1 What is JVM and explain me the Java memory allocation

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

The JVM performs following operation:
a. Loads code
b. Verifies code
c. Executes code
d. Provides runtime environment

The JVM memory consists of the following segments: Code memory segment loads the java code ?
a. Heap Segment -- Heap Memory, which is the storage for Java objects
b. Stack Segment -- Non-Heap Memory, which is used by Java to store loaded classes and other meta-data
c. Code Segment JVM code itself, JVM internal structures, loaded profiler agent code and data, etc.

2. What is Polymorphism and encapsulation?

Polymorphism -- poly is many , morph means forms .. Provide a common interface and multiple methods ..Polymorphism is extensively used in implementing inheritance.
So polymorphism is the ability (in programming) to present the same interface for different forms (data types).
For example, integers and floats are implicitly polymorphic since you can add, subtract, multiply and so on, irrespective of the fact that the types are different. They're rarely considered as objects in the usual term.
Two types of Polymorphism.
Static polymorphism - method overloading
Dynamic polymorphism --method overriding
Encapsulation :
Encapsulation is the mechanism (Process) of hiding the implementation or hiding the data ..In encapsulation the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class, therefore it is also known as data hiding.To achieve encapsulation in Java
We will define the variables or methods of a class as private.
We will access them by using getter and setter method ..
in order to let the private variables use outside the class , we use getter and setter method .
getter and setter method is public . We cannot access these getter and setter methods directly . we can access only through private variables .
Data and code you put inside a capsule , they have derived as Encapsulation .

3. What is method overloading and Method overriding?

1.Method overloading -- static polymorphism -- compile time polymorphism -- Allows your class to have more methods with same name and with different data types ..

In the same class you will have 2 or more methods , they differ in return type and number of arguments , the order of arguments . so that was method overloading ..

Same method name but different data types

Overloading methods

```
Class Absolute(){
Absolute a = new Absolute();

Public void  cool(int i)  {        // will return int value  --
}
Public void cool(double i)  {       // will return double
}
```

Here in this above example -- both cool methods are with same name but different arguments -- like different data types , different parameters , different number of parameters

2. Method Overriding -- Overridden methods -- runtime polymorphism .Method Overriding is used in inheritance...Overriding deals with 2 methods , one in parent class and other in child class and both have the same name and signature.

In a class hierarchy when a method in the subclass has the same name and same type of signature as a method in the superclass , then it is said to be override the method in the superclass .

By combining inheritance with overridden methods, a superclass can define the general form of the methods that will be used by all of its subclasses ..

```
Class A {
Void cool()    {
}
}
Class B extends A    {
Void cool()    {
 }
}
```

We have a method cool in class A .

We also have same method cool in class B .

now we can say method cool in B overriding the method in class A .

The method in class B overrides method in class A ..

Methods Are exactly same .. the signature will remain the same and implementation differ .. Method Overriding - deals with 2 methods , one in the parent class and other one in the child class and has the same name and signature .

4. Why string is Immutable?

Strings -- Strings are immutable -- Strings are objects in java ..
In java, string objects are immutable. Immutable simply means unmodifiable or unchangeable once created can not be changed . Once string object is created its data or state can't be changed but a new string object is created . An immutable object is an object which state is guaranteed to stay identical over its entire lifetime. It means that the state of object once initialized, can never be changed anyhow.
Default value for Strings is null .
String class is FINAL it mean you can't create any class to inherit it and change the basic structure and make the Sting mutable.
The object created as a String is stored in the Constant String Pool .
Every immutable object in Java is thread safe ,that implies String is also thread safe . String can not be used by two threads simultaneously.
String once assigned cannot be changed.

String s1 = "Hello";
String s2 = s1; // s1 and s2 now point at the same string - "Hello"
Now, there is nothing we could do to s1 that would affect the value of s2. They refer to the same object - the string "Hello" - but that object is immutable and thus cannot be altered.

Why Strings are immutable --

a. **Synchronization and concurrency:** making String immutable automatically makes them thread safe thereby solving the synchronization issues. Immutable objects are safe when shared between multiple threads in multithreaded applications. If something can't be changed, then even thread can not change it.
b. **Security:** parameters are typically represented as String in network connections, database connection urls, usernames/passwords etc. If it were mutable, these parameters could be easily changed. If Strings were mutable, then anybody could have injected its own class-loading mechanism with very little effort and destroyed or hacked in any application in a minute.

5. What is the difference between String and StringBuffer?

String
String is immutable (once created can not be changed)object . The object created as a String is stored in the Constant String Pool .
Every immutable object in Java is thread safe ,that implies String is also thread safe . String can not be used by two threads simultaneously.
String once assigned cannot be changed.
StringBuffer

StringBuffer is mutable means one can change the value of the object . The object created through StringBuffer is stored in the heap . StringBuffer has the same methods as the StringBuilder , but each method in StringBuffer is synchronized that is StringBuffer is thread safe .

Due to this it does not allow two threads to simultaneously access the same method . Each method can be accessed by one thread at a time .

But being thread safe has disadvantages too as the performance of the StringBuffer hits due to thread safe property . Thus StringBuilder is faster than the StringBuffer when calling the same methods of each class.

StringBuffer value can be changed , it means it can be assigned to the new value . Nowadays it's a most common interview question ,the differences between the above classes .
String Buffer can be converted to the string by using
toString() method.
Example
StringBuffer sf1 = new StringBuffer("Thank you");
Sf1.append("Good");
System.out.println(sf1); -- will print --Thank you.Good
StringBuffer sf2 = new StringBuffer(str1);

StringBuilder
StringBuilder is same as the StringBuffer , that is it stores the object in heap and it can also be modified . The main difference between the StringBuffer and StringBuilder is that StringBuilder is also not thread safe.
StringBuilder is fast as it is not thread safe .
StringBuilder demo2= new StringBuilder("Hello");
// The above object too is stored in the heap and its value can be modified
demo2=new StringBuilder("Bye");
// Above statement is right as it modifies the value which is allowed in the StringBuilder

	String	StringBuffer	StringBuilder
Storage Area	Constant String Pool	Heap	Heap
Modifiable	No (immutable)	Yes(mutable)	Yes(mutable)
Threadsafe	Yes	Yes	No
Performance	Fast	Very slow	Fast

6. What is the difference between array and arrayList?

Array : Java provides a data structure, the array, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.
Example :

```
public class TestArray {
public static void main(String[] args) {
      double[] myList = {1.9, 2.9, 3.4, 3.5};
      // Print all the array elements
      for (int i = 0; i < myList.length; i++) {
        System.out.println(myList[i] + " ");
      }
      // Summing all elements
      double total = 0;
      for (int i = 0; i < myList.length; i++) {
        total += myList[i];
      }
      System.out.println("Total is " + total);
```

ArrayList : Arraylist is a class which implements List interface. It is widely used because of the functionality and flexibility it offers. Most of the developers choose Arraylist over Array as it's a very good alternative of traditional java arrays.

The issue with arrays is that they are of fixed length so if it is full we cannot add any more elements to it, likewise if there are number of elements gets removed from it the memory consumption would be the same as it doesn't shrink. On the other ArrayList can dynamically grow and shrink as per the need. Apart from these benefits ArrayList class enables us to use predefined methods of it which makes our task easy. Let's see the ArrayList example first then we will discuss it's methods and their usage.
Example -

```
ArrayList<Integer>MyList = new ArrayList<Integer>(5);
MyList.add(1) ;
MyList.add(5) ;
MyList.add(8) ;
MyList.add(0) ;
MyList.add(3) ;
for(Integer x : MyList)
System.out.println(x);
```

To print the size of the ArrayList --
System.out.println(MyList.size()); -- will print size as 5 .
If we delete 2 variables from ArrayList,
ArrayList<Integer>MyList = new ArrayList<Integer>(5);
MyList.add(1) ;
MyList.add(5) ;
MyList.add(8) ;
for(Integer x : MyList)
System.out.println(x);
To print the size of the ArrayList --
System.out.println(MyList.size()); -- will print size as 3 .It won't show error as we declare the size as 5 initially and we have only 3.But in Array if there is difference between initialised size and actual size , it will show error.

Differences between Array and ArrayList :

1. Resizable : Array is static in size that is fixed length data structure, One can not change the length after creating the Array object.
ArrayList is dynamic in size . Each ArrayList object has instance variable capacity which indicates the size of the ArrayList. As elements are added to an ArrayList its capacity grows automatically.

2. Import Library : Java Provides a library called utils and in this utils library there is a keyword called ArrayList helps us maintain the size of it dynamically ..
To Use ArrayList we need to -- import java.utils.ArrayList. To declare the ArrayList , you need a keyword called ArrayList .. You don't need to import any Library to use Array ..

3. Performance : Performance of Array and ArrayList depends on the operation you are performing :
add() or get() operation : adding an element or retrieving an element from the array or arraylist object has almost same performance , as for ArrayList object these operations run in constant time.

4. Primitives : ArrayList can not contains primitive data types (like int , float , double) it can only contains Object while Array can contain both primitive data types as well as objects.
5.Length : Length of the ArrayList is provided by the size() method .to print the size - System.out.println(MyList.size());
To print size of Array -System.out.println(intArray.length());

6. Multi-dimensional : Array can be multi dimensional , while ArrayList is always single dimensional.

7. What is the difference between HashMap and HashTable?

HashMap maintains key and value pairs and often denoted as HashMap<Key, Value> or HashMap<K, V>. HashMap implements Map interface. HashMap is similar to HashTable with two exceptions – HashMap methods are not synchronized and it allows null key and null values unlike Hashtable. It is used for maintaining key and value mapping.

Eg-

```
Map<String,String> phonebook= new  HashMap<>();
phonebook.put("x"  , "7");
phonebook.put("y"  , "9");
phonebook.put("z"  , "4");
phonebook.put("A"  , "8");
phonebook.put("B"  , "5");
To Print any one value --
System.out.println(phonebook.get("z")); -- prints --4
To print all the values and keys
Set<String> keys = phonebook.keyset();
for(String i : keys)  {
System.out.println(i+" : " + phonebook.get(i));
```

HashTable:

This class implements a hash table, which maps keys to values. Any non-null object can be used as a key or as a value. Hashtable is similar to HashMap except it is synchronized. Hashtable is now integrated into the collections framework. It is similar to HashMap, but is synchronized.

Like HashMap, Hashtable stores key/value pairs in a hash table. When using a Hashtable, you specify an object that is used as a key, and the value that you want linked to that key. The key is then hashed, and the resulting hash code is used as the index at which the value is stored within the table.

Difference between HashMap and HashTable

HashMap	**HashTable**
a. Introduced in 1.2 version	a . HashTable is there since Java introduced

b. It is not thread.safe and synchronized	b. It is thread safe and is synchronized
c. It is fast	c. It is slow
d. Works with single thread	d . Works with multiple threads .
e. Allows one null key	e . Does not allow null key

Synchronised : means that in a multiple threaded environment, a synchronised object does not let two threads access a method/block of code at the same time. This means that one thread can't be reading while another updates it. The second thread will instead wait until the first is done.

8. What is a vector in Java?

Vector is a type of list .Vector is a dynamic array where we can increase the size of the array .
Vector implements an arrayList. It is similar to ArrayList, but with two differences:
Vector is synchronized.
Vector contains many legacy methods that are not part of the collections framework.
Vector proves to be very useful if you don't know the size of the array in advance or you just need one that can change sizes over the lifetime of a program.

Eg -

```
Vector v  = new vector ();
v.add (6);
v.add(9);
v.add(3);
v.remove(1) ;    --index 1
System.out.println(a.capacity());
for(int i : v)  {
System.out.println(i);    }
```

Difference between vector and ArrayList

a. Vector introduced at 1.0 that when java introduced .. later it is modified to use lists ..
b. Capacity .. Initially the capacity of ArrayList or vector is 10 .. if it exceeds the capacity , vector will increase the capacity by 100% ..
and ArrayList will increase by 50% .. vector will waste so much space ..ArrayList will save the memory ..

c. vectors are synchronized and theoretically thread safe , ArrayList is not thread safe ..

9. What is set in java?

A Set is a Collection that cannot contain duplicate elements.The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited.
There are three main implementations of Set interface:

HashSet,
TreeSet,
LinkedHashSet.

Comparison of Hashset and Treeset and LinkedHashset
HashSet doesn't maintain any kind of order of its elements.
TreeSet sorts the elements in ascending order.
LinkedHashSet maintains the insertion order. Elements gets sorted in the same sequence in which they have been added to the Set.

HashSet : HashSet, which stores its elements in a hash table, is the best-performing implementation; however it makes no guarantees concerning the order of iteration. Points to note .

a. HashSet doesn't maintain any order, the elements would be returned in any random order.
b. HashSet doesn't allow duplicates. If you try to add a duplicate element in HashSet, the old value would be overwritten.
c. HashSet allows null values however if you insert more than one nulls it would still return only one null value.
d. HashSet is non-synchronized.

Eg --

```
import java.util.HashSet;
public class HashSetExample {
public static void main(String args[]) {
HashSet<String> hset = new HashSet<String>();
    hset.add("Apple");
    hset.add("Mango");
    hset.add("Grapes");
    hset.add("Orange");
    hset.add("Fig");
    //Addition of duplicate elements
    hset.add("Apple");
    hset.add("Mango");
```

```
    //Addition of null values
    hset.add(null);
    hset.add(null);
    System.out.println(hset);  output -[null, Mango, Grapes, Apple, Orange, Fig]
  }       }
```

LinkedHashSet : LinkedHashSet is also an implementation of Set interface, LinkedHashSet maintains the insertion order. Elements gets sorted in the same sequence in which they have been added to the Set.

Example of LinkedHashSet:

```
import java.util.LinkedHashSet;
public class LinkedHashSetExample {
    public static void main(String args[]) {
      LinkedHashSet<String> lhset = new LinkedHashSet<String>();
       lhset.add("Z");
       lhset.add("PQ");
       lhset.add("N");
       lhset.add("O");
       lhset.add("KK");
       lhset.add("FGH");
       System.out.println(lhset);   Output -- [Z, PQ, N, O, KK, FGH] -same order
       lhset.add("AB");
       lhset.add("XY");
       System.out.println(lhset1);  Output -- [Z, PQ, N, O, KK, FGH , AB , XY]

  }       }
```

TreeSet : TreeSet is similar to HashSet except that it sorts the elements in the ascending order while HashSet doesn't maintain any order. TreeSet allows null element but like HashSet it doesn't allow. Like most of the other collection classes this class is also not synchronized,

Example

```
import java.util.TreeSet;
public class TreeSetExample {
public static void main(String args[]) {
       // TreeSet of String Type
       TreeSet<String> tset = new TreeSet<String>();
       tset.add("ABC");
```

```
tset.add("String");
tset.add("Test");
tset.add("Pen");
tset.add("Ink");
tset.add("Jack");
System.out.println(tset);  --Output - [ABC, Ink, Jack, Pen, String, Test]
// TreeSet of Integer Type
TreeSet<Integer> tset2 = new TreeSet<Integer>();
tset2.add(88);
tset2.add(7);
tset2.add(101);
tset2.add(0);
tset2.add(3);
tset2.add(222);
System.out.println(tset2);   -- Output -- [0, 3, 7, 88, 101, 222]
    }   }
```

Output: You can see both the TreeSet have been sorted in ascending order implicitly.

10. What is an abstract class?

Abstraction -

Showing essential things and hiding the non- essential things from the user is called Abstraction

You don't know what is inside TV. We will use remote to watch TV and we will go to the channel we want from remote .We don't know the internal details .. internal details are hidden now and they will let you see see, what you want to see .

Abstract class

a. A class automatically becomes Abstract class when any one of the method is Abstract ..
b. Abstract class is an incomplete class where we cannot instantiate (create) an object ..
c. You will create object for the class that extends Abstract class ..
d. Abstract classes will not be 100% abstract ,because it will accept non abstract methods also.Abstract classes will have non abstract methods also **Abstract method** -- there will be only name of the method , but, there is body in it ..
e. Variable cannot be made Abstract , it's only methods or behavior that can be Abstract.

Abstract Methods:

If you want a class to contain a particular method but you want the actual implementation of that method to be determined by child classes, you can declare the method in the parent class as abstract.
abstract keyword is used to declare the method as abstract.
You have to place the abstract keyword before the method name in the method declaration.
An abstract method contains a method signature, but no method body.
Instead of curly braces an abstract method will have a semicolon (;) at the end.

Below given is an example of the abstract method.

```
Eg- public abstract class Employee  {
   private String name;
   private String address;
   private int number;
   public abstract double computePay();
  }
```

Declaring a method as abstract has two consequences:
The class containing it must be declared as abstract.
Any class inheriting the current class must either override the abstract method or declare itself as abstract.

```
Eg - public class Salary extends Employee
{
   private double salary;
   public double computePay()      {
      System.out.println();
   }
}
```

11. What is an interface?

Interface

Interface looks like class but it is not a class. An interface can have methods and variables just like the class but the methods declared in interface are by default abstract (only method signature, no body). Also, the variables declared in an interface are public, static & final by default
What is the use of interfaces?

As mentioned above they are used for abstraction. Since methods in interfaces do not have body, they have to be implemented by the class before you can access them. The class that implements interface must implement all the methods of that interface. Also, java programming language does not support multiple inheritance, using interfaces we can achieve this as a class can implement more than one interfaces, however it cannot extend more than one classes.
we cannot instantiate object in abstract class ..

An interface is similar to a class in the following ways:

An interface can contain any number of methods.

An interface is written in a file with a .java extension, with the name of the interface matching the name of the file.

The byte code of an interface appears in a .class file.

Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

However, an interface is different from a class in several ways, including:

You cannot instantiate an interface.

An interface does not contain any constructors.

All of the methods in an interface are abstract.

An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.

An interface is not extended by a class; it is implemented by a class.

An interface can extend multiple interfaces.

Example --

```
interface Animal {

   public void eat();
   public void travel();
}
-----------------------------------------------------------------
/* File name : MammalInt.java */
public class  implements Animal{

   public void eat(){
      System.out.println("Mammal eats");
   }

   public void travel(){
```

```
    System.out.println("Mammal travels");
  }

  public int noOfLegs(){
    return 0;
  }

  public static void main(String args[]){
    MammalInt m = new MammalInt();
    m.eat();
    m.travel();
  }
}
```

Output --
Mammal eats
Mammal travels

Extending Multiple Interfaces:
A Java class can only extend one parent class. Multiple inheritance is not allowed. Interfaces are not classes, however, and an interface can extend more than one parent interface.

The extends keyword is used once, and the parent interfaces are declared in a comma-separated list.

For example, if the Hockey interface extended both Sports and Event, it would be declared as:

public interface Hockey extends Sports, Event

Even though interface doesn't have abstract methods , it is 100% abstract .. That means it will not have body ..it can contain only method , no body ..
What ever variable you are going to give here , it is final , you can just start using it ..
It is just name in interface . all implementation has to happen in implementation classes ..
No calculations , nothing ..
All implemented methods are overriden ..
In the below example both CityBank and ChaseBank implements interface Bank
---

```
public interface Bank {
        int minDep=25;
```

```
        public void createAccount();
        public void depositMoney();
        public void withDrawMoney();
        public void onlineBanking();
}
--------------------------------------------------------------------------------
Eg--
public class  CityBank implements Bank {
        @Override
        public void createAccount(){
                System.out.println("CityBank creates accounts");
        }
        @Override
        public void depositMoney() {
                System.out.println("CityBank deposit accounts");
                }
        @Override
        public void withDrawMoney() {
                System.out.println("CityBankwithdraw amount");
        }
        @Override
        public void onlineBanking() {
                System.out.println("CityBank online accounts");
        }
        public void ATM()  {
                System.out.println("ATM");
        }
}
```

12. Why Java is Platform independent?

With Java, you can compile source code on Windows and the compiled code (bytecode to be precise) can be executed (interpreted) on any platform running a JVM. So yes you need a JVM but the JVM can run any compiled code, the compiled code is platform independent.

We say Java is a platform independent language in the sense that a program written in Java can be executed on any operating system with any hardware. Now, of course, many programs we are all familiar with, such as Skype or Excel, can be used on multiple operating systems like Windows 8 and OS X. But notice that these programs have different versions for different operating systems, which means that they are actually not platform independent. For programs written in Java, you can directly use them on either Windows or Mac OS or even Linux without any modification. This is the special feature of Java.

The reason why Java is platform independent is that it is structured differently from other programming languages like C or C++. With C, the source code (that is, the original content you write in the C programming environment) must be compiled into the machine code for the computer to execute. This is a very basic and yet crucial point: computers can only execute machine code. The source code that we write in most of the programming languages is Greek to computers, just like machine code is Greek to us. The process of compiling the source code to the machine code is done by something called, well, compiler. Now here comes the second important point: a compiler uses the functions of the current operating system to compile the source code, which means that the compiled machine code can only be executed on this very operating system, thus making the program platform dependent.

For programs written in Java, this process is a bit different. Instead of directly being compiled into the machine code, Java source code is compiled into an intermediate code called byte code. The byte code is not specific to any operating system, in other words, it is platform independent. Now of course, as mentioned just now, for any computer to execute any program, it must be in the form of machine code. This also holds true for Java, but Java programs are executed somehow indirectly, via the so-called Java virtual machine aka JVM. Within JVM, Java byte code is "interpreted" to a sequence of subroutines that are pre-compiled in machine code.

13. What are access modifiers? Give me an example?

Java provides a number of access modifiers to set access levels for classes, variables, methods and constructors. The four access levels are:
Access Modifiers --- Private , Public , Protected and default..
Java provides a number of access modifiers to set access levels for classes, variables, methods and constructors. The four access levels are:
a. Visible to the package. (default)... No modifiers are needed.
b. Visible to the class only (private).
c. Visible to the world (public).
d. Visible to the package and all subclasses (protected).
Default Access Modifier - No keyword:
Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc.
A variable or method declared without any access control modifier is available to any other class in the same package. The fields in an interface are implicitly public static final and the methods in an interface are by default public.

Private Access Modifier - private:
Methods, Variables and Constructors that are declared private can only be accessed within the declared class itself.

Private access modifier is the most restrictive access level. Class and interfaces cannot be private.
Variables that are declared private can be accessed outside the class if public getter methods are present in the class.
Using the private modifier is the main way that an object encapsulates itself and hide data from the outside world.
Example:
The following class uses private access control:

```
  public class Logger {
  private String format;
  public String getFormat()    {
    return this.format;     }
  public void setFormat(String format) {
    this.format = format;
  }
}
```

Here, the format variable of the Logger class is private, so there's no way for other classes to retrieve or set its value directly.So to make this variable available to the outside world, we defined two public methods:
getFormat(), which returns the value of format,
setFormat(String), which sets its value.

Public Access Modifier - public:
A class, method, constructor, interface etc declared public can be accessed from any other class. Therefore fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe.
However if the public class we are trying to access is in a different package, then the public class still need to be imported.
Because of class inheritance, all public methods and variables of a class are inherited by its subclasses.
Protected Access Modifier - protected:
Variables, methods and constructors which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class.
The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared protected, however methods and fields in a interface cannot be declared protected.

14. What are java exceptions? Give me an example

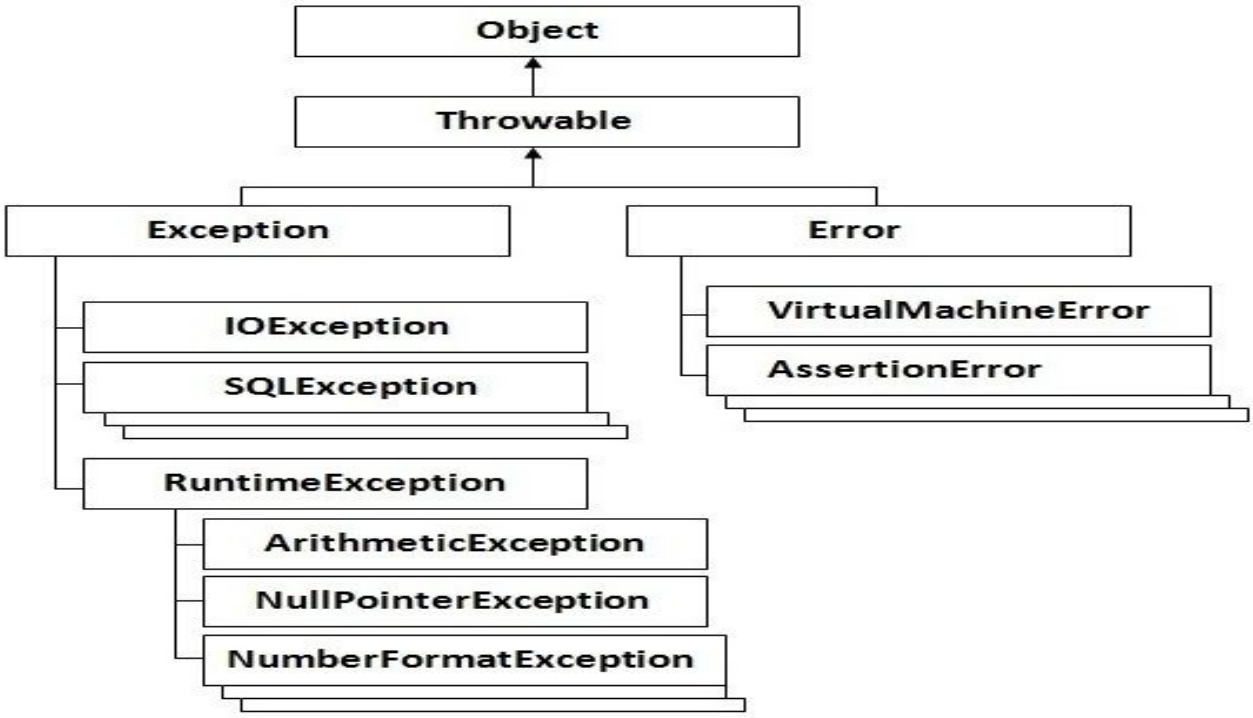
Exceptions --Exceptions are conditions failure at Run-time within the code. A developer can handle such conditions and take necessary corrective actions.In java,

Exceptions is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

Types of Exceptions -

Checked Exceptions -Compile time exceptions - checked exceptions are checked at Compile-time ..IOException, SQLExceptions

Unchecked Exceptions -run-time Exception -- Unchecked Exceptions are checked at Run-time ..Arithmetic Exceptions, NullPointerExceptions, ArrayIndexOutOfBounds Exceptions ..



NullPointerException : Java.lang.NullPointer Exception ...An object of this class gets created whenever a member is invoked with a "null" object. Example ---try{

```
        String str=null;
        System.out.println (str.length());
}catch(NullPointerException e){
        System.out.println("NullPointerException..");
}
```

ArithmeticException----Class: Java.lang.ArithmeticException..This is a built-in-class present in java.lang package. This exception occurs when an integer is divided by zero. Example try{

```
int num1=30, num2=0;
int output=num1/num2;
System.out.println ("Result = " +output);
```

```
    }
    catch(ArithmeticException e){
      System.out.println ("Arithmetic Exception: You can't divide an integer by 0");
    }
```

ArrayIndexOutOfBoundsException :

Java.lang.ArrayIndexOutOfBoundsException ….This is a built in class present in java.lang package. This exception occurs when the referenced element does not exist in the array. For e.g. If array is having only 5 elements and we are trying to display 7th element then it would throw this exception. Example --------

```
class ExceptionDemo2
{
  public static void main(String args[])
  {
    try{
      int a[]=new int[10];
      //Array has only 10 elements
      a[11] = 9;
    }
    catch(ArrayIndexOutOfBoundsException e){
      System.out.println ("ArrayIndexOutOfBounds");
    }
  }
}
```

To handle Exceptions we have 3 concepts

a. Try and catch
b. Throws
c. Throwable .

15. What is the difference between throws and throwable?

Throwable are used in class level ..Throws is at method level , .. Throw is a statement level .

Throwable : Throwable are used in class level .. throwable is a class , it's a parent class for all the exceptions class and errors class .. Throwable class is already defined in JDK Library .. It's a pre-defined class ..Throwable handle both exceptions and errors . Eg - Class A extends Throwable

Eg -

```
Class A Extends throwable {
Main method () throws exception {
Int i = 10/0; throw AI
Good code
}
}
```

Throws : Throws is at method level ,.Throws handle only exceptions ..By using Throw keyword in java you cannot throw more than one exception but using throws you can declare multiple exceptions.To throw an exception we use throw keyword while to handle (catch) an exception we use throws clause.

a. Exceptions -In java, exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

Types of Exceptions

a. IOExceptions

b. SQLExceptions

c. RunTimeExceptions -- Arithmetic Exceptions , Null Pointer Exceptions,NumberPointerExceptions

16. What is the difference between Error and exception?

Difference between error and exception

Error : Errors are also unchecked exception & the programmer is not required to do anything with these.Error is irrecoverable condition , that are not expected to be caught under normal circumstances by our program. For example memory error, hardware error, JVM error etc .Error is a compile time error it may be syntax error or else..

Exceptions : Exception occurs during the execution time it cannot get identified by compiler. Exceptions are conditions failure at Run-time within the code. A developer can handle such conditions and take necessary corrective actions.In java, Exceptions is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.The following are exceptions

a. IOExceptions

b. SQLExceptions

c. RunTimeExceptions -- Arithmetic Exceptions , Null Pointer Exceptions,NumberPointerExceptions

Types of Exceptions -

Checked Exceptions - checked exceptions are checked at Compile-time ..IOException, SQLExceptions

Unchecked Exceptions - Unchecked Exceptions are checked at Run-time ..Arithmetic Exceptions,
NullPointerExceptions, ArrayIndexOutOfBounds Exceptions ..

17. What is the difference between Error, throwable and exception?

Throwable : Throwable are used in class level .. throwable is a class , it's a parent class for all the exceptions class and errors class .. Throwable class is already defined in JDK Library .. It's a pre-defined class ..Throwable handle both exceptions and errors . Eg - Class A extends Throwable

Eg -

Class A Extends throwable {

Main method () throws exception {
Int i = 10/0; throw AI
Good code
}
}
Error : Errors are also unchecked exception & the programmer is not required to do anything with these.Error is irrecoverable condition , that are not expected to be caught under normal circumstances by our program. For example memory error, hardware error, JVM error etc .Error is a compile time error it may be syntax error or else..
Exceptions : Exception occurs during the execution time it cannot get identified by compiler. Exceptions are conditions failure at Run-time within the code. A developer can handle such conditions and take necessary corrective actions.In java, Exceptions is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.The following are exceptions
a. IOExceptions
b. SQLExceptions
c. RunTimeExceptions -- Arithmetic Exceptions , Null Pointer Exceptions,NumberPointerExceptions
Types of Exceptions -
Checked Exceptions - checked exceptions are checked at Compile-time .They are IOException, SQLExceptions
Unchecked Exceptions - Unchecked Exceptions are checked at Run-time . They are Arithmetic Exceptions, NullPointerExceptions, ArrayIndexOutOfBounds Exceptions ..

18. What are collection APIs, give me an example

The Java Collections Framework is a collection of interfaces and classes which helps in storing and processing the data efficiently. This framework has several useful classes which have tons of useful functions which makes a programmer task super easy.
Collection represents a single unit of objects i.e. a group.

The Java collections framework (JCF) is a set of classes and interfaces that implement commonly reusable collection data structures. Although referred to as a framework, it works in a manner of a library. The JCF provides both interfaces that define various collections and classes that implement them.
Collections in Java

Collections in java is a framework that provides an architecture to store and manipulate the group of objects.

All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections.

Java Collection simply means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque etc.) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet etc).

What is framework in java

provides readymade architecture.
represents set of classes and interface.
is optional.
What is Collection framework

Collection framework represents a unified architecture for storing and manipulating group of objects.
It has: Interfaces and its implementations i.e. classes
Algorithm

19. What is the difference between final and finally?

Java finally block

Java finally block is a block that is used to execute important code such as closing connection, stream etc.
Java finally block is always executed whether exception is handled or not.
Java finally block must be followed by try or catch block.

Example --exception doesn't occur.

```
class TestFinallyBlock{
  public static void main(String args[]){
  try{
   int data=25/5;
   System.out.println(data);
  }
  catch(NullPointerException e){System.out.println(e);}
  finally{System.out.println("finally block is always executed");}
  System.out.println("rest of the code...");
  }
}
```

Output:5
finally block is always executed
rest of the code..

Example-- exception occurs and handled.

```
public class TestFinallyBlock2{
  public static void main(String args[]){
  try{
```

```
   int data=25/0;
   System.out.println(data);
  }
  catch(ArithmeticException e){System.out.println(e);}
  finally{System.out.println("finally block is always executed");}
  System.out.println("rest of the code...");
  }
}
```

Test it Now

Output:Exception in thread main java.lang.ArithmeticException:/ by zero
finally block is always executed
rest of the code...

Final Keyword In Java

The final keyword in java is used to restrict the user. The java final keyword can be used in many context.
The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. Final can be:

variable
method
class

1) Java final variable
If you make any variable as final, you cannot change the value of final variable(It will be constant).
2) Java final method
If you make any method as final, you cannot override it.
3) Java final class
If you make any class as final, you cannot extend it.

Finalise
The java.lang.Object.finalize() is called by the garbage collector on an object when garbage collection determines that there are no more references to the object. A subclass overrides the finalize method to dispose of system resources or to perform other cleanup.The garbage collector is working automatically in the background (although it can be explicitly invoked, but the need for this should be rare). It basically cleans up only objects which are not referenced by other objects

Declaration

Following is the declaration for **java.lang.Object.finalize()** method

Final	**Finally**	**Finalise**
Final is used to apply restrictions on class, method and variable. Final class can't be inherited, final method can't be overridden and final variable value can't be changed.	Finally is used to place important code, it will be executed whether exception is handled or not.	Finalize is used to perform clean up processing just before object is garbage collected.
Final is a keyword.	Finally is a Block	Final is a Method.

20. Will java supports multiple inheritance?

Multiple inheritance in java - no we cannot have multiple inheritance . Java does not support multiple inheritance ..
Inheritance - Reusability is the main aim for Inheritance ..
A class that is derived from another class is called subclass and inherits all fields and methods of its superclass. In Java, only single inheritance is allowed and thus, every class can have at most one direct superclass..
Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.
The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

Example --

```
Class A  {    // parent class , superclass
Int i ;
Int j ;
}

Class B extends class A  { // child class / sub-class
Int k;
}
```

Single Inheritance

```
Class A {
Void cool() {
```

}
}
Class B extends class A { --method in class B overrides the method in class A
Void cool() {
}

}
Class C extends class A { -- Single level Inheritance
Void cool() { ---- method in class C overrides the method in class A
}
}

Multiple inheritance in java - no we cannot have multiple inheritance . Java cannot support multiple inheritance ..
Class A {
Void cool() {
}
}
Class B extends class A { --method in class B overrides the method in class A
Void cool() {
}
}
Class C extends A , B { -- Not possible
Void cool() { ---- method in class C overrides the method in class A
}
}

Multi-levels inheritance -
Class A {
Void cool() {
}
}
Class B extends class A { --method in class B overrides the method in class A
Void cool() {
}
}
Class C extends class B { -- Multi-level Inheritance
Void cool() { ---- method in class C overrides the method in class A
}
}

22. What are wrapper class? Give me an example

a simple data type can be converted into an object (with wrapper classes). Wrapper classes are used to convert any data type into an object.

Wrapper class in java provides the mechanism to convert primitive into object and object into primitive.

Since J2SE 5.0, autoboxing and unboxing feature converts primitive into object and object into primitive automatically. The automatic conversion of primitive into object is known and autoboxing and vice-versa unboxing.

One of the eight classes of java.lang package are known as wrapper class in java.

Importance of Wrapper classes

There are mainly two uses with wrapper classes.

To convert simple data types into objects, that is, to give object form to a data type; here constructors are used.
To convert strings into data types (known as parsing operations), here methods of type parseXXX() are used.

The list of eight wrapper classes are given below:

Primitive Type	Wrapper class
boolean	Boolean
char	Character
Byte	Byte

short	Short
int	Integer
long	Long
float	Float
double	Double

Wrapper class Example: Primitive to Wrapper

```
public class WrapperExample1{
public static void main(String args[]){
//Converting int into Integer
int a=20;
Integer i=Integer.valueOf(a);//converting int into Integer
Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally

System.out.println(a+" "+i+" "+j);
} }
```

Output : 20 20 20

Wrapper class Example: Wrapper to Primitive

```
public class WrapperExample2{
public static void main(String args[]){
//Converting Integer to int
Integer a=new Integer(3);
int i=a.intValue();//converting Integer to int
int j=a;//unboxing, now compiler will write a.intValue() internally

System.out.println(a+" "+i+" "+j);
}  }
```

Output : 3 3 3

23. What is boxing and unboxing in Java? Explain with an example

The automatic conversion of primitive data types into its equivalent Wrapper type is known as boxing and opposite operation is known as unboxing. This is the new feature of Java5. So java programmer doesn't need to write the conversion code.

AutoBoxing -- Conversion of primitive data types into its equivalent Wrapper type is known as boxing ..

Example :

```
class BoxingExample1{
  public static void main(String args[]){
    int a=50;
        Integer a2=new Integer(a);    //Boxing
          Integer a3=5;    //Boxing
        System.out.println(a2+" "+a3);    -- Output  :   50   5
 }     }
```

Unboxing : The automatic conversion of wrapper class type into corresponding primitive type, is known as Unboxing.

Example :

```
class UnboxingExample1{
  public static void main(String args[]){
    Integer i=new Integer(50);
        int a=i;
        System.out.println(a);  --Output  :  5   0
 }      }
```

21. What are the different types of interface?

Different types of interface --

List

The List interface extends Collection and declares the behavior of a collection that stores a sequence of elements.

Elements can be inserted or accessed by their position in the list, using a zero-based index.

A list may contain duplicate elements.

Set

A Set is a Collection that cannot contain duplicate elements. It models the mathematical set abstraction.

The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited.

Set also adds a stronger contract on the behavior of the equals and hashCode operations, allowing Set instances to be compared meaningfully even if their implementation types differ.

Queue

The java.util.Queue interface is a subtype of the java.util.Collection interface. It represents an ordered list of objects just like a List, but its intended use is slightly different. A queue is designed to have elements inserted at the end of the queue, and elements removed from the beginning of the queue. Just like a queue in a supermarket.
A queue is a FIFO sequence. Addition takes place only at the tail, and removal takes place only at the head.

24. Explain for each loop

In Java we have three types of basic loops: for, while & do-while.
What is For loop?
It executes a block of statements repeatedly until the specified condition returns false.Syntax of for loop:

```
for (initialization; condition; increment/decrement) {
   statement(s)          //block of statements
}
```

Initialization expression executes only once during the beginning of loop
Condition(Boolean Expression) gets evaluated each time the loop iterates. Loop executes the block of statement repeatedly until this condition returns false.
Increment/Decrement : It executes after each iteration of loop.

For loop example:

```
class ForLoopExample {
   public static void main(String args[]){
       for(int i=10; i>1; i--){
           System.out.println("The value of i is: "+i);
       }
   }
}
```

25. What are iterators, explain with an example ?

Iterators : Often, you will want to cycle through the elements in a collection. For example, you might want to display each element.
The easiest way to do this is to employ an iterator, which is an object that implements either the Iterator or the ListIterator interface.
Iterator enables you to cycle through a collection, obtaining or removing elements. ListIterator extends Iterator to allow bidirectional traversal of a list, and the modification of elements.
Before you can access a collection through an iterator, you must obtain one. Each of the collection classes provides an iterator( ) method that returns an iterator to the start of the collection. By using this iterator object, you can access each element in the collection, one element at a time.

In general, to use an iterator to cycle through the contents of a collection, follow these steps:

a. Obtain an iterator to the start of the collection by calling the collection's iterator( ) method.
b. Set up a loop that makes a call to hasNext( ). Have the loop iterate as long as hasNext( ) returns true.
c. Within the loop, obtain each element by calling next( ).
d. import java.util.*;

Example --

```
public class IteratorDemo {

   public static void main(String args[]) {
     // Create an array list
     ArrayList al = new ArrayList();
     // add elements to the array list
     al.add("C");
     al.add("A");
     al.add("E");
     al.add("B");
     al.add("D");
     al.add("F");

     // Use iterator to display contents of al
     System.out.print("Original contents of al: ");
     Iterator itr = al.iterator();
     while(itr.hasNext()) {
       Object element = itr.next();
       System.out.print(element + " ");
     }
     System.out.println();

         // Modify objects being iterated
     ListIterator litr = al.listIterator();
     while(litr.hasNext()) {
       Object element = litr.next();
       litr.set(element + "+");
     }
     System.out.print("Modified contents of al: ");
     itr = al.iterator();
     while(itr.hasNext()) {
       Object element = itr.next();
```

```
      System.out.print(element + " ");
    }
    System.out.println();

    // Now, display the list backwards
    System.out.print("Modified list backwards: ");
    while(litr.hasPrevious()) {
      Object element = litr.previous();
      System.out.print(element + " ");
     }
     System.out.println();
  }
}
```

26. How do you access Private variables in different class ?

To Access Private variables we have to use setter and getter methods ..

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as as single unit. In encapsulation the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class, therefore it is also known as data hiding.

To achieve encapsulation in Java

a. Declare the variables of a class as private.
b. Provide public setter and getter methods to modify and view the variables values.

Example:

```
/* File name : EncapTest.java */
public class EncapTest{
  private String name;
  private String idNum;
  private int age;

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

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

The public setter() and getter() methods are the access points of the instance variables of the EncapTest class. Normally, these methods are referred as getters and setters. Therefore any class that wants to access the variables should access them through these getters and setters.

The variables of the EncapTest class can be accessed as below::

```
/* File name : RunEncap.java */
public class RunEncap{

   public static void main(String args[]){
      EncapTest encap = new EncapTest();
      encap.setName("James");
      encap.setAge(20);
      encap.setIdNum("12343ms");

      System.out.print("Name : " + encap.getName() + " Age : " + encap.getAge());
    }
}
```

This would produce the following result: -- Name : James Age : 20

27. Prepare for one java program to write on the board

```
//Palindrome String
package assignment_1.txt;
import java.util.Scanner;
public class PalindromeString {

	public static void main(String[] args) {
		System.out.println("Enter the String : ");
		Scanner sc =new Scanner(System.in);
		String  str = sc.nextLine();
		System.out.print("your Original String : "+str);
	   String result="";
	for (int i=str.length()-1; i>=0; i--) {
	result = result + str.charAt(i);
	 }

		System.out.println("\nReverse String  : " +result );
		if(str.equals(result)){
			System.out.println("Palindrome");  }
			else {
				System.out.println("not palindrome ");
			}

		sc.close();

	}

}
```

28. What is Constructor Overloading?

Constructor in Java

Java constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object that is why it is known as constructor.

Rules for creating java constructor

a. There are basically two rules defined for the constructor.
b. Constructor name must be same as its class name
c. Constructor must have no explicit return type

Types of java constructors -- There are two types of constructors:

Default constructor (no-arg constructor)
Parameterized constructor

Java Default Constructor

Default constructor (no-arg constructor)
A constructor that have no parameter is known as default constructor.

```
class Bike1{
Bike1()  {
System.out.println("Bike is created");    }   -- output -- Bike is created
public static void main(String args[])  {
Bike1 b=new Bike1();    }
}
```

Java parameterized constructor
A constructor that have parameters is known as parameterized constructor.
Example of parameterized constructor
In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
class Student4{
    int id;
    String name;

    Student4(int i,String n){
    id = i;
    name = n;
    }
    void display(){System.out.println(id+" "+name);}
       public static void main(String args[]){
    Student4 s1 = new Student4(111,"Karan");
    Student4 s2 = new Student4(222,"Aryan");
    s1.display();
    s2.display();
   }
}
```

Output :
111 Karan
222 Aryan

Constructor Overloading in Java
Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists.The compiler differentiates

these constructors by taking into account the number of parameters in the list and their type.

Example of Constructor Overloading

```
class Student5{
    int id;
    String name;
    int age;
    Student5(int i,String n){
    id = i;
    name = n;
    }
    Student5(int i,String n,int a){
    id = i;
    name = n;
    age=a;
    }
    void display(){System.out.println(id+" "+name+" "+age);}

    public static void main(String args[]){
    Student5 s1 = new Student5(111,"Karan");
    Student5 s2 = new Student5(222,"Aryan",25);
    s1.display();
    s2.display();
   }
}
```

Output: 111 Karan 0 and 222 Aryan 25

Difference between constructor and method in java

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

29.Without using sync key word how do you perform synchronization ?

What is Synchronization in Java
Synchronization in Java is an important concept since Java is a multi-threaded language where multiple threads run in parallel to complete program execution. In multi-threaded environment synchronization of Java object or synchronization of Java class becomes extremely important. Synchronization in Java is possible by using Java keywords "synchronized" and "volatile”.

Concurrent access of shared objects in Java introduces to kind of errors: thread interference and memory consistency errors and to avoid these errors you need to properly synchronize your Java object to allow mutual exclusive access of critical section to two threads.

Why do we need Synchronization in Java?
If your code is executing in a multi-threaded environment, you need synchronization for objects, which are shared among multiple threads, to avoid any corruption of state or any kind of unexpected behavior. Synchronization in Java will only be needed if shared object is mutable. if your shared object is either read-only or immutable object, then you don't need synchronization, despite running multiple threads. Same is true with what threads are doing with an object if all the threads are only reading value then you don't require synchronization in Java. JVM guarantees that Java synchronized code will only be executed by one thread at a time.

Functionality of synchronized keyword in Java ..

1) The synchronized keyword in Java provides locking, which ensures mutually exclusive access to the shared resource and prevents data race.

2) synchronized keyword also prevent reordering of code statement by the compiler which can cause a subtle concurrent issue if we don't use synchronized or volatile keyword.

3) synchronized keyword involve locking and unlocking. before entering into synchronized method or block thread needs to acquire the lock, at this point it reads data from main memory than cache and when it release the lock, it flushes write operation into main memory which eliminates memory inconsistency errors.

30. What is Super keyword ? when and where do you use it ?

super keyword in java
The super keyword in java is a reference variable that is used to refer immediate parent class object.

Usage of java super Keyword

a. super is used to refer immediate parent class instance variable.
b. super() is used to invoke immediate parent class constructor.
c. super is used to invoke immediate parent class method.

example -- 1 --Super is used to refer immediate parent class instance variable.

```
class Vehicle{
  int speed=50;
}
class Bike4 extends Vehicle{
  int speed=100;
  void display()    {
   System.out.println(super.speed);//will print speed of Vehicle now
  }
  public static void main(String args[]){
   Bike4 b=new Bike4();
   b.display();
}
}
```

Example - 2 --Super is used to invoke parent class constructor.

```
class Vehicle{
  Vehicle(){System.out.println("Vehicle is created");}
```

```
}

class Bike5 extends Vehicle{
  Bike5(){
   super();//will invoke parent class constructor
   System.out.println("Bike is created");
  }
  public static void main(String args[]){
   Bike5 b=new Bike5();
}
}
```