Core Java
Interview Questions & Answers

Prepared By : Krishna Agrawal

Link : https://www.linkedin.com/in/staylearner-krishna-agrawal/

Topics Covered

- THREADING
- Collections
- EXCEPTION HANDLING
- GARBAGE COLLECTOR
- OOPS Concepts
- SERIALIZATION
- Immutable Class and String
- Basic Core Java

THREADING

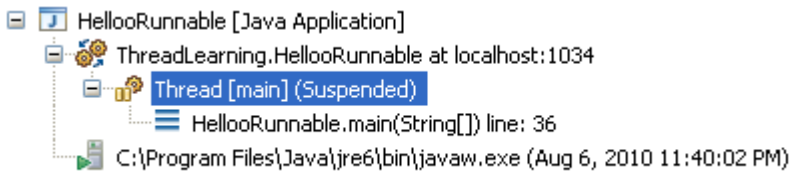
What is a Thread?

In Java, "thread" means two different things:

- ❖An instance of class java.lang.Thread.
- ❖A thread of execution.

An instance of Thread is just an object. Like any other object in Java, it has variables and methods, and lives and dies on the heap. But a thread of execution is an individual process (a "lightweight" process) that has its own call stack. In Java, there is one thread per call stack—or, to think of it in reverse, one call stack per thread. Even if we don't create any new threads in our program, threads are running at back.

The main() method, that starts the whole ball rolling, runs in one thread, called the main thread. If we looked at the main call stack, we would see that main() is the first method on the stack— the method at the bottom. But as soon as you create a new thread, a new stack materializes and methods called from that thread run in a call stack that's separate from the main() call stack.



What are the advantages or usage of threads?

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. Unfortunately, this loop basically simulates an operating system scheduler. In Java, each view can be assigned a thread to provide continuous updates.
• Programs that need to respond to user-initiated events can set up service routines to handle the events without having to insert code in the main routine to look for these events.

Threads provide a high degree of control.

• Imagine launching a complex computation that occasionally takes longer than is satisfactory. A "watchdog" thread can be activated that will "kill" the computation if it becomes costly, perhaps in favor of an alternate, approximate solution. Note that sequential programs must muddy the computation with termination code, whereas, a Java program can use thread control to non-intrusively supervise any operation.

Threaded applications exploit parallelism.

• A computer with multiple CPUs can literally execute multiple threads on different functional units without having to simulating multi-tasking ("time sharing").
• On some computers, one CPU handles the display while another handles computations or database accesses, thus, providing extremely fast user interface response times.

What is difference between thread and process?

Differences between threads and processes are:-
1. Threads share the address space of the process that created it; processes have their own address.

2. Threads have direct access to the data segment of its process; processes have their own copy of the data segment of the parent process.

3. Threads can directly communicate with other threads of its process; processes must use inter-process communication to communicate with sibling processes.

4. Threads have almost no overhead; processes have considerable overhead.

5. New threads are easily created; new processes require duplication of the parent process.

6. Threads can exercise considerable control over threads of the same process; processes can only exercise control over child processes.

7. Changes to the main thread (cancellation, priority change, etc.) may affect the behavior of the other threads of the process; changes to the parent process do not affect child processes.

What is the difference between preemptive scheduling and time slicing?

- ❖ Under preemptive scheduling, the highest priority task executes until it enters the waiting or dead states or a higher priority task comes into existence.
- ❖ Under time slicing, a task executes for a predefined slice of time and then reenters the pool of ready tasks. The scheduler then determines which task should execute next, based on priority and other factors.

Does each thread have its own thread stack?

Yes each thread has its own call stack. For e.g.

```
Thread t1 = new Thread();
Thread t2 = new Thread();
Thread t3 = t1
```

In the above example t1 and t3 will have the same stack and t2 will have its own independent stack.

What all constructors are present in the Thread class?

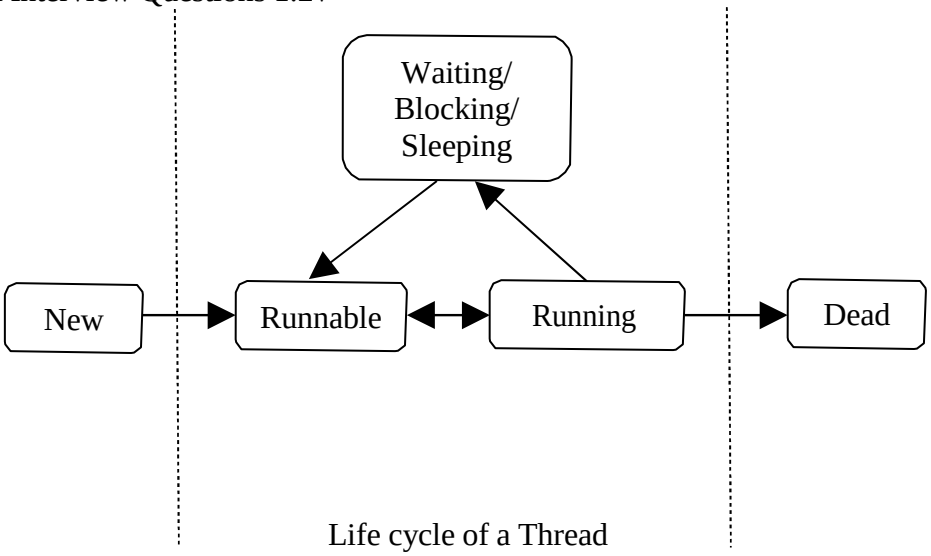
Thread()
Thread(Runnable target)
Thread(Runnable target, String name)
Thread(String name)

Why threads block or enter into waiting state on I/O?
Threads block on I/O (i.e. enters the waiting state), so that other threads may execute while the I/O operation is performed.

What is the initial state of a thread when it is created and started?
The thread is in ready state

What are the different states of a thread's lifecycle?

1) 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.
2) 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' also. In this state the thread is considered to be alive.
3) 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'.
4) Waiting/Blocked/Sleeping – In these states the thread is said to be alive but not Runnable. The thread switches to this state because of reasons like **wait method** called or **sleep method** has been called on the running thread or thread might be **waiting for some I/O** resource so blocked.
5) 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 can't be started again**. If a start() method is invoked on a dead thread a runtime exception will occur.



Life cycle of a Thread

What is synchronization?

Synchronization is a process of controlling the access of shared resources (like instance variables, static variables etc) by the multiple threads in such a manner that only one thread can access one resource at a time. In non synchronized multithreaded application, it is possible for one thread to modify a shared object while another thread is in the process of using or updating the object's value. Synchronization prevents such type of data corruption.

Can the variables or classes be synchronized?

No. Only methods can be synchronized.

Why would you use a synchronized block vs. synchronized method?

Synchronized blocks place locks for shorter periods (fine grained locking) than synchronized methods.

What is the difference when the synchronized keyword is applied to a static method or to a non static method?

When a synch non static method is called a lock is obtained on the object. When a synch static method is called a lock is obtained on the class, 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 is accessing a synch non static method, then the other thread can access the synch static method but can’t access the synch non static method.

Can a class have both synchronized and non-synchronized methods?

Yes a class can have both synchronized and non-synchronized methods.

If a class has a synchronized method and non-synchronized method, can multiple threads execute the non-synchronized methods?

Yes, multiple threads can access the non-synchronized methods.

Can a thread call multiple synchronized methods on the object of which it hold the lock?

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.

Can static methods be synchronized?

Yes. As 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.

Can two threads call two different static synchronized methods of the same class?

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.

Does a static synchronized method block a non-static synchronized method?

No, as 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.

What is the difference between yield() and sleep()?

- yield() allows the current the 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**.

What is the difference between wait() and sleep()?

- wait() allows thread to release the lock and goes to suspended state. The thread is only active when a notify() or notifAll() method is called for the same object. wait() is a method of Object class.
- 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. sleep() is a method of Object class.

What is difference between notify() and notfiyAll()?

- notify( ) wakes up the first thread that called wait( ) on the same object.
- notifyAll( ) wakes up all the threads that called wait( ) on the same object. The highest priority thread will run first.

There are two classes: A and B. The class B need to inform a class A when some important event has happened. What Java technique would you use to implement it?

If these classes are threads we would consider notify() or notifyAll(). For regular classes we can use the Observer interface.

What is the purpose of the wait(), notify(), and notifyAll() methods?

The wait(),notify(), and notifyAll() methods are used to provide an efficient way for threads to wait for a shared resource. When a thread executes an object's wait() method, it enters the waiting state. It only enters the ready state after another thread invokes the object's notify() or notifyAll() methods..

What happens when you invoke a thread's interrupt method while it is sleeping or waiting?

When a task's interrupt() method is executed, the task enters the ready state. The next time the task enters the running state, an InterruptedException is thrown.

What happens if a start method is not invoked and the run method is directly invoked?

If we do not call a start() method on the newly created thread instance, thread is not considered to be alive. If the start() method is not invoked and the run() method is directly called on the Thread instance, the code inside the run() method **will not run in a separate new thread** but it will start running in the existing thread.

What is a volatile keyword?

In general each thread has its own copy of variable, such that one thread is not concerned with the value of same variable in the other thread. But sometime this may not be the case. Consider a scenario in which the count variable is holding the number of times a method is called for a given class irrespective of any thread calling, in this case irrespective of thread access the count has to be increased so the count variable is declared as volatile. The copy of volatile variable is stored in the main memory, so every time a thread access the variable even for reading purpose the local copy is updated each time from the main memory. The volatile variable also has performance issues.

What happens when start() method is called?

A new thread of execution with a new call stack starts. The state of thread changes from new to ‘Runnable state’. When the thread gets chance to execute its target run() method starts to run.

If code running is a thread creates a new thread what will be the initial priority of the newly created thread?

When a code running in a thread creates a new thread object, the priority of the new thread is set equal to the priority of the thread which has created it.

Once a thread has been started can it be started again?

No. Only a thread can be started only once in its lifetime. If we 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 'dead state' thread can't be restarted.

Can the start() method of the Thread class be overridden? If yes should it be overridden?

Yes the start() method can be overridden. But it should not be overridden as its implementation in thread class has the code to create a new executable thread and is specialized.

When JVM starts up, which thread will be started up first?

When JVM starts up the thread executing main method is started.

What are the daemon threads?

Daemon threads are service provider threads running in the background, these not used to run the application code generally. When all user threads (non-daemon threads) complete their execution, 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 thread. 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.

What is an object's lock and which objects have locks?

An object's lock is a mechanism that is used by multiple threads to obtain synchronized access to the object. A thread may execute a synchronized method of an object only after it has acquired the object's lock. All objects and classes have locks. A class's lock is acquired on the class's Class object.

How many locks does an object have?

Each object has only one lock.

What happens when a thread cannot acquire a lock on an object?

If a thread attempts to execute a synchronized method or synchronized statement and is unable to acquire an object's lock, it enters the waiting state until the lock becomes available.

If a thread goes to sleep does it hold the lock?

Yes, when a thread goes to sleep it does not release the lock.

Can a thread hold multiple locks at the same time?

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.

When does deadlock occur and how to avoid it?

When a locked object tries to access a locked object which is trying to access the first locked object. When the threads are waiting for each other to release the lock on a particular object, deadlock occurs.

What is a better way of creating multithreaded application? Extending Thread class or implementing Runnable?

If a class is made to extend the thread class to have a multithreaded application then this subclass of Thread can't extend any other class and the required application will have to be added to this class as it can not 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.

The first strategy, which employs a `Runnable` object, is more general, because the `Runnable` objects can subclass a class other than `Thread`. The second strategy is easier to use in simple applications, but is limited by the fact that your task class must be a descendant of `Thread`. *The first approach is more flexible*, which separates the `Runnable` task from the `Thread` object that executes the task. Not only is this approach more flexible, but it is applicable to the high-level thread management APIs. Extending the Thread class is not a good OO practice, as we will not be able to extend other class. Hence we use this approach when we have a more specialized version of a Thread class.

What is thread starvation?

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.

What is threadLocal variable?

ThreadLocal is a class. If a variable is declared as threadLocal then each thread will have its own copy of variable and would not interfere with the other's thread copy. Typical scenario to use this would be giving JDBC connection to each thread so that there is no conflict.

ThreadLocal class by JAVA API

```
public class ThreadLocal {
  public Object get();
  public void set(Object newValue);
  public Object initialValue();
}
```

Implementation of ThreadLocal

```
public class ConnectionDispenser {
  private static class ThreadLocalConnection extends ThreadLocal {
    public Object initialValue() {
      return      DriverManager.getConnection(ConfigurationSingleton.getDbUrl());
    }
  }

  private static ThreadLocalConnection conn = new ThreadLocalConnection();

    public static Connection getConnection() {
      return (Connection) conn.get();
    }
  }
```

What is a task's priority and how is it used in scheduling?

A task's priority is an integer value that identifies the relative order in which it should be executed with respect to other tasks. The scheduler attempts to schedule higher priority tasks before lower priority tasks.

Collections

What is an Iterator?

- The Iterator interface is used to step through the elements of a Collection.
- Iterators let you process each element of a Collection.
- Iterators are a generic way to go through all the elements of a Collection no matter how it is organized.
- Iterator is an Interface implemented a different way for every Collection.

How do you traverse through a collection using its Iterator?

To use an iterator to traverse through the contents of a collection, follow these steps:

- Obtain an iterator to the start of the collection by calling the collectionâ€™s ***iterator()*** method.
- Set up a loop that makes a call to ***hasNext()***. Have the loop iterate as long as ***hasNext()*** returns **true**.
- Within the loop, obtain each element by calling **next()**.

How do you remove elements during Iteration?

Iterator also has a method ***remove()*** when remove is called, the current element in the iteration is deleted.

What is the difference between Enumeration and Iterator?

Enumeration	Iterator
Enumeration doesn't have a remove() method	Iterator has a remove() method
Enumeration acts as Read-only interface, because it has the methods only to traverse and fetch the objects	Can be *abstract, final, native, static*, or *synchronized*

Note: So Enumeration is used whenever we want to make Collection objects as Read-only.

How is ListIterator?

ListIterator is just like Iterator, except it allows us to access the collection in either the forward or backward direction and lets us modify an element

What is the List interface?

- The List interface provides support for ordered collections of objects.
- Lists may contain duplicate elements.

What are the main implementations of the List interface ?

The main implementations of the List interface are as follows :

❖**ArrayList** : Resizable-array implementation of the List interface. The best all-around implementation of the List interface.
❖**Vector** : Synchronized resizable-array implementation of the List interface with additional "legacy methods."
❖**LinkedList** : Doubly-linked list implementation of the List interface. May provide better performance than the ArrayList implementation if elements are frequently inserted or deleted within the list. Useful for queues and double-ended queues (deques).

What are the advantages of ArrayList over arrays?
Some of the advantages ArrayList has over arrays are:
❖It can grow dynamically
❖It provides more powerful insertion and search mechanisms than arrays.

Difference between ArrayList and Vector?

ArrayList	Vector
ArrayList is **NOT** synchronized by default.	Vector List is synchronized by default.
ArrayList can use only Iterator to access the elements.	Vector list can use Iterator and Enumeration Interface to access the elements.
The ArrayList increases its array size by 50 percent if it runs out of room.	A Vector defaults to doubling the size of its array if it runs out of room
ArrayList has no default size.	While vector has a default size of 10.

How to obtain Array from an ArrayList ?
Array can be obtained from an ArrayList using ***toArray()*** method on ArrayList.

```
List arrayList = new ArrayList();

ObjectA  a[] = arrayList.toArray();
```

Why insertion and deletion in ArrayList is slow compared to LinkedList?
ArrayList internally uses and array to store the elements, when that array gets filled by inserting elements a new array of roughly 1.5 times the size of the original array is created and all the data of old array is copied to new array.
During deletion, all elements present in the array after the deleted elements have to be moved one step back to fill the space created by deletion. In linked list data is stored in nodes that have reference to the previous node and the next node so adding element is simple as creating the node an updating the next pointer on the last node and the previous pointer on the new node. Deletion in linked list is fast because it involves only updating the next pointer in the node before the deleted node and updating the previous pointer in the node after the deleted node.

Why are Iterators returned by ArrayList called Fail Fast ?

Because, if list is structurally modified at any time after the iterator is created, in any way except through the iterator's own remove or add methods, the iterator will throw a ConcurrentModificationException. Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.

How do you decide when to use ArrayList and When to use LinkedList?

If you need to support random access, without inserting or removing elements from any place other than the end, then ArrayList offers the optimal collection. If, however, you need to frequently add and remove elements from the middle of the list and only access the list elements sequentially, then LinkedList offers the better implementation.

What is the Set interface?

- The Set interface provides methods for accessing the elements of a finite mathematical set
- Sets do not allow duplicate elements
- Contains no methods other than those inherited from Collection
- It adds the restriction that duplicate elements are prohibited
- Two Set objects are equal if they contain the same elements

What are the main Implementations of the Set interface?

The main implementations of the List interface are as follows:

- HashSet
- TreeSet
- LinkedHashSet
- EnumSet

What is a HashSet?

- A HashSet is an unsorted, unordered Set.
- It uses the hashcode of the object being inserted (so the more efficient your hashcode() implementation the better access performance you'll get).
- Use this class when you want a collection with no duplicates and you don't care about order when you iterate through it.

What is a TreeSet?

TreeSet is a Set implementation that keeps the elements in sorted order. The elements are sorted according to the natural order of elements or by the comparator provided at creation time.

What is an EnumSet?

An EnumSet is a specialized set for use with enum types, all of the elements in the EnumSet type that is specified, explicitly or implicitly, when the set is created.

Difference between HashSet and TreeSet?

HashSet	TreeSet
HashSet is under set interface i.e. it does not guarantee for either sorted order or sequence order.	TreeSet is under set i.e. it provides elements in a sorted order (acceding order).
We can add any type of elements to hash set.	We can add only similar types of elements to tree set.

What is a Map?

- A map is an object that stores associations between keys and values (key/value pairs).
- Given a key, you can find its value. Both keys and values are objects.
- The keys must be unique, but the values may be duplicated.
- Some maps can accept a null key and null values, others cannot.

What are the main Implementations of the Map interface?

The main implementations of the List interface are as follows:

- HashMap
- HashTable
- TreeMap
- EnumMap

What is a TreeMap?

TreeMap actually implements the SortedMap interface which extends the Map interface. In a TreeMap the data will be sorted in ascending order of keys according to the natural order for the key's class, or by the comparator provided at creation time. TreeMap is based on the Red-Black tree data structure.

How do you decide when to use HashMap and when to use TreeMap?

For inserting, deleting, and locating elements in a Map, the HashMap offers the best alternative. If, however, you need to traverse the keys in a sorted order, then TreeMap is your better alternative. Depending upon the size of your collection, it may be faster to add elements to a HashMap, and then convert the map to a TreeMap for sorted key traversal.

Difference between HashMap and Hashtable?

HashMap	Hashtable
HashMap lets you have null values as well as one null key.	HashTable does not allows null values as key and value.
The iterator in the HashMap is fail-safe (If you change the map while iterating, you'll know).	The enumerator for the Hashtable is not fail-safe.
HashMap is unsynchronized.	Hashtable is synchronized.

Note: Only one NULL is allowed as a key in HashMap. HashMap does not allow multiple keys to be NULL. Nevertheless, it can have multiple NULL values.

How does a Hashtable internally maintain the key-value pairs?
TreeMap actually implements the SortedMap interface which extends the Map interface. In a TreeMap the data will be sorted in ascending order of keys according to the natural order for the key's class, or by the comparator provided at creation time. TreeMap is based on the Red-Black tree data structure.

What are the different Collection Views That Maps Provide?
Maps Provide Three Collection Views.

- **Key Set** - allow a map's contents to be viewed as a set of keys.
- **Values Collection** - allow a map's contents to be viewed as a set of values.
- **Entry Set** - allow a map's contents to be viewed as a set of key-value mappings.

EXCEPTION HANDLING

What is an Exception?
The exception is said to be thrown whenever an exceptional event occurs in java which signals that something is not correct with the code written and may give unexpected result. An exceptional event is an occurrence of condition which alters the normal program flow. Exceptional handler is the code that does something about the exception.

What is error?
An Error indicates that a **non-recoverable condition** has occurred that should not be caught. **Error, a subclass of Throwable**, is intended for drastic problems, such as OutOfMemoryError, which would be reported by the JVM itself.

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

What is difference between Error and Exception?
An *error is an irrecoverable condition* occurring at runtime, such as "OutOfMemory" error. These are JVM errors and we can't repair them at runtime. Though error can be caught in catch block but the execution of application will come to a halt and is not recoverable.
While exceptions are conditions that occur because of bad input etc. e.g. "FileNotFoundException" will be thrown if the specified file does not exist. Or a "NullPointerException" will take place if we try to use 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.)

Which is superclass of Exception?
"Throwable", the parent class of all exception related classes.

What are the advantages of using exception handling?
Exception handling provides the following advantages over "traditional" error management techniques:

- ❖Separating Error Handling Code from "Regular" Code.
- ❖Propagating Errors up the Call Stack.
- ❖Grouping Error Types and Error Differentiation.

Exceptions are defined in which java package?
All the exceptions are subclasses of java.lang.Exception

Explain the exception hierarchy in java.

Throwable is a parent class off all Exception classes. They are two types of Exceptions: Checked exceptions and Unchecked Exceptions. Both types of exceptions extends Exception class.

How are the exceptions handled in java?

When an exception occur the execution of the program is transferred to an appropriate exception handler. The try-catch-finally block is used to handle the exception. 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

What are the types of Exceptions in Java

There are two types of exceptions in Java, unchecked exceptions and checked exceptions.

- **Checked exceptions**: A checked exception is some **subclass of Exception** (or Exception itself), excluding class RuntimeException and its subclasses. **Each method must either handle all checked exceptions by supplying a catch clause or list each unhandled checked exception as a thrown exception**.
- **Unchecked exceptions:** All Exceptions that **extend the RuntimeException** class are unchecked exceptions. **Class Error and its subclasses also are unchecked.**

What is Runtime Exception or unchecked exception?

Runtime exceptions represent problems that are the result of a programming problem. Such problems include arithmetic exceptions, such as dividing by zero; pointer exceptions, such as trying to access an object through a null reference; and indexing exceptions, such as 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 they can be very numerous. Having to add runtime exceptions in every method declaration would reduce a program's clarity. Thus, the compiler does not require that you catch or specify runtime exceptions (although you can). *The solution to rectify is to correct the programming logic where the exception has occurred or provide a check.*

What is checked exception?

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

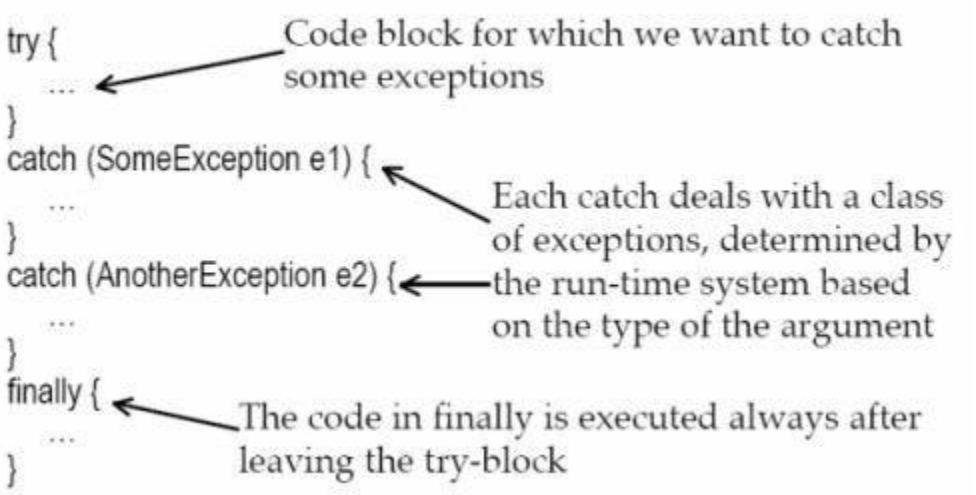
Why did the designers decide to force a method to specify all uncaught checked exceptions that can be thrown within its scope?

Any Exception that can be thrown by a method is part of the method's public programming interface. Those who call a method must know about the exceptions that a method can throw so that they can decide what to do about them. These exceptions are as much a part of that method's programming interface as its parameters and return value.

Why Errors are not checked?
An unchecked exception classes which are the *error* classes (Error and its subclasses) are exempted from compile-time checking because they can occur at many points in the program and recovery from them is difficult or impossible. A program declaring such exceptions would be pointlessly.

Why Runtime Exceptions are not checked?
The *runtime exception* classes (RuntimeException and its subclasses) are exempted from compile-time checking because, in the judgment of the designers of the Java programming language, having to declare such exceptions would not aid significantly in establishing the correctness of programs. Many of the operations and constructs of the Java programming language can result in runtime exceptions. The information available to a compiler, and the level of analysis the compiler performs, are usually not sufficient to establish that such run-time exceptions cannot occur, even though this may be obvious to the programmer. Requiring such exception classes to be declared would simply be an irritation to programmers.

Explain the significance of try-catch blocks?
Whenever the exception occurs in Java, we need a way to tell the JVM what code to execute. To do this, we use the try and catch keywords. The try is used to define a block of code in which exceptions may occur. One or more catch clauses match a specific exception to a block of code that handles it.



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

```
OPTION1
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.
}
```

```
OPTION2
try
{
        // Any code
}finally{}


OPTION3
try
{
}catch(Exception e)
{
        //lines of code to handle the exception thrown in try block
}
```

The catch blocks must always follow the try block. If there are more than one catch blocks they all must follow each other without any block in between. The finally block must follow the catch block if one is present or if the catch block is absent the finally block must follow the try block.

What is the use of finally block?
The finally block encloses code that is always executed at some point after the try block, whether an exception was thrown or not. This is right place to close files, release your network sockets, connections, and perform any other cleanup your code requires.
Note: If the try block executes with no exceptions, the finally block is executed immediately after the try block completes. It there was an exception thrown, the finally block executes immediately after the proper catch block completes

What if there is a break or return statement in try block followed by finally block?
If there is a return statement in the try block, the *finally block executes right after the return statement encountered, and before the return executes.*

Can we have the try block without catch block?
Yes, we can have the try block without catch block, *but finally block should follow the try block.*
Note: It is not valid to use a try clause without either a catch clause or a finally clause.

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

Where is the clean up code like release of resources is put in try-catch-finally block and why?
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.

Is it valid to have a try block without catch or finally?
NO. This will result in a compilation error. The try block must be followed by a catch or a finally block. It is legal to omit the either catch or the finally block but not both.
e.g. The following code is illegal.

```
try{
int i =0;
}
int a = 2;
System.out.println("a = "+a);
```

Is it valid to place some code in between try the catch/finally block that follows it?
No. There should not be any line of code present between the try and the catch/finally block. e.g. The following code is wrong.

```
try{}
String str = "ABC";
System.out.println("str = "+str);
catch(Exception e){}
```

What happens if the exception is never caught and throws down the method stack?
If the exception is not caught by any of the method in the method's stack till you get to the main() method, the main method throws that exception and the JVM halts its execution.

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

Can you catch more than one exception in a single catch block?
Yes. If the exception class specified in the catch clause has subclasses, any exception object that is a subclass of the specified Exception class will be caught by that single catch block.
E.g..

```
try {
// Some code here that can throw an IOException
}
catch (IOException e) {
e.printStackTrace();
}
```

The catch block above will catch IOException and all its subclasses e.g. FileNotFoundException etc.

Why is not considered as a good practice to write a single catchall handler to catch all the exceptions?
We can write a single catch block to handle all the exceptions thrown during the program execution as follows :

```
try {
// code that can throw exception of any possible type
}catch (Exception e) {
e.printStackTrace();
}
```

If we use the Superclass Exception in the catch block then we will not get the valuable information about each of the exception thrown during the execution, though we can find out the class of the exception occurred. Also it will reduce the readability of the code as the programmer will not understand the exact reason for putting the try-catch block.

What is exception matching?
Exception matching is the process by which 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 exception matching.

What happens if the handlers for the most specific exceptions are placed below the more general exceptions handler?
Compilation fails. The catch block for handling the most specific exceptions must always be placed above the catch block written to handle the more general exceptions.
e.g. The code below will not compile.

```
1 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
}
```

Does the order of the catch blocks matter if the Exceptions caught by them are not subtype or supertype of each other?
No. If the exceptions are siblings in the Exception class's hierarchy i.e. if one Exception class is not a subtype or supertype of the other, then the order in which their handlers (catch clauses) are placed does not matter.

What happens if a method does not throw a checked Exception directly but calls a method that does? What does 'Ducking' the exception mean?
If a method does not throw a checked Exception directly but calls a method that throws an exception then the calling method must handle the throw exception or declare the exception in its throws clause. *If the calling method does not handle and declares the exception, the exception is passed to the next method in the method stack. This is called as ducking the exception down the method stack.*

e.g. The code below will not compile as the getCar() method has not declared the CarNotFoundException which is thrown by the getColor () method.

```
void getCar() {
getColor();
}
void getColor () {
throw new CarNotFoundException();
}
```

Fix for the above code is

```
void getCar() throws CarNotFoundException {
getColor();
}
void getColor () {
throw new CarNotFoundException();
}
```

Is an empty catch block legal?
Yes you can leave the catch block without writing any actual code to handle the exception caught.

Can a catch block throw the exception caught by itself?
Yes. This is called 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();
```

```
}
}
```

What is throw keyword?
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");
 }
}
```

What is use of throws keyword?
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.
}
```

What is the difference throw and throws?
throws: Used in a method's signature if a method is capable of causing an exception that it does not handle, so that callers of the method can guard themselves against that exception. If a method is declared as throwing a particular class of exceptions, then any other method that calls it must either have a try-catch clause to handle that exception or must be declared to throw that exception (or its superclass) itself.

A method that does not handle an exception it throws has to announce this:

```
public void myfunc(int arg) throws MyException {
    …
}
```

throw: Used to trigger an exception. The exception will be caught by the nearest try-catch clause that can catch that type of exception. The flow of execution stops immediately after the throw statement; any subsequent statements are not executed.

To throw an user-defined exception within a block, we use the throw command:

```
throw new MyException("I always wanted to throw an exception!");
```

How to create custom exceptions?

A. By extending the Exception class or one of its subclasses.

Example:

```
class MyException extends Exception {
public MyException() { super(); }
public MyException(String s) { super(s); }
}
```

As shown above in example we can easily create our own exception by extending the Exception class or any of its subclasses. Some other examples are:-

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

What are the different ways to handle exceptions?

There are two ways to handle exceptions:

- ❖Wrapping the desired code in a try block followed by a catch block to catch the exceptions.
- ❖List the desired exceptions in the throws clause of the method and let the caller of the method handle those exceptions.

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

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 and hence the class from the child classloader is not visible.

Consider if NoClassDefFoundError occurs for "src/com/TestClass" it does not mean that TestClass class is not in the CLASSPATH. It means that the class TestClass was found by the ClassLoader however when trying to load the class, it ran into an error reading the class definition. This typically

happens when the class in question has static blocks or members which use a Class that's not found by the ClassLoader. So 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.

Can static block throw exception?

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.

GARBAGE COLLECTOR

Explain garbage collection?

Garbage collection is one of the most important feature of Java. Garbage collection is also called automatic memory management as JVM automatically removes the unused variables/objects (value is null) from the memory. User program can't directly free the object from memory; instead it is the job of the garbage collector to automatically free the objects that are no longer referenced by a program. Every class inherits **finalize()** method from **java.lang.Object**, the finalize() method is called by garbage collector when it determines no more references to the object exists. In Java, it is good idea to explicitly assign **null** into a variable when no more in use.

What is the responsibility of Garbage Collector?

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.

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

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.

Does garbage collection guarantee that a program will not run out of memory?

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

Is garbage collector a dameon thread?

Yes garbage collector is a dameon thread. A dameon thread runs behind the application. It is started by JVM. The thread stops when all non-dameon threads stop.

Garbage Collector is controlled by whom?

The JVM controls the Garbage Collector; it decides when to run the Garbage Collector. JVM runs the Garbage Collector when it realizes that the memory is running low, but this behavior of JVM can't be guaranteed.
One can request the Garbage Collection to happen from within the java program but there is no guarantee that this request will be taken care of by JVM.

Which part of the memory is involved in Garbage Collection? Stack or Heap?

Heap

When does an object become eligible for garbage collection?

An object becomes eligible for Garbage Collection when no live thread can access it. Or an object is subject to garbage collection when it becomes **unreachable** to the program in which it is used.

Can an object be garbage collected while it is still reachable?

No. A reachable object cannot be garbage collected. Only unreachable objects may be garbage collected.

If an object is garbage collected, can it become reachable again?

No. Once an object is garbage collected, it ceases to exist. It can no longer become reachable again.

Can an unreachable object become reachable again?

An unreachable object may become reachable again. This can happen when the object's finalize() method is invoked and the object performs an operation which causes it to become accessible to reachable objects.

What are the different ways to make an object eligible for Garbage Collection when it is no longer needed?

1. **Set all available object references to null** once the purpose of creating the object has been served:

```
public class GarbageCollnTest1 {

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

```
String str = "Set the object ref to null";
//String object referenced by variable str is not eligible for GC yet

str = null;
/*String object referenced by variable str becomes eligible for GC */
   }

}
```

2. **Make the reference variable to refer to another object** : Decouple the reference variable from the object and set it refer to another object, so the object which it was referring to before reassigning is eligible for Garbage Collection.

```
publc class GarbageCollnTest2 {

   public static void main(String [] args){
String str1 = "Garbage collected after use";
String str2 = "Another String";
System.out.println(str1);
//String object referred by str1 is not eligible for GC yet

str1 = str2;
/* Now the str1 variable referes to the String object "Another String" and the object "Garbage
collected after use" is not referred by any variable and hence is eligible for GC */

   }

}
```

3) **Creating Islands of Isolation:** If you have two instance reference variables which are referring to the instances of the same class, and these two reference variables refer to each other and the objects referred by these reference variables do not have any other valid reference then these two objects are said to form an Island of Isolation and are eligible for Garbage Collection.

```
public class GCTest3 {
GCTest3 g;

   public static void main(String [] str){
GCTest3 gc1 = new GCTest3();
GCTest3 gc2 = new GCTest3();
gc1.g = gc2; //gc1 refers to gc2
gc2.g = gc1; //gc2 refers to gc1
gc1 = null;
gc2 = null;
//gc1 and gc2 refer to each other and have no other valid //references
```

```
//gc1 and gc2 form Island of Isolation
//gc1 and gc2 are eligible for Garbage collection here
  }

}
```

Can the Garbage Collection be forced by any means?

No. The Garbage Collection can not be forced, though there are few ways by which it can be requested there is no guarantee that these requests will be taken care of by JVM.

How can the Garbage Collection be requested?

There are two ways in which we can request the JVM to execute the Garbage Collection.

- The methods to perform the garbage collections are present in the Runtime class provided by java. The Runtime class is a Singleton for each java main program. The method **getRuntime()** returns a singleton instance of the Runtime class. The method **gc()** can be invoked using this instance of Runtime to request the garbage collection.
- Call the System class **System.gc()** method which will request the JVM to perform garbage collection.

What is the purpose of finalization?

The purpose of finalization is to give an unreachable object the opportunity to perform any cleanup processing before the object is garbage collected.

Can an object's finalize() method be invoked while it is reachable?

An object's finalize() method cannot be invoked by the garbage collector while the object is still reachable. However, an object's finalize() method may be invoked by other objects.

How many times may an object's finalize() method be invoked by the garbage collector?

An object's finalize() method may only be invoked once by the garbage collector.

Under what conditions is an object's finalize() method invoked by the garbage collector?

The garbage collector invokes an object's finalize() method when it detects that the object has become unreachable.

Does Java have destructors?
No garbage collector does this job working in the background.

What is the difference between final, finally and finalize? What do you understand by the java final keyword?

final - declare constant
finally - handles exception
finalize - helps in garbage collection

Variables defined in an interface are implicitly final. A final class can't be extended i.e., final class may not be sub-classed. This is done for security reasons with basic classes like String and Integer. It also allows the compiler to make some optimizations, and makes thread safety a little easier to achieve. A final method can't be overridden when its class is inherited. We can't change value of a final variable (is a constant).

finalize() method is used just before an object is destroyed and garbage collected.

finally, a key word used in exception handling and will be executed whether or not an exception is thrown. For example, closing of open connections is done in the finally method.

What is the purpose of overriding finalize() method?

The finalize() method should be overridden for an object to include the clean up code or to dispose of the system resources that should to be done before the object is garbage collected.

Can we call finalize() method

Yes. Nobody will stop us to call any method , if it is accessible in our class. But a garbage collector cannot call an object's finalize method if that object is reachable.

If an object becomes eligible for Garbage Collection and its finalize() method has been called and inside this method the object becomes accessible by a live thread of execution and is not garbage collected. Later at some point the same object becomes eligible for Garbage collection, will the finalize() method be called again?

No

How many times does the garbage collector calls the finalize() method for an object?

Only once.

What happens if an uncaught exception is thrown from during the execution of the finalize() method of an object?
The exception will be ignored and the garbage collection (finalization) of that object terminates.

How to enable/disable call of finalize() method of exit of the application

Runtime.getRuntime().runFinalizersOnExit(boolean value) . Passing the boolean value will either disable or enable the finalize() call.

What are the different types of *references* in java?

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 anywhere else. Another possible application of weak references is the string intern pool. Semantically, a weak reference means "**get rid of this object when nothing else references it**."

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**."

Does JVM maintain a cache by itself? Does the JVM allocate objects in heap? Is this the OS heap or the heap maintained by the JVM? Why

Yes, the JVM maintains a cache by itself. It creates the Objects on the HEAP, but references to those objects are on the STACK.

What is phantom memory?

Phantom memory is false memory. Memory, that does not exist in reality.

How to change the heap size of a JVM?

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 <initial size> -Xmx <maximum size> program
For example:
java -Xms64m -Xmx128m program

What is *memory leak*?

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.

How can you minimize the need of garbage collection and make the memory use more effective?

Use object pooling and weak object references.

OOPS Concepts

What are the principle concepts of OOPS?

There are four principle concepts upon which object oriented design and programming rest. They are:

- Abstraction
- Polymorphism
- Inheritance
- Encapsulation (i.e. easily remembered as A-PIE).

What is Abstraction?

Abstraction refers to the act of **representing essential features without including the background details** or explanations.

What is Encapsulation?

Encapsulation is a technique used for **hiding the properties and behaviors of an object** and allowing outside access only as appropriate. It prevents other objects from directly altering or accessing the properties or methods of the encapsulated object.

In other words we can, Encapsulation is a process of binding or wrapping the data and the codes that operates on the data into a single entity. This keeps the data safe from outside interface and misuse. One way to think about encapsulation is as a protective wrapper that prevents code and data from being arbitrarily accessed by other code defined outside the wrapper.

What is data encapsulation?

Encapsulation may be used by creating 'get' and 'set' methods in a class (JAVABEAN) which are used to access the fields of the object. Typically the fields are made private while the get and set methods are public. Encapsulation can be used to validate the data that is to be stored, to do calculations on data that is stored in a field or fields, or for use in introspection (often the case when using javabeans in Struts, for instance). **Wrapping of data and function into a single unit is called as data encapsulation.** Encapsulation is nothing but wrapping up the data and associated methods into a single unit in such a way that data can be accessed with the help of associated methods. Encapsulation provides data security. It is nothing but data hiding.

What is the difference between abstraction and encapsulation?

Abstraction focuses on the outside view of an object (i.e. the interface) **Encapsulation** (information hiding) prevents clients from seeing it's inside view, where the behavior of the abstraction is implemented.

Abstraction solves the problem in the design side, while **Encapsulation** is the Implementation. **Encapsulation** is the deliverables of Abstraction. Encapsulation barely talks about grouping up your abstraction to suit the developer needs.

What is Inheritance?

Inheritance is the process by which objects of one class acquire the properties of objects of another class. A class that is inherited is called a superclass. The class that does the inheriting is called a subclass. Inheritance is done by using the keyword extends.

The two most common reasons to use inheritance are:

- To promote code reuse
- To use polymorphism

What are some alternatives to inheritance?

Delegation is an alternative to inheritance. Delegation means that you include an instance of another class as an instance variable, and forward messages to the instance. It is often safer than inheritance because it forces you to think about each message you forward, because the instance is of a known class, rather than a new class, and because it doesn't force you to accept all the methods of the super class: you can provide only the methods that really make sense. On the other hand, it makes you write more code, and it is harder to re-use (because it is not a subclass).

What is Polymorphism?

Polymorphism is briefly described as "one interface, many implementations." The meaning of Polymorphism is something like one name many forms. Polymorphism is a characteristic of being able to assign a different meaning or usage to something in different contexts - specifically, to allow an entity such as a variable, a function, or an object to have more than one form.

How does Java implement polymorphism?

(Inheritance, Overloading and Overriding are used to achieve Polymorphism in java).
Polymorphism manifests itself in Java in the form of multiple methods having the same name.

In some cases, multiple methods have the same name, but different formal argument lists (overloaded methods).

In other cases, multiple methods have the same name, same return type, and same formal argument list (overridden methods).

Explain the different forms of Polymorphism.

There are two types of polymorphism one is **Compile time polymorphism** and the other is run time polymorphism. Compile time polymorphism is method overloading. **Runtime time polymorphism** is

done using inheritance and interface.
Note: *From a practical programming viewpoint, polymorphism manifests itself in three distinct forms in Java:*

- *Method overloading*
- *Method overriding through inheritance*
- *Method overriding through the Java interface*

What is runtime polymorphism or dynamic method dispatch?

In Java, runtime polymorphism or dynamic method dispatch is a process in which a call to an overridden method is resolved at runtime rather than at compile-time. In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the **object being referred** to by the reference

What is Dynamic Binding?

Binding refers to the linking of a procedure call to the code to be executed in response to the call. Dynamic binding (also known as late binding) means that the code associated with a given procedure call is not known until the time of the call at run-time. It is associated with polymorphism and inheritance.

What is method overloading?

Method Overloading means to have two or more methods with same name in the same class with different arguments. The benefit of method overloading is that it allows you to implement methods that support the same semantic operation but differ by argument number or type.
Note:

- *Overloaded methods MUST change the argument list*
- *Overloaded methods CAN change the return type*
- *Overloaded methods CAN change the access modifier*
- *Overloaded methods CAN declare new or broader checked exceptions*
- *A method can be overloaded in the same class or in a subclass*

What is method overriding?

Method overriding occurs when sub class declares a method that has the same type arguments as a method declared by one of its superclass. The key benefit of overriding is the ability to define behavior that's specific to a particular subclass type.
Note:

- *The overriding method cannot have a more restrictive access modifier than the method being overridden (Ex: You can't override a method marked public and make it protected).*
- *You cannot override a method marked final*
- *You cannot override a method marked static*

What are the differences between method overloading and method overriding?

	Overloaded Method	Overridden Method
Arguments	Must change	Must not change
Return type	Can change	Can't change except for covariant returns
Exceptions	Can change	Can reduce or eliminate. Must not throw new or broader checked exceptions
Access	Can change	Must not make more restrictive (can be less restrictive)
Invocation	Reference type determines which overloaded version is selected. Happens at compile time.	Object type determines which method is selected. Happens at runtime.

Can overloaded methods be override too?

Yes, derived classes still can override the overloaded methods. Polymorphism can still happen. Compiler will not binding the method calls since it is overloaded, because it might be overridden now or in the future.

Is it possible to override the main method?

NO, because main is a static method. A static method can't be overridden in Java.

How to invoke a superclass version of an Overridden method?

To invoke a superclass method that has been overridden in a subclass, you must either call the method directly through a superclass instance, or use the super prefix in the subclass itself. From the point of the view of the subclass, the super prefix provides an explicit reference to the superclass' implementation of the method.

```
// From subclass
        super.overriddenMethod();
```

What is super?

super is a keyword which is used to access the method or member variables from the superclass. If a method hides one of the member variables in its superclass, the method can refer to the hidden variable through the use of the super keyword. In the same way, if a method overrides one of the methods in its superclass, the method can invoke the overridden method through the use of the super keyword.
Note:

- *You can only go back one level.*
- *In the constructor, if you use super(), it must be the very first code, and you cannot access any this.xxx variables or methods to compute its parameters.*

How do you prevent a method from being overridden?

To prevent a specific method from being overridden in a subclass, use the **final modifier** on the method declaration, which means "this is the final implementation of this method", the end of its inheritance hierarchy.

What is an Interface?

An interface is a description of a set of methods that conforming implementing classes must have.
Note:

- *You can't mark an interface as final.*
- *Interface variables must be static.*
- *An Interface cannot extend anything but another interfaces.*

Can we instantiate an interface?

You can't instantiate an interface directly, but you can instantiate a class that implements an interface.

Can we create an object for an interface?

Yes, it is always necessary to create an object implementation for an interface. Interfaces cannot be instantiated in their own right, so you must write a class that implements the interface and fulfill all the methods defined in it.

Do interfaces have member variables?

Interfaces may have member variables, but these are **implicitly public, static, and final**- in other words, interfaces can declare only constants, not instance variables that are available to all implementations and may be used as key references for method arguments for example.

What modifiers are allowed for methods in an Interface?

Only public and abstract modifiers are allowed for methods in interfaces.

What is a marker interface?

Marker interfaces are those which do not declare any required methods, but signify their compatibility with certain operations. The java.io.Serializable interface and Cloneable are typical marker interfaces. These do not contain any methods, but classes must implement this interface in order to be serialized and de-serialized.

What is an abstract class?

Abstract classes are classes that contain one or more abstract methods. An abstract method is a method that is declared, but contains no implementation.

Note:

- *If even a single method is abstract, the whole class must be declared abstract.*
- *Abstract classes may not be instantiated, and require subclasses to provide implementations for the abstract methods.*
- *You can't mark a class as both abstract and final.*

Can we instantiate an abstract class?

An abstract class can never be instantiated. Its sole purpose is to be extended (subclassed).

What are the differences between Interface and Abstract class?

Abstract Class	Interfaces
An abstract class can provide complete, default code and/or just the details that have to be overridden.	An interface cannot provide any code at all,just the signature.
In case of abstract class, a class may extend only one abstract class.	A Class may implement several interfaces.
An abstract class can have non-abstract methods.	All methods of an Interface are abstract.
An abstract class can have instance variables.	An Interface cannot have instance variables.
An abstract class can have any visibility: public, private, protected.	An Interface visibility must be public (or) none.
If we add a new method to an abstract class then we have the option of providing default implementation and therefore all the existing code might work properly.	If we add a new method to an Interface then we have to track down all the implementations of the interface and define implementation for the new method.
An abstract class can contain constructors .	An Interface cannot contain constructors .
Abstract classes are fast.	Interfaces are slow as it requires extra indirection to find corresponding method in the actual class.

When should I use abstract classes and when should I use interfaces?

Use Interfaces when…

- You see that something in your design will change frequently.
- If various implementations only share method signatures then it is better to use Interfaces.
- you need some classes to use some methods which you don't want to be included in the class, then you go for the interface, which makes it easy to just implement and make use of the methods defined in the interface.

Use Abstract Class when…

- If various implementations are of the same kind and use common behavior or status then abstract class is better to use.
- When you want to provide a generalized form of abstraction and leave the implementation task with the inheriting subclass.
- Abstract classes are an excellent way to create planned inheritance hierarchies. They're also a good choice for nonleaf classes in class hierarchies.

When you declare a method as abstract, can other nonabstract methods access it?

Yes, other nonabstract methods can access a method that you declare as abstract.

Can there be an abstract class with no abstract methods in it?

Yes, there can be an abstract class without abstract methods.

SERIALIZATION

What is Serialization?
Serialization is a mechanism by which we can save the state of an object by converting it to a byte stream. When an object has to be transferred over a network (typically through rmi or EJB) or persist the state of an object to a file, the object Class needs to implement Serializable interface. Implementing this interface will allow the object converted into bytestream and transfer over a network.

What is the need of Serialization?
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

How do I serialize an object to a file?
The class whose instances are to be serialized should implement an interface Serializable. Then we pass the instance to the ObjectOutputStream which is connected to a FileOutputStream. This will save the object to a file.

Do we need to implement any method of Serializable interface to make an object Serializable?
No. Serializable is a Marker Interface. It does not have any methods.

Other than Serialization what are the different approach to make object Serializable?

Besides the Serializable interface, at least three 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. We 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.

When we serialize an object, what happens to the object references included in the object?

The serialization mechanism generates an **object graph for serialization**. Thus it determines whether the included object references are Serializable or not. This is a recursive process. Thus when an object is serialized, all the included objects are also serialized along with the original object.

What happens if the object to be serialized includes the references to other Serializable objects?
If the object to be serialized includes the references to other objects whose class implements Serializable then all those object's state also will 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.

What happens if an object is Serializable but it includes a reference to a non-Serializable object?
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.

e.g.

```
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: NonSerial
at java.io.ObjectOutputStream.writeObject0(ObjectOutputStream.java)

What one should take care of while serializing the object?
One should make sure that all the included objects are also serializable. If any of the objects is not serializable then it throws a NotSerializableException.

Are the static variables saved as the part of serialization?
No. The static variables belong to the class and not to an object they are not the part of the state of the object so they are not saved as the part of serialized object.

What is a transient variable?
These variables are not included in the process of serialization and are not the part of the object's serialized state. Transient variable can't be serialize. For example if a variable is declared as transient in a Serializable class and the class is written to an ObjectStream, the value of the variable can't be written to the stream instead when the class is retrieved from the ObjectStream the value of the variable becomes **null**

What will be the value of transient variable after de-serialization?
It's default value.
e.g. if the transient variable in question is an int, it's 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("testTransients.ser");
     ObjectOutputStream os = new ObjectOutputStream(fs);
     os.writeObject(c);
     os.close();
     } catch (Exception e) { e.printStackTrace(); }
```

```
        try {
        FileInputStream fis = new FileInputStream("testTransients.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 executing above piece of code –
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 –
The transient variable is not saved as the part of the state of the serailized variable, it's value after de-serialization is it's default value.
```

Does the order in which the value of the transient variables and the state of the object using the defaultWriteObject() method are saved during serialization matter?
Yes. As 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.

How can one customize the Serialization process? Or What is the purpose of implementing the writeObject() and readObject() method?
When we want to store the transient variables state as a part of the serialized object at the time of serialization the class must implement the following methods –

```
private void wrtiteObject(ObjectOutputStream outStream)
{
//code to save the transient variables state as a part of serialized object
}

private void readObject(ObjectInputStream inStream)
{
//code to read the transient variables state and assign it to the de-serialized object
}
```

e.g.

```
public class TestCustomizedSerialization implements Serializable{

   private static final long serialVersionUID =-22L;
```

```
   private String noOfSerVar;
   transient private int noOfTranVar;

   TestCustomizedSerialization(int noOfTranVar, String noOfSerVar) {
      this.noOfTranVar = noOfTranVar;
      this.noOfSerVar = noOfSerVar;
   }

   private void writeObject(ObjectOutputStream os) {

    try {
    os.defaultWriteObject();
    os.writeInt(noOfTranVar);
    } catch (Exception e) { e.printStackTrace(); }
    }

    private void readObject(ObjectInputStream is) {
    try {
    is.defaultReadObject();
    int noOfTransients = (is.readInt());
    } catch (Exception e) {
        e.printStackTrace(); }
    }

    public int getNoOfTranVar() {
      return noOfTranVar;
   }

}
```

The value of transient variable 'noOfTranVar' is saved as part of the serialized object manually by implementing writeObject() and restored by implementing readObject().
The normal serializable variables are saved and restored by calling defaultWriteObject() and defaultReadObject()respectively. These methods perform the normal serialization and de-sirialization process for the object to be saved or restored respectively.

If a class is Serializable but its superclass in not , what will be the state of the instance variables inherited from super class after deserialization?

The values of the instance variables inherited from superclass will be reset to the values they were given during the original construction of the object as the non-serializable super-class constructor will run.
E.g.

```
public class ParentNonSerializable {
   int noOfWheels;

   ParentNonSerializable(){
```

```
        this.noOfWheels = 4;
    }

}
```

```
public class ChildSerializable extends ParentNonSerializable implements Serializable {

    private static final long serialVersionUID = 1L;
    String color;



    ChildSerializable() {
        this.noOfWheels = 8;
        this.color = "blue";
    }
}
```

```
public class SubSerialSuperNotSerial {

    public static void main(String [] args) {

        ChildSerializable c = new ChildSerializable();
        System.out.println("Before : - " + c.noOfWheels + " "+ c.color);
        try {
        FileOutputStream fs = new FileOutputStream("superNotSerail.ser");
        ObjectOutputStream os = new ObjectOutputStream(fs);
        os.writeObject(c);
        os.close();
        } catch (Exception e) { e.printStackTrace(); }

        try {
        FileInputStream fis = new FileInputStream("superNotSerail.ser");
        ObjectInputStream ois = new ObjectInputStream(fis);
        c = (ChildSerializable) ois.readObject();
        ois.close();
        } catch (Exception e) { e.printStackTrace(); }
        System.out.println("After :- " + c.noOfWheels + " "+ c.color);
        }

}
```

Result on executing above code –
Before : - 8 blue
After :- 4 blue

The instance variable 'noOfWheels' is inherited from superclass which is not serializable. Therefore while restoring it the non-serializable superclass constructor runs and its value is set to 8 and is not same as the value saved during serialization which is 4

To serialize an array or a collection all the members of it must be Serializable. True /False?
True

What is Externalizable?
Externalizable is an Interface that extends Serializable Interface. And sends data into Streams in Compressed Format. It has two methods, writeExternal(ObjectOuput out) and readExternal(ObjectInput in)

What is use of serialVersionUID?
During object serialization, the default Java serialization mechanism writes the metadata about the object, which includes the class name, field names and types, and superclass. This class definition is stored as a part of the serialized object. This stored metadata enables the deserialization process to reconstitute the objects and map the stream data into the class attributes with the appropriate type Everytime an object is serialized the java serialization mechanism automatically computes a hash value. ObjectStreamClass's *computeSerialVersionUID*() method passes the class name, sorted member names, modifiers, and interfaces to the secure hash algorithm (SHA), which returns a hash value.The serialVersionUID is also called *suid*.
So when the serilaize object is retrieved , the JVM first evaluates the *suid* of the serialized class and compares the *suid* value with the one of the object. If the suid values match then the object is said to be compatible with the class and hence it is de-serialized. If not *InvalidClassException* exception is thrown.

Changes to a serializable class can be compatible or incompatible. Following is the list of changes which are compatible:

- Add fields
- Change a field from static to non-static
- Change a field from transient to non-transient
- Add classes to the object tree

List of incompatible changes:

- Delete fields
- Change class hierarchy
- Change non-static to static
- Change non-transient to transient
- Change type of a primitive field

So, if no suid is present , inspite of making compatible changes, jvm generates new *suid* thus resulting in an exception if prior release version object is used .
The only way to get rid of the exception is to recompile and deploy the application again.

If we explicitly metion the suid using the statement:

private final static long serialVersionUID = <integer value>

then if any of the metioned compatible changes are made the class need not to be recompiled. But for incompatible changes there is no other way than to compile again.

Immutable Class and String

What is an immutable class?
Immutable class is a class which once created; its contents can not be changed. Immutable objects are the objects whose state can not be changed once constructed. **e.g. String class**

How to create an immutable class?
To create an immutable class following steps should be followed:

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:
    1. Don't provide methods that modify the mutable objects.
    2. Don't share references to the mutable objects. Never store references to external, mutable objects passed to the constructor; if necessary, create copies, and store references to the copies. Similarly, create copies of your internal mutable objects when necessary to avoid returning the originals in your methods.

Example:-

```
public final class FinalEmployee
{
    private final String name;
    private final int    age;

    public FinalEmployee(final String name,
                         final int    age)
    {
        super();
        this.name = name;
        this.age = age;
    }

    public int getAge()
    {
        return age;
    }

    public String getName()
    {
        return name;
    }
}
```

Immutable objects are automatically thread-safe –true/false?

True. Since the state of the immutable objects can not be changed once they are created they are automatically synchronized/thread-safe.

Which classes in java are immutable?
All wrapper classes in java.lang are immutable –
String, Integer, Boolean, Character, Byte, Short, Long, Float, Double, BigDecimal, BigInteger

What are the advantages of immutability?
The advantages are:
1) Immutable objects are automatically thread-safe, the overhead caused due to use of synchronization is avoided.
2) Once created the state of the immutable object can not be changed so there is no possibility of them getting into an inconsistent state.
3) The references to the immutable objects can be easily shared or cached without having to copy or clone them as there state can not be changed ever after construction.
4) The best use of the immutable objects is as the keys of a map.

STRING

To what value is a variable of the String type automatically initialized?
The default value of an String type is null.

How is it possible for two String objects with identical values not to be equal under the == operator?
The == operator compares two objects to determine if they are the same object in memory. It is possible for
two String objects to have the same value, but located indifferent areas of memory.

What is the difference between StringBuffer and String class?
A string buffer implements a mutable sequence of characters. A string buffer is like a String, but can be
modified. At any point in time it contains some particular sequence of characters, but the length and content
of the sequence can be changed through certain method calls.
The String class represents character strings. All string literals in Java programs, such as "abc" are constant
and implemented as instances of this class; their values cannot be changed after they are created.

General Question

```
public class EqualsTest {

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

```
                String s1 = “abc”;
                String s2 = s1;
                String s5 = “abc”;
                String s3 = new String(”abc”);
                String s4 = new String(”abc”);
                System.out.println(”== comparison : ” + (s1 == s5));
                System.out.println(”== comparison : ” + (s1 == s2));
                System.out.println(”Using equals method : ” + s1.equals(s2));
                System.out.println(”== comparison : ” + s3 == s4);
                System.out.println(”Using equals method : ” + s3.equals(s4));
        }
}
```

Output

== comparison : true
== comparison : true
Using equals method : true
false
Using equals method : true

What is difference between String and StringTokenizer?

A StringTokenizer is utility class used to break up string.

Example:

```
StringTokenizer st = new StringTokenizer(”Hello World”);
while (st.hasMoreTokens()) {
System.out.println(st.nextToken());
}
```

Output:

Hello
World

Basic Core Java

What is Byte Code? Or what do you mean by "write once and run anywhere"?
All Java programs are compiled into class files that contain bytecodes. These byte codes can be run in any platform and hence java is said to be platform independent

In System.out.println(), what is System, out and println?
System is a predefined final class, out is a PrintStream object and println is a built-in overloaded method in the out object.

What are Java Access Specifiers?
Access specifiers are keywords that determine the type of access to the member of a class. These keywords are for allowing
privileges to parts of a program such as functions and variables. These are:
• *Public* : accessible to all classes
• *Protected* : accessible to the classes within the same package and any subclasses.
• *Private* : accessible only to the class to which they belong
• *Default :* accessible to the class to which they belong and to subclasses within the same package

Name primitive Java types.
The 8 primitive types are byte, char, short, int, long, float, double, and boolean.

What is the difference between the boolean & operator and the && operator?
If an expression involving the boolean & operator is evaluated, both operands are evaluated, whereas the && operator is a short cut operator. When an expression involving the && operator is evaluated, the first operand is evaluated. If the first operand returns a value of true then the second operand is evaluated. If the first operand evaluates to false, the evaluation of the second operand is skipped.

What if I write static public void instead of public static void?
Program compiles and runs properly.

What do you understand by numeric promotion?
The Numeric promotion is the conversion of a smaller numeric type to a larger numeric type, so that integral and floating-point operations may take place. In the numerical promotion process the byte, char, and short values are converted to int values. The int values are also converted to long values, if necessary. The long and float values are converted to double values, as required.

How can I swap two variables without using a third variable?

Add two variables and assign the value into First variable. Subtract the Second value with the result Value and assign to Second variable. Subtract the Result of First Variable With Result of Second Variable and Assign to First Variable. Example:
int a=5,b=10;a=a+b; b=a-b; a=a-b;
An other approach to the same question
You use an XOR swap.

Why is *main()* method static?
To access the static method the object of the class is not needed. The method can be access 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.

What is difference between instanceof and isInstance(Object obj)?
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 method is used to check the type of an object which are known at compile time and isInstance() could only be called on class, San instance of java.lang.Class.

```
if (obj instanceof MyType) {
...
}else if (MyType.class.isInstance(obj)) {
...
}
```

3) 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.

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. This means the references passed to the method are actually copies of the original references.Java copies and passes the reference by value, not the object. Thus, method manipulation will alter the objects, since the references point to the original objects.Consider the example:

```
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
```

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.

How to make sure that Childclass method actually overrides the method of the superclass?

The @Override annotation can be added to the javadoc for the new method. If you accidently miss an argument or capitalize the method name wrong, the compiler will generate a compile-time error.

How to find the size of an object?

The heap size of an object can be found using -

Runtime.totalMemory()-Runtime.freeMemory() .

What is Constructor?

- A constructor is a special method whose task is to initialize the object of its class.
- It is special because its name is the **same as the class name**.
- They do not have return types, not even **void** and therefore they cannot return values.
- They **cannot be inherited**, though a derived class can call the base class constructor.
- Constructor is invoked whenever an object of its associated class is created.

How does the Java default constructor be provided?

If a class defined by the code does **not** have any constructor, compiler will automatically provide one no-parameter-constructor (default-constructor) for the class in the byte code. The access modifier (public/private/etc.) of the default constructor is the same as the class itself.

Can constructor be inherited?

No, constructor cannot be inherited, though a derived class can call the base class constructor.

What are the differences between Contructors and Methods?

	Constructors	Methods
Purpose	Create an instance of a class	Group Java statements
Modifiers	Cannot be *abstract, final, native, static,* or *synchronized*	Can be *abstract, final, native, static,* or *synchronized*
Return Type	No return type, not even void	void or a valid return type
Name	Same name as the class (first letter is capitalized by convention) -- usually a noun	Any name except the class. Method names begin with a lowercase letter by convention -- usually the name of an action
this	Refers to another constructor in the same class. If used, it must be the first line of the constructor	Refers to an instance of the owning class. Cannot be used by static methods.
super	Calls the constructor of the parent class. If used, must be the first line of the constructor	Calls an overridden method in the parent class
Inheritance	Constructors are not inherited	Methods are inherited

How are this() and super() used with constructors?

- Constructors use *this* to refer to another constructor in the same class with a different parameter list.
- Constructors use *super* to invoke the superclass's constructor. If a constructor uses *super*, it must use it in the first line; otherwise, the compiler will complain.

What are the differences between Class Methods and Instance Methods?

Class Methods	Instance Methods
Class methods are methods which are declared as static. The method can be called without creating an instance of the class	Instance methods on the other hand require an instance of the class to exist before they can be called, so an instance of a class needs to be created by using the new keyword. Instance methods operate on specific instances of classes.
Class methods can only operate on class members and not on instance members as class methods are unaware of instance members.	Instance methods of the class can also not be called from within a class method unless they are being called on an instance of that class.

Class methods are methods which are declared as static. The method can be called without creating an instance of the class.

Instance methods are not declared as static.

How are this() and super() used with constructors?

- Constructors use *this* to refer to another constructor in the same class with a different parameter list.
- Constructors use *super* to invoke the superclass's constructor. If a constructor uses super, it must use it in the first line; otherwise, the compiler will complain.

What are Access Specifiers?

One of the techniques in object-oriented programming is *encapsulation*. It concerns the hiding of data in a class and making this class available only through methods. Java allows you to control access to classes, methods, and fields via so-called *access specifiers*..

What are Access Specifiers available in Java?

Java offers four access specifiers, listed below in decreasing accessibility:

- **Public**- *public* classes, methods, and fields can be accessed from everywhere.
- **Protected**- *protected* methods and fields can only be accessed within the same class to which the methods and fields belong, within its subclasses, and within classes of the same package.
- **Default(no specifier)-** If you do not set access to specific level, then such a class, method, or field will be accessible from inside the same package to which the class, method, or field belongs, but not from outside this package.
- **Private**- *private* methods and fields can only be accessed within the same class to which the methods and fields belong. *private* methods and fields are not visible within subclasses and are not inherited by subclasses.

Situation	public	Protected	default	private
Accessible to class from same package?	yes	yes	yes	no
Accessible to class from different package?	yes	no, *unless it is a subclass*	no	no

What is final modifier?

The final modifier keyword makes that the programmer cannot change the value anymore. The actual meaning depends on whether it is applied to a class, a variable, or a method.

- ***final* Classes**- A final class cannot have subclasses.
- ***final* Variables**- A final variable cannot be changed once it is initialized.
- ***final* Methods**- A final method cannot be overridden by subclasses.

What are the uses of final method?

There are two reasons for marking a method as final:

- Disallowing subclasses to change the meaning of the method.
- Increasing efficiency by allowing the compiler to turn calls to the method into inline Java code.

What is static block?

Static block which exactly executed exactly once when the class is first loaded into JVM. Before going to the main method the static block will execute.

What are static variables?

Variables that have only one copy per class are known as static variables. They are not attached to a particular instance of a class but rather belong to a class as a whole. They are declared by using the static keyword as a modifier.

```
static type  varIdentifier;
```

where, the name of the variable is varIdentifier and its data type is specified by type.
Note: Static variables that are not explicitly initialized in the code are automatically initialized with a default value. The default value depends on the data type of the variables.

What is the difference between static and non-static variables?

A static variable is associated with the class as a whole rather than with specific instances of a class. Non-static variables take on unique values with each object instance.

What are static methods?

Methods declared with the keyword static as modifier are called static methods or class methods. They are so called because they affect a class as a whole, not a particular instance of the class. Static methods are always invoked without reference to a particular instance of a class.
Note:The use of a static method suffers from the following restrictions:

- *A static method can only call other static methods.*
- *A static method must only access static data.*
- *A static method **cannot** reference to the current object using keywords super or this.*

How could Java classes direct program messages to the system console, but error messages, say to a file?

The class System has a variable *out* that represents the standard output, and the variable *err* that represents the standard error device. By default, they both point at the system console. This how the standard output could be re-directed:
Stream st = new Stream(new FileOutputStream("output.txt")); System.setErr(st); System.setOut(st);

How do you know if an explicit object casting is needed?
If you assign a superclass object to a variable of a subclass's data type, you need to do explicit casting. For example:
Object a; Customer b; b = (Customer) a;

When you assign a subclass to a variable having a supeclass type, the casting is performed automatically.

What's the difference between constructors and other methods?
Constructors must have the same name as the class and can not return a value. They are only called once while regular methods could be called many times.

Can you call one constructor from another if a class has multiple constructors
Use this() syntax.

Explain the usage of Java packages.
This is a way to organize files when a project consists of multiple modules. It also helps resolve naming conflicts when different packages have classes with the same names. Packages access level also allows you to protect data from being used by the non-authorized classes.

If a class is located in a package, what do you need to change in the OS environment to be able to use it?
You need to add a directory or a jar file that contains the package directories to the CLASSPATH environment variable. Let's say a class Employee belongs to a package com.xyz.hr; and is located in the file c:\dev\com\xyz\hr\Employee.java. In this case, you'd need to add c:\dev to the variable CLASSPATH. If this class contains the method main(), you could test it from a command prompt window as follows:
c:\>java com.xyz.hr.Employee

Can an inner class declared inside of a method access local variables of this method?
It's possible if these variables are final.

How can a subclass call a method or a constructor defined in a superclass?
Use the following syntax: super.myMethod(); To call a constructor of the superclass, just write super(); in the first line of the subclass's constructor.

You can create an abstract class that contains only abstract methods. On the other hand, you can create an interface that declares the same methods. So can you use abstract classes instead of interfaces?

Sometimes, but your class may be a descendent of another class and in this case the interface is your only option.

How would you make a copy of an entire Java object with its state?
Have this class implement Cloneable interface and call its method clone().

What access level do you need to specify in the class declaration to ensure that only classes from the same directory can access it?
You do not need to specify any access level, and Java will use a default package access level.

Can we declare an anonymous class as both extending a class and implementing an interface?
No. An anonymous class can extend a class or implement an interface, but it cannot be declared to do both

What is a native method? - A native method is a method that is implemented in a language other than Java.

What are wrapped classes?
Wrapped classes are classes that allow primitive types to be accessed as objects.

What restrictions are placed on the location of a package statement within a source code file? - A package statement must appear as the first line in a source code file (excluding blank lines and comments).

How Observer and Observable are used?
Subclass of Observable class maintain a list of observers. Whenever an Observable object is updated, it invokes the update() method of each of its observers to notify the observers that it has a changed state. An observer is any object that implements the interface Observer

What does it mean that a method or field is "static"? - Static variables and methods are instantiated only once per class. In other words they are class variables, not instance variables. If you change the value of a static variable in a particular object, the value of that variable changes for all instances of that class. Static methods can be referenced with the name of the class rather than the name of a particular object of the class (though that works too). That's how library methods like System.out.println() work. out is a static field in the java.lang.System class.

When is static variable loaded? Is it at compile time or runtime? When exactly a static block is loaded in Java?

Static variable are loaded when classloader brings the class to the JVM. It is not necessary that an object has to be created. Static variables will be allocated memory space when they have been loaded. The code in a static block is loaded/executed only once i.e. when the class is first initialized. A class can have any number of static blocks. Static block is not member of a class, they do not have a return statement and they cannot be called directly. Cannot contain this or super. They are primarily used to initialize static fields.

What comes to mind when someone mentions a shallow copy in Java?

Object cloning.

The **clone( )** method generates a duplicate copy of the object on which it is called. Only classes that implement the **Cloneable** interface can be cloned.
The **Cloneable** interface defines no members. It is used to indicate that a class allows a bitwise copy of an object (that is, a *clone*) to be made. If we try to call **clone( )** on a class that does not implement **Cloneable**, a **CloneNotSupportedException** is thrown. When a clone is made, the constructor for the object being cloned is *not* called. A clone is simply an exact copy of the original.
Cloning is a potentially dangerous action, because it can cause unintended side effects. For example, if the object being cloned contains a reference variable called *obRef,* then when the clone is made, *obRef* in the clone will refer to the same object as does *obRef* in the original. If the clone makes a change to the contents of the object referred to by *obRef,* then it will be changed for the original object, too. Here is another example. If an object opens an I/O stream and is then cloned, two objects will be capable of operating on the same stream. Further, if one of these objects closes the stream, the other object might still attempt to write to it, causing an error.
Because cloning can cause problems, **clone( )** is declared as **protected** inside **Object**. This means that it must either be called from within a method defined by the class that implements **Cloneable**, or it must be explicitly overridden by that class so that it is public.

Implementing a deep copy of an object can be a learning experience -- you learn that you don't want to do it! If the object in question refers to other complex objects, which in turn refer to others, then this task can be daunting indeed. Traditionally, each class in the object must be individually inspected and edited to implement the Cloneable interface and override its clone() method in order to make a deep copy of itself as well as its contained objects. This article describes a simple technique to use in place of this time-consuming conventional deep copy.

The concept of deep copy
In order to understand what a *deep copy* is, let's first look at the concept of shallow copying.
In a previous *JavaWorld* article, "How to avoid traps and correctly override methods from java.lang.Object," Mark Roulo explains how to clone objects as well as how to achieve shallow copying instead of deep copying. To summarize briefly here, a shallow copy occurs when an object is copied without its contained objects. To illustrate, Figure 1 shows an object, obj1, that contains two objects, containedObj1 and containedObj2

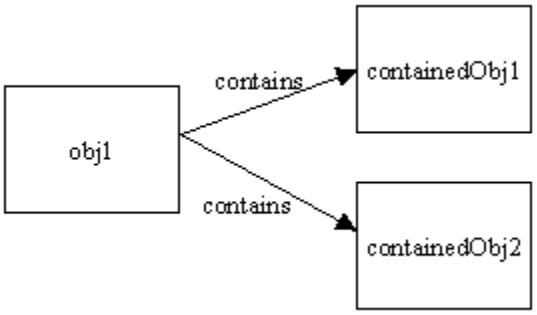


Figure 1. The original state of obj1

An alternative to the deep copy technique
Use serialization to make deep copies and avoid extensive manual editing or extending of classes

Implementing a deep copy of an object can be a learning experience -- you learn that you don't want to do it! If the object in question refers to other complex objects, which in turn refer to others, then this task can be daunting indeed. Traditionally, each class in the object must be individually inspected and edited to implement the Cloneable interface and override its clone() method in order to make a deep copy of itself as well as its contained objects. This article describes a simple technique to use in place of this time-consuming conventional deep copy.

The concept of deep copy
In order to understand what a *deep copy* is, let's first look at the concept of shallow copying.
In a previous *JavaWorld* article, "How to avoid traps and correctly override methods from java.lang.Object," Mark Roulo explains how to clone objects as well as how to achieve shallow copying instead of deep copying. To summarize briefly here, a shallow copy occurs when an object is copied without its contained objects. To illustrate, Figure 1 shows an object, obj1, that contains two objects, containedObj1 and containedObj2.

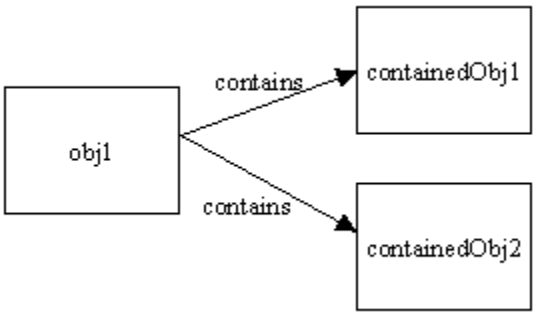


Figure 1. The original state of obj1

If a shallow copy is performed on obj1, then it is copied but its contained objects are not, as shown in Figure 2.

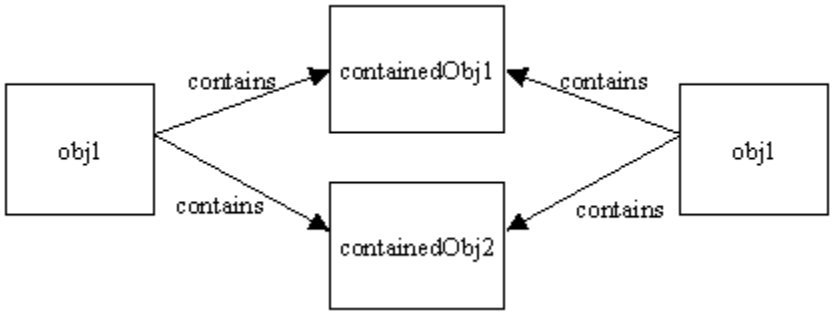


Figure 2. After a shallow copy of obj1

A deep copy occurs when an object is copied along with the objects to which it refers. Figure 3 shows obj1 after a deep copy has been performed on it. Not only has obj1 been copied, but the objects contained within it have been copied as well.

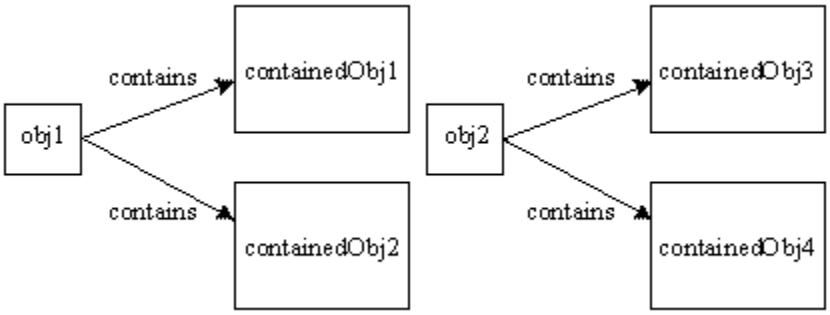


Figure 3. After a deep copy of obj1

If either of these contained objects themselves contain objects, then, in a deep copy, those objects are copied as well, and so on until the entire graph is traversed and copied. Each object is responsible for cloning itself via its clone() method. The default clone() method, inherited from Object, makes a shallow copy of the object. To achieve a deep copy, extra logic must be added that explicitly calls all contained objects' clone() methods, which in turn call their contained objects' clone() methods, and so on. Getting this correct can be difficult and time consuming, and is rarely fun. To make things even more complicated, if an object can't be modified directly and its clone() method produces a shallow copy, then the class must be extended, the clone() method overridden, and this new class used in place of the old. (For example, Vector does not contain the logic necessary for a deep copy.) And if you want to write code that defers until runtime the question of whether to make a deep or shallow copy an object, you're in for an even more complicated situation. In this case, there must be two copy functions for each object: one for a deep copy and one for a shallow. Finally, even if the object being deep copied contains multiple references to another object, the latter object should still only be copied once. This prevents the proliferation of objects, and heads off the special situation in which a circular reference produces an infinite loop of copies

A common solution to the deep copy problem is to use Java Object Serialization (JOS). The idea is simple: Write the object to an array using JOS's ObjectOutputStream and then use ObjectInputStream to reconsistute a copy of the object. The result will be a completely distinct object, with completely distinct referenced objects. JOS takes care of all of the details: superclass fields, following object graphs, and handling repeated references to the same object within the graph. Figure 3 shows a first draft of a utility class that uses JOS for making deep copies.

```
import java.io.IOException;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;

/**
 * Utility for making deep copies (vs. clone()'s shallow copies) of
 * objects. Objects are first serialized and then deserialized. Error
 * checking is fairly minimal in this implementation. If an object is
 * encountered that cannot be serialized (or that references an object
 * that cannot be serialized) an error is printed to System.err and
 * null is returned. Depending on your specific application, it might
 * make more sense to have copy(...) re-throw the exception.
 *
 * A later version of this class includes some minor optimizations.
 */
public class UnoptimizedDeepCopy {

   /**
    * Returns a copy of the object, or null if the object cannot
    * be serialized.
    */
   public static Object copy(Object orig) {
      Object obj = null;
      try {
         // Write the object out to a byte array
         ByteArrayOutputStream bos = new ByteArrayOutputStream();
         ObjectOutputStream out = new ObjectOutputStream(bos);
         out.writeObject(orig);
         out.flush();
         out.close();

         // Make an input stream from the byte array and read
         // a copy of the object back in.
         ObjectInputStream in = new ObjectInputStream(
            new ByteArrayInputStream(bos.toByteArray()));
         obj = in.readObject();
      }
      catch(IOException e) {
```

```
            e.printStackTrace();
        }
        catch(ClassNotFoundException cnfe) {
            cnfe.printStackTrace();
        }
        return obj;
    }

}
```

Figure 3. Using Java Object Serialization to make a deep copy

Unfortunately, this approach has some problems, too

1. It will only work when the object being copied, as well as all of the other objects references directly or indirectly by the object, are serializable. (In other words, they must implement java.io.Serializable.) Fortunately it is often sufficient to simply declare that a given class implements java.io.Serializable and let Java's default serialization mechanisms do their thing.
2. Java Object Serialization is slow, and using it to make a deep copy requires both serializing and deserializing. There are ways to speed it up (e.g., by pre-computing serial version ids and defining custom readObject() and writeObject() methods), but this will usually be the primary bottleneck.
3. The byte array stream implementations included in the java.io package are designed to be general enough to perform reasonable well for data of different sizes and to be safe to use in a multi-threaded environment. These characteristics, however, slow down ByteArrayOutputStream and (to a lesser extent) ByteArrayInputStream.