas Map can contain unique Keys with duplicate values.
- Set holds a single number of null value whereas Map can include a single null key with n number of null values.

13)What is the difference between Set and Map?

Ans: Set contains values only whereas Map contains key and values both.

- Set contains unique values whereas Map can contain unique Keys with duplicate values.
- Set holds a single number of null value whereas Map can include a single null key with n number of null values.

14)What is the difference between Collection and Collections?

Ans: The Collection is an interface whereas Collections is a class.

- The Collection interface provides the standard functionality of data structure to List, Set, and Queue. However, Collections class is to sort and synchronize the collection elements.

15)What is the difference between ArrayList and LinkedList?

Ans:

ArrayList	LinkedList
ArrayList uses a dynamic array.	LinkedList uses a doubly linked list.
ArrayList is not efficient for manipulation because too much is required	LinkedList is efficient for manipulation.
ArrayList is better to store and fetch data.	LinkedList is better to manipulate data.
ArrayList takes less memory overhead as it stores only object	LinkedList takes more memory overhead, as it stores the object as well as the address of that object.

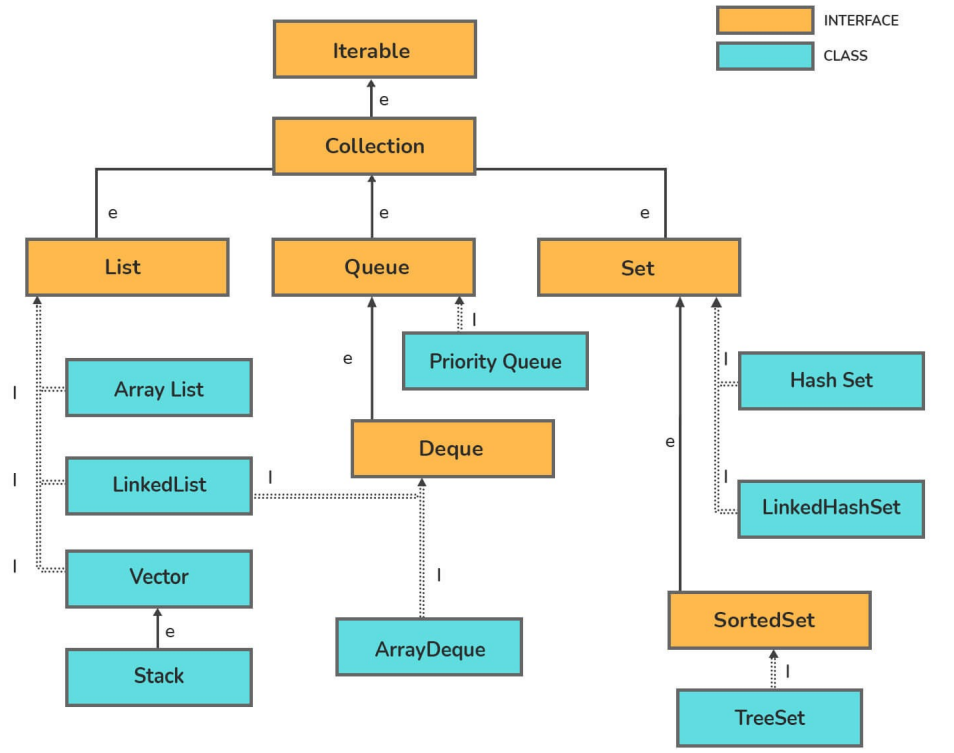
16)What is the difference between Iterator and ListIterator?

Ans:

Iterator	ListIterator
The Iterator traverses the elements in the forward direction only.	ListIterator traverses the elements in backward and forward directions both.
The Iterator can be used in List, Set, and Queue.	ListIterator can be used in List only.

17)Explain the hierarchy of the Collection framework in Java.

Ans: The entire collection framework hierarchy is made up of four fundamental interfaces: Collection, List, Set, Map, and two specific interfaces for sorting called SortedSet and SortedMap. The java.util package contains all of the collection framework's interfaces and classes. The following diagram depicts the Java collection structure.



Collection Hierarchy in Java

InterviewBit

18)What is the difference between Comparable and Comparator?

Ans:

Comparable	Comparator
Comparable provides only one sort of sequence.	The Comparator provides multiple sorts of sequences.

It provides one method named compareTo().	It provides one method named compare().
It is found in java.lang package.	It is located in java.util package.
If we implement the Comparable interface, The actual class is modified.	The actual class is not changed.

19)If one does not override hashcode() method then does it have any performance implication ?

Ans: This is a good question and is open to all. According to me a poor hashcode function will result into frequent collisions in HashMap which eventually increases time for adding an object into Hash Map.

20)What is wrong in using HashMap in multithreaded environment? When wil get() method go into an infinite loop ?

Ans: This is another good question. The get() method will go into an infinite loop during concurrent access and re-sizing.

21)What will happen if we put a key object in a HashMap which is already present ?

Ans: The difficult part of this Java questions is how does the HashMap works in Java which is also a popular topic to prompt confusing and tricky question in Java. If we put the same key again then it will replace the old mapping because HashMap doesn't allow duplicate keys.

22)Can we use any class as Map key?

Ans:

- We can use any class as Map Key, however following points should be considered before using them.
- If the class overrides equals() method, it should also override hashCode() method. The class should follow the rules associated with equals() and hashCode() for all instances.
- If a class field is not used in equals(), you should not use it in hashCode() method. Best practice for user defined key class is to make it immutable, so that hashCode() value can be cached for fast performance.
- Also immutable classes make sure that hashCode() and equals() will not change in future that will solve any issue with mutability.
- For example, let’s say I have a class MyKey that I am using for HashMap key. MyKey name argument passed is used for equals() and hashCode()

```
MyKey key = new MyKey("Kajal"); //assume hashCode=1234
myHashMap.put(key, "Value");
// Below code will change the key hashCode() and equals()
// but it's location is not changed.
key.setName("Profound"); //assume new hashCode=7890
```

```
//below will return null, because HashMap will try to look for key
//in the same index as it was stored but since key is mutated,
//there will be no match and it will return null.
myHashMap.get(new MyKey("Kajal"));
```

This is the reason why String and Integer are mostly used as HashMap keys.

23)What are the differences between Queue and Stack in java?

Ans:

- A collection designed for holding elements prior to processing. Besides basic Collection operations, queues provide additional insertion, extraction, and inspection operations.
- Queues typically, but do not necessarily, order elements in a FIFO (first-in-first-out) manner.
- Stack is also a form of Queue but one difference, it is LIFO (last-in-first-out). Whatever the ordering used, the head of the queue is that element which would be removed by a call to remove() or poll(). Also note that Stack and Vector are both synchronized.
- Usage:
- Use a queue if you want to process a stream of incoming items in the order that they are received. Good for work lists and handling requests.
- Use a stack if you want to push and pop from the top of the stack only. Good for recursive algorithms.

24)How can we make HashMap synchronized?

Ans: HashMap can be synchronized by:

```
Map m = Collections.synchronizedMap(hashMap);
```

25)What is NavigableMap in Java ? What are its benefits over Map?

Ans:

- NavigableMap Map was added in Java 1.6, it adds navigation capability to Map data structure. It provides methods like lowerKey() to get keys which is less than specified key, floorKey() to return keys which is less than or equal to specified key, ceilingKey() to get keys which is greater than or equal to specified key and higherKey() to return keys which is greater specified key from a Map.
- It also provide similar methods to get entries e.g. lowerEntry(), floorEntry(), ceilingEntry() and higherEntry().
- Apart from navigation methods, it also provides utilities to create sub-Map e.g. creating a Map from entries of an exsiting Map like tailMap, headMap and subMap. headMap() method returns a NavigableMap whose keys are less than specified, tailMap() returns a NavigableMap whose keys are greater than the specified and subMap() gives a NavigableMap between a range, specified by toKey to fromKey.

26)What is the Difference between the Iterator and ListIterator?

Ans:

- **Iterator :** Iterator Can Only get Data From forward Direction.

- **ListIterator :** An iterator for lists that allows one to traverse the list in either direction, modify the list during iteration, and obtain the iterator's current position in the list.A ListIterator has no current element. its cursor position always lies between the element that would be returned by a call to previous() and the element that would be returned by a call to next(). In a list of length n, there are n+1 valid index values, from 0 to n, inclusive.

27)Difference between Vector and ArrayList?

Ans: Vector & ArrayList both classes are implemented using dynamically resizable arrays, providing fast random access and fast traversal. ArrayList and Vector class both implement the List interface.

- **Synchronization -** ArrayList is not thread-safe whereas Vector is thread-safe. In Vector class each method like add(), get(int i) is surrounded with a synchronized block and thus making Vector class thread-safe.
- **Data growth -** Internally, both the ArrayList and Vector hold onto their contents using an Array. When an element is inserted into an ArrayList or a Vector, the object will need to expand its internal array if it runs out of room. A Vector defaults to doubling the size of its array, while the ArrayList increases its array size by 50 percent.

28)Which two method you need to implement for key Object in HashMap ?

Ans: In order to use any object as Key in HashMap, it must implement equals and hashcode method in Java.

29)What is BlockingQueue?

Ans: A Queue that additionally supports operations that wait for the queue to become non-empty when retrieving an element, and wait for space to become available in the queue when storing an element. BlockingQueue methods come in four forms: one throws an exception, the second returns a special value (either null or false, depending on the operation), the third blocks the current thread indefinitely until the operation can succeed, and the fourth blocks for only a given maximum time limit before giving up.

30)Which implementation of the List interface provides for the fastest insertion of a new element into the middle of the list?

Ans: We have three implementation of List : Vector, ArrayList, LinkedList.
ArrayList and Vector both use an array to store the elements of the list.

31)Which collection classes provide random access of it's elements?

Ans: ArrayList, HashMap, TreeMap, Hashtable classes provide random access to it's elements.

32)What will happen if we put a key object in a HashMap which is already there ?

Ans: This tricky Java questions is part of How HashMap works in Java, which is also a popular topic to create confusing and tricky question in Java. Well if you put the same key again than it will replace the old mapping because HashMap doesn't allow duplicate keys.

33)What will be the problem if you don't override hashcode() method ?
Ans: You will not be able to recover your object from hash Map if that is used as key in HashMap.

34)How to convert a string array to arraylist?
Ans: new ArrayList(Arrays.asList(myArray));

35)How can you suppress unchecked warning in Java ?
Ans: javac compiler for Java 5 generates unchecked warnings if you use combine raw types and generics types. You can be suppress those warnings by using @SuppressWarnings("unchecked") annotation.

36)How can an Arraylist be synchronized without using Vector?
Ans: Arraylist can be synchronized using :

- Collections.synchronizedList(List list)
- Other collections can be synchronized:
- Collections.synchronizedMap(Map map)
- Collections.synchronizedCollection(Collection c)

37)What is difference between an ArrayList and a vector?
Ans:

- **Synchronization -** ArrayList is not thread-safe whereas Vector is thread-safe. In Vector class each method like add(), get(int i) is surrounded with a synchronized block, thus making Vector class thread-safe.
- **Data growth -** Internally, both the ArrayList and Vector hold onto their contents using an Array. When an element is inserted into an ArrayList or a Vector, the object will need to expand its internal array if it runs out of room. A Vector defaults to doubling the size of its array, while the ArrayList increases its array size by 50 percent.
- **Performance -** Since vector is thread-safe, the performance is slower than ArrayList.

38)If an Employee class is present and its objects are added in an arrayList. Now I want the list to be sorted on the basis of the employeeID of Employee class. What are the steps?
Ans:

- Implement Comparable interface for the Employee class and override the compareTo(Object obj) method in which compare the employeeID
- Now call Collections.sort() method and pass the list as an argument.
- Now consider that Employee class is a jar file.
- Since Comparable interface cannot be implemented, create Comparator and override the compare(Object obj, Object obj1) method.

- Call Collections.sort() on the list and pass comparator as an argument

39)What is difference between List and a Set?

Ans:

- List can contain duplicate values but Set doesn't allow.
- List allows retrieval of data to be in same order in the way it is inserted but Set doesn't ensures the sequence in which data can be retrieved.(Except HashSet)

40)What are the classes implementing the List interface?

Ans:

There are three implementation of List interface:

- **ArrayList :** It is a resizable array implementation. The size of the ArrayList can be increased dynamically also operations like add,remove and get can be formed once the object is created. It also ensures that the data is retrieved in the manner it was stored. The ArrayList is not thread-safe.
- **Vector :** It is thread-safe implementation of ArrayList. The methods are wrapped around a synchronized block.
- **LinkedList :** The LinkedList implements Queue interface too and provide FIFO (First In First Out) operation for add operation. It is faster than ArrayList if its mainly used forinsertion and deletion of elements.

41)What is difference between a HashMap and a HashTable?

Ans: Both collections implements Map. Both collections store value as key-value pairs. The key differences between the two are:

- HashMap is not synchronized in nature but HashTable is.
- Another difference is that, iterator in the HashMap is fail-safe while the enumerator for the HashTable isn't. Fail-safe - if the HashTable is structurally modified at any time after the iterator is created, in any way except through the iterator's own remove method, the iterator will throw a ConcurrentModificationException.
- HashMap permits null values and only one null key, while HashTable doesn't allow key or value as null.

42)Which all classes implement Set interface ?

Ans: A Set is a collection that contains no duplicate elements. More formally, a set contains no pair of elements e1 and e2 such that e1.equals(e2), and at most one null element. HashSet, SortedSet and TreeSet are the commonly used class which implements Set interface.

- **SortedSet -** It is an interface which extends Set. A the name suggest, the interface allows the data to be iterated in the ascending order or sorted on the basis of Comparator or Comparable interface. All elements inserted into the interface must implement Comparable or Comparator interface.

- **TreeSet -** It is the implementation of SortedSet interface. This implementation provides guaranteed log(n) time cost for the basic operations (add, remove and contains). The class is not synchronized. The class uses Red-Black tree data structure.
- **HashSet -** This class implements the Set interface, backed by a hash table (actually a HashMap instance). It makes no guarantees as to the iteration order of the set; in particular, it does not guarantee that the order will remain constant over time. This class permits the null element. This class offers constant time performance for the basic operations (add, remove, contains and size), assuming the hash function disperses the elements properly among the buckets.

43)What is difference between Arrays and ArrayList ?

Ans:

- Arrays are created of fix size whereas ArrayList is dynamic in nature and can vary its length. Also the size of array cannot be incremented or decremented. But with arrayList the size is variable.
- Once the array is created elements cannot be added or deleted from it. But with ArrayList the elements can be added and deleted at runtime.
- List list = new ArrayList(); list.add(1); list.add(3); list.remove(0); // will remove the element from the 1st location.
- ArrayList is one dimensional but array can be multidimensional.
- int[][][] intArray = new int[3][2][1]; // 3 dimensional array
- Array can contain objects of a single data type or class. ArrayList if not used with generic can contain objects of different classes.

44)When to use ArrayList or LinkedList ?

Ans:

- Adding new elements is pretty fast for either type of list. Inserting element to nth location in arraylist and to first location in linkedlist takes O(1).
- For the ArrayList, doing random lookup using "get" is faster O(1), but for LinkedList O(n), it's slow. It's slow because there's no efficient way to index into the middle of a linked list. Linkedlist lookup always start from 1st location.
- When removing elements, using ArrayList is slow. This is because all remaining elements in the underlying array of Object instances must be shifted down for each remove operation. But LinkedList is fast, because deletion can be done simply by changing a couple of links. So an ArrayList works best for cases where you're doing random access on the list and a LinkedList works better if you're doing a lot of editing in the middle of the list.

45)What are advantages of iterating a collection using iterator?

Ans: For loop does not allow updating the collection(add or remove) whereas Iterator does. Also Iterator can be used where there is no clue what type of collections will be used because all collections implement Iterator interface.

46)Which design pattern Iterator follows?

Ans:

- It follows Iterator design pattern. Iterator Pattern is a type of behavioral pattern. The Iterator pattern is one, which allows you to navigate through a collection of data using a common interface without knowing about the underlying implementation. Iterator should be implemented as an interface. This allows the user to implement it anyway its easier for him/ her to return data. The benefits of Iterator are about their strength to provide a common interface for iterating through collections without bothering about underlying implementation.
- Example of Iteration design pattern - Enumeration The class java.util.Enumeration is an example of the Iterator pattern. It represents and abstract means of iterating over a collection of elements in some sequential order without the client having to know the representation of the collection being iterated over. It can be used to provide a uniform interface for traversing collections of all kinds.

47)Why is it preferred to declare: List list = new ArrayList(); instead of ArrayList = new ArrayList();

Ans: It is preferred because:

- If later on code needs to be changed from ArrayList to Vector then only at the declaration place we can do that.
- The most important one – If a function is declared such that it takes list. E.g. void showDetails(List list);
- When the parameter is declared as List to the function it can be called by passing any subclass of List like ArrayList, Vector, LinkedList making the function more flexible.

48)Which data structure HashSet implements ?

Ans: HashSet implements hashTable internally to store the data. The data passed to hashset is stored as key in hashTable with null as value.

49)How to make a List (ArrayList, Vector, LinkedList) read only ?

Ans:

- A list implementation can be made read only using Collections.unmodifiableList(list). This method returns a new list.
- If a user tries to perform add operation on the new list, UnSupportedOperationException is thrown.

50)How to sort a list in reverse order?

Ans: To sort the elements in the reverse natural order of the strings, get a reverse Comparator from the Collections class with reverseOrder(). Then, pass the reverse Comparator to the sort() method.

- List list = new ArrayList();
- Comparator comp = Collections.reverseOrder();
- Collections.sort(list, comp)

51)Can a null element be added to a Treeset or HashSet ?

Ans: A null element can be added only if the set is of size 1 because when a second element is added then as per set defination a check is made to check duplicate value and comparison with null element will throw NullPointerException. HashSet is based on hashMap and can contain null element.

52)What is WeakHashMap?

Ans: A hashtable-based Map implementation with weak keys. An entry in a WeakHashMap will automatically be removed when its key is no longer in ordinary use.More precisely, the presence of a mapping for a given key will not prevent the key from being discarded by the garbage collector, that is, made finalizable, finalized, and then reclaimed.When a key has been discarded its entry is effectively removed from the map, so this class behaves somewhat differently than other Map implementations

53)Which is faster to iterate LinkedHashSet or LinkedList?

Ans: LinkedList.

54)Arrange in the order of speed - HashMap, HashTable, Collections.synchronizedMap, concurrentHashmap

Ans:

- HashMap is fastest.
- ConcurrentHashMap
- Collections.synchronizedMap

HashTable

55)What is identityHashMap?

Ans: The IdentityHashMap uses == for equality checking instead of equals(). This can be used for both performance reasons, if you know that two different elements will never be equals and for preventing spoofing, where an object tries to imitate another.

56)Can you pass List< String > to a method which accepts List< Object > ?

Ans: In first glance it looks like : String is Object so List can be used where List< Object > is required but this is not true. It will result in compilation error. This is because List< Object > can store any any thing including String, Integer etc but List< String > can only store Strings !!.

57)How HashMap works in Java ?

Ans:

- HashMap stores key-value pair in Map & Map.Entry is static nested class implementation. HashMap works on hashing algorithm and uses hashCode() and equals() method input and get methods.

- When we call put method by passing key-value pair : HashMap uses Key hashCode() with hashing to find out the index to store the key-value pair. The Map.Entry is stored in the LinkedList, so if there are already existing entry, it uses equals() method to check if the passed key already exists. If entry already exists then it overwrites the value else it creates a new entry and store this key-value Entry.
- When we call get method by passing Key : Again it uses the hashCode() to find the index in the array and then use equals() method to find the correct Entry and return it's value.
- Remember : hashCode() is used to narrow down the results for fast searching. objects having same hashCode() doesn't mean that they're same, It only means that those objects are stored in same bucket.

58)What are advantages of using Generics ?
Ans: Generics provides compile time type-safety and ensures that you only insert correct Type in collection and avoids ClassCastException during runtime

59)Map implements collection. True or false?
Ans: False, as map does not implement collection.

Generics Interview Question

1)What is Generics in Java ? What are advantages of using Generics?
Ans:

- This is one of the first interview questions asked on generics in any Java interview, mostly at beginners and intermediate level.
- Those who are coming from prior to Java 5 background knows that how inconvenient it was to store object in Collection and then cast it back to correct Type before using it.
- Generics prevents from those, it provides compile time type-safety and ensures that you only insert correct Type in collection and avoids ClassCastException in runtime.

2)Can you pass List to a method which accepts List < Object > ?
Ans: This generic interview question in Java may look confusing to any one who is not very familiar with Generics as in fist glance it looks like String is object so List can be used where List< Object > is required but this is not true. It will result in compilation error. It does make sense if you go one step further because List< Object > can store any any thing including String, Integer etc but List can only store Strings.

```
List<Object> objectList;
List stringList;
objectList = stringList; // compilation error incompatible types
```

3)How to write a generic method which accepts generic argument and return Generic Type ?

Ans: Writing generic method is not difficult, instead of using raw type you need to use Generic Type like T, E or K,V which are well known placeholders for Type, Element and Key, Value.
In simplest form a generic method would look like :

```
public V put(K key, V value){
   return cache.put(key, value);
}
```

4)How to write parameterized class in Java using Generics ?

Ans: This is an extension of previous Java generics interview question. Instead of asking to write Generic method Interviewer may ask to write a type safe class using generics. Again key is instead of using raw types you need to used generic types and always use standard place holder used in JDK.

5)Difference between List < Object > and raw type List in Java?

Ans: This Java Generics interview question is based on correct understanding of raw type in Generics.
Main difference between raw type and parameterized type List< Object > is that, compiler will not check type-safety of raw type at compile time but it will do that for parameterized type and by using Object as Type it inform compiler that it can hold any Type of Object e.g. String or Integer.
Second difference between them is that you can pass any parameterized type to raw type List but you cannot pass List to any method which accept List< Object > it will result in compilation error

6)Difference between List< ? > and List < Object > in Java?

Ans: This generics interview question may look related to previous interview questions but completely different.
List< ? > is List of unknown type while List< Object > is essentially List of any Type. You can assign List , List to List< ? > but you can not assign List to List< Object >.

```
List<?> listOfAnyType;
List<Object> listOfObject = new ArrayList<Object>();
List<String> listOfString = new ArrayList<String>();
List<Integer> listOfInteger = new ArrayList<Integer>();
listOfAnyType = listOfString;  //legal
listOfAnyType = listOfInteger; //legal
listOfObjectType = (List<Object>) listOfString; // compiler error - in-convertible types
```

Garbage Collection Interview Question

1)What is Garbage Collection in Java?

Ans:

- Garbage Collection is an automatic memory management feature.

- The process of destroying unreferenced objects is called Garbage Collection.
- Once object is unreferenced it is considered as unused object, hence JVM automatically destroys that object.
- In java it is developers responsibility to create objects and unreference those objects after usage.
-

2)Can we force garbage collector?

Ans: No, we can not force garbage collector to destroy objects , but we can request it.

3)How JVM can destroy unreferenced object?

Ans:

- JVM internally uses a daemon thread called "garbage collector" to destroy all unreferenced objects.
- A daemon thread is a service thread. Garbage Collector thread is called daemon thread because it provides services to JVM to destroy unreferenced objects.
- This thread is low priority thread. Since it is a low priority thread we cannot guarantee its execution.

3)So can you guarantee objects destruction?

Ans:

- No, we cannot guarantee objects destruction even though it is unreferenced, because we can not guarantee garbage collector's execution.
- So, we can confirm whether object is eligible for garbage collection or not.

4)Explain Java Garbage Collection in detail?

Ans:

- Java Garbage Collection is the process to identify and remove the unused objects from the memory and free space to be allocated to objects created in the future processing.
- One of the best feature of java programming language is the automatic garbage collection, unlike other programming languages such as C where memory allocation and deallocation is a manual process.
- Garbage Collector is the program running in the background that looks into all the objects in the memory and find out objects that are not referenced by any part of the program.
- All these unreferenced objects are deleted and space is reclaimed for allocation to other objects.

One of the basic way of garbage collection involves three steps:

- **Marking :** This is the first step where garbage collector identifies which objects are in use and which ones are not in use.
- **Normal Deletion :** Garbage Collector removes the unused objects and reclaim the free space to be allocated to other objects.

- **Deletion with Compacting :** For better performance, after deleting unused objects, all the survived objects can be moved to be together. This will increase the performance of allocation of memory to newer objects.

There are two problems with simple mark and delete approach:

- First one is that it's not efficient because most of the newly created objects will become unused.
- Secondly objects that are in-use for multiple garbage collection cycle are most likely to be in-use for future cycles too.

The above shortcomings with the simple approach is the reason that Java Garbage Collection is Generational and we have Young Generation and Old Generation spaces in the heap memory.

5)Which part of the memory is involved in Garbage Collection?
Ans: Heap.

6)How can we request JVM to start garbage collection process?
Ans:

- We have a method called gc() in system class as static method and also in Runtime class as non static method to request JVM to start garbage collector execution.
- System.gc();
- Runtime.getRuntime().gc();

7)What is the algorithm JVM internally uses for destroying objects?
Ans: JVM internally uses "mark and swap" algorithm for destroying objects.

8)What is responsibility of Garbage Collector?
Ans:

- Garbage Collector frees the memory occupied by the unreachable objects during the java program by deleting these unreachable objects.
- It ensures that the available memory will be used efficiently, but does not guarantee that there will be sufficient memory for the program to run.

9)When does an object become eligible for garbage collection?
Ans: An object becomes eligible for garbage collection when no live thread can access it.

10)How many times does the garbage collector calls the finalize() method for an object?
Ans: Only once.

11)How to enable /disable call of finalize() method on exit of application?
Ans: Runtime.getRuntime().runFinalizersOnExit(boolean value). Passing the boolean value true or false in the mentioned method here will enable or disable the call to finalize() method.

12)What happens if an uncaught exception is thrown during the execution of finalize() method of an object?

Ans: The exception will be ignored and the garbage collection (finalization) of that object terminates.

13)What are the different ways to call garbage collector?

Ans:

- System.gc();
- Runtime.getRuntime().gc();

14)Java Garbage Collection Types.

Ans: There are five types of garbage collection types that we can use in our applications. We just need to use JVM switch to enable the garbage collection strategy for the application. Let's look at each of them one by one.

- **Serial GC (-XX:+UseSerialGC) :** Serial GC uses the simple mark-sweep-compact approach for young and old generations garbage collection i.e Minor and Major GC. Serial GC is useful in client-machines such as our simple stand alone applications and machines with smaller CPU. It is good for small applications with low memory footprint.
- **Parallel GC (-XX:+UseParallelGC) :** Parallel GC is same as Serial GC except that is spawns N threads for young generation garbage collection where N is the number of CPU cores in the system. We can control the number of threads using -XX:ParallelGCThreads=n JVM option. Parallel Garbage Collector is also called throughput collector because it uses multiple CPUs to speed up the GC performance. Parallel GC uses single thread for Old Generation garbage collection.
- **Parallel Old GC (-XX:+UseParallelOldGC) :** This is same as Parallel GC except that it uses multiple threads for both Young Generation and Old Generation garbage collection.
- **Concurrent Mark Sweep (CMS) Collector (-XX:+UseConcMarkSweepGC) :** CMS Collector is also referred as concurrent low pause collector. It does the garbage collection for Old generation. CMS collector tries to minimize the pauses due to garbage collection by doing most of the garbage collection work concurrently with the application threads. CMS collector on young generation uses the same algorithm as that of the parallel collector. This garbage collector is suitable for responsive applications where we can't afford longer pause times. We can limit the number of threads in CMS collector using -XX:ParallelCMSThreads=n JVM option.
- **G1 Garbage Collector (-XX:+UseG1GC) :** The Garbage First or G1 garbage collector is available from Java 7 and it's long term goal is to replace the CMS collector. The G1 collector is a parallel, concurrent, and incrementally compacting low-pause garbage collector. Garbage First Collector doesn't work like other collectors and there is no concept of Young and Old generation space. It divides the heap space into multiple equal-sized heap regions. When a garbage collection is invoked, it first collects the region with lesser live data, hence "Garbage First". You can find more details about it at Garbage-First Collector Oracle Documentation.

15)List features of Garbage Collection in Java?

Ans:

- All the Garbage Collections are “Stop the World” events because all application threads are stopped until the operation completes.
- Since Young generation keeps short-lived objects, Minor GC is very fast and the application doesn’t get affected by this.
- However Major GC takes longer time because it checks all the live objects. Major GC should be minimized because it will make your application unresponsive for the garbage collection duration. So if you have a responsive application and there is a lot of Major Garbage Collection happening, you will notice timeout errors.
- The duration taken by garbage collector depends on the strategy used for garbage collection. That’s why it’s necessary to monitor and tune the garbage collector to avoid timeouts in the highly responsive applications.

Inner Class Interview Questions

1)What is an inner class?

Ans: Inner class is a class defined inside other class and act like a member of the enclosing class.

2)What are the advantages of Inner classes?

Ans:

- The embedding of inner class into the outer class in the case when the inner class is to be used only by one class i.e. the outer class makes the package more streamlined.
- Nesting the inner class code where it is used (inside the outer class) makes the code more readable and maintainable.
- The inner class shares a special relationship with the outer class i.e. the inner class has access to all members of the outer class and still have its own type is the main advantages of Inner class.
- Advantage of inner class is that they can be hidden from the other classes in the same package and still have the access to all the members (private also) of the enclosing class.
- So the outer class members which are going to be used by the inner class can be made private and the inner class members can be hidden from the classes in the same package. This increases the level of encapsulation.
- If a class A is written requires another class B for its own use, there are two ways to do this. One way is to write a separate class B or to write an inner class B inside class A. Advantage of writing the inner class B in the class A is you can avoid having a separate class.
- Inner classes are best used in the event handling mechanism and to implement the helper classes. The advantage of using inner class for event handling mechanism is that the use of if/else to select the component to be handled can be avoided. If inner classes are used each

component gets its own event handler and each event handler implicitly knows the component it is working for.

3)What is static member class?

Ans: A static member class behaves much like an ordinary top-level class, except that it can access the static members of the class that contains it. The static nested class can be accessed as the other static members of the enclosing class without having an instance of the outer class. The static class can contain non-static and static members and methods.

```
public class InnerClass {
   static class StaticInner {
      static int i = 9;
      int no = 6;
      private void method() { }
      public void method1() { }
      static void method2() { }
      final void method3() { }
   }
}
```

The static inner class can be accessed from Outer Class in the following manner :

```
InnerClass.StaticInner staticObj= new InnerClass StaticInner ();
```

No outer class instance is required to instantiate the nested static class because the static class is a static member of the enclosing class.

4)What are the different types of inner classes?

Ans: The below mentioned are the types of inner classes –

- Static member class
- Inner class
- Member class
- Anonymous class
- Local class

5)What are non static inner classes?

Ans: The different type of static inner classes are :

- **Non-static inner classes :–** Classes associated with the object of the enclosing class.
- **Member class :-** Classes declared outside a function (hence a "member") and not declared "static".

The member class can be declared as public, private, protected, final and abstract.

Example :

```
public class InnerClass {
   class MemberClass {
      public void method1() { }
   }
```

```
}
```

- **Method local class :–** The inner class declared inside the method is called method local inner class. Method local inner class can only be declared as final or abstract. Method local class can only access global variables or method local variables if declared as final.

```
public class InnerClass {
   int i = 9;
   public void method1() {
      final int k = 6;
      class MethodLocal {
         MethodLocal() {
            System.out.println(k + i);
         }
      }
   }
}
```

- **Anonymous inner class :-** These are local classes which are automatically declared and instantiated in the middle of an expression. Also, like local classes, anonymous classes cannot be public, private, protected, or static. They can specify arguments to the constructor of the superclass, but cannot otherwise have a constructor. They can implement only one interface or extend a class.

```
public class MyFrame extends JFrame {
   JButton btn = new JButton();
   MyFrame() {
      btn.addActionListener(
         new ActionListener() {
         public void actionPerformed(ActionEvent e) {}
      });
   }
}
// Anonymous class used with comparator
List l = new ArrayList();
l.add(new Parent(2));
l.add(new Parent(3));
Collections.sort(l, new Comparator() {
   public int compare(Object o1, Object o2) {
      Parent prt1 = (Parent) o1;
      Parent prt2 = (Parent) o2;
      if (prt1.getAge() > prt2.getAge()) {
         return -1;
      }
      else if(prt1.getAge() {
         return 1;
```

```
        }
        else {
            return 0;
        }
    }
});
```

6)Can a static nested class have access to the enclosing class's non-static methods or instance variables?

Ans: No.

7)What are disadvantages of using inner classes?

Ans:

- Using inner class increases the total number of classes being used by the application. For all the classes created by JVM and loaded in the memory, jvm has to perform some tasks like creating the object of type class. Jvm may have to perform some routine tasks for these extra classes created which may result slower performance if the application is using more number of inner classes.
- Inner classes get limited support of ide/tools as compared to the top level classes, so working with the inner classes is sometimes annoying for the developer.

8)What are different types of anonymous classes?

Ans: Plain old anonymous class type one–

```
class superClass{
    void get() {
        System.out.println(“ In Super class”);
    }
}
class hasAnonymous{
    superClass sc = new superClass(){
    void get() {
        System.out.println(“ In Anonymous class”);
    }
};
```

Here sc is the reference which is of type superClass which is the class extended by the anonymous class i.e. superclass of the anonymous class. The method get() is the super class method overridden by the anonymous class.

Plain old anonymous class type two –

```
interface Eat{
    public void Sweets();
}
class Meal {
```

```
    Eat food = new Eat(){
        public void
        Sweets(){
            //come implementation code goes here
        }
    };
}
```

food is reference variable of type Eatable interface which refers to the anonymous class which is the implementer of the interface Eat. The anonymous implementer class of the interface Eat implements its method Sweets() inside it.

Argument defined anonymous class –

```
interface Vehicle {
    void Wheels();
}
class Car {
    void Type(Vehical v) { }
}
class BeautifulCars {
    void getTheBeautifilCar() {
        Car c = new Car ();
        c.Type (new Vehicle (){
            public void Wheels () {
                System.out.println("It has four wheels");
            }
        });
    }
}
```

Anonymous class is defined as the argument of the method getTheBeautifilCar(), this anonymous class is the implementer of the interface Vehicle. The method of class Car getTheBeautifilCar() expects the argument as an object of type Vehicle. So first we create an object of Car referenced by the variable ‘c’. On this object of Car we call the method getTheBeautifilCar() and in the argument we create an anonymous class in place which is the implementer of interface Vehicle hence of type Vehicle.

9)What is a 'throw' keyword?If you compile a file containing inner class how many .class files are created and are all of them accessible in usual way?

Ans: If a inner class enclosed with an outer class is compiled then one .class file for each inner class and a .class file for the outer class is created.

Example :

```
class Outer {
    class Inner{ }
}
```

If you compile the above code with command % javac Outer.java Two files will be created. Though a separate inner class file is generated, the inner class file is not accessible in the usual way.
Outer.class Outer$Inner.class

10)How to access the inner class from code within the outer class?

Ans: The inner class is instantiated only through the outer class instance.

```
class Outer {
   private int InnerClass = 1;
   public void getInnerClasses() {
      Inner in = new Inner();
      System.out.println("No Of Inner classes is
         :"+ in.getClassesFromOuter());
   }
class Inner {
   public int getClassesFromOuter() {
      return InnerClass;
   }
}
```

Here the method getInnerClasses() is called on the outer class's instance and through this outer class instance the inner class instance is created.

11)How to create an inner class instance from outside the outer class instance code?

Ans: To create an instance of the inner class you must have the instance of its enclosing class.

```
class Outer {
   class Inner{ }
}
```

To create the instance of inner class from class other than the enclosing class.

```
// 1)
class OtherOuter {
   Outer out = new Outer();
   Outer.Inner in = out.new Inner();
}
// 2)
class OtherOuter {
   Outer.Inner_ out = new Outer.Inner();
}
```

12)Which modifiers can be applied to the inner class?

Ans: Following modifiers can be applied to the inner class:

- public
- private
- abstract

- final
- protected
- strictfp
- static
- static nested class

13)Can the method local inner class object access method's local variables?
Ans: No, a method local inner class object cannot access the method local variable.
Reason :

- The local variables are not guaranteed to live as long as the local inner class object.
- The method local variable live on stack and exist only till the method lives, their scope is limited upto the code inside the method they are declared in.
- But the local inner class object created within the method lives on heap and it may exist even after the method ends if in case the reference of this local inner class is passed into some other code and is stored in an instance variable.
- So we cannot be sure that the local variables will live till the method local inner class object lives, therefore the method local inner class object cannot access the method local variable.
- To access the method local variables, the variable has to be declared as final.

14)Can a method local inner class access the local final variables? Why?
Ans: Yes. Because the final variables are stored on heap and live as long as the method local inner class object may live.

15)Which modifiers can be applied to the method local inner class?
Ans: Only abstract or final keyword is allowed.

16)Can a local class declared inside a static method have access to the instance members of the outer class?
Ans: No. There is no reference available in the static method about the instance members of the outer class. The static method class cannot have access to any members of the outer class other than static members.

17)Can a method which is not in the definition of the superclass of an anonymous class be invoked on that anonymous class reference?
Ans: No. As the reference variable type of the anonymous class will be of superclass which will not know about any method defined inside the anonymous class and hence the compilation will fail.

```
class SuperClass {
   void doSomething() {
      System.out.println("In the Super class");
   }
}
class hasAnonymous {
```

```
    SuperClass sc = new SuperClass() {
        void doSomething() {
            System.out.println("In the Anonymous class");
        }
        void doStuff() {
            System.out.println("An Anonymous class method not
            present in superClass");
        }
    };
    public void doIt() {
        sc.doSomething(); // legal superClass has this method
        sc.doStuff();     // Not legal
    }
}
```

The above code will not compile as the superClass does not know about the anonymous class method doStuff().

18)Can an anonymous class define method of its own?

Ans: Yes. But there will be no way by which the methods defined in the anonymous class which are not present in its superclass be invoked. As only those methods which are defined in the superclass which the anonymous class extends be invoked. Hence defining the methods in the anonymous class will be of no use.

19)Can an anonymous class implement an interface and also extend a class at the same time?

Ans: No. An anonymous class can either extend a class or implement a single interface. If the anonymous class is extending a class then it automatically becomes the implementer of all the interfaces implemented by its superclass.

20)Can inner class have static members?

Ans: No, inner class cannot have static members.

21)Why inner class can access only final variable?

Ans:

- Local classes can most definitely reference instance variables. The reason they cannot reference non final local variables is because the local class instance can remain in memory after the method returns.
- When the method returns the local variables go out of scope, so a copy of them is needed. If the variables weren't final then the copy of the variable in the method could change, while the copy in the local class didn't, so they'd be out of synch.
- Anonymous inner classes require final variables because of the way they are implemented in Java. An anonymous inner class (AIC) uses local variables by creating a private instance field which holds a copy of the value of the local variable.

- The inner class isn't actually using the local variable, but a copy. It should be fairly obvious at this point that a "Bad Thing" can happen if either the original value or the copied value changes; there will be some unexpected data synchronization problems.
- In order to prevent this kind of problem, Java requires you to mark local variables that will be used by the AIC as final (i.e., unchangeable). This guarantees that the inner class' copies of local variables will always match the actual values.
-

22)What is a static member class?

Ans: A static member class behaves just like an ordinary top-level class, except that it can access the static members of the class that contains it.
The static nested class can be accessed just like the other static members of the enclosing class without having an instance of the outer class. The static class can contain non-static and static members as well as methods.

```
public class InnerClass {
   static class StaticInner {
      static int i = 9;
      int no = 6;
      private void method() {}
      public void method1() {}
      static void method2() {}
      final void method3() {}
   }
}
```

The static inner class can be accessed by the Outer Class in the following manner :

```
InnerClass.StaticInner staticObj = new InnerClass. StaticInner ();
```

No outer class instance is required to instantiate the nested static class because the static class is a static member of the enclosing class.

23)What are non-static inner classes?

Ans: The different types of static inner classes are:

- **Non-static inner classes -** classes associated with the object of the enclosing class.
- **Member class -** Classes declared outside a function (hence a "member") and not declared "static". The member class can be declared as public, private, protected, final and abstract. E.g.

```
public class InnerClass {
   class MemberClass {
      public void method1() { }
   }
}
```

- **Method local class -** The inner class declared inside the method is called 'method local inner class'. 'Method local inner class' can only be declared as final or abstract and can only access global variables or method local variables if declared as 'final'.

```
public class InnerClass {
   int i = 9;
   public void method1() {
      final int k = 6;
      class MethodLocal {
         MethodLocal() {
            System.out.println(k + i);
         }
      }
   }
}
```

- **Anonymous inner class -** These are the local classes which are automatically declared and instantiated in the middle of an expression. Apart from this, just like local classes, anonymous classes cannot be public, private, protected, or static. They can specify arguments to the constructor of the superclass, but cannot have a constructor of its own. They can implement only one interface or extend a class.

```
public class MyFrame extends JFrame {
   JButton btn = new JButton();
   MyFrame() {
      btn.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {}
      });
   }
}
// Anonymous class used with comparator
List l = new ArrayList();
l.add(new Parent(2));
l.add(new Parent(3));
Collections.sort(l, new Comparator() {
   public int compare(Object o1, Object o2) {
      Parent prt1 = (Parent) o1;
      Parent prt2 = (Parent) o2;
      if (prt1.getAge() > prt2.getAge()) {
         return -1;
      }
      else if(prt1.getAge() {
         return 1;
      }
      else {
         return 0;
      }
   }
```

});

24)Can a static nested class have access to the enclosing class's 'non-static methods' or 'instance variables'?

Ans: No, a static nested class cannot have an access to the enclosing class's 'non-static methods' or 'instance variables'.

25)What are the advantages of Inner classes?

Ans: The embedding of inner class into the outer class is possible only by one class when the inner class is to be used i.e. the outer class makes the package more streamlined. Nesting the inner class code where it is used (inside the outer class) makes the code more readable and maintainable.

The inner class shares a special relationship with the outer class because the inner classes has access to all members of the outer class and still have its own type which is the main advantage of Inner class. One of the other advantages of inner class is that they can be hidden from the other classes in the same package and still have the access to all the members (private also) of the enclosing class. So the outer class members which are going to be used by the inner class can be made private and the inner class members can be hidden from the classes in the same package. This increases the level of encapsulation.

If a 'Class A' is requires another 'Class B' for its own use, there are two ways to do this. One way is to write a separate 'Class B' or to write an inner 'Class B' inside 'Class A'. Advantage of writing the inner 'Class B' in the 'Class A' is that one can avoid having a separate class. Inner classes are best used in the event handling mechanism and to implement the helper classes. The advantage of using inner class for event handling mechanism is that the use of if/else to select the component to be handled can be avoided. If inner classes are used each component gets its own event handler and each event handler implicitly knows the component it is working for.

26)What are the disadvantages of using inner classes?

Ans:

- Using inner class increases the total number of classes being used by the application. For all the classes created by JVM and loaded in the memory, JVM has to perform some tasks like creating the object of type class. JVM may have to perform some routine tasks for these extra classes created which may result into slower performance if the application is using more number of inner classes.
- Inner classes get limited support of ide/tools as compared to the top level classes, so working with the inner classes is sometimes annoying for the developer.

27)What are different types of anonymous classes?

Ans: Plain old anonymous class type one –

```
class superclass {
   void doSomething() {
      System.out.println(“In the Super class”);
   }
```

```
}
class hasAnonymous {
superClass anon = new superClass(){
      void doSomething() {
         System.out.println(“In the Anonymous class”);
      }
   };
}
```

Here 'anon' is a reference variable which is of type 'superClass' that is extended by the anonymous class i.e. superclass of the anonymous class. The method 'doSomething()' is the super class method overridden by the anonymous class. Plain old anonymous class type two –

```
interface Eatable {
   public void prepareSweets();
}
class serveMeal {
   Eatable food = new Eatable(){
      public void prepareSweets(){
         //come implementation code goes here
      }
   };
}
```

Here, 'food' is reference variable of type 'Eatable interface' which refers to the anonymous class that is an implementer of the interface 'Eatable'. The anonymous implementer class of the interface 'Eatable' implements its method 'prepareSweets()' inside it.

Argument defined anonymous class –

```
interface Vehicle {
   void getNoOfWheels();
}
class Car {
   void getType(Vehicle v) { }
}
class BeautifulCars {
   void getTheBeautifilCar() {
      Car c = new Car();
      c.getType (new Vehicle () {
         public void getNoOfWheels () {
            System.out.println("It has four wheels");
         }
      });
   }
}
```

'Anonymous class' is defined as the argument of the method 'getTheBeautifilCar()' and this anonymous class is the implementer of the interface 'Vehicle'. The method of class 'Car', 'getTheBeautifilCar()' expects the argument as an object of type 'Vehicle'. So first we create an object of 'Car' referenced by the variable ‘c’. On this object of 'Car' we call the method 'getTheBeautifilCar()' and in the argument we create an anonymous class in place which is the implementer of interface 'Vehicle' hence of type 'Vehicle'

28)If you compile a file containing inner class, then how many ".class" files are created and how are all of them accessible?

Ans: If an inner class enclosed with an outer class is compiled, then there is one ".class" file and a inner class ".class" file for each outer class. e.g.

```
class EnclosingOuter
{
   class Inner{ }
}
```

If you compile the above code with command :

```
% javac EnclosingOuter.java
```

Two files will be created. Though a separate inner class file is generated, the inner class file is not accessible in the usual way.

- EnclosingOuter.class
- EnclosingOuter$Inner.class

Java 8 Interview Question

1)Describe the newly added features in Java 8?

Ans: Here are the newly added features of Java 8:

Feature Name	Description
Lambda expression	A function that can be shared or referred to as an object.
Functional Interfaces	Single abstract method interface.
Method References	Uses function as a parameter to invoke a method.
Default method	It provides an implementation of methods within interfaces enabling 'Interface evolution' facilities.

2)What are the significant advantages of Java 8?

Ans: Compact, readable, and reusable code.

- Less boilerplate code.
- Parallel operations and execution.
- Can be ported across operating systems.
- High stability.
- Stable environment.

3)What are static methods in Interfaces?

Ans: Static methods, which contains method implementation is owned by the interface and is invoked using the name of the interface, it is suitable for defining the utility methods and cannot be overridden.

4)What are some standard Java pre-defined functional interfaces?

Ans: Some of the famous pre-defined functional interfaces from previous Java versions are Runnable, Comparator, and Comparable.

Runnable: use to execute the instances of a class over another thread with no arguments and no return value.

Runnable: use to execute the instances of a class over another thread with no arguments and no return value.

Comparator: use to sort different objects in a user-defined order

Comparable: use to sort objects in the natural sort order

5)What is the lambda expression in Java and How does a lambda expression relate to a functional interface?

Ans: Lambda expression is a type of function without a name. It may or may not have results and parameters. It is known as an anonymous function as it does not have type information by itself. It is executed on-demand. It is beneficial in iterating, filtering, and extracting data from a collection. As lambda expressions are similar to anonymous functions, they can only be applied to the single abstract method of Functional Interface. It will infer the return type, type, and several arguments from the signature of the abstract method of functional interface

Reflection Interview Question

1)Explain Reflection

Ans: Java Reflection is a process of examining or modifying the run time behavior of a class at run time.

The java.lang.Class class provides many methods that can be used to get metadata, examine and change the run time behavior of a class.

The java.lang and java.lang.reflect packages provide classes for java reflection

2)What is reflection ?

Ans: It is the process of examining / modifying the runtime behaviour of an object at runtime.

3)Where reflection is used?

Ans: The Reflection API is mainly used in:

- IDE (Integrated Development Environment) e.g. Eclipse, MyEclipse, NetBeans etc.
- Debugger
- Test Tools etc.
-

4)Is It Good to use Reflection in an application ? Why ?

Ans: No. It's like challenging the design of application.

5)Why is Reflection slower ?

Ans: Because it has to inspect the metadata in the bytecode instead of just using pre compiled addresses and constants.

6)What is Java Reflection API ?

Ans:

- Java Reflection is a process of examining or modifying the run time behavior of a class at run time.
- The java.lang.Class class provides many methods that can be used to get metadata, examine and change the run time behavior of a class.
- The java.lang and java.lang.reflect packages provide classes for java reflection.

7)How Commonly used methods of Class class ?

Ans:

Method	Description
public String getName()	returns the class name
public static Class forName(String className) throws ClassNotFoundException	loads the class and returns the reference of Class class.
public Object newInstance()throws InstantiationException, IllegalAccessException	creates new instance.
public boolean isInterface()	checks if it is interface.
public boolean isArray()	checks if it is array.
public boolean isPrimitive()	checks if it is primitive.
public Class getSuperclass()	returns the superclass class reference
public Field[ ] getDeclaredFields()throws SecurityException	returns the total number of fields of this class.
public Method[ ] getDeclaredMethods()throws SecurityException	returns the total number of methods of this class.
public Constructor[ ] getDeclaredConstructors()throws SecurityException	returns the total number of constructors of this class.
public Method getDeclaredMethod(String name,Class[ ] parameterTypes)throws NoSuchMethodException,SecurityException	returns the method class instance.

8)How to get the object of Class class?

Ans: There are 3 ways to get the instance of Class class. They are as follows:

- forName() method of Class class
- getClass() method of Object class
- the .class syntax

9)How to determine class object?

Ans: Following methods of Class class is used to determine the class object:

1. **public boolean isInterface():** determines if the specified Class object represents an interface type.
2. **public boolean isArray():** determines if this Class object represents an array class.
3. **public boolean isPrimitive():** determines if the specified Class object represents a primitive type.

Let's see the simple example of reflection api to determine the object type.

```
class Simple{}
interface My{}

class Test{
   public static void main(String args[]){
      try{
         Class c=Class.forName("Simple");
         System.out.println(c.isInterface());

         Class c2=Class.forName("My");
         System.out.println(c2.isInterface());
      }
      catch(Exception e){
         System.out.println(e);
      }
   }
}
```

Output:

false true

10)What is forName() method of Class class?

Ans:

- Is used to load the class dynamically.
- Returns the instance of Class class.
- It should be used if you know the fully qualified name of class. This cannot be used for primitive types.

Let's see the simple example of forName() method.

```
class Simple{}
class Test {
   public static void main(String args[]){
      Class c=Class.forName("Simple");
      System.out.println(c.getName());
   }
}
```

Output:

Simple

Annotation Interview Question

1)What is Annotation?

Ans: Annotations are nothing but meta-data added to the Java Elements. Annotations define a java type which is similar to classes, Enums, Interfaces and they can be applied to several Java Elements.

2)Explain Annotations.

Ans: Annotations is a form of metadata which provides data about a program but itself is not part of the program. Annotations have no direct effect on the operation of the code they annotate. Annotations have a number of uses, of which some of them are as follows :

Information for the compiler : Annotations can be used by the compiler either to detect errors or suppress warnings.

Compile-time and deployment-time processing : Software tools can process annotation information to generate code, XML files and so on.

Runtime processing : Some annotations are available to be examined at runtime.

3)What are few of the Annotations that are predefined by Java?

Ans: 'Override annotation' informs the compiler that the element is meant to override an element declared in a superclass.

'Deprecated annotation' indicates that the marked element is deprecated and should no longer be used. The compiler generates a warning whenever a program uses a method, class or a field with the 'Deprecated annotation'.

'SuppressWarnings annotation' tells the compiler to suppress specific warnings which would be generated otherwise.

'FunctionalInterface annotation', introduced in Java SE 8, indicates that the type declaration is intended to be a 'functional interface', as defined by the 'Java Language Specification'.

'SafeVarargs annotation', when applied to a method or a constructor, asserts that the code does not perform potentially unsafe operations on its 'varargsparameter'. When this annotation type is used, unchecked warnings relating to 'varargs' usage are suppressed.

'FunctionalInterface annotation', introduced in Java SE 8, indicates that the type declaration is intended to be a 'functional interface', as defined by the 'Java Language Specification'.

4)Annotations were introduced with which version of Java ?

Ans: Annotations were introduced with 5th version of Java.

5)Give an Example of Annotation.

Ans: Let's say for instance a software group traditionally starts the body of every class with comments providing important information:

```
public class List3 extends List2 {
   // Author: ProfoundEdutech
   // Date: 10/08/2020
   // Current revision: 1
   // Last modified: 07/07/2021
   // By: ProfoundEdutech
   // Reviewers: Rohit
   // class code goes here
}
```

To add this same metadata with an annotation, you must first define the annotation type. The syntax for doing this is :

```
@interface ClassPreamble {
   String author();
   String date();
   int currentRevision() default 1;
   String lastModified() default "N/A";
   String lastModifiedBy() default "N/A";

   // Note use of array
   String[] reviewers();
}
```

- The annotation type definition looks similar to an interface definition where the keyword interface is preceded by the at sign (@) (@ = AT, as in annotation type). Annotation types are a form of interface, which will be covered in a later lesson. For the moment, you do not need to understand interfaces.
- The body of the previous annotation definition contains annotation type element declarations, which look a lot like methods. Note that they can define optional default values. After the annotation type is defined, you can use annotations of that type, with the values filled in, like this :

```
@ClassPreamble (
   author = "ProfoundEdutech",
   date = "10/08/2020",
   currentRevision = 1,
   lastModified = "07/07/2021",
   lastModifiedBy = "ProfoundEdutech",
   // Note array notation
   reviewers = {"Ram", "Patil"}
)
```

6)Name few meta-annotations ?

- '**Retention annotation**' specifies how the marked annotation is stored.

- '**Documented annotation**' indicates that whenever the specified annotation is used those elements should be documented using the Javadoc tool. (By default, annotations are not included in Javadoc.)
- '**Target annotation**' marks another annotation to restrict what kind of Java elements the annotation can be applied to.
- '**Inherited annotation**' indicates that the annotation type can be inherited from the super class (though this is not true by default). When the user query's the annotation type and if the class has no annotation for this type, the class's superclass is queried for the annotation type. This annotation applies only to class declarations.
- '**Repeatable annotation**' that was introduced in Java SE 8, indicates that the marked annotation can be applied more than once to the same 'declaration' or 'type' in use. For more information, refer 'Repeating Annotations'.

7)What is meta annotations ?

Ans: Annotations that apply to other annotations are called "meta-annotations".

8)What is the difference between the below given two annotations ?

Entity

Entity (name="STUDENT")

Ans: The first annotation will try to map the Class with the Table having the same name as Class whereas the second annotation will specify the Entity name as "STUDENT" and hence will try to map with Table Name "STUDENT".

9)Which are the annotations used in Junit with Junit4 ?

Ans:

- **Test Annotation :**The 'Test annotation' indicates that the public void method to which it is attached can be run as a test case.
- **Before Annotation** :The 'Before annotation' indicates that this method must be executed before each test in the class, so as to execute some pre-conditions necessary for the test.
- **BeforeClassAnnotation** :The 'BeforeClass annotation' indicates that the static method to which it is attached must be executed once and before all tests in the class.
- **After Annotation** :The 'After annotation' indicates that this method gets executed after execution of each test.
- **AfterClass Annotation** :The 'AfterClass annotation' can be used when a method needs to be executed after executing all the tests in a JUnit Test Case class so as to clean-up the set-up.
- **Ignore Annotation** :The 'Ignore annotation' can be used when you want temporarily disable the execution of a specific test.

10)What are the different ID generating strategies that are used with the @GeneratedValue annotation ?

Ans: Auto , Identity , Sequence and Table are some of the strategies that are used with the @GeneratedValue annotation.

11)How should we ignore or avoid executing set of tests ?

Ans: We can remove @Test from the respective test in order to avoid its execution. Alternatively we can put @Ignore annotation on the Junit file if we want to ignore all tests in a particular file.

12)How can we test methods individually which are not visible or declared private ?

Ans: We can either increase the test method's visibility and mark them with annotation @VisibleForTesting or we can use reflection to individually test those methods.

13)Which of the following annotation is used to avoid executing Junits ?

- **@explicit**
- **@ignore**
- **@avoid**
- **@NoTest**

Ans: "@ignore"

14)What is the @FunctionalInterface annotation ?

Ans:

- This is an informative annotation which specify that the interface is a functional interface. A 'Functional Interface' has only one 'abstract method' and many 'default methods'.
- Compiler generates an error if the interface specified with the annotation doesn't abide by the specifications for functional interface.

15)Which is the Parent Class of Annotation class?

Ans: 'Object' is said to be the parent class of annotation class.

16)How annotations are retrieved or read?

Ans: Annotations that are once defined and used in some class, can be read using 'reflection package' methods like getAnnotations(). We have to first obtain the reference to the class which contains or uses the Annotations and then we can write code as shown below:

```
Class Profound = MyClass.class;
Annotation[] annotations = Profound.getAnnotations();
for (Annotation annotation : annotations) {
}
```

17)Annotations were introduced with which version of Java ?

Ans: Java 5.

18)Explain Annotations ?

Ans: Annotations a form of metadata, provide data about a program that is not part of the program itself. Annotations have no direct effect on the operation of the code they annotate.

19)What are few of the Annotations pre defined by Java?

Ans: @Deprecated annotation indicates that the marked element is deprecated and should no longer be used. The compiler generates a warning whenever a program uses a method, class, or field with the
@Deprecated annotation.
@Override annotation informs the compiler that the element is meant to override an element declared in a superclass.
@SuppressWarnings annotation tells the compiler to suppress specific warnings that it would otherwise generate.
@FunctionalInterface annotation, introduced in Java SE 8, indicates that the type declaration is intended to be a functional interface, as defined by the Java Language Specification.

Cloning Interview Question

1)What are different type of cloning in Java?

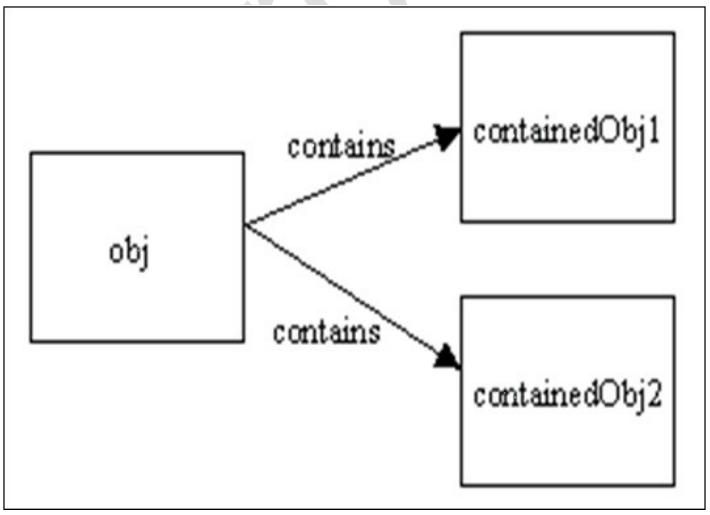
Ans: Java supports two type of cloning :-

- Deep Cloning
- Shallow Cloning

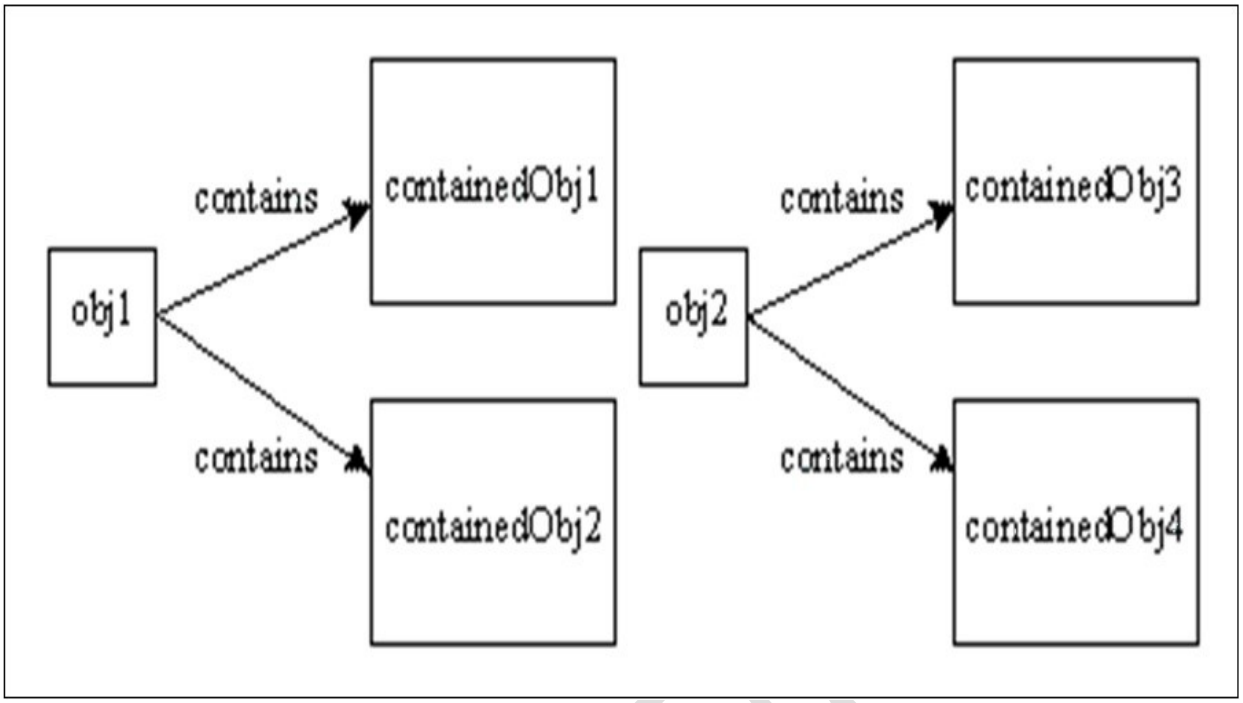
By default shallow clone is used in Java. Object class has a method clone() which does shallow cloning.

2)What is Shallow copy?

Ans: Shallow clone is copying the reference pointer to the object, which means the new object is pointing to the same memory reference of the old object. The memory usage is lower.



The shallow copy is done for obj and new object obj1 is created but contained objects of obj are not copied.



It can be seen that no new objects are created for obj1 and it is referring to the same old contained objects. If either of the contained Obj contain any other object than no new reference is created.

3)What is difference between deep and shallow cloning?

Ans: The differences are as follows :

Consider a class : public class MyData { String id; Map myData; }

The shallow copying of this object will be pointing to the same memory reference as the original object. So a change in myData by either original or cloned object will be reflected in other also.

But in deep copying there will memory allocated and values assigned to the property will be same. Any change in object will not be reflected in other.

Shallow copying is default cloning in Java which can be achieved using Object.clone() method of Object class. For deep copying override the clone method to create new object and copy its values.

4)What are disadvantages of deep cloning ?

Ans: Disadvantages of using Serialization to achieve deep cloning :-

- Serialization is more expensive than using object.clone().
- Not all objects are serializable.
- Serialization is not simple to implement for deep cloned object.

Design Pattern Interview Question

1)What is design pattern? Have you used any design pattern in your code ?

Ans: Design patterns are tried and tested way to solve particular design issues by various programmers in the world. Design patterns are extension of code reuse.

2)Can you name few design patterns used in standard JDK library?

Ans:

- Decorator design pattern which is used in various Java IO classes.
- Singleton pattern which is used during runtime.
- Calendar and various other classes.
- Factory pattern which is used along with various Immutable classes like Boolean e.g. Boolean.valueOf.

Observer pattern which is used in Swing and many event listener frameworks

3)What is Singleton design pattern in Java ?

Ans: Singleton pattern focuses on sharing expensive object in whole system. Only one instance of a particular class is maintained in the whole application which is shared by all modules.
Java.lang.Runtime is a common example of Singleton design pattern

4)What is the main benefit of using factory pattern? Where do you use it?

Ans: Factory pattern's main benefit is the increased level of encapsulation while creating objects. If one uses Factory to create object one can later replace original implementation of Products or classes with more advanced and high performance implementation without any change on client layer.

5)What is Observer design pattern in Java?

Ans: Observer design pattern is based on communicating changes in state of object to observers so that they can take there action. Simple example is a weather system where change in weather must be reflected in 'Views' to show to public

6)What is Decorator design pattern?

Ans: Decorator pattern enhances capability of individual object. Java IO uses decorator pattern extensively and classical example is Buffered class say for instance BufferedReader and BufferedWriter which enhances Reader and Writer objects to perform Buffer level reading and writing for improved performance

7)When do you overload a method in Java and when do you override it ?

Ans: Rather a simple question for experienced designer in Java. If one sees implementation of a class has different ways of doing certain things then overriding is the way to go while overloading is doing same thing but with different inputs. Method signature varies in case of overloading but not in case of overriding in java.

8)Give an example where we will prefer abstract class over interface.

Ans: Both interface and abstract class follow "writing code for interface then implementation" design principle which adds flexibility in code which is quite important to tackle with changing requirements. Here are some pointers which help you to answer this question:

- In Java we can only extend one class but implement multiple interfaces. So if we extend a class we may lost our chance of extending another class.
- Interfaces are used to represent adjective or behavior e.g. Runnable, Clonable, Serializable etc. So if we use an abstract class to represent behavior our class won't be Runnable and Clonable at same time because we cannot extend two classes in Java but if we use interface our class can have multiple behavior at the same time.
- At a critical time application prefers abstract class as it is slightly faster than interface.
- If there is a genuine common behavior across the inheritance hierarchy which can be coded better at one place, then abstract class is preferred. Sometimes interface and abstract class can work together also where we define function in interface whereas default functionality in abstract class

9)What is Garbage Collection in Java?

Ans:

- Garbage Collection is an automatic memory management feature.
- The process of destroying unreferenced objects is called Garbage Collection.
- Once object is unreferenced it is considered as unused object, hence JVM automatically destroys that object.
- In java it is developers responsibility to create objects and unreference those objects after usage.

10)Can we force garbage collector?

Ans: No, we can not force garbage collector to destroy objects , but we can request it.

11)How JVM can destroy unreferenced object?

Ans:

- JVM internally uses a daemon thread called "garbage collector" to destroy all unreferenced objects.
- A daemon thread is a service thread. Garbage Collector thread is called daemon thread because it provides services to JVM to destroy unreferenced objects.
- This thread is low priority thread. Since it is a low priority thread we cannot guarantee its execution.

12)So can you guarantee objects destruction?

Ans:

- No, we cannot guarantee objects destruction even though it is unreferenced, because we can not guarantee garbage collector's execution.
- So, we can confirm whether object is eligible for garbage collection or not.

13)Explain Java Garbage Collection in detail?

Ans:

- Java Garbage Collection is the process to identify and remove the unused objects from the memory and free space to be allocated to objects created in the future processing.
- One of the best feature of java programming language is the automatic garbage collection, unlike other programming languages such as C where memory allocation and deallocation is a manual process.
- Garbage Collector is the program running in the background that looks into all the objects in the memory and find out objects that are not referenced by any part of the program.
- All these unreferenced objects are deleted and space is reclaimed for allocation to other objects.

One of the basic way of garbage collection involves three steps:

- **Marking :** This is the first step where garbage collector identifies which objects are in use and which ones are not in use.
- **Normal Deletion :** Garbage Collector removes the unused objects and reclaim the free space to be allocated to other objects.
- **Deletion with Compacting :** For better performance, after deleting unused objects, all the survived objects can be moved to be together. This will increase the performance of allocation of memory to newer objects.

There are two problems with simple mark and delete approach:

- First one is that it’s not efficient because most of the newly created objects will become unused.
- Secondly objects that are in-use for multiple garbage collection cycle are most likely to be in-use for future cycles too.

The above shortcomings with the simple approach is the reason that Java Garbage Collection is Generational and we have Young Generation and Old Generation spaces in the heap memory.

14)Which part of the memory is involved in Garbage Collection?

Ans: Heap.

15)How can we request JVM to start garbage collection process?

Ans:

- We have a method called gc() in system class as static method and also in Runtime class as non static method to request JVM to start garbage collector execution.
- System.gc();
- Runtime.getRuntime().gc();

16)What is the algorithm JVM internally uses for destroying objects?

Ans: JVM internally uses "mark and swap" algorithm for destroying objects.

17)What is responsibility of Garbage Collector?

Ans:

- Garbage Collector frees the memory occupied by the unreachable objects during the java program by deleting these unreachable objects.
- It ensures that the available memory will be used efficiently, but does not guarantee that there will be sufficient memory for the program to run.

18)When does an object become eligible for garbage collection?
Ans: An object becomes eligible for garbage collection when no live thread can access it.

19)How many times does the garbage collector calls the finalize() method for an object?
Ans: Only once

20)How to enable /disable call of finalize() method on exit of application?
Ans: Runtime.getRuntime().runFinalizersOnExit(boolean value). Passing the boolean value true or false in the mentioned method here will enable or disable the call to finalize() method.

21)What happens if an uncaught exception is thrown during the execution of finalize() method of an object?
Ans: The exception will be ignored and the garbage collection (finalization) of that object terminates.

22)What are the different ways to call garbage collector?
Ans:

- System.gc();
- Runtime.getRuntime().gc();

23)Java Garbage Collection Types.
Ans: There are five types of garbage collection types that we can use in our applications. We just need to use JVM switch to enable the garbage collection strategy for the application. Let's look at each of them one by one.

- **Serial GC (-XX:+UseSerialGC) :** Serial GC uses the simple mark-sweep-compact approach for young and old generations garbage collection i.e Minor and Major GC. Serial GC is useful in client-machines such as our simple stand alone applications and machines with smaller CPU. It is good for small applications with low memory footprint.
- **Parallel GC (-XX:+UseParallelGC) :** Parallel GC is same as Serial GC except that is spawns N threads for young generation garbage collection where N is the number of CPU cores in the system. We can control the number of threads using -XX:ParallelGCThreads=n JVM option. Parallel Garbage Collector is also called throughput collector because it uses multiple CPUs to speed up the GC performance. Parallel GC uses single thread for Old Generation garbage collection.
- **Parallel Old GC (-XX:+UseParallelOldGC) :** This is same as Parallel GC except that it uses multiple threads for both Young Generation and Old Generation garbage collection.

- **Concurrent Mark Sweep (CMS) Collector (-XX:+UseConcMarkSweepGC) :** CMS Collector is also referred as concurrent low pause collector. It does the garbage collection for Old generation. CMS collector tries to minimize the pauses due to garbage collection by doing most of the garbage collection work concurrently with the application threads. CMS collector on young generation uses the same algorithm as that of the parallel collector. This garbage collector is suitable for responsive applications where we can't afford longer pause times. We can limit the number of threads in CMS collector using -XX:ParallelCMSThreads=n JVM option.

G1 Garbage Collector (-XX:+UseG1GC) : The Garbage First or G1 garbage collector is available from Java 7 and it's long term goal is to replace the CMS collector. The G1 collector is a parallel, concurrent, and incrementally compacting low-pause garbage collector. Garbage First Collector doesn't work like other collectors and there is no concept of Young and Old generation space. It divides the heap space into multiple equal-sized heap regions. When a garbage collection is invoked, it first collects the region with lesser live data, hence "Garbage First". You can find more details about it at Garbage-First Collector Oracle Documentation.

24)List features of Garbage Collection in Java?

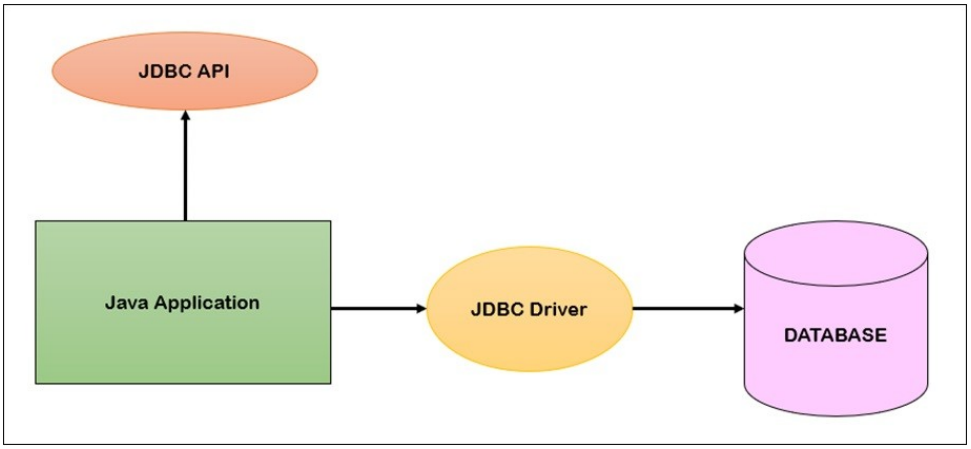
Ans:

- All the Garbage Collections are "Stop the World" events because all application threads are stopped until the operation completes.
- Since Young generation keeps short-lived objects, Minor GC is very fast and the application doesn't get affected by this.
- However Major GC takes longer time because it checks all the live objects. Major GC should be minimized because it will make your application unresponsive for the garbage collection duration. So if you have a responsive application and there is a lot of Major Garbage Collection happening, you will notice timeout errors.
- The duration taken by garbage collector depends on the strategy used for garbage collection. That's why it's necessary to monitor and tune the garbage collector to avoid timeouts in the highly responsive applications.

JDBC Interview Question

1)What is Java JDBC ?

Ans: Java JDBC is a java API to connect and execute query with the database. JDBC API uses jdbc drivers to connect with the database.



2)What are the types of statements in JDBC?

Ans: JDBC API has 3 Interfaces and their key features of these are as follows :

Statement : which is used to run simple SQL statements like select and update. Statement interfaces use for general-purpose access to your database. It is useful when you are using static SQL statements at runtime. The Statement interface cannot accept parameters.

Prepared Statement : A SQL statement is pre-compiled and stored in a Prepared Statement object. It is used to run Pre compiled SQL. This object can then be used to efficiently execute this statement multiple times. The object of Prepared Statement class can be created using Connection.prepareStatement() method. This extends Statement interface.

Callable Statement : This interface is used to execute the stored procedures. This extends Prepared Statement interface. The object of Callable Statement class can be created using Connection.prepareCall() method.

What causes "No suitable driver" error?

"No suitable driver" is occurs during a call to the DriverManager.getConnection method, may be of any of the following reason:

- Due to failing to load the appropriate JDBC drivers before calling the getConnection method.
- It can be specifying an invalid JDBC URL, one that is not recognized by JDBC driver.
- This error can occur if one or more the shared libraries needed by the bridge cannot be loaded.

3)Why use JDBC ?

Ans: Before JDBC, ODBC API was the database API to connect and execute query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

4)What restrictions are placed on method overriding?

Ans: The restriction on method overriding is the signature of the method.

- The signature is the number, type, and order of the arguments passed to a method.

- Overridden methods must have the same name, argument list, and return type
- Any method which has the same name cannot have the same signature.
- They can have the same return types in the same scope.
- The compiler uses the signature to detect which overloaded method to refer when a overloaded method is called.
- If two methods have the same name and signature the compiler will throw a runtime error.

5)What does setAutoCommit do?

Ans:

- setAutoCommit() invoke the commit state query to the database. To perform batch updation we use the setAutoCommit() which enable us to execute more than one statement together, which in result minimize the database call and send all statement in one batch.
- setAutoCommit() allowed us to commit the transaction commit state manually the default values of the setAutoCommit() is true.

6)What are types of JDBC drivers?

Ans: There are four types of drivers defined by JDBC as follows :

- **JDBC/ODBC :** These require an ODBC (Open Database Connectivity) driver for the database to be installed. It is used for local connection.
- **Native API (partly-Java driver) :** This type of driver uses a database API to interact with the database. It also provides no host redirection.
- **Network Protocol Driver :** It makes use of a middle-tier between the calling program and the database. The client driver communicates with the net server using a database-independent protocol and the net server translates this protocol into database calls.
- **Native Protocol Drive :** This has a same configuration as a type 3 driver but uses a wire protocol specific to a particular vendor and hence can access only that vendor's database

7)Why Prepared Statements are faster?

Ans:

- Prepared execution is faster than direct execution for statements executed more than three or four times because the statement is compiled only once.
- Prepared statements and JDBC driver are linked with each other.
- We can bind drivers with columns by triggering the query into the database.
- When we execute Connection.prepareStatement(), all the columns bindings take place, in order to reduce the time.

8)What are the differences between setMaxRows(int) and SetFetchSize(int)?

Ans: The difference between setFetchSize and setMaxRow are :

- setFetchSize(int) defines the number of rows that will be read from the database when the ResultSet needs more rows whereas setMaxRows(int) method of the ResultSet specifies how many rows a ResultSet can contain at a time.

- In setFetchSize(int), method in the java.sql.Statement interface will set the 'default' value for all the ResultSet derived from that Statement whereas in setMaxRow(int) default value is 0, i.e. all rows will be included in the ResultSet.
- The setMaxRows affects the client side JDBC object while the setFetchSize affects how the database returns the ResultSet data.

9)How can I manage special characters when I execute an INSERT query?

Ans: The special characters meaning in SQL can be preceded with a special escape character in strings, e.g. "\". In order to specify the escape character used to quote these characters, include the following syntax on the end of the query :

{escape 'escape-character'}

For example, the query,

SELECT NAME FROM IDENTIFIERS WHERE ID LIKE '_%' {escape '\'}

// finds identifier names that begin with an underscore.

10)How do I insert an image file (or other raw data) into a database?

Ans: All raw data types should be read and uploaded to the database as an array of bytes, byte[].

- Originating from a binary file.
- Read all data from the file using a FileInputStream.
- Create a byte array from the read data.
- Use method setBytes(int index, byte[] data); of java.sql.PreparedStatement to upload the data.

11)Discuss the significances of JDBC.

Ans: The significances are given below :

- JDBC is the acronym stands for Java Database Connectivity.
- Java Database Connectivity (JDBC) is a standard Java API .
- Its purpose is to interact with the relational databases in Java.
- JDBC is having a set of classes & interfaces which can be used from any Java application.
- By using the Database Specific JDBC drivers, it interacts with a database without the applications of RDBMS.

12)Name the new features added in JDBC 4.0.

Ans: The major features introduced in JDBC 4.0 are :

- Auto-loading by JDBC driver class.
- Enhanced Connection management.
- RowId SQL enabled.
- DataSet implemented by SQL by using Annotations.
- Enhancements of SQL exception handling.
- Supporting SQL XML files.

13)Is it possible to connect to multiple databases simultaneously? Using single statement can one update or extract data from multiple databases?

Ans: Yes, it is possible but it depends upon the capabilities of the specific driver implementation, we can connect to multiple databases at the same time. We doing following steps :

- Minimum one driver will be used to handle the commits transaction for multiple connections.
- To update and extract data from the different database we use single statement for this we need special middleware to deal with multiple databases in a single statement or to effectively treat them as one database.

14)What is the benefit of having JdbcRowSet implementation? Why do we need a JdbcRowSet like wrapper around ResultSet?

Ans: The JdbcRowSet implementation is a wrapper around a ResultSet object has following advantages over ResultSet :

- It makes possible to use the ResultSet object as a JavaBeans component.A JdbcRowSet can be used as a JavaBeans component, thus it can be created and configured at design time and executed at run time.
- It can be used to make a ResultSet object scrollable and updatable. All RowSet objects are by default scrollable and updatable.

15)I have the choice of manipulating database data using a byte[] or a java.sql.Blob. Which has best performance?

Ans: We use java.sql.Blob, because of following reason :

- It does not extract any data from the database until we trigger a query to the database.
- Use byte[] for inserting data in the database when data is not upload in the database till yet.
- java.sql.Blob is used when extraction of the data is performed.

16)How do Java applications access the database using JDBC?

Ans: Java applications access the database using JDBC by :

- Communicating with the database for Loading the RDBMS specific JDBC driver
- Opening the connection with database
- Sending the SQL statements and get the results back.
- Creating JDBC Statement object which contains SQL query.
- Executing statement to return the resultset(s) containing the tuples of database table which is a result of SQL query.
- Processing the result set.
- Closing the connection.

17)Explain the life cycle of JDBC.

Ans: The life cycle for a servlet comprises of the following phases :

- **DriverManager :** for managing a list of database drivers.
- **Driver :** for communicating with the database.

- **Connection :** for interfacing with all the methods for connecting a database.
- **Statement :** for encapsulating an SQL statement for passing to the database which had been parsed, compiled, planned and executed.
- **ResultSet :** for representing a set of rows retrieved for the query execution.

18)Describe how the JDBC application works.

Ans: A JDBC application may be divided into two layers :

- Driver layer
- Application layer
- The Driver layer consists of DriverManager class & the JDBC drivers.
- The Application layer begins after putting a request to the DriverManager for the connection.
- An appropriate driver is chosen and used for establishing the connection.
- This connection is linked to the application layer.
- The application needs the connection for creating the Statement kind of objects by which the results are obtained.

19)Briefly tell about the JDBC Architecture.

Ans: The JDBC Architecture consists of two layers :

- The JDBC API
- The JDBC Driver API
  - The JDBC API provides the application-JDBC Manager connection.
  - The JDBC Driver API supports the JDBC Manager-to-Driver Connection.
  - The JDBC API interacts with a driver manager, database-specific driver for providing transparent connectivity for the heterogeneous databases.
  - The JDBC driver manager authenticates that the correct driver has been used to access each data source.
  - The driver manager supports multiple concurrent drivers connected to the multiple heterogeneous databases.

20)What are DML and DDL?

Ans:

- Data Manipulation Language (DDL) this portion of the SQL standard is concerned with manipulating the data in a database as opposed to the structure of a database. The DML deals with the SELECT, INSERT, DELETE, UPDATE, COMMIT and ROLLBACK.
- Data Definition Language (DDL) this portion of the SQL standard is concerned with the creation, deletion and modification of database objects like tables, indexes and views. The core verbs for DDL are CREATE, ALTER and DROP. While most DBMS engines allow DDL to be used dynamically, it is often not supported in transactions.

21)Explain Basic Steps in writing a Java program using JDBC.

Ans: JDBC makes the interaction with RDBMS simple and intuitive. When a Java application needs to access database :

- Load the RDBMS specific JDBC driver because this driver actually communicates with the database.
- Open the connection to database, for sending SQL statements and get results back.
- Create JDBC Statement object containing SQL query.
- Execute statement which returns result set. ResultSet contains the tuples of database table as a result of SQL query.
- Process the result set.
- Close the connection.

22)What does the JDBC Driver interface do?

Ans:

- The JDBC Driver interface provides vendor-specific customized implementations of the abstract classes.
- It is provided normally by the JDBC API.
- For each vendor the driver provides implementations of the java.sql.Connection, PreparedStatement, Driver, Statement, ResultSet and CallableStatement.

23)How a database driver can be loaded with JDBC 4.0 / Java 6?

Ans:

- By providing the JAR file , the driver must be properly configured.
- The JAR file is placed in the classpath.
- It is not necessary to explicitly load the JDBC drivers by using the code like Class.forName() to register in the JDBC driver.
- The DriverManager class looks after this, via locating a suitable driver at the time when the DriverManager.getConnection() method is called.
- This feature provides backward-compatibility, so no change is needed in the existing JDBC code.

24)What is a Statement ?

Ans:

- The Statement acts just like a vehicle via which SQL commands are sent.
- By the connection objects, we create the Statement kind of objects.
  - Statement stmt = conn.createStatement();
- This method returns the object, which implements the Statement interface.

25)What is represented by the connection object?

Ans:

- The connection object represents the communication context.
- All the communication with the database is executed via the connection objects only.
- Connection objects are used as the main linking elements.

26)Define PreparedStatement.

Ans:

- A Preparedstatement is an SQL statement which is precompiled by the database.
- By precompilation, the prepared statements improve the performance of the SQL commands that are executed multiple times (given that the database supports prepared statements).
- After compilation, prepared statements may be customized before every execution by the alteration of predefined SQL parameters.
  PreparedStatement pstmt = conn.prepareStatement("UPDATE data= ? WHERE vl = ?");
  pstmt.setBigDecimal(1, 1200.00);
  pstmt.setInt(2, 192);

27)Differentiate between a Statement and a PreparedStatement.

Ans:

- A standard Statement is used for creating a Java representation for a literal SQL statement and for executing it on the database.
- A PreparedStatement is a precompiled Statement.
- A Statement has to verify its metadata in the database every time.
- But, the prepared statement has to verify its metadata in the database only once.
- If we execute the SQL statement, it will go to the STATEMENT.
- But, if we want to execute a single SQL statement for the multiple number of times, it'll go to the PreparedStatement.

28)What is the function of setAutoCommit?

Ans:

- When a connection is created, it is in auto-commit mode.
- This means that each individual SQL statement is to be treated as a single transaction .
- The setAutoCommit will be automatically committed just after getting executed.
- The way by which two or more statements are clubbed into a transaction to disable the auto-commit mode is :
- con.setAutoCommit (false);
- Once auto-commit mode is disabled, no SQL statements will be committed until we call the method 'commit' explicitly.
- con.setAutoCommit(false);
- PreparedStatement updateSales = con.prepareStatement(
- "UPDATE COFFEE SALES = ? WHERE COF_NAME LIKE ?");
- updateSales.setInt(1, 50); updateSales.setString(2, "Colombian");
- updateSales.executeUpdate();
- PreparedStatement updateTotal = con.prepareStatement(
- "UPDATE COFFEES SET TOTAL = TOTAL + ? WHERE COF_NAME LIKE ?");
- updateTotal.setInt(1, 50);
- updateTotal.setString(2, "Colombian");

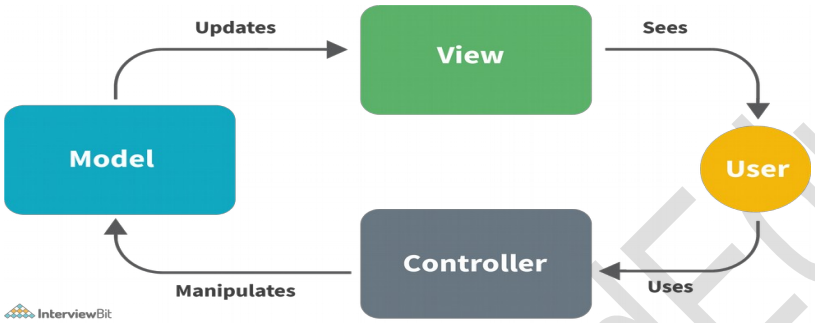
- updateTotal.executeUpdate();
- con.commit();
- con.setAutoCommit(true);

MVC Interview Questions

1)Explain Model, View and Controller in Brief.

Ans: A model can be defined as the data that will be used by the program. Commonly used examples of models in MVC are the database, a simple object holding data (such as any multimedia file or the character of a game), a file, etc.

A view is a way of displaying objects (user interfaces) within an application. This is the particular vertical through which end users will communicate.



A controller is the third vertical which is responsible for updating both models and views. It accepts input from users as well as performs the equivalent update. In other words, it is the controller which is responsible for responding to user actions.

2)How will you define the 3 logical layers of MVC?

Ans: The 3 logical layers of MVC can be defined as follows:

- Model logic acts as a business layer.
- View logic acts as a display layer.
- Controller logic acts as input control.