1. ***What is object-oriented paradigm?***

It is a programming paradigm based on objects having data and methods defined in the class to which it belongs. Object-oriented paradigm aims to incorporate the advantages of modularity and reusability. Objects are the instances of classes which interacts with one another to design applications and programs. There are the following features of the object-oriented paradigm.

- Follows the bottom-up approach in program design.
- Focus on data with methods to operate upon the object's data
- Includes the concept like Encapsulation and abstraction which hides the complexities from the user and show only functionality.
- Implements the real-time approach like inheritance, abstraction, etc.
- The examples of the object-oriented paradigm are C++, Simula, Smalltalk, Python, C#, etc.

2. ***What is Java?***

Java is the high-level, object-oriented, robust, secure programming language, platform-independent, high performance, Multithreaded, and portable programming language. It was developed by **James Gosling** in June 1991. It can also be known as the platform as it provides its own JRE and API.

3. ***What is the most important feature of Java?***
   Outline major features of Java.
   Tell me about Java Programming Language.

Outline major features:
There are the following features in Java Programming Language.

- **Simple:** Java is easy to learn. The syntax of Java is based on C++ which makes easier to write the program in it.

- **Object-Oriented:** Java follows the object-oriented paradigm which allows us to maintain our code as the combination of different type of objects that incorporates both data and behavior.

- **Portable:** Java supports read-once-write-anywhere approach. We can execute the Java program on every machine. Java program (.java) is converted to bytecode (.class) which can be easily run on every machine.

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

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

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

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

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

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

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

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

- **Dynamic:** Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.

---

4. What do you mean by platform independence?
Why Java is platform independent?
Why Java is not 100% object-oriented.

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

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.

---

5. What is JVM?
Is JVM Platform independent?
What is the difference between a JDK and a JVM?
Explain JDK, JRE and JVM?
What is JVM? What is its significance? What do you understand by JVM?
Differentiate between JDK, JRE, JVM.
How many types of memory areas are allocated by JVM?
What is JIT compiler?

JVM stands for Java Virtual Machine. JVM can be explained as an engine that is used by the Java Development Kit to provide a runtime environment to run java code or its applications. The main function of JVM is that it is used for converting Java bytecode into machine language. It is an integral part of the Java Runtime Environment. It works on “Write once, run anywhere” principle and manages the program memory.

JRE

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

JDK

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

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

JDK vs JRE vs 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**.

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.

6. ***How many types of memory areas are allocated by JVM?***

Many types:

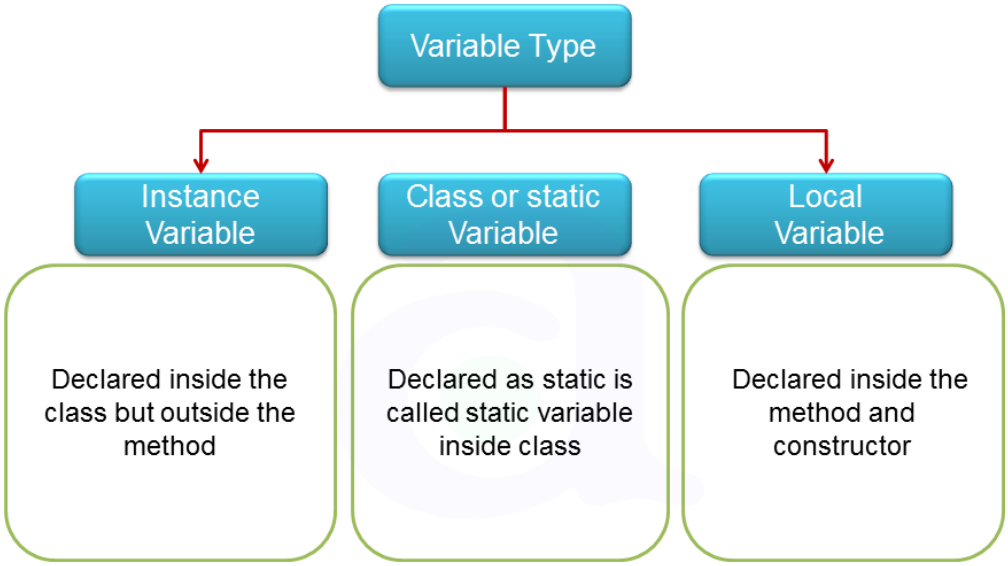
1. **Class(Method) Area:** Class Area stores per-class structures such as the runtime constant pool, field, method data, and the code for methods.
2. **Heap:** It is the runtime data area in which the memory is allocated to the objects
3. **Stack:** Java Stack stores frames. It holds local variables and partial results, and plays a part in method invocation and return. Each thread has a private JVM stack, created at the same time as the thread. A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.
4. **Program Counter Register:** PC (program counter) register contains the address of the Java virtual machine instruction currently being executed.
5. **Native Method Stack:** It contains all the native methods used in the application.

7. ***What gives Java its 'write once and run anywhere' nature?***
The bytecode. Java compiler converts the Java programs into the class file (Byte Code) which is the intermediate language between source code and machine code. This bytecode is not platform specific and can be executed on any computer.

8. What are the default values provided by the JVM for the instance variables?
What are local variables? Does JVM provide any default values to the local variables?

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.

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.

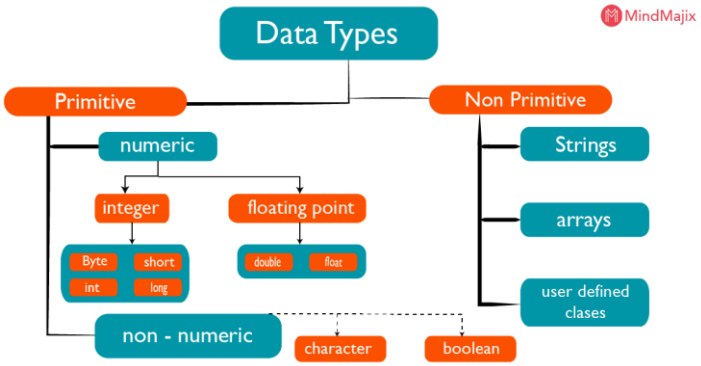


9. ***What is the default value of the local variables?***

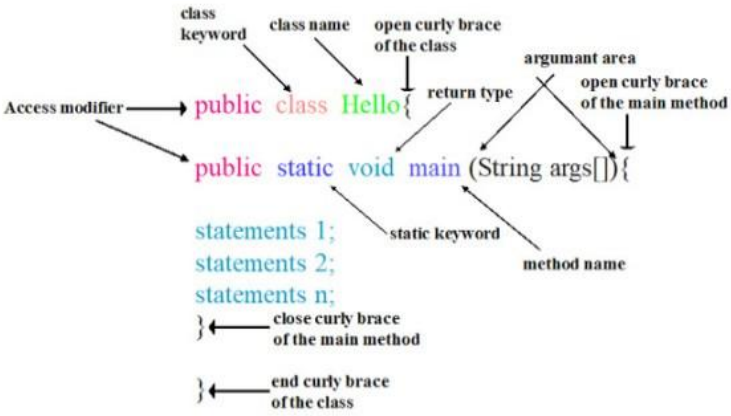
The local variables are not initialized to any default value, neither primitives nor object references.

10. **What is a pointer and does Java support pointers?**
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.

11. **Are arrays primitive data types?**
No, arrays are not primitive datatypes in Java. They are container objects which are created dynamically. All methods of class Object may be invoked on an array. They were considered as reference data types



12. Should a main() method be compulsorily declared in all Java classes?
Differentiate an object and Driver class
What is the return type of the main() method?
Why is the main() method declared static?
What is the argument of the main() method?
Is it possible to overload main() method?
Does the order of public and static declaration matter in the main() method?
Explain public static void main(String args[]) in Java



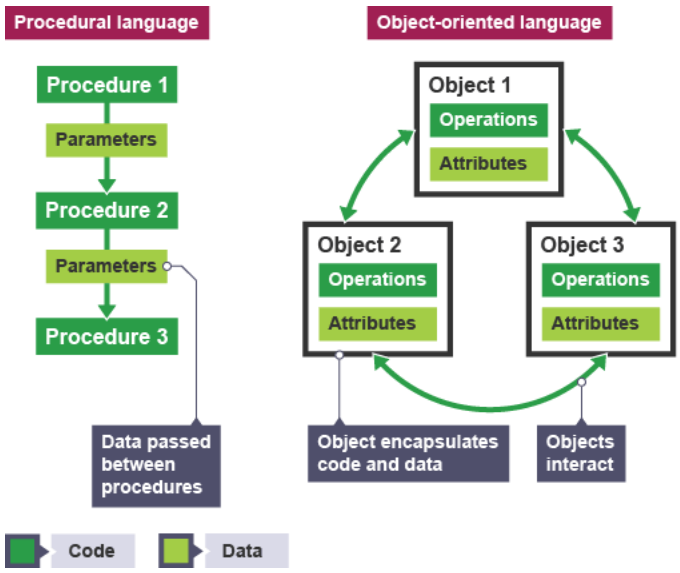
main() in Java is the entry point for any Java program. It is always written as **public static void main(String[] args)**.

- **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.

The main() method doesn't return anything and hence declared void.

The main()method is called by the JVM even before the instantiation of the class hence it is declared as static.
Themain() method accepts an array of String objects as argument.
you can have any number of main() methods with different method signature and implementation in the class.
The order of public and static in main method signature does not matter but void should always come before main()
We can even write classes without main method. They are referred as object classes.
A class with main method is referred as Driver class.

13. What is the difference between procedural and object-oriented programs?



PROCEDURAL ORIENTED PROGRAMMING	OBJECT ORIENTED PROGRAMMING
In procedural programming, program is divided into small parts called ***functions***.	In object oriented programming, program is divided into small parts called ***objects***.
Procedural programming follows ***top down approach***.	Object oriented programming follows ***bottom up approach***.
There is no access specifier in procedural programming.	Object oriented programming have access specifiers like private, public, protected etc.
Adding new data and function is not easy.	Adding new data and function is easy.
Procedural programming does not have any proper way for hiding data so it is ***less secure***.	Object oriented programming provides data hiding so it is ***more secure***.
In procedural programming, overloading is not possible.	Overloading is possible in object oriented programming.
In procedural programming, function is more important than data.	In object oriented programming, data is more important than function.
Procedural programming is based on ***unreal world***.	Object oriented programming is based on ***real world***.
Examples: C, FORTRAN, Pascal, Basic etc.	Examples: C++, Java, Python, C# etc.

14. ***Does java follow any naming conventions?***

Yes.

They must be followed while developing software in java for good maintenance and readability of code. Java uses CamelCase as a practice for writing names of methods, variables, classes, packages and constants.

Camel case in Java Programming : It consists of compound words or phrases such that each word or abbreviation begins with a capital letter or first word with a lowercase letter, rest all with capital.

1. **Classes and Interfaces** : UpperCamelCase
    - Class names should be **nouns**, in mixed case with the **first** letter of each internal word capitalised. Interfaces name should also be capitalised just like class names.
    - Use whole words and must avoid acronyms and abbreviations.

Examples:

interface Bicycle

class MountainBike implements Bicyle

2. **Methods and variables: lowerCamelCase**
    - Methods should be **verbs**, in mixed case with the **first letter lowercase** and with the first letter of each internal word capitalised.

Examples:

void changeGear(int newValue);

int speed = 0;

3. **Constant variables: UPPERCASE**
    - Should be **all uppercase** with words separated by underscores ("_").
    - There are various constants used in predefined classes like Float, Long, String etc.

Examples:

static final int MIN_WIDTH = 4;

4. **Packages: lowercase**

Examples:

java.lang

---

15. Explain few reserved words in Java.

Keywords or Reserved words are the words in a language that are used for some internal process or represent some predefined actions. These words are therefore not allowed to use as a variable names or objects. Doing this will result into a **compile time error**.

Java also contains a list of reserved words or keywords. These are:

1. **abstract** -Specifies that a class or method will be implemented later, in a subclass
2. **assert** -Assert describes a predicate (a true–false statement) placed in a Java program to indicate that the developer thinks that the predicate is always true at that place. If an assertion evaluates to false at run-time, an assertion failure results, which typically causes execution to abort.
3. **boolean** – A data type that can hold True and False values only
4. **break** – A control statement for breaking out of loops
5. **byte** – A data type that can hold 8-bit data values
6. **case** – Used in switch statements to mark blocks of text
7. **catch** – Catches exceptions generated by try statements
8. **char** – A data type that can hold unsigned 16-bit Unicode characters
9. **class** -Declares a new class
10. **continue** -Sends control back outside a loop
11. **default** -Specifies the default block of code in a switch statement
12. **do** -Starts a do-while loop

13. **double** – A data type that can hold 64-bit floating-point numbers
14. **else** – Indicates alternative branches in an if statement
15. **enum** – A Java keyword used to declare an enumerated type. Enumerations extend the base class.
16. **extends** -Indicates that a class is derived from another class or interface
17. **final** -Indicates that a variable holds a constant value or that a method will not be overridden
18. **finally** -Indicates a block of code in a try-catch structure that will always be executed
19. **float** -A data type that holds a 32-bit floating-point number
20. **for** -Used to start a for loop
21. **if** -Tests a true/false expression and branches accordingly
22. **implements** -Specifies that a class implements an interface
23. **import** -References other classes
24. **instanceof** -Indicates whether an object is an instance of a specific class or implements an interface
25. **int** – A data type that can hold a 32-bit signed integer
26. **interface** – Declares an interface
27. **long** – A data type that holds a 64-bit integer
28. **native** -Specifies that a method is implemented with native (platform-specific) code
29. **new** – Creates new objects
30. **null** -Indicates that a reference does not refer to anything
31. **package** – Declares a Java package
32. **private** -An access specifier indicating that a method or variable may be accessed only in the class it's declared in
33. **protected** – An access specifier indicating that a method or variable may only be accessed in the class it's declared in (or a subclass of the class it's declared in or other classes in the same package)
34. **public** – An access specifier used for classes, interfaces, methods, and variables indicating that an item is accessible throughout the application (or where the class that defines it is accessible)
35. **return** -Sends control and possibly a return value back from a called method
36. **short** – A data type that can hold a 16-bit integer
37. **static** -Indicates that a variable or method is a class method (rather than being limited to one particular object)
38. **strictfp** – A Java keyword used to restrict the precision and rounding of floating point calculations to ensure portability.
39. **super** – Refers to a class's base class (used in a method or class constructor)
40. **switch** -A statement that executes code based on a test value
41. **synchronized** -Specifies critical sections or methods in multithreaded code
42. **this** -Refers to the current object in a method or constructor
43. **throw** – Creates an exception
44. **throws** -Indicates what exceptions may be thrown by a method
45. **transient** -Specifies that a variable is not part of an object's persistent state
46. **try** -Starts a block of code that will be tested for exceptions
47. **void** -Specifies that a method does not have a return value
48. **volatile** -Indicates that a variable may change asynchronously
49. **while** -Starts a while loop

****** The keywords **const** and **goto** are reserved, even they are not currently in use.

- **const** -Reserved for future use
- **goto** – Reserved for future use

**** true, false** and **null** look like keywords, but in actual they are **literals**. However they still can't be used as identifiers in a program.

16. What is Object oriented programming?
What is a class?
How do you create a class?
What are objects?
What are the properties of an object?
differentiate class and object

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.

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.

```
classAbc {

      member variables // class body

      methods
}
```

An object is a real-world entity that has a state and behavior. An object has three characteristics:

1. State
2. Behavior
3. Identity

An object is created using the 'new' keyword. For example:
ClassNameobj = new ClassName();

Difference between Class and object In JAVA:

- **Class -** A class is like a blueprint. It is the class with the help of which objects are created. The class holds together data and methods in one single unit. And the class needs to be declared only a single time.
- **Object -** This is an instance of the class. It is with the help of an object that data and methods that exist within a class can be used. Objects can be called upon several times as and when required.

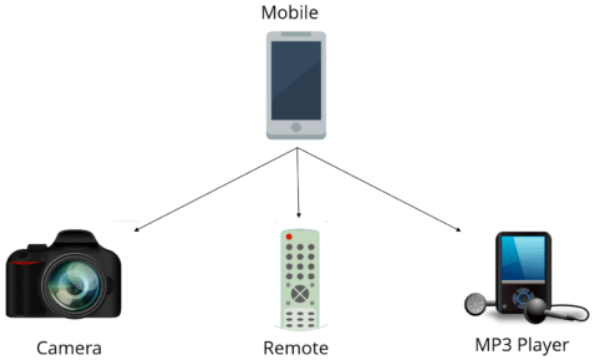
17. What are the basic principles of OOPS?
What is Encapsulation?
What is Inheritance?
what is base class? and derived class?
What is Polymorphism?
How does Java achieves polymorphism?
What are different forms of polymorphim?
What is Abstraction?
Difference between Encapsulation and Abstraction.

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.

Polymorphism is briefly described as “one interface, many implementations”. Polymorphism is a characteristic of being able to assign a different meaning or usage to something in different contexts – specifically, to allow an entity such as a variable, a function, or an object to have more than one form. There are two types of polymorphism:



1. Compile time polymorphism
2. Run time polymorphism

Compile time polymorphism is method overloading whereas Runtime time polymorphism is done using inheritance and interface.

Runtime polymorphism or Dynamic Method Dispatch
In Java, runtime polymorphism or dynamic method dispatch is a process in which a call to an overridden method is resolved at runtime rather than at compile-time. In this process, an overridden method is called through the reference variable of a superclass. Let’s take a look at the example below to understand it better.

Abstraction:
Abstraction refers to the quality of dealing with ideas rather than events. It basically deals with hiding the details and showing the essential things to the user. Thus you can say that abstraction in Java is the process of hiding the implementation details from the user and revealing only the functionality to them. Abstraction can be achieved in two ways:

1. **Abstract Classes** (0-100% of abstraction can be achieved)
2. **Interfaces** (100% of abstraction can be achieved)

Encapsulation:
Encapsulation is a mechanism where you bind your data(variables) and code(methods) together as a single unit. Here, the data is hidden from the outer world and can be accessed only via current class methods. This helps in protecting the data from any unnecessary modification. We can achieve encapsulation in Java by:

- Declaring the variables of a class as private.

Providing public setter and getter methods to modify and view the values of the variables.

18. What is a constructor?
When does the compiler supply a default constructor for a class?
What is the difference between the constructor and a method?
Does java support destructors?

What is a singleton class in Java and how can we make a class singleton?

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.

What is the constructor?
The constructor can be defined as the special type of method that is used to initialize the state of an object. It is invoked when the class is instantiated, and the memory is allocated for the object. Every time, an object is created using the **new** keyword, the default constructor of the class is called. The name of the constructor must be similar to the class name. The constructor must not have an explicit return type

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

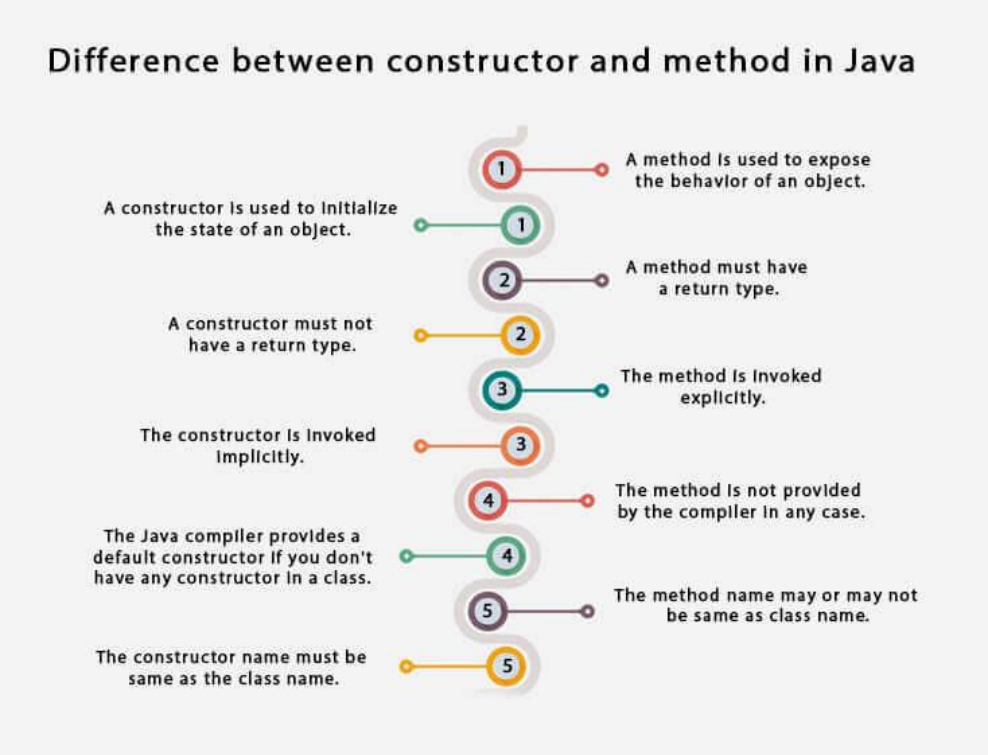
20. Is constructor inherited?
No, The constructor is not inherited.

21. Can you make a constructor final?
No, the constructor can't be final.

22. Can we overload the constructors?
Yes, the constructors can be overloaded by changing the number of arguments accepted by the constructor or by changing the data type of the parameters.

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.

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 Java, constructor overloading is a technique of adding any number of constructors to a class each having a different parameter list. The compiler uses the number of parameters and their types in the list to differentiate the overloaded constructors.

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()

23. Explain Garbage Collection.
Does garbage collection guarantee that a program will not run out of memory?
When is an object subject to garbage collection?
The System.gc() method may be used to call it explicitly.

Garbage collection is a process of reclaiming the unused runtime objects. It is performed for memory management. In other words, we can say that It is the process of removing unused objects from the memory to free up space and make this space available for Java Virtual Machine. Due to garbage collection java gives 0 as output to a variable whose value is not set, i.e., the variable has been defined but not initialized. For this purpose, we were using free() function in the C language and delete() in C++. In Java, it is performed automatically. So, java provides better memory management

How is garbage collection controlled?
Garbage collection is managed by JVM. It is performed when there is not enough space in the memory and memory is running low. We can externally call the System.gc() for the garbage collection. However, it depends upon the JVM whether to perform it or not.

The purpose of garbage collection is to identify and discard objects that are no longer needed by a program, so that their resources can be reclaimed and reused.

A Java object is subject to garbage collection when it becomes unreachable to the program in which it is used.

24. What kind of thread is the Garbage collector thread?
Daemon thread.

An object is subject to garbage collection when it becomes unreachable to the program in which it is used. An object which is not refered by any reference variable is subject to garbage collection.
when an object is no longer referred to by any reference varible, Java automatically reclaims memory used by that object. This is known as garbage collection.
The finalize() method is called automatically just before the an object is destroyed and can be called just prior to garbage collection.

---

25. ***What is meant by binding?***
What is static binding?
What is dynamic binding?
can a class be declared as static?
When will you define a method as static?
What are the restrictions imposed on a static method or a static block of code?
How to print some message before main() is executed?
What is the importance of static variables?
can we declare a static variable inside a method?
differentiate local, instance and class variables.
What is the difference between inner class and nested class?
What is static in Java?
Differentiate static and non-static variables.

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

27. ***What is the purpose of static methods and variables?***
The methods or variables defined as static are shared among all the objects of the class. The static is the part of the class and not of the object. The static variables are stored in the class area, and we do not need to create the object to access such variables. Therefore, static is used in the case, where we need to define variables or methods which are common to all the objects of the class.

For example, In the class simulating the collection of the students in a college, the name of the college is the common attribute to all the students. Therefore, the college name will be defined as **static**.

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

Static means one per class, not one for each object no matter how many instance of a class might exist. This means that you can use them without creating an instance of a class. The static methods are implicitly final because overriding is done based on the type of the object and static methods are attached to a class, not to an object.

A static method in a superclass can be shadowed by another static method in a subclass as long as the original method was not declared final. However, you can't override a static method with a non-static method. In other words, you can't change a static method into an instance method in a subclass

What is the static method?

- A static method belongs to the class rather than the object.
- There is no need to create the object to call the static methods.
- A static method can access and change the value of the static variable.

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

Two main restrictions are applied to the static methods.

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

28. Why is the main method static?

Because the object is not required to call the static method. If we make the main method non-static, JVM will have to create its object first and then call main() method which will lead to the extra memory allocation.

29. Can we override the static methods?

No, we can't override static methods.

30. What is the static block?

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

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

static or class method	instance method
1)A method that is declared as static is known as the static method.	A method that is not declared as static is known as the instance method.
2)We don't need to create the objects to call the static methods.	The object is required to call the instance methods.
3)Non-static (instance) members cannot be accessed in the static context (static method, static block, and static nested class) directly.	Static and non-static variables both can be accessed in instance methods.
4)For example: public static int cube(int n){ return n*n*n;}	For example: public void msg(){...}.

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

33.Why String is immutable in Java? Mention atleast two reasons?
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.

34. what is the difference between equals() method and == operator.

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 booleanequals(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.

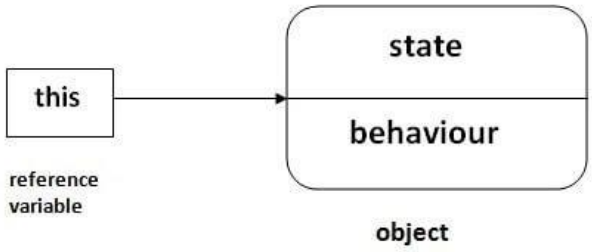
35. Difference between String, StringBuilder and StringBuffer

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

36. What is the purpose of toString() method in Java?
The toString() method returns the string representation of an object. If you print any object, java compiler internally invokes the toString() method on the object. So overriding the toString() method, returns the desired output, it can be the state of an object, etc. depending upon your implementation. By overriding the toString() method of the Object class, we can return the values of the object, so we don't need to write much code.

37***. Explain this keyword in Java.***
Explain super keyword in Java?
Differentiate this and super keywords

this keyword:
The **this** keyword is a reference variable that refers to the current object. There are the various uses of this keyword in Java. It can be used to refer to current class properties such as instance methods, variable, constructors, etc. It can also be passed as an argument into the methods or constructors. It can also be returned from the method as the current class instance.g



main uses of this keyword:
There are the following uses of **this** keyword.

- **this** can be used to refer to the current class instance variable.
- **this** can be used to invoke current class method (implicitly)
- **this()** can be used to invoke the current class constructor.
- **this** can be passed as an argument in the method call.
- **this** can be passed as an argument in the constructor call.
- **this** can be used to return the current class instance from the method.

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

main uses of the super keyword:
There are the following uses of super keyword.

- super can be used to refer to the immediate parent class instance variable.
- super can be used to invoke the immediate parent class method.
- super() can be used to invoke immediate parent class constructor.

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

38. What is the inheritance?
What are the benefits of inheritance?
Which class is the super class for all the classes?
Why is multiple inheritance not supported in Java?
What is aggregation?

Inheritance in Java is the concept where the properties of one class can be inherited by the other. It helps to reuse the code and establish a relationship between different classes. Inheritance is performed between two types of classes:

1. Parent class (Super or Base class)

2. Child class (Subclass or Derived class)

A class which inherits the properties is known as Child Class whereas a class whose properties are inherited is known as Parent class.

Java supports four types of inheritance which are:

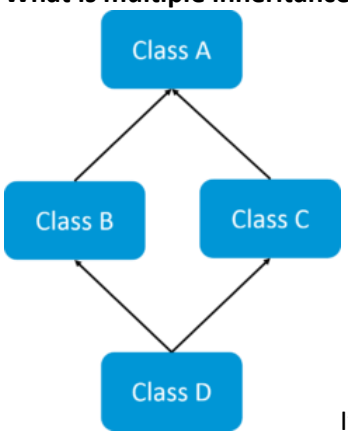
1. **Single Inheritance:** In single inheritance, one class inherits the properties of another i.e there will be only one parent as well as one child class.
2. **Multilevel Inheritance:** When a class is derived from a class which is also derived from another class, i.e. a class having more than one parent class but at different levels, such type of inheritance is called Multilevel Inheritance.
3. **Hierarchical Inheritance:** When a class has more than one child classes (subclasses) or in other words, more than one child classes have the same parent class, then such kind of inheritance is known as hierarchical.
4. **Hybrid Inheritance:** Hybrid inheritance is a combination of two *or more types* of inheritance.

Why is Inheritance used in Java?

There are various advantages of using inheritance in Java that is given below.

- Inheritance provides code reusability. The derived class does not need to redefine the method of base class unless it needs to provide the specific implementation of the method.
- Runtime polymorphism cannot be achieved without using inheritance.
- We can simulate the inheritance of classes with the real-time objects which makes OOPs more realistic.
- Inheritance provides data hiding. The base class can hide some data from the derived class by making it private.
- Method overriding cannot be achieved without inheritance. By method overriding, we can give a specific implementation of some basic method contained by the base class.

What is multiple inheritance? Is it supported by Java?



If a child class inherits the property from multiple classes is known as multiple inheritance. Java does not allow to extend multiple classes.

The problem with multiple inheritance is that if multiple parent classes have the same method name, then at runtime it becomes difficult for the compiler to decide which method to execute from the child class.

Therefore, Java doesn't support multiple inheritance. The problem is commonly referred to as Diamond Problem.

What is an association?

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

and both have their own lifecycle. These relationships can be one to one, one to many, many to one and many to many.

What do you mean by aggregation?
An aggregation is a specialized form of Association where all object has their own lifecycle but there is ownership and child object can not belong to another parent object. Let's take an example of Department and teacher. A single teacher can not belong to multiple departments, but if we delete the department teacher object will not destroy.

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

The Object class is the highest-level class in the Java class hierarchy.

---

39. **What is method overloading?**
What is method overriding?
Can we override the overloaded method?
Differentiate between method overloading and overriding.
Can we override the private methods?

Method Overloading :

- In Method Overloading, Methods of the same class shares the same name but each method must have a different number of parameters or parameters having different types and order.
- Method Overloading is to "add" or "extend" more to the method's behavior.
- It is a compile-time polymorphism.
- The methods must have a different signature.

It may or may not need inheritance in Method Overloading.

Method Overriding:

- In Method Overriding, the subclass has the same method with the same name and exactly the same number and type of parameters and same return type as a superclass.
- Method Overriding is to "Change" existing behavior of the method.
- It is a run time polymorphism.
- The methods must have the same signature.
- It always requires inheritance in Method Overriding.

Rules for Method overriding

- The method must have the same name as in the parent class.
- The method must have the same signature as in the parent class.
- Two classes must have an IS-A relationship between them.

Can we override the static method?

No, you can't override the static method because they are the part of the class, not the object.

Difference between method Overloading and Overriding.

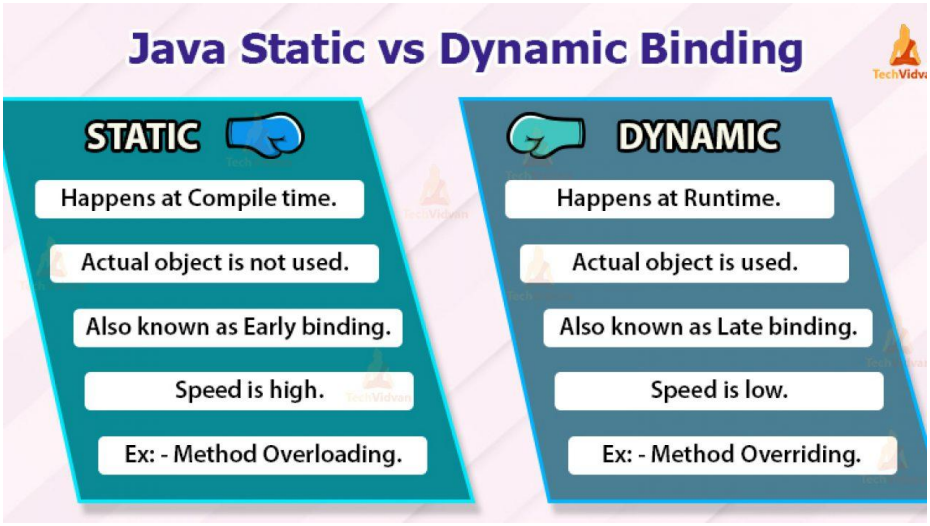
Method Overloading	Method Overriding
1) Method overloading increases the readability of the program.	Method overriding provides the specific implementation of the method that is already provided by its superclass.
2) Method overloading occurs within the class.	Method overriding occurs in two classes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

Can we override the private methods?

No.

Can you override a private or static method in Java?
You cannot override a private or static method in Java. If you create a similar method with the same return type and same method arguments in child class then it will hide the superclass method; this is known as method hiding. Similarly, you cannot override a private method in subclass because it's not accessible there. What you can do is create another private method with the same name in the child class. Let's take a look at the example below to understand it better.

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



40. ***What are the different types of access modifiers?***
What is the default access scope of a class, variable & methods if they are declared without any access specifiers?
Can a top level class be private or protected?

What type of parameter passing does Java support?
What is the access scope of a protected method?
What do you understand by private, protected and public?
If a class is declared without any access modifiers, where the class can be accessed?
what modifiers may be used with a top-level class?
what modifiers may be used with an inner class that is a member of an outer class?

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

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.

we cannot declare a **top-level class** as **private or protected**. It **can** be either public or default (no modifier). If it **does** not have a modifier, it is supposed to have a default access

The **top-level** classes can only have **public, abstract** and **final** modifiers,

We can declare the **inner classes** as **private** or **protected**, but it is not allowed in **outer classes**.

More than one **top-level class** can be defined in a Java source file, but there can be at most one **public top-level class** declaration. The file name must match the name of the public class.

41. ***What is a package?***
Which package is imported by default in all Java programs?
name some packages.
What is meant by user defined package?
CAn I import same package / class twice in my Java program?

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

java.lang package gets imported to all java classes by default.

we **can import same package/class** multiple times. It **will** not create any problem . Neither compiler nor JVM complains wil complain about it. JVM **will** internally load **the class** only once no matter how many times you **import the same class**.

What are the advantages of Packages in Java?

There are various advantages of defining packages in Java.

- Packages avoid the name clashes.
- The Package provides easier access control.
- We can also have the hidden classes that are not visible outside and used by the package.
- It is easier to locate the related classes.

Few packages in java

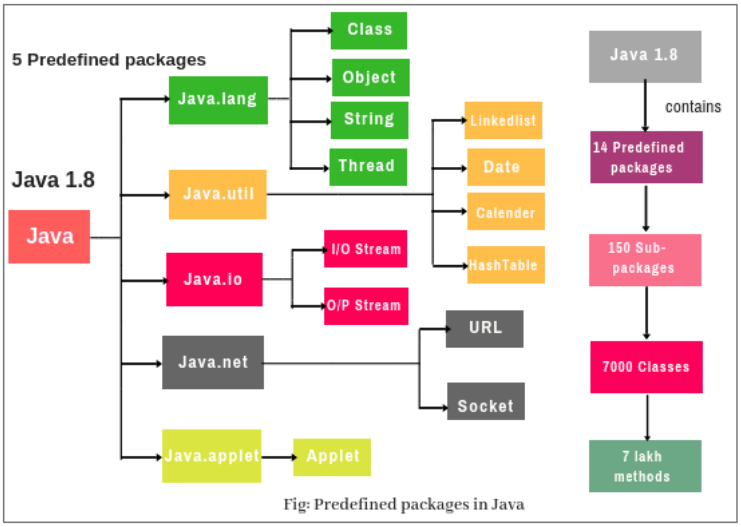


Fig: Predefined packages in Java

42. About Final keyword

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.

43. ***What is an abstract class and what is its purpose?***
can an abstract class be declared final?
Can you create an object of an abstract class?
Can an abstract class be defined without any abstract methods?
What is an abstract method?
What is the difference between abstract class and interface?
What is interface in JAva and what are its uses?
Can a method inside an interface be declared as final?
Can an interface implement another interface?
can an interface extend another interface?
can a class extend more than one class?
can an interface be final?

What is the abstract class?
A class that is declared as abstract is known as an abstract class. It needs to be extended and its method implemented. It cannot be instantiated. It can have abstract methods, non-abstract methods, constructors, and static methods. It can also have the final methods which will force the subclass not to change the body of the method.

Interface:
An interface in Java is a blueprint of a class or you can say it is a collection of abstract methods and static constants. In an interface, each method is public and abstract but it does not contain any constructor. Thus, interface basically is a group of related methods with empty bodies. Example:

```
public interface Animal {
  public void eat();
  public void sleep();
  public void run();
}
```

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

	to find the corresponding method in the actual class

Can you use abstract and final both with a method?
No, because we need to override the abstract method to provide its implementation, whereas we can't override the final method.

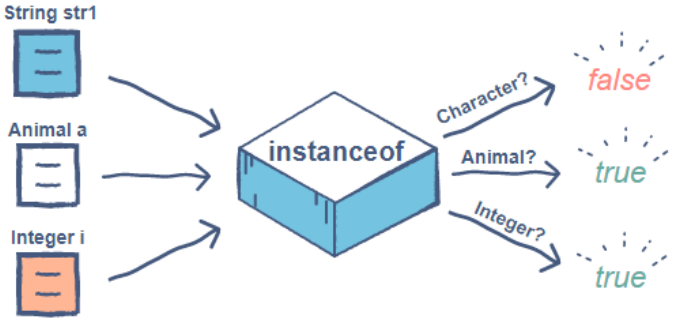
Is it possible to instantiate the abstract class?
No, the abstract class can never be instantiated even if it contains a constructor and all of its methods are implemented.

Interfaces can be used to achieve full abstraction and multiple inheritance. It is a mechanism to achieve abstraction.
Java Interface also represents the IS-A relationship. It cannot be instantiated just like the abstract class. However, we need to implement it to define its methods. Since Java 8, we can have the default, static, and private methods in an interface.

Can you declare an interface method static?
No, because methods of an interface are abstract by default, and we can not use static and abstract together.

Can the Interface be final?
No, because an interface needs to be implemented by the other class and if it is final, it can't be implemented by any class.

What is Java instanceOf operator?
The instanceof in Java is also known as type comparison operator because it compares the instance with type. It returns either true or false. If we apply the instanceof operator with any variable that has a null value, it returns false.



44. ***Exception Handling:***
How many types of exception can occur in a Java program?
What is Exception Handling?
Explain the Exception Class Hierarchy.
What is finally block?
What is the difference between throw and throws?
What is exception propagation?

An error is an irrecoverable condition occurring at runtime. Such as OutOfMemory error. These JVM errors you cannot repair them at runtime. Though error can be caught in the catch block but the execution of application will come to a halt and is not recoverable.

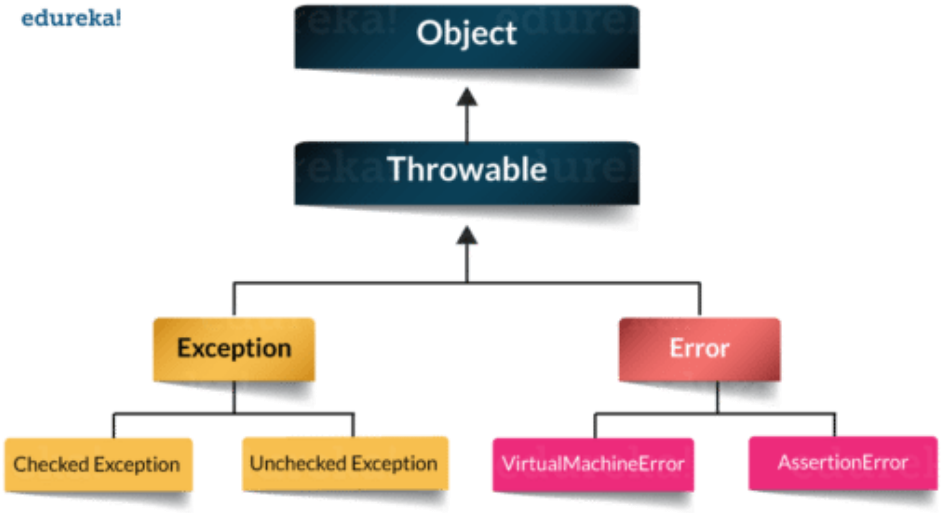
While exceptions are conditions that occur because of bad input or human error etc. e.g. FileNotFoundException will be thrown if the specified file does not exist. Or a NullPointerException will take place if you try using a null reference. In most of the cases it is possible to recover from an exception (probably by giving the user feedback for entering proper values etc.

Handling Java Exceptions:
There are five keywords used to handle exceptions in Java:

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

Excepiton Class Hierarchy

Throwable is a parent class of all Exception classes. There are two types of Exceptions: Checked exceptions and UncheckedExceptions or RunTimeExceptions. Both type of exceptions extends Exception class whereas errors are further classified into Virtual Machine error and Assertion error.




```
public String getUserName(String userId) throws UserNotExistException
    {
        String userName =null;
        try
        {
            if(userId.equals("123"))
            {
                userName ="Saurabh Gupta";
            }
            else
            {
                throw new UserNotExistException();
            }
        }
        catch(Exception ex)
        {
            ex.printStackTrace();
        }
        finally
        {
            System.out.println("Will call at last..but sure");
        }
        return userName;

    }
```

Throws exception to handle by calling method If not handle on current method.

Try block to guard exceptions

Throw Exception if user id not found

Exception Handler to catch Exception

Finally must execute even no exception

differences between throw and throws?

throw keyword	throws keyword
Throw is used to explicitly throw an exception.	Throws is used to declare an exception.
Checked exceptions can not be propagated with throw only.	Checked exception can be propagated with throws.
Throw is followed by an instance.	Throws is followed by class.
Throw is used within the method.	Throws is used with the method signature.
You cannot throw multiple exception	You can declare multiple exception e.g. public void method()throws IOException,SQLException.

differences between Checked Exception and Unchecked Exception?

Checked Exception

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

Unchecked Exception

- The classes that extend RuntimeException are known as unchecked exceptions.
- Unchecked exceptions are not checked at compile-time.

Example: ArithmeticException, NullPointerException etc.

create a custom Exception?

To create you own exception extend the Exception class or any of its subclasses.

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

What is a finally block? Is there a case when finally will not execute?

Finally block is a block which always executes a set of statements. It is always associated with a try block regardless of any exception that occurs or not.

Yes, finally will not be executed if the program exits either by calling System.exit() or by causing a fatal error that causes the process to abort.

Can we write multiple catch blocks under single try block?

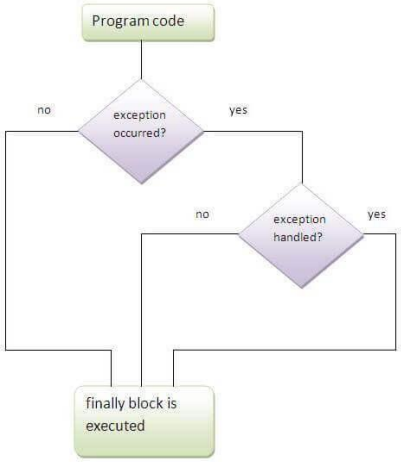
Yes we can have multiple catch blocks under single try block but the approach should be from specific to general. Let's understand this with a programmatic example.

```
1       public class Example {
2       public static void main(String args[]) {
3       try {
4       int a[]= new int[10];
5       a[10]= 10/0;
6       }
7       catch(ArithmeticException e)
8       {
9       System.out.println("Arithmetic exception in first catch block");
10      }
11      catch(ArrayIndexOutOfBoundsException e)
```

```
12      {
13      System.out.println("Array index out of bounds in second catch block");
14      }
15      catch(Exception e)
16      {
17      System.out.println("Any exception in third catch block");
18      }
19      }
```

What is finally block?

The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not. In other words, we can say that finally block is the block which is always executed. Finally block follows try or catch block. If you don't handle the exception, before terminating the program, JVM runs finally block, (if any). The finally block is mainly used to place the cleanup code such as closing a file or closing a connection. Here, we must know that for each try block there can be zero or more catch blocks, but only one finally block. The finally block will not be executed if program exits(either by calling System.exit() or by causing a fatal error that causes the process to abort).



Can finally block be used without a catch?

Yes, According to the definition of finally block, it must be followed by a try or catch block, therefore, we can use try block instead of catch.

Is there any case when finally will not be executed?

Finally block will not be executed if program exits(either by calling System.exit() or by causing a fatal error that causes the process to abort)

What is the difference between throw and throws?

throw keyword	throws keyword
1) The **throw** keyword is used to throw an exception explicitly.	The **throws** keyword is used to declare an exception.
2) The checked exceptions cannot be propagated with throw only.	The checked exception can be propagated with throws

3) The **throw** keyword is followed by an instance.	The **throws** keyword is followed by class.
4) The **throw** keyword is used within the method.	The **throws** keyword is used with the method signature.
5) You cannot throw multiple exceptions.	You can declare multiple exceptions, e.g., public void method()throws IOException, SQLException.

What is exception propagation?
An exception is first thrown from the top of the stack and if it is not caught, it drops down the call stack to the previous method, If not caught there, the exception again drops down to the previous method, and so on until they are caught or until they reach the very bottom of the call stack. This procedure is called exception propagation. By default, checked exceptions are not propagated.

45. We have final, finally, and finalize in java. Are all three related?

Final:
Final is used to apply restrictions on class, method, and variable. A final class can't be inherited, final method can't be overridden and final variable value can't be changed. Let's take a look at the example below to understand it better.

```
1       class FinalVarExample {
2       public static void main( Stringargs[])
3       {
4       final int a=10;   // Final variable
5       a=50;           //Error as value can't be changed
6       }
```

Finally
Finally is used to place important code, it will be executed whether the exception is handled or not. Let's take a look at the example below to understand it better.

```
1         class FinallyExample {
2         public static void main(String args[]){
3         try {
4         int x=100;
5         }
6         catch(Exception e) {
7         System.out.println(e);
8         }
9         finally {
10        System.out.println("finally block is executing");}
11        }}
12        }
```

Finalize
Finalize is used to perform clean up processing just before the object is garbage collected. Let's take a look at the example below to understand it better.

What is the difference between final, finally and finalize?

No.	final	finally	finalize
1)	Final is used to apply restrictions on	Finally is used to place	Finalize is used to perform

	class, method, and variable. The final class can't be inherited, final method can't be overridden, and final variable value can't be changed.	important code, it will be executed whether an exception is handled or not.	clean up processing just before an object is garbage collected.
2)	Final is a keyword.	Finally is a block.	Finalize is a method.

46. What is a Thread?

A thread is the smallest piece of programmed instructions which can be executed independently by a scheduler. In Java, all the programs will have at least one thread which is known as the main thread. This main thread is created by the JVM when the program starts its execution. The main thread is used to invoke the main() of the program.

47. What are the two ways to create a thread?

In Java, threads can be created in the following two ways:-

- By implementing the Runnable interface.
- By extending the Thread

If a class is declared without any access modifiers, where may the class be accessed?

A class that is declared without any access modifiers is said to have package access. This means that the class can only be accessed by other classes and interfaces that are defined within the same package.

48. What is the purpose of wrapper classes?

Wrapper classes allow primitive types to be accessed as objects.

Wrapper classes are classes that allow primitive types to be accessed as objects. In other words, we can say that wrapper classes are built-in java classes which allow the conversion of objects to primitives and primitives to objects. The process of converting primitives to objects is called autoboxing, and the process of converting objects to primitives is called unboxing.

Primitive Types	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

49. What is a daemon thread in Java?

Daemon thread is a low need string (in setting of JVM) that runs in foundation to perform assignments, for example, trash assortment (GC) and so forth., they don't keep the JVM from leaving (regardless of whether the daemon string itself is running) when all the client strings (non-daemon strings) finish their execution. JVM ends itself when all client strings (non-daemon strings) finish their execution, JVM couldn't care less whether Daemon string is running or not, if JVM discovers running daemon (endless supply of client strings), it ends the string and after that shutdown itself.

50. What are Mutable classes?

Mutable classes are the classes that are being prepared can change its internal state anytime without any issue. Also, it can change the state even after it has been already developed. The classes here are not the ones that are actually mutable. But the object that consists inside it, they are actually the mutated ones. Until we can prevent it from any change after it has been developed. It also depends strongly on the language that is being used.

51. What is an applet program?

An **applet is a Java** application that could be installed into a web page and runs within the web browser and operates at the client-side. It is usually embedded in an HTML page using the APPLET or OBJECT tag and treated on a web server.
Applets are employed to make the web site more productive and entertaining. All applets are sub-classes (directly or indirectly) of java.applet. Applet class and are not stand-alone applications. Instead, they work in a web browser or an applet viewer. JDK provides a standard applet viewing tool known as applet viewer. In general, the working of an applet does not begin at the main() method. When an applet starts, the init(), start() and paint() methods are called in a sequence and when an applet is terminated, stop( )and destroy( ) method calls take place.

52. The differences between C++ and Java are given in the following table.

Comparison Index	C++	Java
Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in window, web-based, enterprise and mobile application.
Design Goal	C++ was designed for systems and applications programming. It was an extension of C programming language.	Java was designed and created as an interpreter for printing systems but later extended as a support network computing. It was designed with a goal of being easy to use and accessible to a broader audience.
Goto	C++ supports the goto statement.	Java doesn't support the goto statement.
Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by interfaces in java.
Operator Overloading	C++ supports operator overloading.	Java doesn't support operator overloading.
Pointers	C++ supports pointers. You can write pointer program in C++.	Java supports pointer internally. However, you can't write the pointer program in java. It means java has restricted pointer support in Java.
Compiler and Interpreter	C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent.	Java uses compiler and interpreter both. Java source code is converted into bytecode at compilation time. The interpreter executes this bytecode at runtime and produces output. Java is interpreted that is why it is platform

		independent.
Call by Value and Call by reference	C++ supports both call by value and call by reference.	Java supports call by value only. There is no call by reference in java.
Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.
Thread Support	C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support.	Java has built-in thread support.
Documentation comment	C++ doesn't support documentation comment.	Java supports documentation comment (/** ... */) to create documentation for java source code.
Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not override a function.	Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are virtual by default.
unsigned right shift >>>	C++ doesn't support >>> operator.	Java supports unsigned right shift >>> operator that fills zero at the top for the negative numbers. For positive numbers, it works same like >> operator.
Inheritance Tree	C++ creates a new inheritance tree always.	Java uses a single inheritance tree always because all classes are the child of Object class in java. The object class is the root of the inheritance tree in java.
Hardware	C++ is nearer to hardware.	Java is not so interactive with hardware.
Object-oriented	C++ is an object-oriented language. However, in C language, single root hierarchy is not possible.	Java is also an object-oriented language. However, everything (except fundamental types) is an object in Java. It is a single root hierarchy as everything gets derived from java.lang.Object.

Collections
++++++++++
What is the difference between Array list and vector in Java?

What are the differences between Heap and Stack Memory in Java?
What is hashing in Java?

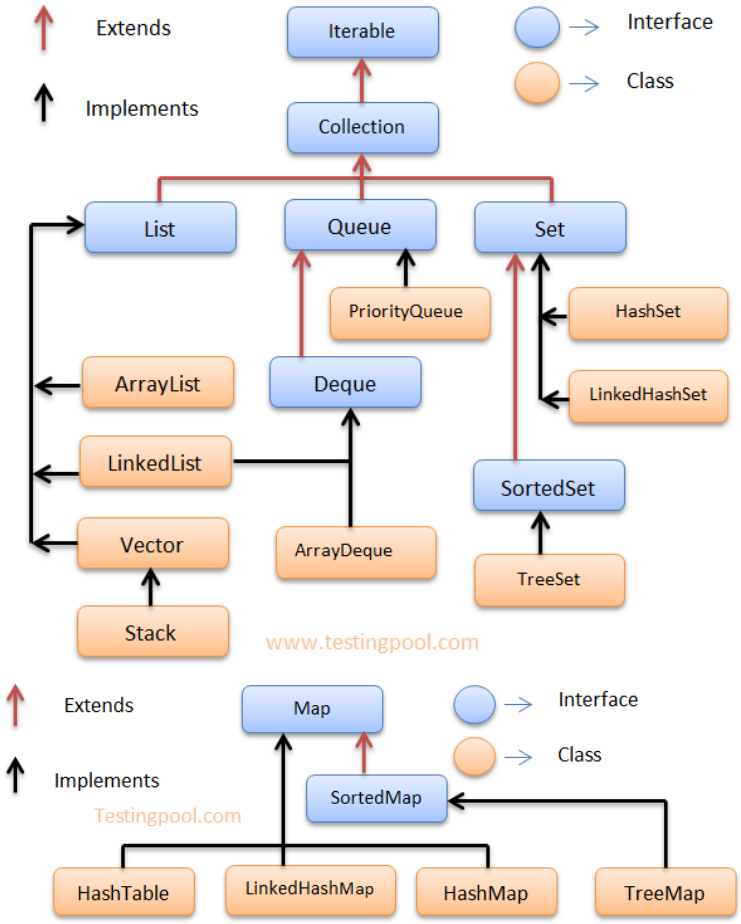
53. What is the Collection framework in Java?

Collection Framework is a combination of classes and interface, which is used to store and manipulate the data in the form of objects. It provides various classes such as ArrayList, Vector, Stack, and HashSet, etc. and interfaces such as List, Queue, Set, etc. for this purpose.

54. ***What are the main differences between array and collection?***
Array and Collection are somewhat similar regarding storing the references of objects and manipulating the data, but they differ in many ways. The main differences between the array and Collection are defined below:

- Arrays are always of fixed size, i.e., a user can not increase or decrease the length of the array according to their requirement or at runtime, but In Collection, size can be changed dynamically as per need.
- Arrays can only store homogeneous or similar type objects, but in Collection, heterogeneous objects can be stored.
- Arrays cannot provide the ready-made methods for user requirements as sorting, searching, etc. but Collection includes readymade methods to use.

55. **Explain various interfaces used in Collection framework?**
Collection framework implements various interfaces, Collection interface and Map interface (java.util.Map) are the mainly used interfaces of Java Collection Framework. List of interfaces of Collection Framework is given below:



1. Collection interface: Collection (java.util.Collection) is the primary interface, and every collection must implement this interface.
Syntax:

1. **public interface** Collection<E>**extends** Iterable

Where <E> represents that this interface is of Generic type

2. List interface: List interface extends the Collection interface, and it is an ordered collection of objects. It contains duplicate elements. It also allows random access of elements.

Syntax:

1. **public interface** List<E> **extends** Collection<E>

3. Set interface: Set (java.util.Set) interface is a collection which cannot contain duplicate elements. It can only include inherited methods of Collection interface

Syntax:

1. **public interface** Set<E> **extends** Collection<E>

Queue interface: Queue (java.util.Queue) interface defines queue data structure, which stores the elements in the form FIFO (first in first out).

Syntax:

1. **public interface** Queue<E> **extends** Collection<E>

4. Dequeue interface: it is a double-ended-queue. It allows the insertion and removal of elements from both ends. It implants the properties of both Stack and queue so it can perform LIFO (Last in first out) stack and FIFO (first in first out) queue, operations.

Syntax:

1. **public interface** Dequeue<E> **extends** Queue<E>

5. Map interface: A Map (java.util.Map) represents a key, value pair storage of elements. Map interface does not implement the Collection interface. It can only contain a unique key but can have duplicate elements. There are two interfaces which implement Map in java that are Map interface and Sorted Map.

56. **What is the difference between ArrayList and Vector?**

No.	ArrayList	Vector
1)	ArrayList is not synchronized.	Vector is synchronized.
2)	ArrayList is not a legacy class.	Vector is a legacy class.
3)	ArrayList increases its size by 50% of the array size.	Vector increases its size by doubling the array size.
4)	ArrayList is not ?thread-safe? as it is not synchronized.	Vector list is ?thread-safe? as it?s every method is synchronized.

57. What is the difference between ArrayList and LinkedList?

No.	ArrayList	LinkedList
1)	ArrayList uses a dynamic array.	LinkedList uses a doubly linked list.
2)	ArrayList is not efficient for manipulation because too much is required.	LinkedList is efficient for manipulation.
3)	ArrayList is better to store and fetch data.	LinkedList is better to manipulate data.
4)	ArrayList provides random access.	LinkedList does not provide random access.
5)	ArrayList takes less memory overhead as it stores only object	LinkedList takes more memory overhead, as it stores the object as well as the address of that object.

	ArrayList	LinkedList	Vector	Stack
Duplicates	Allow	Allow	Allow	Allow
Order	Insertion order	Insertion order	Insertion order	Insertion order
Insert / Delete	Slow	Fast	Slow	Slow
Accessibility	Radom and fast	Sequential and slow	Random and fast	Slow
Traverse	Uses Iterator / ListIterator	Uses Iterator / ListIterator	Enumeration	Enumeration
Synchronization	No	No	Yes	Yes
Increment size	50%	No initial size	100%	100%

58. ***What is the difference between Iterator and ListIterator?***

Iterator traverses the elements in the forward direction only whereas ListIterator traverses the elements into forward and backward direction.

No.	Iterator	ListIterator
1)	The Iterator traverses the elements in the forward direction only.	ListIterator traverses the elements in backward and forward directions both.
2)	The Iterator can be used in List, Set, and Queue.	ListIterator can be used in List only.
3)	The Iterator can only perform remove operation while traversing the collection.	ListIterator can perform ?add,? ?remove,? and ?set? operation while traversing the collection.

59. **What is the difference between Iterator and Enumeration?**

No.	Iterator	Enumeration
1)	The Iterator can traverse legacy and non-legacy elements.	Enumeration can traverse only legacy elements.
2)	The Iterator is fail-fast.	Enumeration is not fail-fast.
3)	The Iterator is slower than Enumeration.	Enumeration is faster than Iterator.
4)	The Iterator can perform remove operation while traversing the collection.	The Enumeration can perform only traverse operation on the collection.

60. ***What is the difference between List and Set?***

The List and Set both extend the collection interface. However, there are some differences between the both which are listed below.

- The List can contain duplicate elements whereas Set includes unique items.
- The List is an ordered collection which maintains the insertion order whereas Set is an unordered collection which does not preserve the insertion order.
- The List interface contains a single legacy class which is Vector class whereas Set interface does not have any legacy class.
- The List interface can allow n number of null values whereas Set interface only allows a single null value.

	TreeSet	HashSet	LinkedHashSet
Duplicates	No	No	No
Thread safety	No	No	No
Speed	Slow	Faster	Medium
Order	Sorted order	No order	Insertion order
Null values	No	Allow	Allow
Implementation	By NavigableMap	HashMap	LinkedList & HashSet

61. What is the difference between HashSet and TreeSet?

The HashSet and TreeSet, both classes, implement Set interface.

HashSet maintains no order whereas TreeSet maintains ascending order.

HashSet impended by hash table whereas TreeSet implemented by a Tree Structure.

HashSet performs faster than TreeSet.

HashSet is backed by HashMap whereas TreeSet is backed by TreeMap

62. What is the difference between Set and Map?

Set Contains values only whereas Map contains key and values both.

Set contains unique values whereas Map can contain unique Keys and duplicate values.

Set holds a single number of null value whereas Map can include a single null key with n number of null values.

63. What is difference between HashSet and HashMap?

HashSet contains only values whereas HashMap includes the entry(key, value). HashSet can be iterated, but HashMap needs to convert into Set to be iterated.

HashSet implements Set interface whereas HashMap implements the Map interface.

HashSet connot have any duplicate value whereas HashMap can contain duplicate values with unique keys.

HashSet contains the only single number of null value whereas HashMap can hold a single null key with n number of null values.

64. What is the difference between HashMap and TreeMap?

HashMap maintains no order, but TreeMap maintains ascending order.

HashMap is implemented by hash table whereas TreeMap is implemented by a Tree Structure.

HashMap can be sorted by Key or value whereas TreeMap can be sorted by Key.

HashMap may contain a null key with multiple null values whereas TreeMap cannot hold a null key but can have multiple null values.

65. What is the difference between HashMap and Hashtable?

No.	HashMap	Hashtable
1)	HashMap is not synchronized.	Hashtable is synchronized.
2)	HashMap can contain one null key and multiple null values.	Hashtable cannot contain any null key or null value.
3)	HashMap is not thread-safe, so it is useful for non-threaded applications.	Hashtable is thread-safe, and it can be shared between various threads.
4)	4) HashMap inherits the AbstractMap class	Hashtable inherits the Dictionary class.

66. What is the difference between Collection and Collections?

The Collection is an interface whereas Collection is a class.

The Collection interface provides the standard functionality of data structure to List, Set and Queue. However, Collections class is to sort and synchronize the collection elements.

The Collection interface provides the methods that can be used for data structure whereas Collections class provides the static methods which can be used for various operation on a colletion.

67. What is the difference between Comparable and Comparator?

No.	Comparable	Comparator
1)	Comparable provides only one sort of sequence.	The Comparator provides multiple sorts of sequences.
2)	It provides one method named compareTo().	It provides one method named compare().
3)	It is found in java.lang package.	It is located in java.util package.
4)	If we implement the Comparable interface, The actual class is modified.	The actual class is not changed.

68. What does the hashCode() method?

Returns a hashcode value (an integer number)

Returns the same integer number if two keys (by calling equals()) method are identical.

69. Differences between Array and ArrayList?

SN	Array	ArrayList
1	The Array is of fixed size, means we cannot resize the array as per need.	ArrayList is not of the fixed size we can change the size dynamically.
2	Arrays are of the static type.	ArrayList is of dynamic size.
3	Arrays can store primitive data types as well as	ArrayList cannot store the primitive data types it can

	objects.	only store the objects.