Basic Java Questions for Freshers

Q1. Explain JDK, JRE and JVM?

JDK	JRE	JVM
It stands for Java Development Kit.	It stands for Java Runtime Environment.	It stands for Java Virtual Machine.
It is the tool necessary to compile, document and package Java programs.	JRE refers to a runtime environment in which Java bytecode can be executed.	It is an abstract machine. It is a specification that provides a run-time environment in which Java bytecode can be executed.
It contains JRE + development tools.	It's an implementation of the JVM which physically exists.	JVM follows three notations: Specification, **Implementation,** and **Runtime Instance**.

Q2. Explain public static void main(String args[]) in Java.

main() in Java is the entry point for any Java program. It is always written as **public static void main(String[] args)**.

- **public**: Public is an access modifier, which is used to specify who can access this method. Public means that this Method will be accessible by any Class.
- **static**: It is a keyword in java which identifies it is class-based. main() is made static in Java so that it can be accessed without creating the instance of a Class. In case, main is not made static then the compiler will throw an error as **main**() is called by the JVM before any objects are made and only static methods can be directly invoked via the class.
- **void**: It is the return type of the method. Void defines the method which will not return any value.
- **main**: It is the name of the method which is searched by JVM as a starting point for an application with a particular signature only. It is the method where the main execution occurs.
- **String args[]**: It is the parameter passed to the main method.

Q3. Why Java is platform independent?

Java is called platform independent because of its byte codes which can run on any system irrespective of its underlying operating system.

Q4. Why Java is not 100% Object-oriented?

Java is not 100% Object-oriented because it makes use of eight primitive data types such as Boolean, byte, char, int, float, double, long, short which are not objects.

Q5. What are wrapper classes in Java?

Wrapper classes convert the Java primitives into the reference types (objects). Every primitive data type has a class dedicated to it. These are known as wrapper classes because they "wrap" the primitive data type into an object of that class. Refer to the below image which displays different primitive type, wrapper class and constructor argument.

Q6. What are constructors in Java?
In Java, constructor refers to a block of code which is used to initialize an object. It must have the same name as that of the class. Also, it has no return type and it is automatically called when an object is created.

There are two types of constructors:

1. **Default Constructor:** In Java, a default constructor is the one which does not take any inputs. In other words, default constructors are the no argument constructors which will be created by default in case you no other constructor is defined by the user. Its main purpose is to initialize the instance variables with the default values. Also, it is majorly used for object creation.
2. **Parameterized Constructor:** The parameterized constructor in Java, is the constructor which is capable of initializing the instance variables with the provided values. In other words, the constructors which take the arguments are called parameterized constructors.

Q7. What is singleton class in Java and how can we make a class singleton?
Singleton class is a class whose only one instance can be created at any given time, in one JVM. A class can be made singleton by making its constructor private.

Q8. What is the difference between Array list and vector in Java?

Array list	**vector**
Array List is not synchronized.	Vector is synchronized.
Array List is fast as it's non-synchronized.	Vector is slow as it is thread safe.
If an element is inserted into the Array List, it increases its Array size by 50%.	Vector defaults to doubling size of its array.
Array List does not define the increment size.	Vector defines the increment size.
Array List can only use Iterator for traversing an Array List.	Vector can use both Enumeration and Iterator for traversing.

Q9. What is the difference between equals() and == in Java?
Equals() method is defined in Object class in Java and used for checking equality of two objects defined by business logic.

"==" or equality operator in Java is a binary operator provided by Java programming language and used to compare primitives and objects. *public boolean equals(Object o)* is the method provided by the Object class. The default implementation uses == operator to compare two objects. For example: method can be overridden like String class. equals() method is used to compare the values of two objects.

Q10. What are the differences between Heap and Stack Memory in Java?
The major difference between Heap and Stack memory are:

Features	**Stack**	**Heap**
Memory	Stack memory is used only by one thread of execution.	Heap memory is used by all the parts of the application.
Access	Stack memory can't be accessed by other threads.	Objects stored in the heap are globally accessible.

Memory Management	Follows LIFO manner to free memory.	Memory management is based on the generation associated with each object.
Lifetime	Exists until the end of execution of the thread.	Heap memory lives from the start till the end of application execution.
Usage	Stack memory only contains local primitive and reference variables to objects in heap space.	Whenever an object is created, it's always stored in the Heap space.

Q11. What is a package in Java? List down various advantages of packages.
Packages in Java, are the collection of related classes and interfaces which are bundled together. By using packages, developers can easily modularize the code and optimize its reuse. Also, the code within the packages can be imported by other classes and reused. Below I have listed down a few of its advantages:

- Packages help in avoiding name clashes
- They provide easier access control on the code
- Packages can also contain hidden classes which are not visible to the outer classes and only used within the package
- Creates a proper hierarchical structure which makes it easier to locate the related classes

Q12. Why pointers are not used in Java?
Java doesn't use pointers because they are unsafe and increases the complexity of the program. Since, Java is known for its simplicity of code, adding the concept of pointers will be contradicting. Moreover, since JVM is responsible for implicit memory allocation, thus in order to avoid direct access to memory by the user, pointers are discouraged in Java.

Q13. What is JIT compiler in Java?
JIT stands for Just-In-Time compiler in Java. It is a program that helps in converting the Java bytecode into instructions that are sent directly to the processor. By default, the JIT compiler is enabled in Java and is activated whenever a Java method is invoked. The JIT compiler then compiles the bytecode of the invoked method into native machine code, compiling it "just in time" to execute. Once the method has been compiled, the JVM summons the compiled code of that method directly rather than interpreting it. This is why it is often responsible for the performance optimization of Java applications at the run time.

Q14. What are access modifiers in Java?
In Java, access modifiers are special keywords which are used to restrict the access of a class, constructor, data member and method in another class. Java supports four types of access modifiers:

1. *Default*
2. *Private*
3. *Protected*
4. *Public*

Modifier	**Default**	**Private**	**Protected**	**Public**
Same class	YES	YES	YES	YES
Same Package subclass	YES	NO	YES	YES
Same Package non-subclass	YES	NO	YES	YES

Different package subclass	NO	NO	YES	YES
Different package non-subclass	NO	NO	NO	YES

Q15. Define a Java Class.
A class in Java is a blueprint which includes all your data. A class contains fields (variables) and methods to describe the behavior of an object. Let's have a look at the syntax of a class.

```
class Demo
{
        member variables
        methods

}
```

Q16. What is an object in Java and how is it created?
An object is a real-world entity that has a state and behavior. An object has three characteristics:

1. State
2. Behavior

An object is created using the 'new' keyword. For example:

```
ClassName obj = new ClassName();
```

Q17. What is Object Oriented Programming?
Object-oriented programming or popularly known as OOPs is a programming model or approach where the programs are organized around objects rather than logic and functions. In other words, OOP mainly focuses on the objects that are required to be manipulated instead of logic. This approach is ideal for the programs large and complex codes and needs to be actively updated or maintained.

Q18. What are the main concepts of OOPs in Java?
Object-Oriented Programming or OOPs is a programming style that is associated with concepts like:

1. *Inheritance:* Inheritance is a process where one class acquires the properties of another.
2. *Encapsulation:* Encapsulation in Java is a mechanism of wrapping up the data and code together as a single unit.
3. *Abstraction:* Abstraction is the methodology of hiding the implementation details from the user and only providing the functionality to the users.
4. *Polymorphism:* Polymorphism is the ability of a variable, function or object to take multiple forms.

Q19. What is the difference between a local variable and an instance variable?
In Java, a **local variable** is typically used inside a method, constructor, or a **block** and has only local scope. Thus, this variable can be used only within the scope of a block. The best benefit of having a local variable is that other methods in the class won't be even aware of that variable.

Example

```
if(x > 100)
```

```
{
String test = "Destination Technologies";
}
```

Whereas, an **instance variable** in Java, is a variable which is bounded to its object itself. These variables are declared within a **class**, but outside a method. Every object of that class will create it's own copy of the variable while using it. Thus, any changes made to the variable won't reflect in any other instances of that class and will be bound to that particular instance only.

```
class Test
{
	public String EmpName;
	public int empAge;
}
```

Q20. Differentiate between the constructors and methods in Java?

Methods	Constructors
1. Used to represent the behavior of an object	1. Used to initialize the state of an object
2. Must have a return type	2. Do not have any return type
3. Needs to be invoked explicitly	3. Is invoked implicitly
4. No default method is provided by the compiler	4. A default constructor is provided by the compiler if the class has none
5. Method name may or may not be same as class name	5. Constructor name must always be the same as the class name

In case you are facing any challenges with these Java interview questions, please comment on your problems in the section below.

Q21. What is final keyword in Java?
final is a special keyword in Java that is used as a non-access modifier. A final variable can be used in different contexts such as:

- **final variable**

When the final keyword is used with a variable then its value can't be changed once assigned. In case the no value has been assigned to the final variable then using only the class constructor a value can be assigned to it.

- **final method**

When a method is declared final then it can't be overridden by the inheriting class.

- **final class**

When a class is declared as final in Java, it can't be extended by any subclass class but it can extend other class.

Q22. What is the difference between break and continue statements?

break	continue
1. Can be used in switch and loop (for, while, do while) statements	1. Can be only used with loop statements
2. It causes the switch or loop statements to terminate the moment it is executed	2. It doesn't terminate the loop but causes the loop to jump to the next iteration
3. It terminates the innermost enclosing loop or switch immediately	3. A continue within a loop nested with a switch will cause the next loop iteration to execute

Q23.What is an infinite loop in Java? Explain with an example.
An infinite loop is an instruction sequence in Java that loops endlessly when a functional exit isn't met. This type of loop can be the result of a programming error or may also be a deliberate action based on the application behavior. An infinite loop will terminate automatically once the application exits.

For example:

```
public class InfiniteForLoopDemo
{
public static void main(String[] arg) {
for(;;)
System.out.println("Welcome to Destination Technologies!");
// To terminate this program press ctrl + c in the console.
}
}
```

Q24. What is the difference between this() and super() in Java?
In Java, super() and this(), both are special keywords that are used to call the constructor.

this()	super()
1. this() represents the current instance of a class	1. super() represents the current instance of a parent/base class
2. Used to call the default constructor of the same class	2. Used to call the default constructor of the parent/base class
3. Used to access methods of the current class	3. Used to access methods of the base class
4. Used for pointing the current class instance	4. Used for pointing the superclass instance
5. Must be the first line of a block	5. Must be the first line of a block

Q25. What is Java String Pool?
Java String pool refers to a collection of Strings which are stored in heap memory. In this, whenever a new object is created, String pool first checks whether the object is already present in the pool or not. If it is present, then the same reference is returned to the variable else new object will be created in the String pool and the respective reference will be returned.

Q26. Differentiate between static and non-static methods in Java.

Static Method	Non-Static Method

1. *The static* keyword must be used before the method name	1. No need to use the *static* keyword before the method name
2. It is called using the class (className.methodName)	2. It is can be called like any general method
3. They can't access any non-static instance variables or methods	3. It can access any static method and any static variable without creating an instance of the class

Q27. What is constructor chaining in Java?
In Java, constructor chaining is the process of calling one constructor from another with respect to the current object. Constructor chaining is possible only through legacy where a subclass constructor is responsible for invoking the superclass' constructor first. There could be any number of classes in the constructor chain. Constructor chaining can be achieved in two ways:

1. Within the same class using this()
2. From base class using super()

Q28. Difference between String, StringBuilder, and StringBuffer.

Factor	String	StringBuilder	StringBuffer
Storage Area	Constant String Pool	Heap Area	Heap Area
Mutability	Immutable	Mutable	Mutable
Thread Safety	Yes	No	Yes
Performance	Fast	More efficient	Less efficient

If you think this article on Java Interview Questions is helpful, you can check out Destination Technologies's Java Training in Chennai as well.

Q29. What is a classloader in Java?
The **Java ClassLoader** is a subset of JVM (Java Virtual Machine) that is responsible for loading the class files. Whenever a Java program is executed it is first loaded by the classloader. Java provides three built-in classloaders:

1. Bootstrap ClassLoader
2. Extension ClassLoader
3. System/Application ClassLoader

Q30. Why Java Strings are immutable in nature?
In Java, string objects are immutable in nature which simply means once the String object is created its state cannot be modified. Whenever you try to update the value of that object instead of updating the values of that particular object, Java creates a new string object. Java String objects are immutable as String objects are generally cached in the String pool. Since String literals are usually shared between multiple clients, action from one client might affect the rest. It enhances security, caching, synchronization, and performance of the application.

Q31. What is the difference between an array and an array list?

Array	Array List
Cannot contain values of different data types	Can contain values of different data types.
Size must be defined at the time of	Size can be dynamically changed

declaration	
Need to specify the index in order to add data	No need to specify the index
Arrays are not type parameterized	Arraylists are type
Arrays can contain primitive data types as well as objects	Arraylists can contain only objects, no primitive data types are allowed

Q32. What is a Map in Java?
In Java, Map is an interface of Util package which maps unique keys to values. The Map interface is not a subset of the main Collection interface and thus it behaves little different from the other collection types. Below are a few of the characteristics of Map interface:

1. Map doesn't contain duplicate keys.
2. Each key can map at max one value.

Q33. What is collection class in Java? List down its methods and interfaces.
In Java, the collection is a framework that acts as an architecture for storing and manipulating a group of objects. Using Collections you can perform various tasks like searching, sorting, insertion, manipulation, deletion, etc. Java collection framework includes the following:

- Interfaces
- Classes
- Methods

In case you are facing any challenges with these java interview questions, please comment on your problems in the section below.

OOPS Java Questions

1. What is meant by the term OOPs?

OOPs refers to Object-Oriented Programming. It is the programming paradigm that is defined using objects. Objects can be considered as real-world instances of entities like class, that have some characteristics and behaviours.

2. What is the need for OOPs?

There are many reasons why OOPs is mostly preferred, but the most important among them are:

- OOPs helps users to understand the software easily, although they don't know the actual implementation.
- With OOPs, the readability, understandability, and maintainability of the code increase multifold.
- Even very big software can be easily written and managed easily using OOPs.

3. What are some major Object Oriented Programming languages?

The programming languages that use and follow the Object-Oriented Programming paradigm or OOPs, are known as Object-Oriented Programming languages. Some of the major Object-Oriented Programming languages include:

- Java
- C++
- Javascript
- Python
- PHP

5. What is meant by Structured Programming?

Structured Programming refers to the method of programming which consists of a completely structured control flow. Here structure refers to a block, which contains a set of rules, and has a definitive control flow, such as (if/then/else), (while and for), block structures, and subroutines.

Nearly all programming paradigms include Structured programming, including the OOPs model.

6. What are some advantages of using OOPs?

- OOPs is very helpful in solving very complex level of problems.
- Highly complex programs can be created, handled, and maintained easily using object-oriented programming.
- OOPs, promote code reuse, thereby reducing redundancy.
- OOPs also helps to hide the unnecessary details with the help of Data Abstraction.
- OOPs, are based on a bottom-up approach, unlike the Structural programming paradigm, which uses a top-down approach.
- Polymorphism offers a lot of flexibility in OOPs.

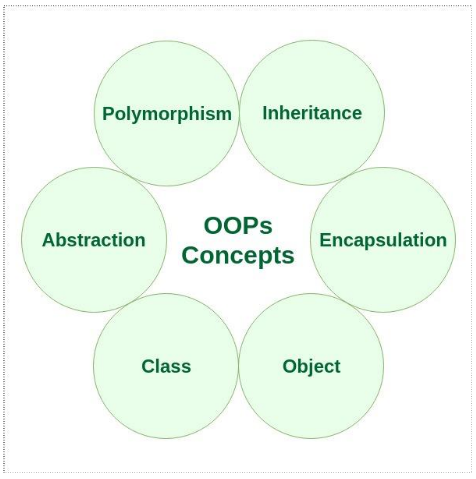
7. Why is OOPs so popular?

OOPs programming paradigm is considered as a better style of programming. Not only it helps in writing a complex piece of code easily, but it also allows users to handle and maintain them easily as well. Not only that, the main pillar of OOPs - Data Abstraction, Encapsulation, Inheritance, and Polymorphism, makes it easy for programmers to solve complex scenarios. As a result of these, OOPs is so popular.

8. What are the main features of OOPs?

OOPs or Object Oriented Programming mainly comprises of the below four features, and make sure you don't miss any of these:

- Inheritance
- Encapsulation
- Polymorphism
- Abstraction
- Class
- Object



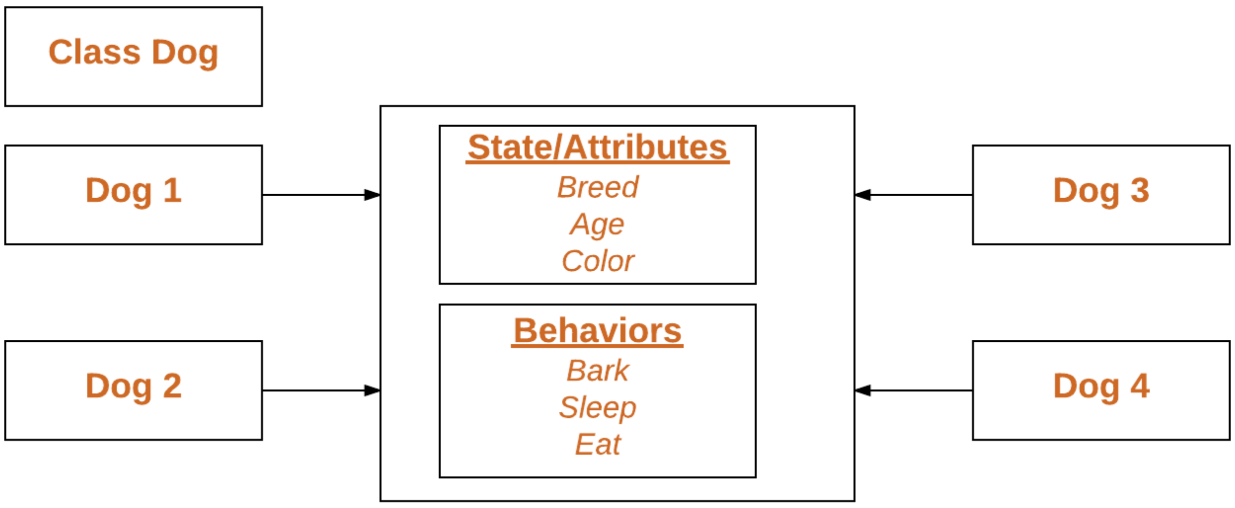
9. What is a class?

A class can be understood as a template or a blueprint, which contains some values, known as member data or member, and some set of rules, known as behaviors or functions. So when an object is created, it automatically takes the data and functions that are defined in the class.
Therefore the class is basically a template or blueprint for objects. Also one can create as many objects as they want based on a class.

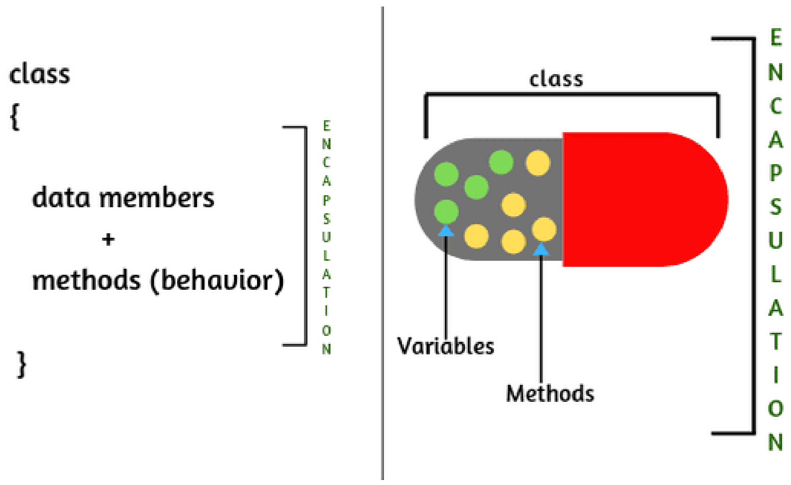
For example, first, a car's template is created. Then multiple units of car are created based on that template.

10. What is an object?

An object refers to the instance of the class, which contains the instance of the members and behaviours defined in the class template. In the real world, an object is an actual entity to which a user interacts, whereas class is just the blueprint for that object. So the objects consume space and have some characteristic behaviour.



11. What is encapsulation?



One can visualize Encapsulation as the method of putting everything that is required to do the job, inside a capsule and presenting that capsule to the user. What it means is that by Encapsulation, all the necessary data and methods are bind together and all the unnecessary details are hidden to the normal user. So Encapsulation is the process of binding data members and methods of a program together to do a specific job, without revealing unnecessary details.

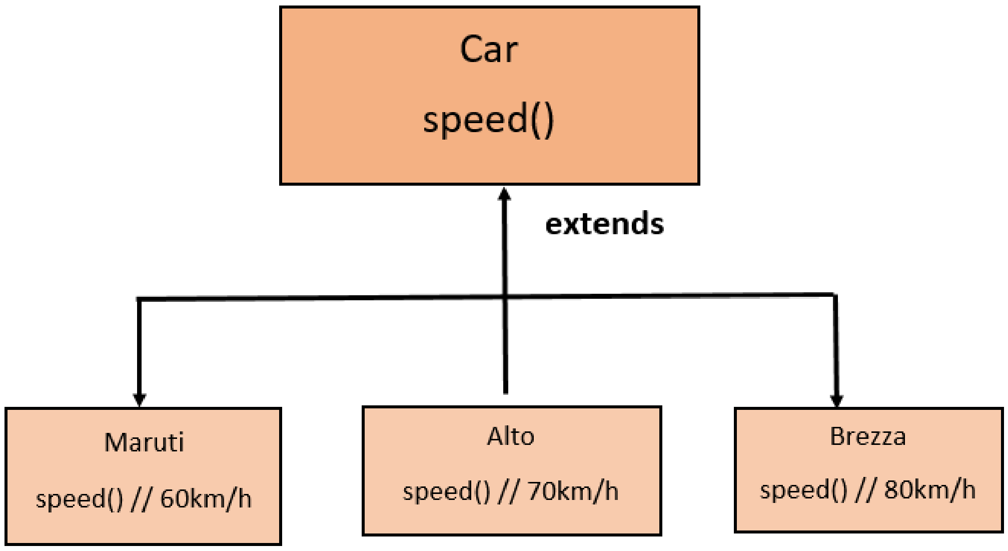
Encapsulation can also be defined in two different ways:

1) **Data hiding:** Encapsulation is the process of hiding unwanted information, such as restricting access to any member of an object.

2) **Data binding:** Encapsulation is the process of binding the data members and the methods together as a whole, as a class.

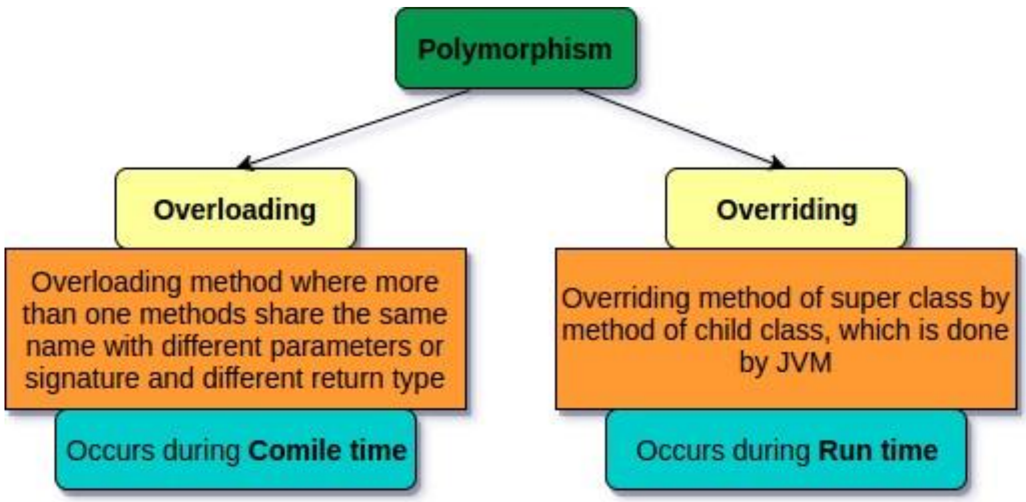
12. What is Polymorphism?

Polymorphism is composed of two words - "poly" which means "many", and "morph" which means "shapes". Therefore Polymorphism refers to something that has many shapes.



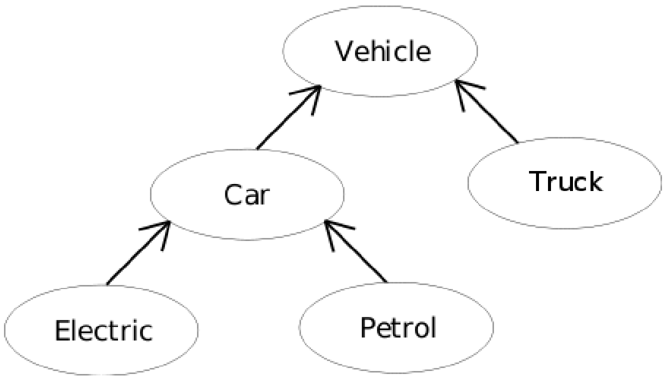
In OOPs, Polymorphism refers to the process by which some code, data, method, or object behaves differently under different circumstances or contexts. Compile-time polymorphism and Run time polymorphism are the two types of polymorphisms in OOPs languages.

13. What are the different types of Polymorphism ?

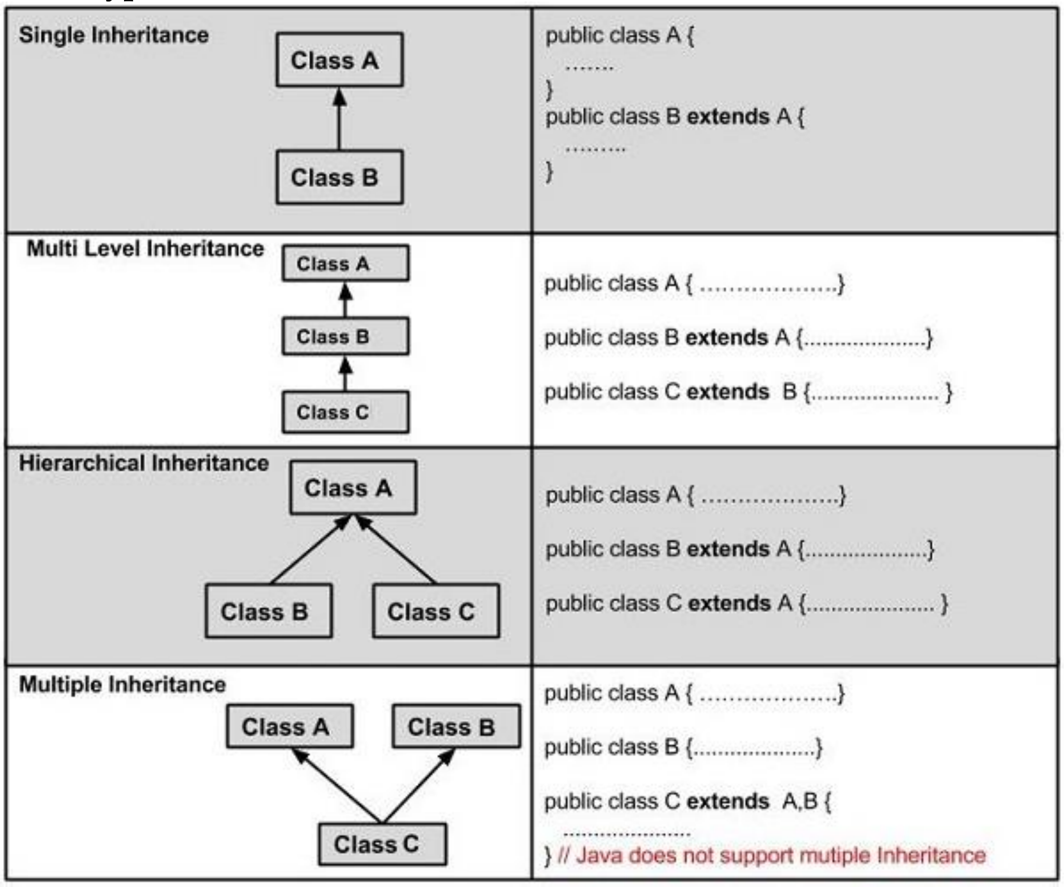

Polymorphism
Overloading
Overloading method where more than one methods share the same name with different parameters or signature and different return type
Occurs during Comile time
Overriding
Overriding method of super class by method of child class, which is done by JVM
Occurs during Run time

15. What is meant by Inheritance?

The term "inheritance" means "receiving some quality or behavior from a parent to an offspring." In object-oriented programming, inheritance is the mechanism by which an object or class (referred to as a child) is created using the definition of another object or class (referred to as a parent). Inheritance not only helps to keep the implementation simpler but also helps to facilitate code reuse.



16.What are the types of Inheritance?



```
public class A {
  .......
}
public class B extends A {
  .........
}
```

```
public class A { ...................}

public class B extends A {....................}

public class C extends  B {..................... }
```

```
public class A { ...................}

public class B extends A {....................}

public class C extends A {..................... }
```

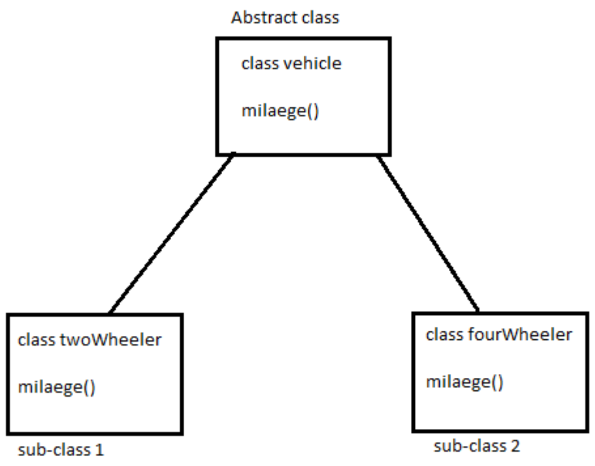
```
public class A { ...................}

public class B {....................}

public class C extends  A,B {
  ......................
} // Java does not support mutiple Inheritance
```

17. What is Abstraction?

If you are a user, and you have a problem statement, you don't want to know how the components of the software work, or how it's made. You only want to know how the software solves your problem. Abstraction is the method of hiding unnecessary details from the necessary ones. It is one of the main features of OOPs.
For example, consider a car. You only need to know how to run a car, and not how the wires are connected inside it. This is obtained using Abstraction.



18. How much memory does a class occupy?

Classes do not consume any memory. They are just a blueprint based on which objects are created. Now when objects are created, they actually initialize the class members and methods and therefore consume memory.

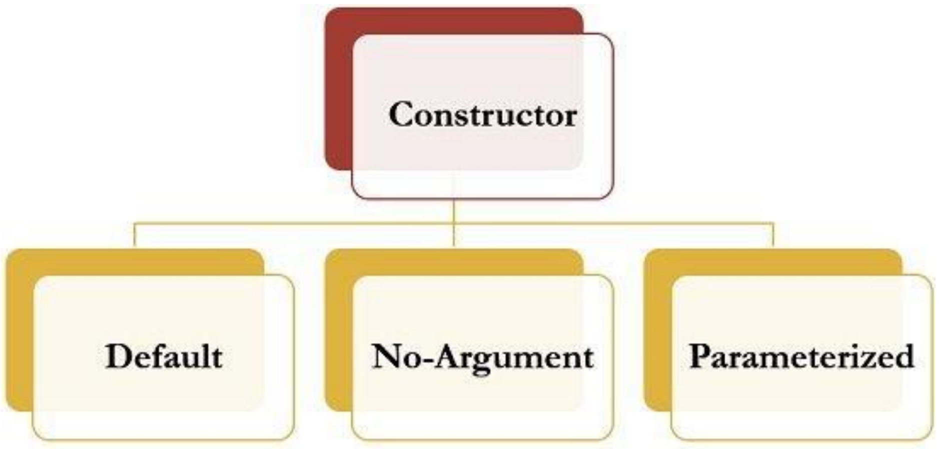
19. Is it always necessary to create objects from class?

No. An object is necessary to be created if the base class has non-static methods. But if the class has static methods, then objects don't need to be created. You can call the class method directly in this case, using the class name.

20. What is a constructor?

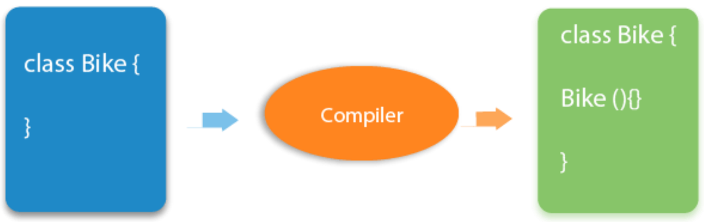
Constructors are special methods whose name is the same as the class name. The constructors serve the special purpose of initializing the objects.

21.What are the types of constructors?



Q) What is the purpose of a default constructor?

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.



Q)Why use the parameterized constructor?

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

Q) What is the use of No-Argument or Zero- parameterized constructor?

A zero parameter (or "no-arg") constructor is one that requires no arguments. A class can have multiple constructors, each with different arguments. Constructors without arguments are particularly helpful for tools that use reflection to populate an object's properties.

22. Are class and structure the same? If not, what's the difference between a class and a structure?

No, class and structure are not the same. Though they appear to be similar, they have differences that make them apart. For example, the structure is saved in the stack memory, whereas the class is saved in the heap memory. Also, Data Abstraction cannot be achieved with the help of structure, but with class, Abstraction is majorly used.

23. What is a subclass?

The subclass is a part of Inheritance. The subclass is an entity, which inherits from another class. It is also known as the child class.

24. Define a superclass?

Superclass is also a part of Inheritance. The superclass is an entity, which allows subclasses or child classes to inherit from itself.

25. What is an interface?

An interface refers to a special type of class, which contains methods, but not their definition. Only the declaration of methods is allowed inside an interface. To use an interface, you cannot create objects. Instead, you need to implement that interface and define the methods for their implementation.

26. What is meant by static polymorphism?

Static Polymorphism is commonly known as the Compile time polymorphism. Static polymorphism is the feature by which an object is linked with the respective function or operator based on the values during the compile time. Static or Compile time Polymorphism can be achieved through Method overloading or operator overloading.

27. What is meant by dynamic polymorphism?

Dynamic Polymorphism or Runtime polymorphism refers to the type of Polymorphism in OOPs, by which the actual implementation of the function is decided during the runtime or execution. The dynamic or runtime polymorphism can be achieved with the help of method overriding.

28. What is the difference between overloading and overriding?

Overloading is a compile-time polymorphism feature in which an entity has multiple implementations with the same name. For example, Method overloading and Operator overloading.

Whereas Overriding is a runtime polymorphism feature in which an entity has the same name, but its implementation changes during execution. For example, Method overriding.

29. How is data abstraction accomplished?

Data abstraction is accomplished with the help of abstract methods or abstract classes.

30. What is an abstract class?

An abstract class is a special class containing abstract methods. The significance of abstract class is that the abstract methods inside it are not implemented and only declared. So as a result, when a subclass inherits the abstract class and needs to use its abstract methods, they need to define and implement them.

31. How is an abstract class different from an interface?

Interface and abstract class both are special types of classes that contain only the methods declaration and not their implementation. But the interface is entirely different from an abstract class. The main difference between the two is that, when an interface is implemented, the subclass must define all its methods and provide its implementation. Whereas when an abstract class is inherited, the subclass does not need to provide the definition of its abstract method, until and unless the subclass is using it.

36. What are access specifiers and what is their significance?

Access specifiers, as the name suggests, are a special type of keywords, which are used to control or specify the accessibility of entities like classes, methods, etc. Some of the access specifiers or access modifiers include “private”, “public”, etc. These access specifiers also play a very vital role in achieving Encapsulation - one of the major features of OOPs.

37. What is an exception?

An exception can be considered as a special event, which is raised during the execution of a program at runtime, that brings the execution to a halt. The reason for the exception is mainly due to a position in the program, where the user wants to do something for which the program is not specified, like undesirable input.

38. What is meant by exception handling?

No one wants its software to fail or crash. Exceptions are the major reason for software failure. The exceptions can be handled in the program beforehand and prevent the execution from stopping. This is known as exception handling.
So exception handling is the mechanism for identifying the undesirable states that the program can reach and specifying the desirable outcomes of such states.
Try-catch is the most common method used for handling exceptions in the program.

39. What is meant by Garbage Collection in OOPs world?

Object-oriented programming revolves around entities like objects. Each object consumes memory and there can be multiple objects of a class. So if these objects and their memories are not handled properly, then it might lead to certain memory-related errors and the system might fail.

Garbage collection refers to this mechanism of handling the memory in the program. Through garbage collection, the unwanted memory is freed up by removing the objects that are no longer needed.

40. Can we run a Java application without implementing the OOPs concept?

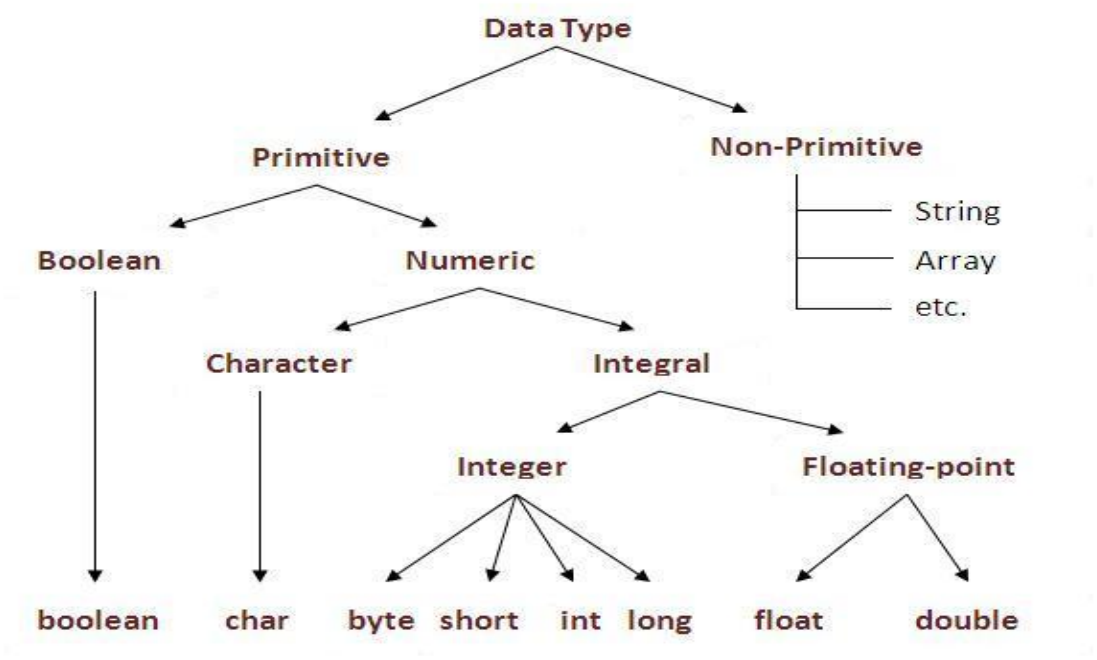
No. Java applications are based on Object-oriented programming models or OOPs concept, and hence they cannot be implemented without it.

Other Important Points

What is Data types in Java?

Data types represent the different values to be stored in the variable. In java, there are two types of data types:

- Primitive data types
- Non-primitive data types



What is static and non static in java?

Static Variables:
- A static variable is associated with the class has only one copy per class but not for each object. An instance of a class does not have static variables.

- Static variables can be accessed by static or instance methods

- Memory is allocated when the class is loaded in context area at run time.

Non-Static Variables:
- Non-static variables will have one copy each per object. Each instance of a class will have one copy of non-static variables.

- Instance variables can be accessed only by the instance methods.

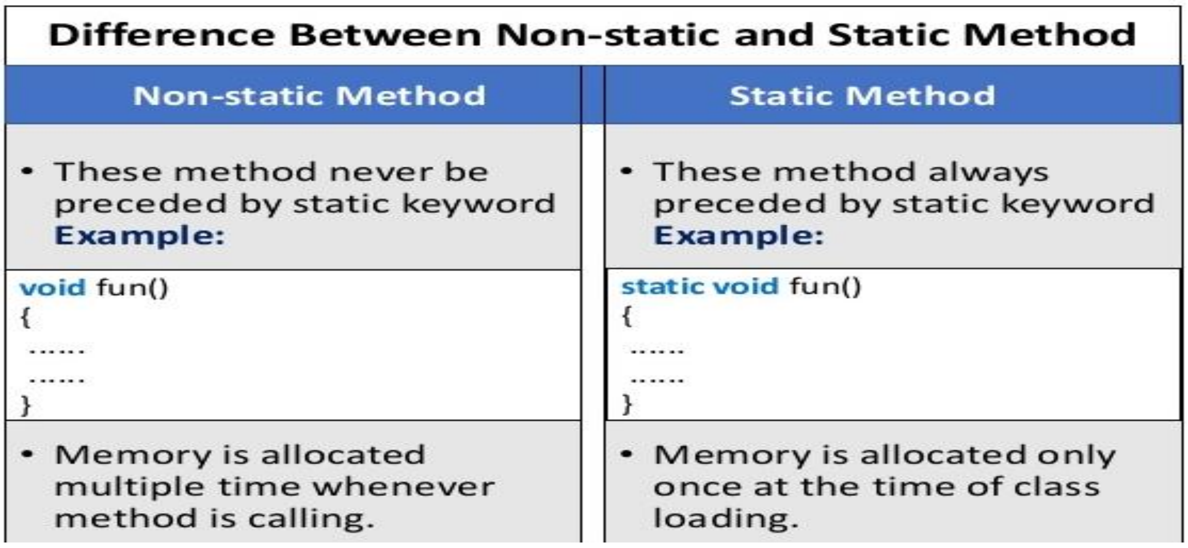
- Instance variables are allocated at compile time.

Static blocks:

1. A static block is a block of code which contains code that executes at class loading time.
2. A static keyword is prefixed before the start of the block.
3. All static variables can be accessed freely
4. Any non-static fields can only be accessed through object reference,thus only after object construction.
5. multiple static blocks would execute in the order they are defined in the class.
6. All static blocks executes only once per classloader
7. A typical static block looks like

Non-static block:

1. A non-static block executes when the object is created, before the constructor
2. There is no keyword prefix to make a block non-static block,unlike static blocks.
3. Incase of multiple non-static blocks , the block executes the order in which it is defined in the class.
4. All static and non-static fields can be access freely.
5. All non-static block executes everytime an object of the containing class is created.
6. A typical non-static block looks like below



Difference Between Non-static and Static Method

Non-static Method	Static Method
• These method never be preceded by static keyword **Example:**	• These method always preceded by static keyword **Example:**

void fun()
{
......
......
}
	static void fun()
	{
	......
	......
	}

• Memory is allocated multiple time whenever method is calling.	• Memory is allocated only once at the time of class loading.

Java Command Line Arguments.

Command Line Arguments in Java: Important Points

- Command Line Arguments can be used to specify configuration information while launching your application.
- There is no restriction on the number of java command line arguments.You can specify any number of arguments
- Information is passed as Strings.
- They are captured into the String args of your main method

Example of command-line argument.

```
Class A
{
    public static void main(String args[])
    {
    for (int i=0;i<args.length;i++)
   System.out.println (args[i]);
   }
```

```
Output:
     sonoo
      Jaiswal
      1
      3
      abc
```

```
    }
Now take output from compiler:
    Compile by > javac A.java
    Run by > java A sonoo Jaiswal 1 3 abc
```

Operators in java

Operator in java is a symbol that is used to perform operations. For example: +, -, *, / etc.

There are many types of operators in java which are given below:

- Unary Operator,
- Arithmetic Operator,
- shift Operator,
- Relational Operator,
- Bitwise Operator,
- Logical Operator,
- Ternary Operator and
- Assignment Operator.

Arithmetic operators

Arithmetic operators are used in mathematical expression in the same way that are used in algebra.

Operator	Description
`+`	adds two operands
`-`	subtract second operands from first
`*`	multiply two operand
`/`	divide numerator by denumerator
`%`	remainder of division
`++`	Increment operator increases integer value by one
`--`	Decrement operator decreases integer value by one

Relation operators

The following table shows all relation operators supported by Java.

Operator	Description
`==`	Check if two operand are equal

`!=`	Check if two operand are not equal.
`>`	Check if operand on the left is greater than operand on the right
`<`	Check operand on the left is smaller than right operand
`>=`	check left operand is greater than or equal to right operand
`<=`	Check if operand on left is smaller than or equal to right operand

Logical operators

Java supports following 3 logical operator. Suppose a=1 and b=0;

Operator	**Description**	**Example**
`&&`	Logical AND	(a && b) is false
`||`	Logical OR	(a || b) is true
`!`	Logical NOT	(!a) is false

Bitwise operators

Java defines several bitwise operators that can be applied to the integer types long, int, short, char and byte

Operator	**Description**
`&`	Bitwise AND
`|`	Bitwise OR
`^`	Bitwise exclusive OR
`>>`	left shift
`<<`	right shift

Now lets see truth table for bitwise `&`, `|` and `^`

a	b	a & b	a | b	a ^ b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

The bitwise shift operators shifts the bit value. The left operand specifies the value to be shifted and the right operand specifies the number of positions that the bits in the value are to be shifted. Both operands have the same precedence. **Example**

```
a = 0001000
b = 2
a << b = 0100000
a >> b = 0000010
```

Assignment Operators

Assignment operator supported by Java are as follows:

Operator	**Description**	**Example**
`=`	assigns values from right side operands to left side operand	`a = b`
`+=`	adds right operand to the left operand and assign the result to left	`a+=b` is same as `a=a+b`
`-=`	subtracts right operand from the left operand and assign the result to left operand	`a-=b` is same as a`=a-b`
`*=`	mutiply left operand with the right operand and assign the result to left operand	`a*=b` is same as `a=a*b`
`/=`	divides left operand with the right operand and assign the result to left operand	`a/=b` is same as `a=a/b`
`%=`	calculate modulus using two operands and assign the result to left operand	`a%=b` is same as `a=a%b`

Misc operator

There are few other operator supported by java language.

Conditional operator

It is also known as ternary operator and used to evaluate Boolean expression,

```
epr1 ? expr2 : expr3
```

If `epr1`Condition is true? Then value `expr2` : Otherwise value `expr3`

`instanceOf` operator

This operator is used for object reference variables. The operator checks whether the object is of particular type (class type or interface type)

What is Assertion?

Assertion is a statement in java. It can be used to test your assumptions about the program.

While executing assertion, it is believed to be true. If it fails, JVM will throw an error named Assertion Error. It is mainly used for testing purpose.

```
import java.util.Scanner;
class AssertionExample{
 public static void main( String args[] ){
  Scanner scanner = new Scanner( System.in );
  System.out.print("Enter ur age ");
  int value = scanner.nextInt();
  assert value>=18:" Not valid";
  System.out.println("value is "+value);
 }
}
```

Output:

```
Enter ur age 11
Exception in thread "main" java.lang.AssertionError: Not valid
```

WRITE SOME PROGRAM TO BITWISE OPERATER?

```
public class Test {

   public static void main(String args[]) {

      int a = 60;           /* 60 = 0011 1100 */

      int b = 13;           /* 13 = 0000 1101 */

      int c = 0;

      c = a & b;       /* 12 = 0000 1100 */

      System.out.println("a & b = " + c );

      c = a | b;       /* 61 = 0011 1101 */

      System.out.println("a | b = " + c );

      c = a ^ b;       /* 49 = 0011 0001 */

      System.out.println("a ^ b = " + c );

      c = ~a;          /*-61 = 1100 0011 */

      System.out.println("~a = " + c );

      c = a << 2;      /* 240 = 1111 0000 */

      System.out.println("a << 2 = " + c );

      c = a >> 2;      /* 15 = 1111 */

      System.out.println("a >> 2  = " + c );
      c = a >>> 2;     /* 15 = 0000 1111 */

      System.out.println("a >>> 2 = " + c );

   }

}
```

Output

```
a & b = 12
a | b = 61
a ^ b = 49
~a = -61
a << 2 = 240
a >> 2  = 15
a >>> 2 = 15
```

Java Unary Operator Example: ++ and --

```
class OperatorExample{
public static void main(String args[]){
int x=10;
System.out.println(x++);//10 (11)
System.out.println(++x);//12
System.out.println(x--);//12 (11)
System.out.println(--x);//10
}}
```

Output:

```
10
12
12
10
```

Java Unary Operator Example: ~ and !

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=-10;
boolean c=true;
boolean d=false;
System.out.println(~a);
System.out.println(~b);
System.out.println(!c);
System.out.println(!d);
}}
```

Output:

```
-11
9
false
true
```

Java Shift Operator Example: Left Shift

```
class OperatorExample{
public static void main(String args[]){
System.out.println(10<<2);//10*2^2=10*4=40
System.out.println(10<<3);//10*2^3=10*8=80
System.out.println(20<<2);//20*2^2=20*4=80
System.out.println(15<<4);//15*2^4=15*16=240
}}
```

Output:

```
40
```

Java Unary Operator Example 2: ++ and --

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=10;
System.out.println(a++ + ++a);//10+12=22
System.out.println(b++ + b++);//10+11=21

}}
```

Output:

```
22
21
```

Java Arithmetic Operator Example

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
System.out.println(a+b);//15
System.out.println(a-b);//5
System.out.println(a*b);//50
System.out.println(a/b);//2
System.out.println(a%b);//0
}}
```

Output:

```
15
5
50
2
0
```

```
80
80
240
```

Java Shift Operator Example: Right Shift

```
class OperatorExample{
public static void main(String args[]){
System.out.println(10>>2);//10/2^2=10/4=2
System.out.println(20>>2);//20/2^2=20/4=5
System.out.println(20>>3);//20/2^3=20/8=2
}}
```

Output:

```
2
5
2
```

Java AND Operator Example: Logical && and Bitwise &

The logical && operator doesn't check second condition if first condition is false. It checks second condition only if first one is true.

The bitwise & operator always checks both conditions whether first condition is true or false.

```
class OperatorExample{
public static void main(String args[])
{
int a=10;
int b=5;
int c=20;
System.out.println(a<b&&a<c);
System.out.println(a<b&a<c);//
}}
```

Output:

```
false
false
```

Java AND Operator Example: Logical && vs Bitwise &

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int c=20;
System.out.println(a<b&&a++<c);//false && true = false
System.out.println(a);//10 because second condition is not checked
System.out.println(a<b&a++<c);//false && true = false
System.out.println(a);//11 because second condition is checked
}}
```

Output:

```
false
10
false
11
```

Java OR Operator Example: Logical || and Bitwise |

The logical || operator doesn't check second condition if first condition is true. It checks second condition only if first one is false.

The bitwise | operator always checks both conditions whether first condition is true or false.

```
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int c=20;
System.out.println(a>b||a<c);//true || true = true
System.out.println(a>b|a<c);//true | true = true
//|| vs |
System.out.println(a>b||a++<c);//true || true = true
System.out.println(a);//10 because second condition is not checked
System.out.println(a>b|a++<c);//true | true = true
System.out.println(a);//11 because second condition is checked
}}
```

Output:

```
true
true
```

```
true
10
true
11
```

Java Ternary Operator Example

```java
class OperatorExample{
public static void main(String args[]){
int a=2;
int b=5;
int min=(a<b)?a:b;
System.out.println(min);
}}
```

Output:

```
2
```

Java Assignment Operator Example

```java
class OperatorExample{
public static void main(String args[]){
int a=10;
int b=20;
a+=4;//a=a+4 (a=10+4)
b-=4;//b=b-4 (b=20-4)
System.out.println(a);
System.out.println(b);
}}
```

Output:

```
14
16
```

Variables and Data Types in Java

Variable is a name of memory location. There are three types of variables in java: local, instance and static.

There are two types of data types in java: primitive and non-primitive.

Variable

Variable is name of *reserved area allocated in memory*. In other words, it is a *name of memory location*. It is a combination of "vary + able" that means its value can be changed.

```java
int data=50;//Here data is variable
```

Types of Variable

There are three types of variables in java:

- local variable
- instance variable
- static variable

Java Assignment Operator Example

```java
class OperatorExample{
public static void main(String[] args){
int a=10;
a+=3;//10+3
System.out.println(a);
a-=4;//13-4
System.out.println(a);
a*=2;//9*2
System.out.println(a);
a/=2;//18/2
System.out.println(a);
}}
```

Output:

```
13
9
18
9
```

Java Assignment Operator Example: Adding short

```java
class OperatorExample{
public static void main(String args[]){
short a=10;
short b=10;
//a+=b;//a=a+b internally so fine
a=a+b;//Compile time error because 10+10=20 now int

System.out.println(a);
}}
```

Output:

```
Compile time error
```

After type cast:

```java
class OperatorExample{
public static void main(String args[]){
short a=10;
short b=10;
a=(short)(a+b);//20 which is int now converted to short
 System.out.println(a);
}}
```

Output:

```
20
```

1) Local Variable

A variable which is declared inside the method is called local variable.

2) Instance Variable

A variable which is declared inside the class but outside the method, is called instance variable . It is not declared as static.

3) Static variable

A variable that is declared as static is called static variable. It cannot be local.

Q) What is the scope of the variable in Java?
ANS)
Scope and Lifetime of **Variables**. The **scope** of a **variable** defines the section of the code in which the **variable** is visible. As a general rule, **variables** that are defined within a block are not accessible outside that block. The lifetime of a **variable** refers to how long the **variable** exists before it is destroyed.

Explain Access Modifiers in java?

There are two types of modifiers in java: **access modifiers** and **non-access modifiers**.The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class.

There are 4 types of java access modifiers:

1. private
2. default
3. protected
4. public

There are many non-access modifiers such as static, abstract, synchronized, native, volatile, transient etc. Here, we will learn access modifiers.

Default:
When no access modifier is specified for a class, method or data member – It is said to be having the **default** access modifier by default.

- The data members, class or methods which are not declared using any access modifiers i.e. having default access modifier are accessible **only within the same package**.

Private:
The private access modifier is specified using the keyword **private**.

- The methods or data members declared as private are accessible only **within the class** in which they are declared.
- Any other **class of same package will not be able to access** these members.
- Classes or interface cannot be declared as private.

Protected:
The protected access modifier is specified using the keyword **protected**.

- The methods or data members declared as protected are **accessible within same package or sub classes in different package.**

Public:
The public access modifier is specified using the keyword **public**.

- The public access modifier has the **widest scope** among all other access modifiers.
- Classes, methods or data members which are declared as public are **accessible from everywhere** in the program. There is no restriction on the scope of a public data members.

Q) What is Constructors in Java?

Constructor in java is a *special type of method* that is used to initialize the object.
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.

When is a Constructor called?

Each time an object is created using **new ()** keyword at least one constructor (it could be default constructor) is invoked to assign initial values to the **data members** of the same class.

Rules for creating java constructor

There are basically two rules defined for the constructor.

1. Constructor name must be same as its class name
2. Constructor must have no explicit return type

Types of java constructors

<u>Default constructor (no-arg constructor)</u>

A constructor that have no parameter is known as default constructor.

<u>Parameterized constructor</u>

Default constructor provides the default values to the object like 0, null etc. depending on the type.

Java Copy Constructor

There is no copy constructor in java. But, we can copy the values of one object to another like copy constructor in C++. There are many ways to copy the values of one object into another in java. They are:

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

How constructors are different from methods in Java?

- Constructor(s) must have the same name as the class within which it defined while it is not necessary for the method in java.
- Constructor(s) do not any return type while method(s) have the return type or **void** if does not return any value.
- Constructor is called only once at the time of Object creation while method(s) can be called any numbers of time.

Constructor Chaining in Java with Examples

Constructor chaining is the process of calling one constructor from another constructor with respect to current object.

Constructor chaining can be done in two ways:

- **<u>Within same class</u>**: It can be done using **this()** keyword for constructors in same class
- **<u>From base class</u>:** by using **super ()** keyword to call constructor from the base class.

```
public class MyChaining {

    public MyChaining(){
        System.out.println ("In default constructor...");
    }
    public MyChaining(int i){
        this();
        System.out.println ("In single parameter constructor...");
    }
    public MyChaining(int i,int j){
        this(j);
        System.out.println("In double parameter constructor...");
    }

  public static void main(String a[]){
        MyChaining ch = new MyChaining(10,20);
    }
}
```

Example Output

```
In default constructor...
In single parameter constructor...
In double parameter constructor...
```

Question: What is the difference between “==” and equals() method?
The == (double equals) returns true, if the variable reference points to the same object in memory. This is called “shallow comparison”.

The equals () method calls the user implemented equals () method, which compares the object attribute values. The equals() method provides “deep comparison” by checking if two objects are logically equal as opposed to the shallow comparison provided by the operator ==.

If equals () method does not exist in a user supplied class then the inherited Object class's equals () method will be called which evaluates if the references point to the same object in memory. In this case, the object. Equals () works just like the "==" operator.

This Keyword In Java

There can be a lot of usage of **java this keyword**. In java, this is a **reference variable** that refers to the current object.

Usage of java this keyword

Here is given the 6 usage of java this keyword.

1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this can be used to return the current class instance from the method.

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.

Important points related to Constructor overloading:
1. Constructor overloading is similar to method overloading in Java.

2. You can call overloaded constructor by using this() keyword in Java.

3. Overloaded constructor must be called from another constructor only.

4. make sure you add no argument default constructor because once compiler will not add if you have added any constructor in Java.

5. if an overloaded constructor called , it must be the first statement of constructor in java.

6. Its best practice to have one primary constructor and let overloaded constructor calls that. this way

Example of constructor overloading:

```
public class Student
{
  public Student()                         // I ,  default constructor
  {
    System.out.println("Hello 1");
  }
  public Student(String name)              // II,  parameterized constructor with single parameter
  {
    System.out.println("Student name is " + name);
  }

  public Student(String name, int marks)   // III,  parameterized constructor with two parameters
  {
    System.out.println("Student name is " + name + " and marks are " + marks);
  }

  public static void main(String args[])
  {
    Student std1 = new Student();          // calls I
    Student std2 = new Student("Mr.Reddy");       // calls II
    Student std3 = new Student("Mr.Raju", 56);    // calls III
  }
}
```

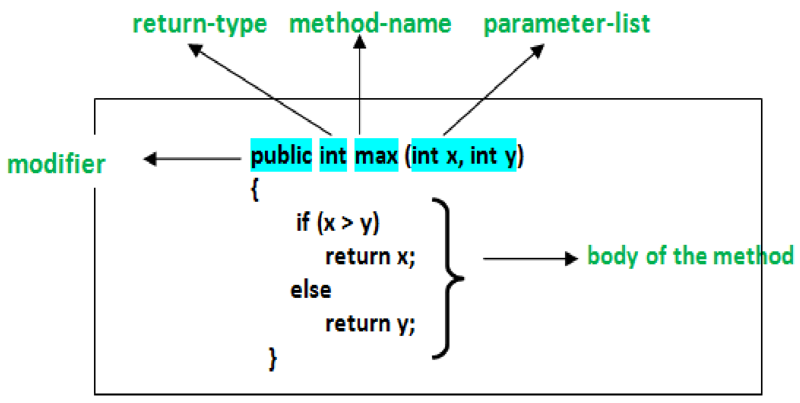
Java - Methods

A Java method is a collection of statements that are grouped together to perform an operation. When you call the System.out.**println()** method, for example, the system actually executes several statements in order to display a message on the console.

Creating Method

Considering the following example to explain the syntax of a method −

- **public static** − modifier
- **int** − return type
- **methodName** − name of the method
- **a, b** − formal parameters
- **Int x, int y** − list of parameters



```
public int max (int x, int y)
{
    if (x > y)
        return x;
    else
        return y;
}
```

<u>Calling a method</u>

The method needs to be called for using its functionality. There can be three situations when a method is called:

A method returns to the code that invoked it when:

- It completes all the statements in the method
- It reaches a return statement
- Throws an exception

Types Of Method In Java?

1). Static methods: A static method is a method that can be called and executed without creating an object. In general, static methods are used to create instance methods.

Static method can be invoked directly via class name i.e.; we don't have to create an object for a class in order to initiate static method.

2. Instance methods: These methods act upon the instance variables of a class. Instance methods are being invoked with below syntax

Method Overloading in Java

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

Advantage of method overloading

Method overloading *increases the readability of the program*.

Different ways to overload the method

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

Q) Why Method Overloading is not possible by changing the return type of method only?

In java, method overloading is not possible by changing the return type of the method only because of ambiguity. Let's see how ambiguity may occur:

Can we overload java main() method?

Yes, by method overloading. You can have any number of main methods in a class by method overloading. But JVM calls main() method which receives string array as arguments only. L

Example of Method Overloading with ambiguity

If there are no matching type arguments in the method, and each method promotes similar number of arguments, there will be ambiguity.

```java
class OverloadingCalculation3{
 void sum(int a,long b){System.out.println("a method invoked");}
 void sum(long a,int b){System.out.println("b method invoked");}
 public static void main(String args[]){
 OverloadingCalculation3 obj=new OverloadingCalculation3();
 obj.sum(20,20);//now ambiguity
  }
}
```

Output: Compile Time Error

Call by Value and Call by Reference in Java

There is only call by value in java, not call by reference. If we call a method passing a value, it is known as call by value. The changes being done in the called method, is not affected in the calling method.
In case of call by value original value is not changed. Let's take a simple example

Example of call by value in java

```java
class Operation{
 int data=50;
   void change(int data){
 data=data+100;//changes will be in the local variable only
  }
    public static void main(String args[]){
   Operation op=new Operation();
   System.out.println("before change "+op.data);
   op.change(500);
   System.out.println("after change "+op.data);
 }  }
```

```
Output:
Before change
50
after change 50
```

Call By Reference

In case of call by reference original value is changed if we made changes in the called method. If we pass object in place of any primitive value, original value will be changed. In this example we are passing object as a value. Let's take a simple example:

Example of call by reference in java

```java
class Operation2
{
 int data=50;
 void change(Operation2 op)
{
 op.data=op.data+100;//changes will be in the instance variable
  }
 public static void main(String args[])
{
   Operation2 op=new Operation2();
   System.out.println("before change "+op.data);
   op.change(op);//passing object
   System.out.println("after change "+op.data);
 }
}
```

```
Output:
Before change 50
After change 150
(in java there is no call by reference)
```

Inheritance in Java

Inheritance in java is a mechanism in which one object acquires all the properties and behaviours of parent object.

Why use inheritance in java

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

Types of inheritance in java

Three types of inheritance in java:

1) Single inheritance

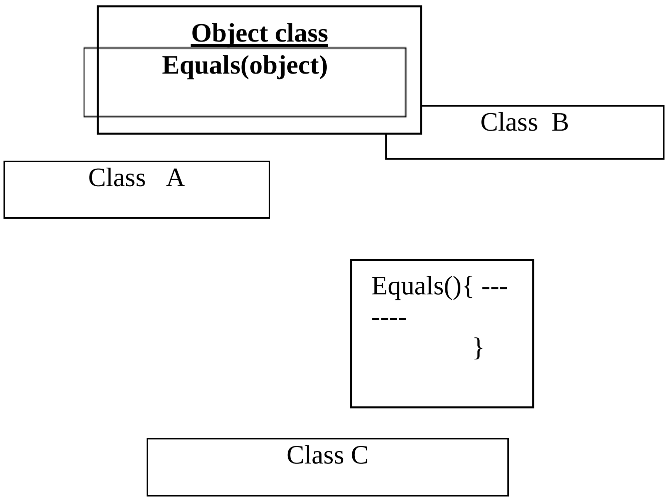
   A derived class is created from a single base class. ...

2) Multilevel (not supported in java).

Q) Why multiple inheritance is not supported in java?

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Consider a scenario where A, B and C are three classes. The C class inherits A and B classes. If A and B classes have same method and you call it from child class object, there will be ambiguity to call method of A or B class.



```
class A{
void mgs(){
System.out.println ("Hello");

}
class B
{
void mgs()
{
System.out.println ("Welcome");
}
}
class C extends A,B{//suppose if it were
 Public Static void main (String args [])
{
   C obj=new C ();
   obj.msg ();    //Now which msg() method would be invoked?
}
}
```

Compile Time Error

NOTE: when a class extends to super class if both super class have same behaviour but different implementation then there is ambiguity to subclass to inherit implementation .This ambiguity as diamond problem.so there is no solution for diamond problem multiple inheritance is not supported in java…..

x

Q) What is Singleton Class ?
A class which allows maximum of one instance thought out the execution is known as singeleton class.
Q) When we need singleton class:
We need singleton class when there is only one instance of class is created in java virtual machine
Q) How to make singleton class ?
1) create private constructor
2)create private static variable of same class type.
3) create a public static method which create new instance and return for 1st call return same instance for next call.
Program Of Singleton Class?

```
Public class A
{
```

```
Private static A obje;
Private A()
{
}
Public static A getInstance ();
    {
 if (obj==null) obj=new A();
Return obj;
     }
}
```

Difference between final, finally and finalize?

Final :

- 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 is a keyword.
- Final variable value can't be changed

Finally:

- Finally Is Used To Place Important Code,
- It Will Be Executed Whether Exception Is Handled Or Not.
- Finally Is A Block.

Finalize

- Finalize is used to perform clean up processing just before object is garbage collected.
- Finalize is a method.

Exception Handling in Java

- The **exception handling in java** is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained.
- Exception is an event that disrupts the normal flow of the program.
- It is an object which is thrown at runtime.

Types of Exception

There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception. The sun microsystem says there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception
3. Error

Difference between checked and unchecked exceptions

1) Checked Exception

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

2) Unchecked Exception

The classes that extend RuntimeException are known as unchecked exceptions

e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc.

Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

3) Error

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

Java Exception Handling Keywords

There are 5 keywords used in java exception handling.

1. try
2. catch
3. finally
4. throw
5. throws

Java try block

Java try block is used to enclose the code that might throw an exception. It must be used within the method.

Java try block must be followed by either catch or finally block.

Java catch block

Java catch block is used to handle the Exception. It must be used after the try block only.

You can use multiple catch block with a single try.

Exception handling flow chart

What is the difference between throw, throws and Throwable in java?

Throw is a keyword in java which is used to throw an exception manually.

- Using throw keyword, you can throw an exception from any method or block.
- But, that exception must be of type **java.lang.Throwable** class or it's sub classes.

Throws is also a keyword in java which is used in the method signature to indicate that this method may throw mentioned exceptions.

- The caller to such methods must handle the mentioned exceptions either using try-catch blocks or using throws keyword..

Throwable is a super class for all types of errors and exceptions in java.

- This class is a member of **java.lang**package.
- Only instances of this class or it's sub classes are thrown by the java virtual machine by the throw statement.
- The only argument of catch block must be of this type or it's sub classes.
- If you want to create your own customized exceptions, then your class must extend this class.

27. **Inheritance** :-
i. The process of acquiring properties from one class to another class is known as inheritance.
ii. Keyword-extends.
iii. The class from which properties are acquired is known as superclass nd to which properties are acquired is known as subclass.

ADVANTAGES:-

Reusability- we can use the public methods of super class without rewriting the same.
Overriding- with the help of inheritance, we can override the methods of super class in sub class.
2 types of inheritance -1.single inheritance 2. Multiple inheritance.

PROGRAM:-

```
Class phone{
        Void call(){
                Sop(“call connected”);
                }
        Void sendsms(){
                Sop(“sms sent”);
                }
Class cameraphone extends phone{
        Void takephoto(){
```

```
                    Sop("photo captured");
                    }
}
Class test{
          Psvm(string[] args){
                    cameraphone cp=new cameraphone();
                    cp.call();
                    cp.sendsms();
                    cp.takephoto();
          }
}
```

31. **BINDING**:-
Connecting a method call to a method implementation is called as binding.
2types- 1.early binding   2.late binding
PROGRAM:-

```
class x
{
          static int m1()
          {
                    return 10;
          }
          int m2()
          {
                    return 20;
          }
}
class y extends x
{
          static int m1()
          {
                    return 30;
          }
          int m2()
          {
                    return 40;
          }
}
public class Binding {

          public static void main(String[] args) {
                    x x1=new x();
                    System.out.println(x.m1());//EARLY BINDING
                    System.out.println(x1.m2());//LATE BIMDING
                    x2=new y();
                    System.out.println(x.m1());
                    System.out.println(x1.m2());
          }

}
```

32. **OBJECT CLASS**:-
i. Object class is a built in class in java, it is the super most class in java.
ii. A class created in java application is directly or in-directly sub class to object class. If there is class with out extending any super class than by default that class extends to object class.
33. **Methods of object class:-**
i. public String toString():- this method returns string representation. When an object is printed using println method it implicitly make a call to to string method.
ii. public boolean equals(object arg)
iii. public int hashcode()
iv. public void wait(long arg)
v. public void wait(long arg, int arg)
vi. public void wait()
vii. public void notify()
viii. public void notifyAll()
ix. public class getclass()
x. protected object clone()
xi. protected void finalize()

34. **POLYMORPHISM**:-

i. One object or one method call shows different behavior is known as polymorphism.
ii. 2types- 1.Compiletime polymorphism 2.Runtime polymorphism
call to overloaded method is resolved at compile time is known as compiletime.
Call to overriding method is resolved at runtime is known as runtime.
35. **ABSTRACTION:-**
i. Hiding the complex implementation and provide required functionality to the user is known as abstraction.
ii. It is achieved by interface. For user only interface details are provided.
iii. Service provider provides implementation for all implementation methods in implementation calss.
iv. At runtime based on the instance of implementation class program will be executed.
36. **Loose coupling**:-
It is a concept in which both service provider and service user are connected in such a way that ,any new implementation is added in implementation layer is not affected for service user program.
Tight coupling:-
It is a concept in which both service provider and service user are connected in such a way that ,any new implementation is added in implementation layer is affected for service user program.
38. **Abstract class:-**
A class which contains at least one abstract method is known as abstract class. Abstract class are declared with the abstract modifier.
Abstract class can not be instantiated.
When a class is extends a abstract class all the abstract methods of super class must be over ridden in subclass. Otherwise sub class becomes abstract class.
Interface:-
It acts like a bridge between 2 java programs.
In interface all the methods by default public and abstract and all variables are static and final. Variables must be initialized .
No constructor in interface ,it can not be instantiated.
39. String is a class used to store and manage string. Where String is nothing but group of character.
It provides many methods to perform string manipulation.
String is immutable because, in string pool string objects are shared with many program, changes made by one program may affect the other programs. To avoid this string is developed as immutable.
41. **Encapsulation:-**
Declaring a data member as private and provide a public method to access the variable is known as encapsulation.
The method which is used to set a private data member is known as setter.
The method which is used to read a private data member is known as getter.
42. **Comparable:-**
It is an interface which is used to compare 2 objects of same type.
It provides a method compareTo() with one argument of type Object.
It compares the current instance with given instance, and returns,
+ve no if current instance > given instance
-ve no if current instance < given instance
0 if both are equal. Comparable implementation is called as NATURAL ordering.

Comparator:-
It is an interface which is used to compare 2 objects of same type.
It provides a method compare with 2 arguments. It compares 1st argument with 2nd argument and returns
+ve number if 1st is bigger
-ve number if 1st is smaller
0 if both are equal.
Using comparator multiple implementation can be provided for one type.
Comparator implementation is also called as custom ordering.
44. == operator is used for both primitives and non-primitives
== operator compares the object according to object reference.
Equals method is used for only non-primitive types.
Equals method by default compares according to reference but it can override according to properties.
45. A class which allows maximum of one object through out the execution is known as singleton class.
47. **Finally:-**
Finally statements are executed whether exception is thrown or not, whether the exception is handled or not.
Only one finally can be written for a try block.
It is generally used to write statements such as closing database connection, assigning null values for costly objects.
Finalize:-
It is method of object class.
It is protected method .
It is called and executed by garbage collector before an object is removed.
Final:-
Final is used to apply restrictions on class, method and variable.
Final class can not be inherited, final method can not be overridden and final variable value can not be changed.
Final is a keyword.
48. **Custom Exception:-**

If you are creating your own exception that is known as custom exception or user-defined exception. Java custom exceptions are used to customize the exception according to user need. By the help of custom exception, you can have your own exception and message.

Program:-

```
        class TestCustomException1{

      static void validate(int age)throws InvalidAgeException{

        if(age<18)

      throw new InvalidAgeException("not valid");

       else

       System.out.println("welcome to vote");

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

               try{

        validate(13);

               }catch(Exception m){System.out.println("Exception occured: "+m);}

                 System.out.println("rest of the code...");

            }

          }
```

49.**Throw and Throws:-**

1) In simple words throw is used to throw an exception & throws is used to declare an exception.
2) throw is used in method implementation & throws is used in method signature.
3) using throw keyword we can throw only 1 exception at a time & throws can declare multiple exception at a time.
4**) throws is a keyword used to handle checked exceptions. throw is a keyword used to raise user defined exceptions.**
5) throw is internal keyword it throw the exception only once in try and catch block throws is throw the multiple exceptions

51. Difference between throw and throws

•throws clause is used to declare an exception and throw keyword is used to throw an exception explicitly.

•If we see syntax wise then throw is followed by an instance variable and throws is followed by exception class names.

•The keyword throw is used inside method body to invoke an exception and throws clause is used in method declaration (signature).

For ex:
throw
throw new Exception("You have some exception")
throw new IOException("Connection failed!!")
throws
public int myMethod() throws IOException, ArithmeticException, NullPointerException {}

•You cannot declare multiple exceptions with throw. You can declare multiple exception e.g. public void method()throws IOException,SQLException.

•checked exceptions can not be propagated with throw only because it is explicitly used to throw an particular exception. checked exception can be propagated with throws.

52. Flow Chart of collection

Collection Framework in java is a centralized and unified theme to store and manipulate the group of objects. Java Collection Framework provides some pre-defined classes and interfaces to handle the group of objects. Using collection framework, you can store the objects as a list or as a set or as a queue or as a map and perform operations like adding an object or removing an object or sorting the objects without much hard work.
The entire collection framework is divided into four interfaces.
1) List —> It handles sequential list of objects. **ArrayList**, **Vector** and **LinkedList** classes implement this interface.
2) Queue —> It handles special list of objects in which elements are removed only from the head. **LinkedList** and **PriorityQueue** classes implement this interface.
3) Set —> It handles list of objects which must contain unique element. This interface is implemented by **HashSet** and **LinkedHashSet** classes and extended by **SortedSet** interface which in turn, is implemented by **TreeSet**.
4) Map —> This is the one interface in Collection Framework which is not inherited from Collection interface. It handles group of objects as Key/Value pairs. It is implemented by **HashMap** and **HashTable** classes and extended by **SortedMap** interface which in turn is implemented by **TreeMap**.
Three of above interfaces (List, Queue and Set) inherit from Collection interface. Although, Map is included in collection framework it does not inherit from Collection interface.

Collection interface is the root level interface in the collection framework. List, Queue and Set are all sub interfaces of Collection interface. JDK does not provide any direct implementations of this interface. But, JDK provides direct implementations of its sub interfaces.
Collection interface extends Iterable interface which is a member of java.lang package. Iterable interface has only one method called iterator(). It returns an Iterator object, using that object you can iterate over the elements of Collection. Here is the class diagram of Collection interface.

53. Difference between Collection Interface and Collections class
Collections class is a utility class having static methods for doing operations on objects of classes which implement the Collection interface. It consists of only static methods which are used to operate on objects of type Collection.
For example, Collections has methods for finding the max element in a Collection, it has the method to sort the collection, it has the method to search for a particular element in a collection

The **Collection interface** defines methods common to structures which hold other objects. List and Set are sub interfaces of Collection, and ArrayList and HashSet are examples of concrete collections.

54. Difference between Arraylist, LinkedList and Vector
1. ArrayList:

ArrayList is implemented as a resizable array. It's elements can be accessed directly by using the get and set methods, since ArrayList is essentially an array.
2. Linked List:

LinkedList is implemented as a double linked list. Its performance on add and remove is better than Arraylist, but worse on get and set methods.
LinkedList, however, also implements Queue interface which adds more methods than ArrayList and Vector, such as offer(), peek(), poll(), etc.
3. Vector:

Vector is similar with ArrayList, but it is synchronized. Vector each time doubles its array size.

ARRAYLIST	**LINKEDLIST**	**VECTOR**
Contains resizable array.	Contains Linked list to store elements	Contains resizable array
Grows 50% of its size each time		Vector each time doubles its array size.
Add at end is fast and get is fast	Fast add and remove	Equal time to get and insert

	But slow get	
Not synchronized so not thread safe	Synchronized, so thread safe	Not synchronized so not thread safe

56. Difference between Hashmap and Hashtable

1. Hashtable is synchronized, whereas HashMap is not. This makes HashMap better for non-threaded applications, as unsynchronized Objects typically perform better than synchronized ones.

2. Hashtable does not allow null keys or values. HashMap allows one null key and any number of null values.

3. One of HashMap's subclasses is LinkedHashMap, so in the event that you'd want predictable iteration order (which is insertion order by default), you could easily swap out the HashMap for a LinkedHashMap. This wouldn't be as easy if you were using Hashtable.

4. HashMap does not guarantee that the order of the map will remain constant over time.

57. Internals of Hashmap

There are four things we should know about before going into the internals of how does **HashMap** work in Java -

- **HashMap** works on the principal of hashing.

- **Map.Entry interface** - This interface gives a map entry (key-value pair). HashMap in Java stores both key and value object, in bucket, as an object of Entry class which implements this nested interface Map.Entry.

- **hashCode()** - HashMap provides put(key, value) for **storing** and get(key) method for **retrieving** values from HashMap. When **put()** method is used to store (Key, Value) pair, HashMap implementation **calls hashcode** on Key object to calculate a hash that is used to find a bucket where Entry object will be stored.
  When **get()** method is used to retrieve value, again key object (passed with the get() method) is used to calculate a hash which is then used to find a bucket where that particular key is stored.

- **equals()** - equals() method is used to **compare objects for equality**. In case of HashMap key object is used for comparison, also using equals() method Map knows how to handle **hashing collision** (hashing collision means more than one key having the same hash value, thus assigned to the same bucket). In that case objects are stored in a linked list.
  Where **hashCode()** method helps in finding the bucket where that key is stored, **equals()**method helps in finding the right key as there may be more than one key-value pair stored in a single bucket.

Points to note -

- HashMap works on the principal of hashing.

- HashMap uses the hashCode() method to calculate a hash value. Hash value is calculated using the key object. This hash value is used to find the correct bucket where Entry object will be stored.

- HashMap uses the equals() method to find the correct key whose value is to be retrieved in case of get() and to find if that key already exists or not in case of put().

- Hashing collision means more than one key having the same hash value, in that case Entry objects are stored as a linked-list with in a same bucket.

- With in a bucket values are stored as Entry objects which contain both key and value.

- In Java 8 hash elements use balanced trees instead of linked lists after a certain threshold is reached while storing values. This improves the worst case performance from O(n) to O(log n).

58. Difference between Nested class and Inner class

Nested Class:
In java, it is possible to define a class within another class, such classes are known as *nested* classes. They enable you to logically group classes that are only used in one place, thus this increases the use of encapsulation, and create more readable and maintainable code.

- The scope of a nested class is bounded by the scope of its enclosing class. Thus in above example, class *NestedClass* does not exist independently of class *OuterClass*.

- A nested class has access to the members, including private members, of the class in which it is nested. However, reverse is not true i.e. the enclosing class does not have access to the members of the nested class.

- A nested class is also a member of its enclosing class.

- As a member of its enclosing class, a nested class can be declared *private*, *public*, *protected*, or *package private*(default).

- Nested classes are divided into two categories:

1. **static nested class :** Nested classes that are declared *static* are called static nested classes.

2. **inner class :** An inner class is a non-static nested class.

Inner Class:
An inner class is a class declared as a non-static member of another class. Inner classes are a security mechanism in Java. We know a class cannot be associated with the access modifier **private**, but if we have the class as a member of other class, then the inner class can be made private. And this is also used to access the private members of a class.
Inner classes are of three types depending on how and where you define them. They are −

- Inner Class
- Method-local Inner Class
- Anonymous Inner Class

59. Difference between iterator and list iterator.

ITERATOR	**LIST ITERATOR**
Iterator is used for traversing List and Set both.	ListIteratoris used to traverse List only but not set.
Traverse Only forward Direction	Traverse in both directions
Indexes cannot be obtained	Indexes can be obtained at any point of time. The methods nextIndex() and previousIndex() are used.
Cannot add element to collection while traversing it..	We can add element at any point of time.
Cannot replace the existing element value.	By using set(E e) method of ListIterator we can replace the last element returned by next() or previous() methods.
Methods of Iterator: • hasNext() • next() • remove()	Methods of ListIterator: • add(E e) • hasNext() • hasPrevious() • next() • nextIndex() • previous() • previousIndex() • remove() • set(E e)

	ListIterator	**Iterator**
Traversal Direction	Both , forward and backward	Forward
Objects traversal	List only	Map, Set and List
Add and Set operations	Allows both operations	Not possible
Iterator's current position	Yes , can be determined	Not possible.
Retrieve Index	Yes	Not possible

60.Anonymous Class
It is an inner class without a name and for which only a single object is created. An anonymous inner class can be useful when making an instance of an object with certain “extras” such as overloading methods of a class or interface, without having to actually subclass a class.
Anonymous inner classes are useful in writing implementation classes for listener interfaces in graphics programming.
Anonymous inner classes are mainly created in two ways:

- Class (may be abstract or concrete)

- Interface

```
//Java program to demonstrate Anonymous inner class
interface Age
{
    int x = 21;
    void getAge();
}
class AnonymousDemo
{
    public static void main(String[] args) {

        // Myclass is hidden inner class of Age interface
        // whose name is not written but an object to it
        // is created.
        Age oj1 = new Age() {
            @Override
            public void getAge() {
            System.out.print("Age is "+x);
            }
        };
        oj1.getAge();
    }
}
```

Difference between Normal/Regular class and Anonymous class:
A normal class can implement any number of interfaces but anonymous class can implement only one interface at a time.
A regular class can extend a class and implement any number of interface simultaneously. But anonymous class can extend a class or can implement an interface but not both at a time.
For regular/normal class, we can write any number of constructors but we cant write any constructor for anonymous class because anonymous class does not have any name and while defining constructor class name and constructor name must be same.

61. THREAD LIFE Cycle

A thread can be in one of the five states. According to sun, there is only 4 states in **thread life cycle in java** new, runnable, non-runnable and terminated. There is no running state.
But for better understanding the threads, we are explaining it in the 5 states.
The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:

1. New
2. Runnable
3. Running
4. Non-Runnable (Blocked)
5. Terminated

1) New
The thread is in new state if you create an instance of Thread class but before the invocation of start() method.
2) Runnable
The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.
3) Running
The thread is in running state if the thread scheduler has selected it.
4) Non-Runnable (Blocked)
This is the state when the thread is still alive, but is currently not eligible to run.
5) Terminated
A thread is in terminated or dead state when its run() method exits.

62. DEADLOCK

Deadlock in java is a part of multithreading. Deadlock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread. Since, both threads are waiting for each other to release the lock, the condition is called deadlock.

Example of Deadlock in java

```
public class TestDeadlockExample1 {
```

```
public static void main(String[] args) {
final String resource1 = "ratan jaiswal";
final String resource2 = "vimal jaiswal";
   // t1 tries to lock resource1 then resource2
   Thread t1 = new Thread() {
     public void run() {
         synchronized (resource1) {
          System.out.println("Thread 1: locked resource 1");

          try { Thread.sleep(100);} catch (Exception e) {}

          synchronized (resource2) {
           System.out.println("Thread 1: locked resource 2");
          }
        }
     }
   };

   // t2 tries to lock resource2 then resource1
   Thread t2 = new Thread() {
     public void run() {
       synchronized (resource2) {
         System.out.println("Thread 2: locked resource 2");

         try { Thread.sleep(100);} catch (Exception e) {}

         synchronized (resource1) {
           System.out.println("Thread 2: locked resource 1");
         }
       }
     }
   };
 t1.start();
   t2.start();
  }
}
```

64. Sorting and searching algorithims

-> SELECTION SORT

```
class SelectionSort
{
void sort(int arr[])
{
        int n = arr.length;
        // One by one move boundary of unsorted subarray
        for (int i = 0; i < n-1; i++)
        {
        // Find the minimum element in unsorted array
        int min_idx = i;
        for (int j = i+1; j < n; j++)
        if (arr[j] < arr[min_idx])
        min_idx = j;

        // Swap the found minimum element with the first
        // element
        int temp = arr[min_idx];
        arr[min_idx] = arr[i];
        arr[i] = temp;
        }
}

// Prints the array
void printArray(int arr[])
{
        int n = arr.length;
        for (int i=0; i<n; ++i)
        System.out.print(arr[i]+" ");
        System.out.println();
}

// Driver code to test above
public static void main(String args[])
{
        SelectionSort ob = new SelectionSort();
        int arr[] = {64,25,12,22,11};
        ob.sort(arr);
        System.out.println("Sorted array");
        ob.printArray(arr);
}
}
```

-> **BUBBLE SORT**

```
class BubbleSort
{
void bubbleSort(int arr[])
{
int n = arr.length;
for (int i = 0; i < n-1; i++)
for (int j = 0; j < n-i-1; j++)
if (arr[j] > arr[j+1])
{
// swap temp and arr[i]
int temp = arr[j];
arr[j] = arr[j+1];
arr[j+1] = temp;
}
}


/* Prints the array */
void printArray(int arr[])
{
int n = arr.length;
for (int i=0; i<n; ++i)
System.out.print(arr[i] + " ");
System.out.println();
}


// Driver method to test above
public static void main(String args[])
{
BubbleSort ob = new BubbleSort();
int arr[] = {64, 34, 25, 12, 22, 11, 90};
ob.bubbleSort(arr);
System.out.println("Sorted array");
ob.printArray(arr);
}
}
```

-> MERGE SORT

```
class MergeSort
{
// Merges two subarrays of arr[].
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
void merge(int arr[], int l, int m, int r)
{
// Find sizes of two subarrays to be merged
int n1 = m - l + 1;
int n2 = r - m;


/* Create temp arrays */
int L[] = new int [n1];
int R[] = new int [n2];


/*Copy data to temp arrays*/
for (int i=0; i<n1; ++i)
        L[i] = arr[l + i];
for (int j=0; j<n2; ++j)
        R[j] = arr[m + 1+ j];



/* Merge the temp arrays */


// Initial indexes of first and second subarrays
int i = 0, j = 0;


// Initial index of merged subarry array
int k = l;
while (i < n1 && j < n2)
{
        if (L[i] <= R[j])
        {
                arr[k] = L[i];
                i++;
        }
        else
        {
                arr[k] = R[j];
                j++;
        }
        k++;
}


/* Copy remaining elements of L[] if any */
while (i < n1)
{
        arr[k] = L[i];
        i++;
        k++;
}


/* Copy remaining elements of R[] if any */
while (j < n2)
{
        arr[k] = R[j];
        j++;
        k++;
}
}


// Main function that sorts arr[l..r] using
// merge()
void sort(int arr[], int l, int r)
{
if (l < r)
{
        // Find the middle point
        int m = (l+r)/2;


        // Sort first and second halves
        sort(arr, l, m);
        sort(arr , m+1, r);


        // Merge the sorted halves
        merge(arr, l, m, r);
}
}


/* A utility function to print array of size n */
static void printArray(int arr[])
{
int n = arr.length;
for (int i=0; i<n; ++i)
        System.out.print(arr[i] + " ");
System.out.println();
}


// Driver method
public static void main(String args[])
{
int arr[] = {12, 11, 13, 5, 6, 7};


System.out.println("Given Array");
printArray(arr);


MergeSort ob = new MergeSort();
ob.sort(arr, 0, arr.length-1);


System.out.println("\nSorted array");
printArray(arr);
}
}
```

-> QUICK SORT

```
public class Quicksort {
```

```
private int[] numbers;
private int number;
public void sort(int[] values) {
    this.numbers = values;
    number = values.length;
    quicksort(0, number - 1);
}
private void quicksort(int low, int high) {
    int i = low, j = high;
    // Get the pivot element from the middle of the list
    int pivot = numbers[(low + high) / 2];
    // Divide into two lists
    while (i <= j) {
        // If the current value from the left list is smaller then the pivot
        // element then get the next element from the left list
        while (numbers[i] < pivot)
        {
            i++;
        }
    // If the current value from the right list is larger then the pivot
    // element then get the next element from the right list
    while (numbers[j] > pivot) {
        j--;
    }


    // If we have found a values in the left list which is larger then
    // the pivot element and if we have found a value in the right list
    // which is smaller then the pivot elment then we exchange the
    // values.
    if (i <= j) {
        exchange(i, j);
        i++;
        j--;
    }
}
// Recursion
if (low < j)
    quicksort(low, j);
if (i < high)
    quicksort(i, high);
}


private void exchange(int i, int j) {
int temp = numbers[i];
numbers[i] = numbers[j];
numbers[j] = temp;
}
}
```

67. Difference between Serialization and deserialization

Serialization is a process of converting an object into a sequence of bytes which can be persisted to a disk or database or can be sent through streams. The reverse process of creating object from sequence of bytes is called **deserialization**.
A class must implement **Serializable** interface present in java.io package in order to serialize its object successfully. **Serializable** is a **marker interface** that adds serializable behaviour to the class implementing it.

Serializing an Object

```
import java.io.*;
class studentinfo implements Serializable
{
   String name;
   int rid;
   static String contact;
   studentinfo(string n, int r, string c)
   {
   this.name = n;
   this.rid = r;
   this.contact = c;
   }
}

class Test
{
   public static void main(String[] args)
   {
      try
      {
         Studentinfo si = new studentinfo("Abhi", 104, "110044");
         FileOutputStream fos = new FileOutputStream("student.ser");
         Objectoutputstream oos = new ObjectOutputStream(fos);
         oos.writeObject(si);
         oos.close();
         fos.close();
      }
      catch (Exception e)
      {
         e.printStackTrace();
      }
   }
}
```

Object of Studentinfo class is serialized using writeObject() method and written to student.ser file.

Serialization	**Deserialization**
Serialization is the process through which we can store the state of an object into any storage medium. We can store the state of the object into a file, into a database table etc.	Deserialization is the opposite process of serialization where we retrieve the object back from the storage medium.
An object is serialized by writing it an ObjectOutputStream.	An object is deserialized by reading it from an ObjectInputStream.

Example:-
 	
FileOutputStream out = new FileOutputStream("abc.txt");
ObjectOutputStream oos = new ObjectOutputStream(out);
oos.writeObject(new String ());
oos.close ();
	Example:-
	FileInputStream in = new FileInputStream("abc.txt");
	ObjectInputStream ois = new ObjectInputStream(in);
	String s = (String) ois.readObject();
	ois.close();

Deserialization of Object

```
import java.io * ;
class DeserializationTest
{
   public static void main(String[] args)
   {
      studentinfo si=null ;
      try
      {
         FileInputStream fis = new FileInputStream("student.ser");
         ObjectOutputStream ois = new ObjectOutputStream(fis);
         si = (studentinfo)ois.readObject();
      }
      catch (Exception e)
      {
         e.printStackTrace(); }
         System.out.println(si.name);
         System.out. println(si.rid);
         System.out.println(si.contact);
      }
}
```

68. Difference between anonymous class and lambda function

- Lambda Function

- Java lambda expressions are new in Java 8. A **Java lambda expression is thus a function which can be created without belonging to any class**. A lambda expression can be passed around as if it was an object or variable that is executed on demand.

- So basically when a method is assigned to a variable or an object that is known as a lambda function.

- syntax: Variable name() -> { codes };

•The key difference between Anonymous class and Lambda expression is the usage of '**this**' keyword. In the anonymous classes, 'this' keyword resolves to anonymous class itself, whereas for lambda expression 'this' keyword resolves to enclosing class where lambda expression is written.

•Another difference between lambda expression and anonymous class is in the way these two are compiled. Java compiler compiles lambda expressions and convert them into private method of the class.