

25 IMPORTANT OOPS CONCEPT

INTERVIEW QUESTIONS



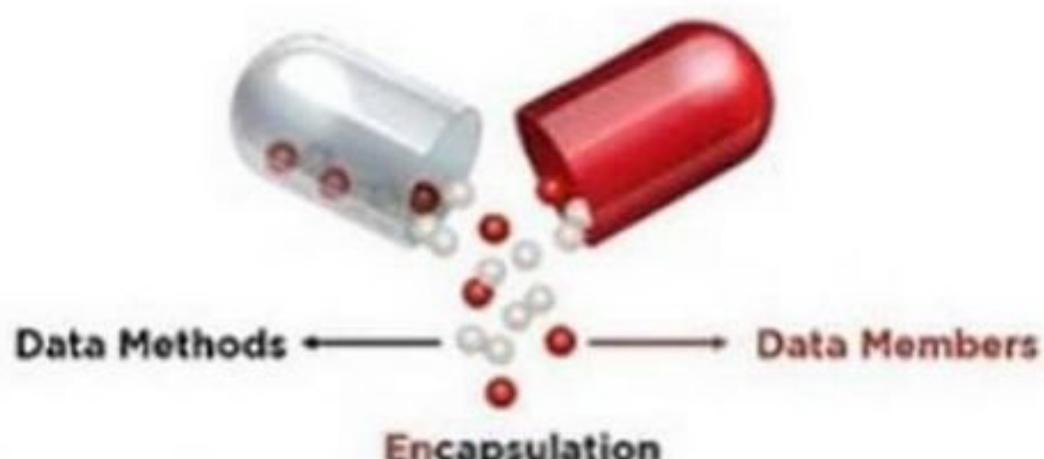
Q 1. What is object-oriented programming (OOPs)?

Ans: Object-Oriented Programming (OOP) is a programming paradigm that organizes code into objects, which are instances of classes.

It focuses on encapsulating data and behavior together, promoting modularity and reusability.

Q 2. What is encapsulation?

Ans: Encapsulation is the concept of bundling data (attributes) and methods (functions) that operate on that data into a single unit (class). It hides the internal details of an object and provides controlled access through methods.



Q 3. What is inheritance?

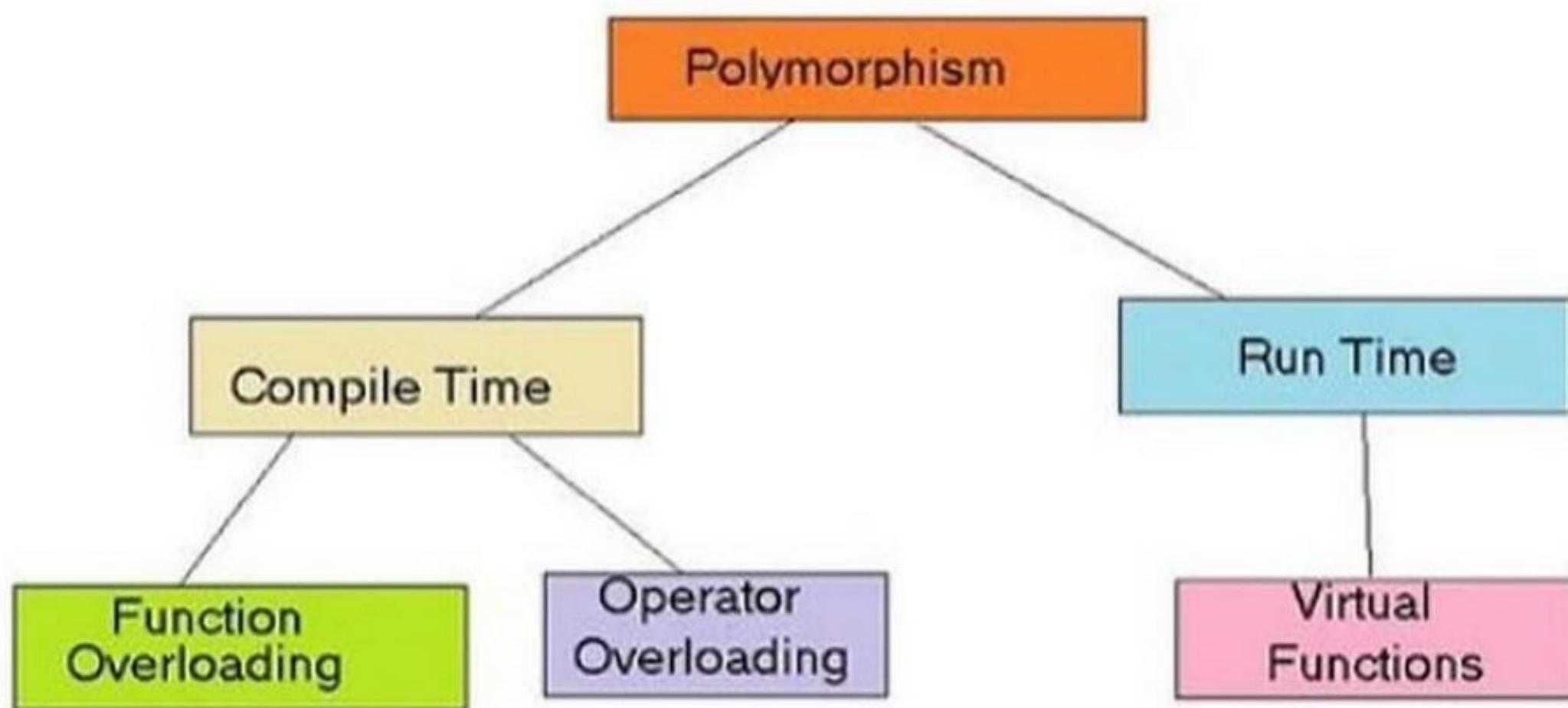
Ans: Inheritance is a mechanism where a new class (subclass or derived class) inherits properties and behaviors from an existing class (superclass or base class).

It promotes code reuse and supports hierarchical relationships.

Q 4. What is polymorphism?

Ans: Polymorphism allows objects of different classes to be treated as instances of a common superclass.

It enables methods to be invoked on objects without knowing their specific types, as long as they adhere to the common interface.



Q 5. What is a constructor?

Ans: A constructor is a special method that is automatically called when an object is created.

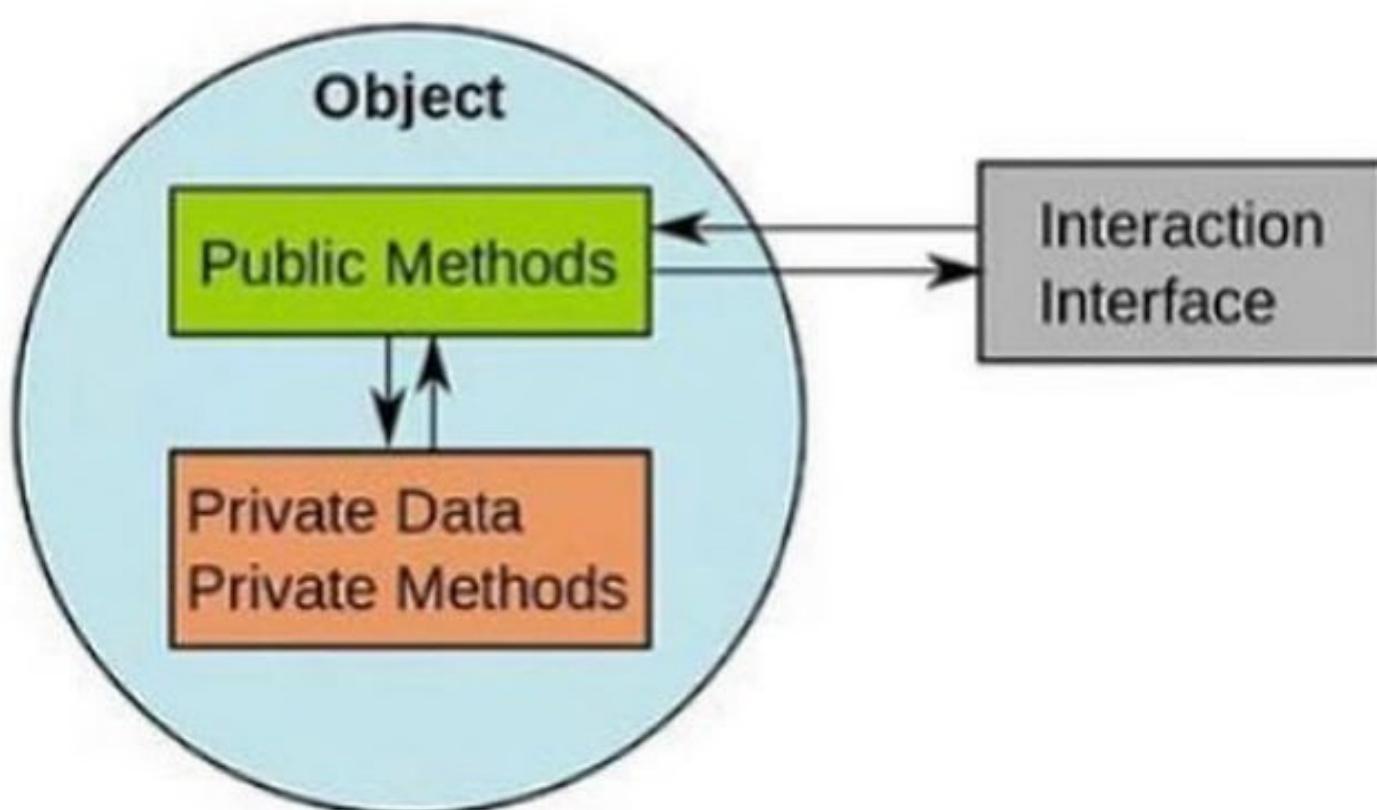
It initializes the object's attributes and prepares the object for use. Constructors often have the same name as the class.

Q 9. What is an interface?

Ans: An interface is a contract that defines a set of methods that a class must implement.

It allows multiple classes to adhere to the same interface, promoting a form of multiple inheritance.

Interfaces only declare method signatures, not implementations.



Q 10. What is a static method?

Ans: A static method belongs to the class itself, not to instances of the class.

It can be called using the class name and is often used for utility functions or operations that don't require instance-specific data.



Q 6. What is method overloading?

Ans: Method overloading is the ability to define multiple methods in a class with the same name but different parameter lists.

The methods are differentiated based on the number or types of parameters they accept.

Q 7. What is method overriding?

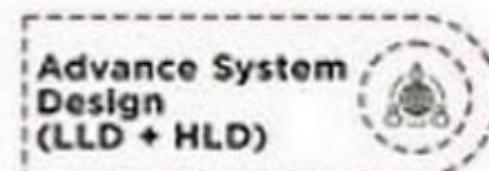
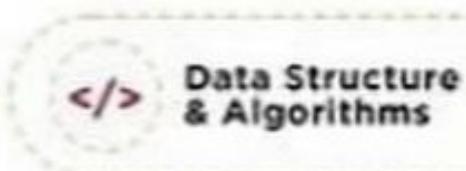
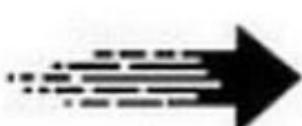
Ans: Method overriding is the process by which a subclass provides a specific implementation for a method that is already defined in its superclass.

The overridden method in the subclass has the same name, return type, and parameters.

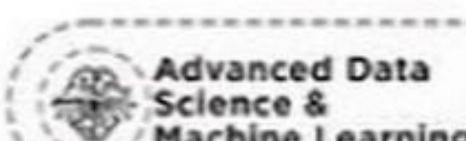
Q 8. What is an abstract class?

Ans: An abstract class is a class that cannot be instantiated on its own.

It may contain abstract methods (methods without a body) that must be implemented by its concrete subclasses. Abstract classes provide a common interface.



Explore Our Popular Courses



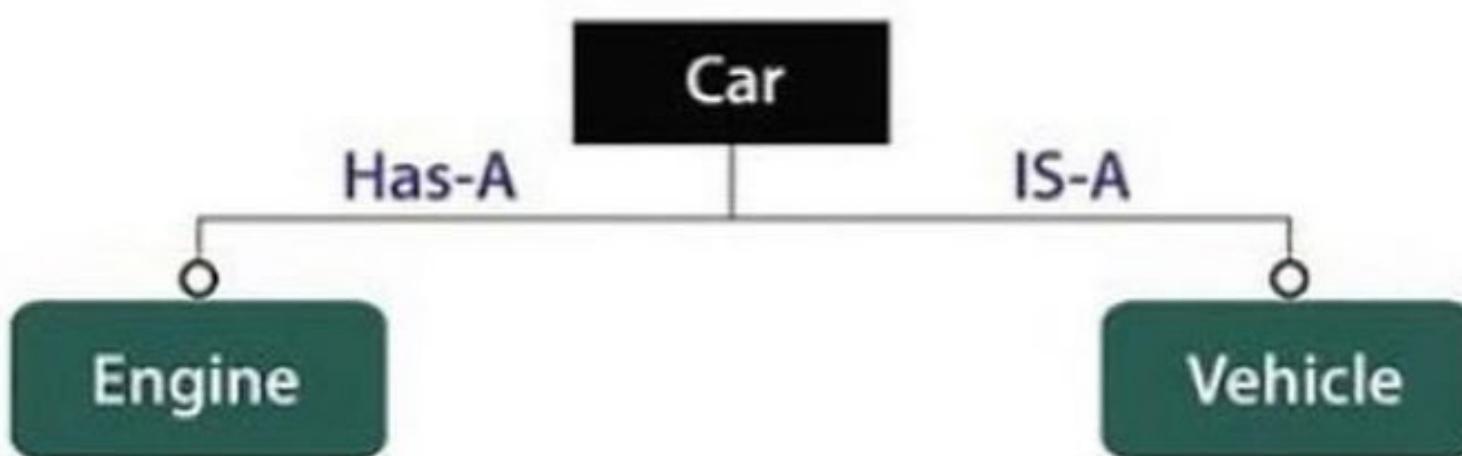
Q 11. What is a final class?

Ans: A final class is a class that cannot be subclassed. It prevents other classes from extending it and inheriting its behavior.

Q 12. What is composition?

Ans: Composition is a design principle where a class contains objects of other classes as part of its attributes.

It allows creating complex structures by combining simpler classes.



Q 13. What is a super keyword?

Ans: The **super keyword** is used to refer to the parent class or superclass. It can be used to call methods and constructors from the superclass.

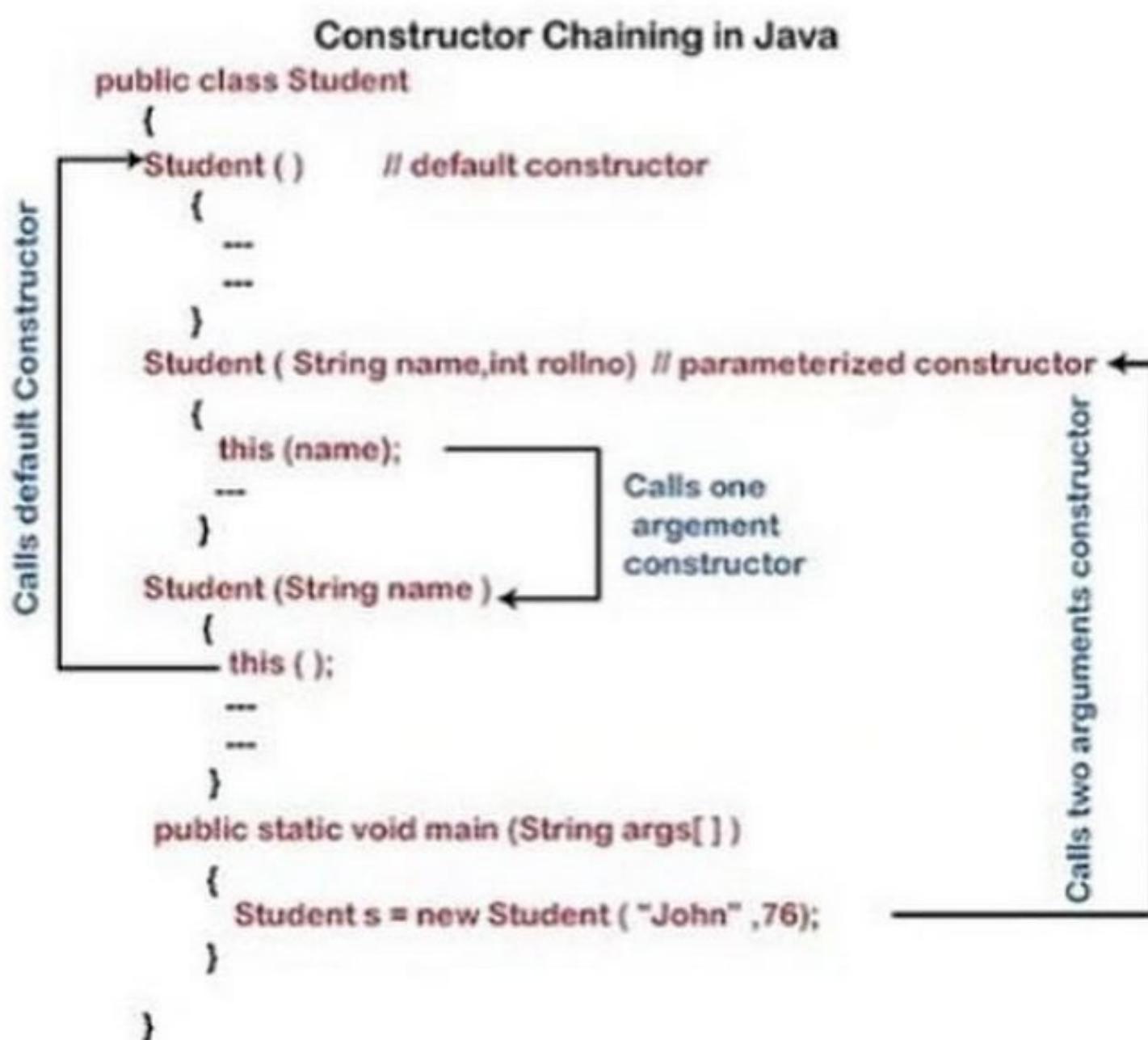
Q 14. What is method hiding?

Ans: Feature engineering involves transforming raw data into meaningful features to improve model performance.

Q 15. What is a constructor chaining?

Ans: Constructor chaining is the process of calling one constructor from another within the same class or between a superclass and a subclass.

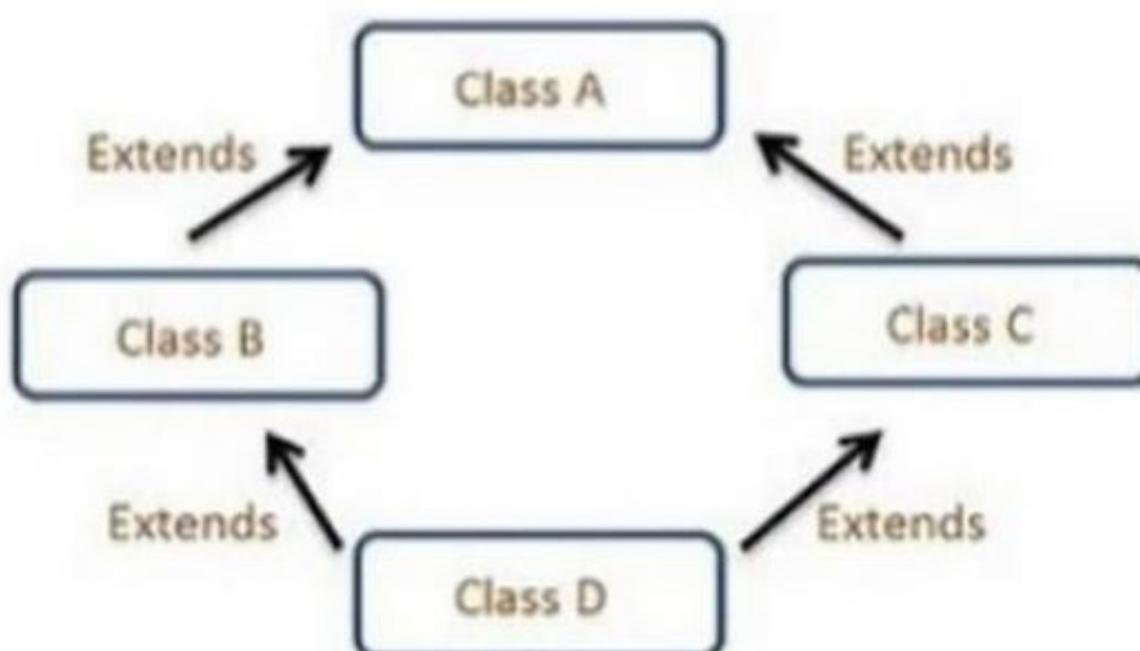
It allows for code reuse and efficient initialization.



Q 16. What is the diamond problem in multiple inheritance?

Ans: The diamond problem occurs in languages that support multiple inheritance, where a class inherits from two classes that have a common base class.

This can lead to ambiguity in method resolution. Some languages provide mechanisms to handle this, like virtual inheritance.



Q 17. What is a shallow copy and a deep copy?

Ans: A shallow copy copies the references of objects contained within an object, while a deep copy creates new instances of the objects contained.

A deep copy results in a completely independent copy of the original object and its contained objects.

Q 21. What is the SOLID principle?

Ans: The SOLID principle is an acronym for a set of design principles that promote maintainable and scalable code:



Q 22. What is the difference between an abstract class and an interface?

Ans: An abstract class can have both abstract and concrete methods, while an interface only has method signatures (no method implementations).

A class can implement multiple interfaces, but it can inherit from only one abstract class.



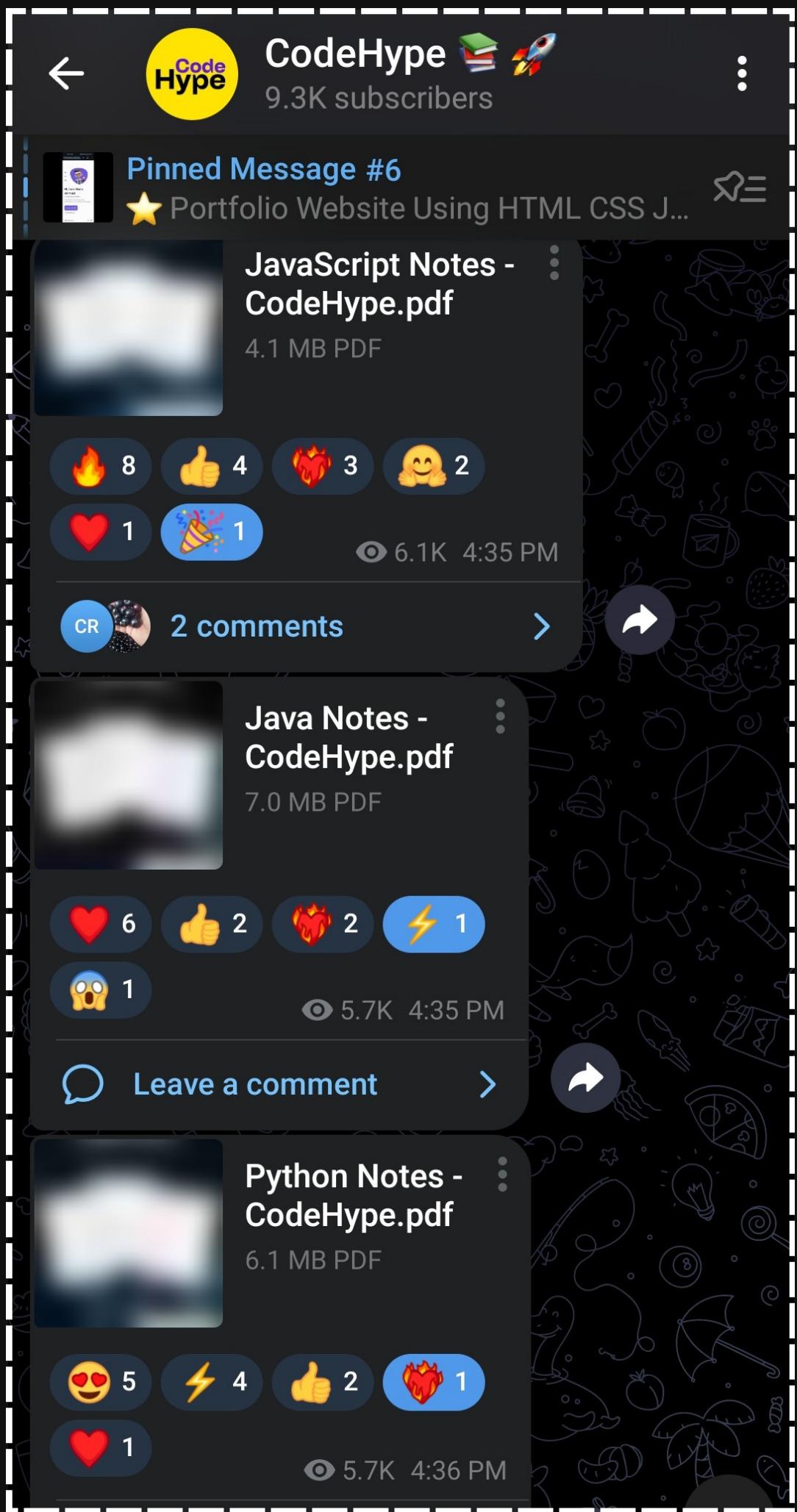
Follow for more useful content



Telegram
@codehype

Join Telegram for
FREE Coding
Resources

CodeHype



Java programming

What is Java?

Java is a **programming language** and a **platform**. Java is a high level, robust, object-oriented and secure programming language.

Java was developed by Sun Microsystems (which is now the subsidiary of Oracle) in the year 1995. James Gosling is known as the father of Java.

Platform: Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.

Application

1. Desktop Applications such as acrobat reader, media player, antivirus, etc.
2. Web Applications such as irctc.co.in, javatpoint.com, etc.
3. Enterprise Applications such as banking applications.
4. Mobile
5. Embedded System
6. Smart Card
7. Robotics
8. Games, etc.

Features of Java

Simple

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun Microsystem, Java language is a simple programming language because:

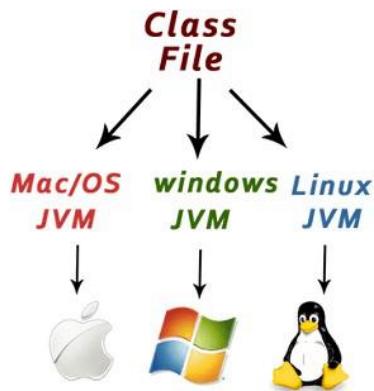
- Java syntax is based on C++ (so easier for programmers to learn it after C++).

- Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
- There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

Object-oriented

Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporate both data and behaviour.

Platform Independent



Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.

Secured

Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

- **No explicit pointer**
- **Java Programs run inside a virtual machine sandbox**

Robust

The English mining of Robust is strong. Java is robust because:

- It uses strong memory management.
- There is a lack of pointers that avoids security problems.
- Java provides automatic garbage collection which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.

Architecture-natural

Java is architecture natural because there are no implementation dependent features, for example, the size of primitive types is fixed.

Portable

Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

High-performance

Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code.

Distributed

Java is distributed because it facilitates users to create distributed applications in Java.

Multi-threaded

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads.

Dynamic

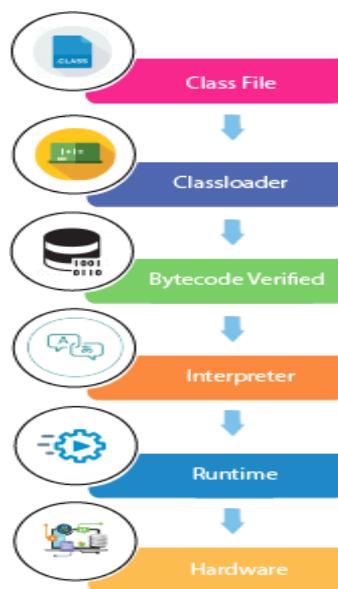
Java is a dynamic language. It supports the dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.

First Java Program

- **class** keyword is used to declare a class in Java.
- **public** keyword is an access modifier that represents visibility. It means it is visible to all.
- **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main() method is executed by the JVM, so it doesn't require creating an object to invoke the main() method. So, it saves memory.
- **void** is the return type of the method. It means it doesn't return any value.
- **main** represents the starting point of the program.
- **String[] args** or **String args[]** is used for command line argument.
- **System.out.println()** is used to print statement. Here, System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class.

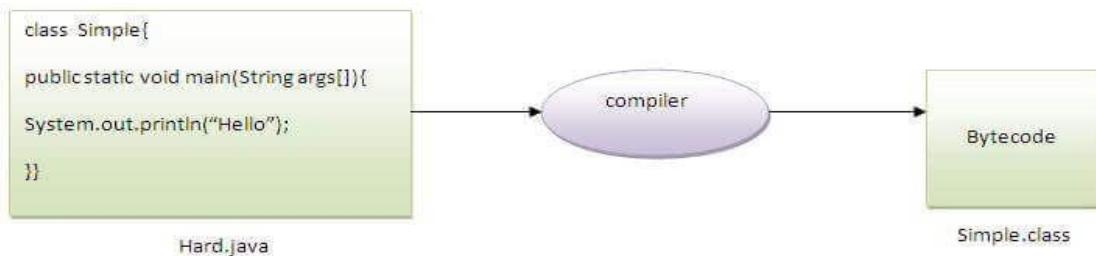
What happens at runtime?

At runtime, the following steps are performed:



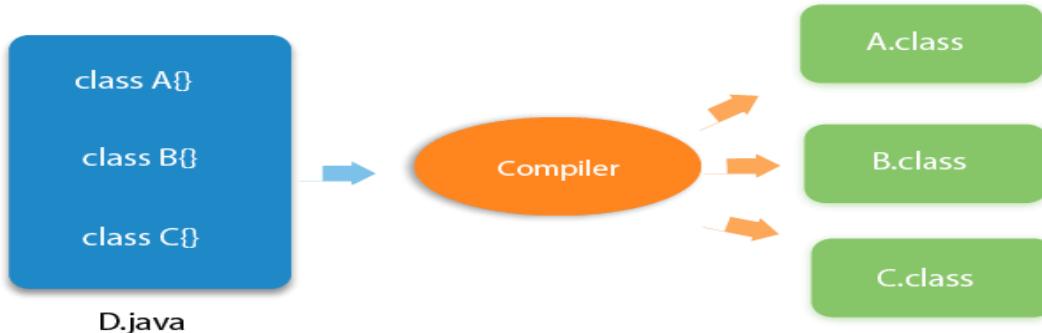
Q) Can you save a Java source file by another name than the class name?

Yes, if the class is not public. It is explained in the figure given below:



Q) Can you have multiple classes in a java source file?

Yes, like the figure given below illustrates:



Difference between JDK, JRE, and JVM

JVM

JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist. It is a specification that provides a runtime environment

JRE

JRE is an acronym for Java Runtime Environment. It is also written as Java RTE. 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.

JDK

JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets. It physically exists. It contains JRE + development tools.

Java Variables

A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type.

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

1) Local Variable

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "static" keyword.

2) Instance Variable

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

3) Static variable

A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class.

Data Types in Java

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include [Classes](#), [Interfaces](#), and [Arrays](#).

Unicode System

Unicode is a universal international standard character encoding that is capable of representing most of the world's written languages.

Why java uses Unicode System?

Before Unicode, there were many language standards:

- **ASCII** (American Standard Code for Information Interchange) for the United States.
- **ISO 8859-1** for Western European Language.
- **KOI-8** for Russian.
- **GB18030 and BIG-5** for Chinese, and so on.

Problem

This caused two problems:

1. A particular code value corresponds to different letters in the various language standards.
2. The encodings for languages with large character sets have variable length.
3. Some common characters are encoded as single bytes, other require two or more byte.

Solution

To solve these problems, a new language standard was developed i.e. Unicode System.

In unicode, character holds 2 byte, so java also uses 2 byte for characters.

lowest value:\u0000

highest value:\uFFFF

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.

Java Keywords

Java keywords are also known as reserved words. Keywords are particular words that act as a key to a code. These are predefined words by Java so they cannot be used as a variable or object name or class name.

Java Control Statements | Control Flow in Java

Java provides three types of control flow statements.

1. Decision Making statements

- if statements
- switch statement

2. Loop statements

- do while loop
- while loop
- for loop
- for-each loop

3. Jump statements

- break statement
- continue statement

Decision-Making statements:

As the name suggests, decision-making statements decide which statement to execute and when.

1) If Statement:

In Java, the "if" statement is used to evaluate a condition. The control of the program is diverted depending upon the specific condition. The condition of the If statement gives a Boolean value, either true or false.

2) if-else statement

The if-else statement is an extension to the if-statement, which uses another block of code, i.e., else block.

3) if-else-if ladder:

The if-else-if statement contains the if-statement followed by multiple else-if statements. In other words, we can say that it is the chain of if-else

statements that create a decision tree where the program may enter in the block of code where the condition is true.

4. Nested if-statement

In nested if-statements, the if statement can contain a **if** or **if-else** statement inside another if or else-if statement.

Switch Statement:

In Java, **Switch statements** are similar to if-else-if statements. The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable which is being switched.

Loop Statements

In programming, sometimes we need to execute the block of code repeatedly while some condition evaluates to true. However, loop statements are used to execute the set of instructions in a repeated order.

1. for loop
2. while loop
3. do-while loop

Java for loop

In Java, **for loop** is similar to **C** and **C++**. It enables us to initialize the loop variable, check the condition, and increment/decrement in a single line of code. We use the for loop only when we exactly know the number of times, we want to execute the block of code.

Java for-each loop

Java provides an enhanced for loop to traverse the data structures like array or collection. In the for-each loop, we don't need to update the loop variable. The syntax to use the for-each loop in java is given below.

```
for(data_type var : array_name/collection_name){  
    //statements
```

}

Java while loop

The while loop is also used to iterate over the number of statements multiple times. However, if we don't know the number of iterations in advance, it is recommended to use a while loop.

Java do-while loop

The do-while loop checks the condition at the end of the loop after executing the loop statements. When the number of iteration is not known and we have to execute the loop at least once, we can use do-while loop.

It is also known as the exit-controlled loop since the condition is not checked in advance. The syntax of the do-while loop is given below.

```
do
{
    //statements
} while (condition);
```

Jump Statements

Jump statements are used to transfer the control of the program to the specific statements. In other words, jump statements transfer the execution control to the other part of the program.

Java break statement

As the name suggests, the break statement is used to break the current flow of the program and transfer the control to the next statement outside a loop or switch statement.

Java continue statement

Unlike break statement, the continue statement doesn't break the loop, whereas, it skips the specific part of the loop and jumps to the next iteration of the loop immediately.

Java Object Class

Java OOPs Concepts

Object-Oriented Programming is a paradigm that provides many concepts, such as **inheritance**, **data binding**, **polymorphism**, etc.

Object

Any entity that has state and behaviour is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

Class

Collection of objects is called class. It is a logical entity.

Inheritance

When one object acquires all the properties and behaviours of a parent object, it is known as inheritance. It provides code reusability.

Polymorphism

If one task is performed in different ways, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing.

Encapsulation

Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines.

Coupling

Coupling refers to the knowledge or information or dependency of another class. It arises when classes are aware of each other.

Cohesion

Cohesion refers to the level of a component which performs a single well-defined task. A single well-defined task is done by a highly cohesive method.

Association

Association represents the relationship between the objects. Here, one object can be associated with one object or many objects.

Aggregation

Aggregation is a way to achieve Association. Aggregation represents the relationship where one object contains other objects as a part of its state.

Composition

The composition is also a way to achieve Association. The composition represents the relationship where one object contains other objects as a part of its state.

Constructors in Java

In [Java](#), a constructor is a block of codes similar to the method. It is called when an instance of the [class](#) is created. At the time of calling constructor, memory for the object is allocated in the memory.

Constructor Overloading in Java

Constructor [overloading in Java](#) is a technique of having more than one constructor with different parameter lists.

Singleton design pattern in Java

Singleton Pattern says that just "**define a class that has only one instance and provides a global point of access to it**".

In other words, a class must ensure that only single instance should be created and single object can be used by all other classes.

There are two forms of singleton design pattern

- **Early Instantiation:** creation of instance at load time.
- **Lazy Instantiation:** creation of instance when required.

Advantage of Singleton design pattern

- Saves memory because object is not created at each request. Only single instance is reused again and again.

Private Constructor in Java

Java allows us to declare a constructor as private. We can declare a constructor private by using the **private** access specifier. Note that if a constructor is declared private, we are not able to create an object of the class. Instead, we can use this private constructor in **Singleton Design Pattern**.

Rules for Private Constructor

The following rules keep in mind while dealing with private constructors.

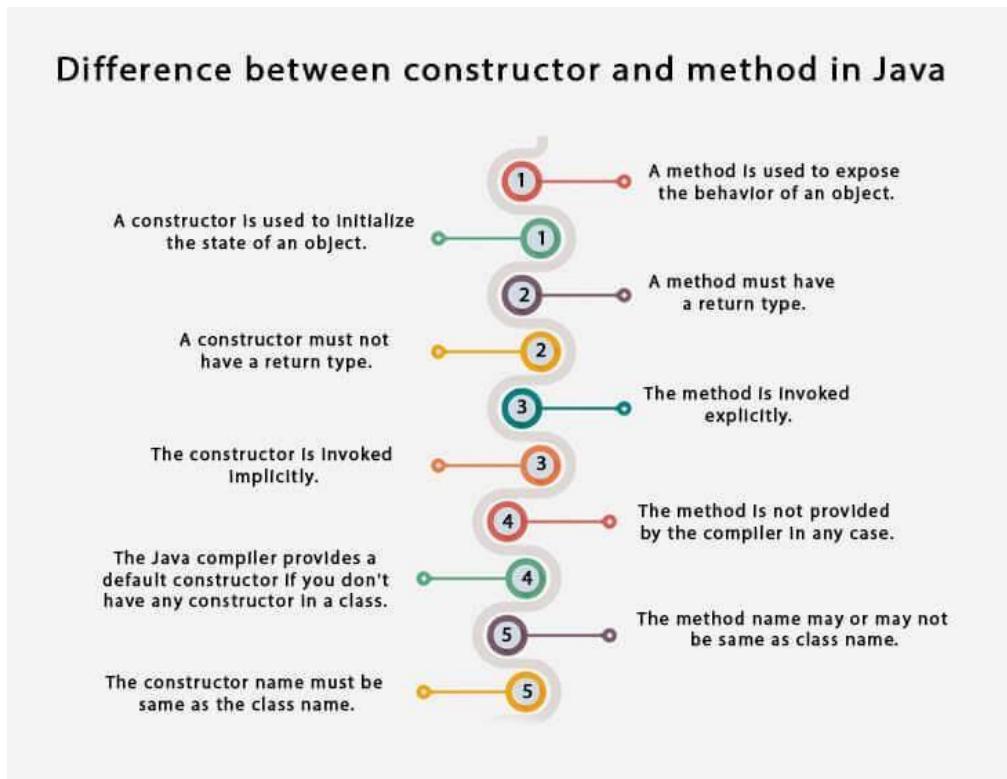
- It does not allow a class to be sub-classed.
- It does not allow to create an object outside the class.
- If a class has a private constructor and when we try to extend the class, a compile-time error occurs.
- We cannot access a private constructor from any other class.
- If all the constant methods are there in our class, we can use a private constructor.

Use Cases of Private Constructor

The main purpose of using a private constructor is **to restrict object creation**. We also use private constructors to implement the singleton design pattern. The use-cases of the private constructor are as follows:

- It can be used with static members-only classes.
- It can be used with static utility or constant classes.
- It can also be used to create singleton classes.
- It can be used to assign a name, for instance, creation by utilizing factory methods.
- It is also used to avoid sub-classing.

Different b/w Constructor and Method



Java static keyword

The **static keyword** in Java is used for memory management mainly. We can apply static keyword with variables, methods, blocks and nested

classes. The static keyword belongs to the class than an instance of the class.

1) Java static variable

If you declare any variable as static, it is known as a static variable.

- The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.

- The static variable gets memory only once in the class area at the time of class loading.

2) Java static method

If you apply static keyword with any method, it is known as static method.

- A static method belongs to the class rather than the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data member and can change the value of it.

Q) Why is the Java main method static?

Ans) It is because the object is not required to call a static method. If it were a non-static method, JVM creates an object first then call main() method that will lead the problem of extra memory allocation.

3) Java static block

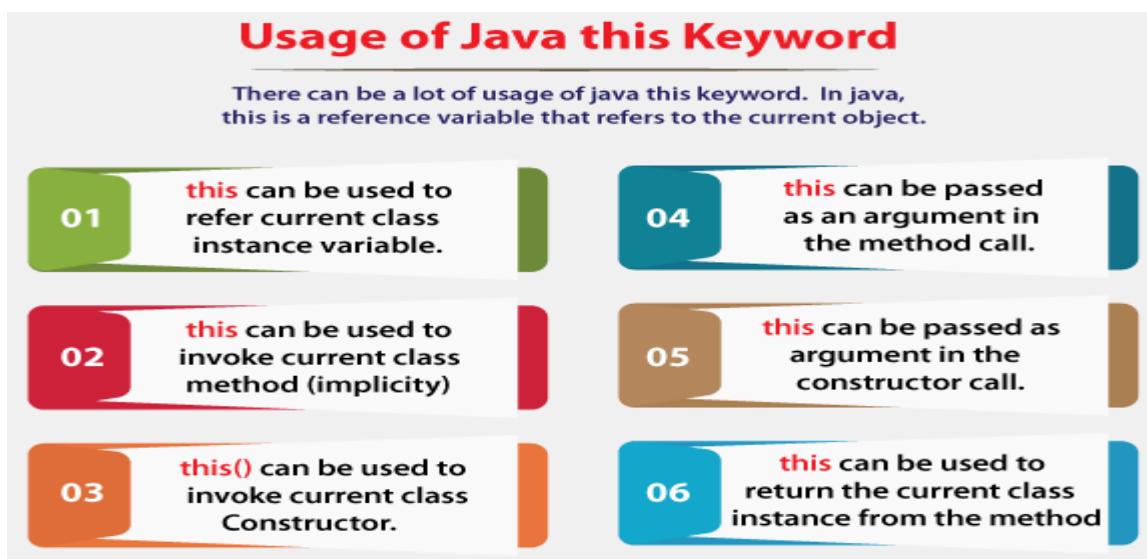
- Is used to initialize the static data member.
- It is executed before the main method at the time of classloading.

Q) Can we execute a program without main() method?

Ans) No, one of the ways was the static block, but it was possible till JDK 1.6. Since JDK 1.7, it is not possible to execute a Java class without the main method.

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.



Java Inheritance

Inheritance in Java

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

Inheritance represents the **IS-A relationship** which is also known as a parent-child relationship.

Why use inheritance in java

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

- For Code Reusability.

Types of inheritance in java

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

1) Single Inheritance

When a class inherits another class, it is known as a single inheritance.

2) Multilevel Inheritance Example

When there is a chain of inheritance, it is known as multilevel inheritance.

3) Hierarchical Inheritance Example

When two or more classes inherit a single class, it is known as hierarchical inheritance.

Q) Why multiple inheritance is not supported in java?

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

Aggregation in Java

If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.

Method Overloading in Java

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

If we have to perform only one operation, having same name of the methods increases the readability of the program.

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.

Q) Can we overload java main() method?

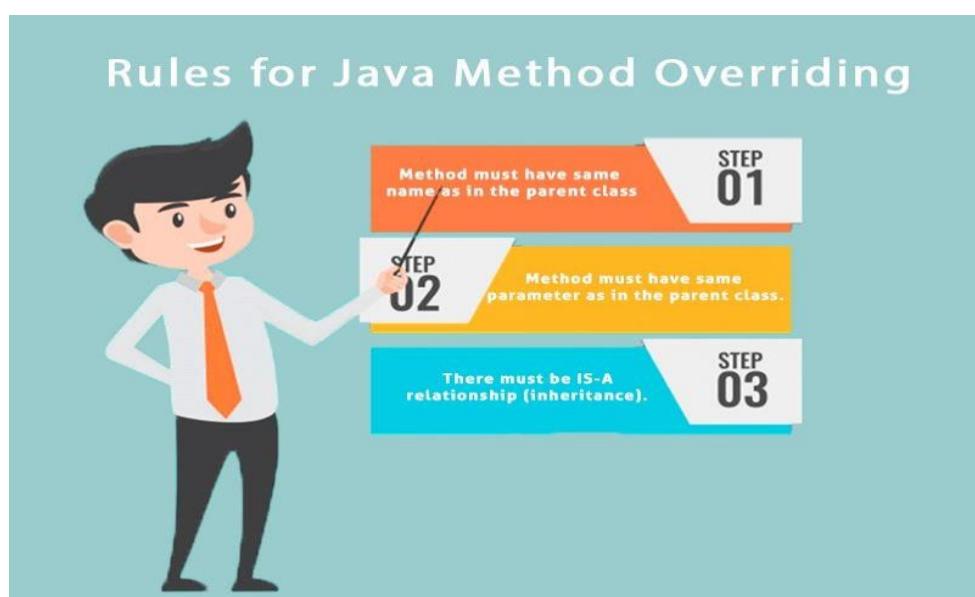
Yes, by method overloading. You can have any number of main methods in a class by method overloading. But JVM calls main() method which receives string array as arguments only.

Java Polymorphism

Method Overriding in Java

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.

In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

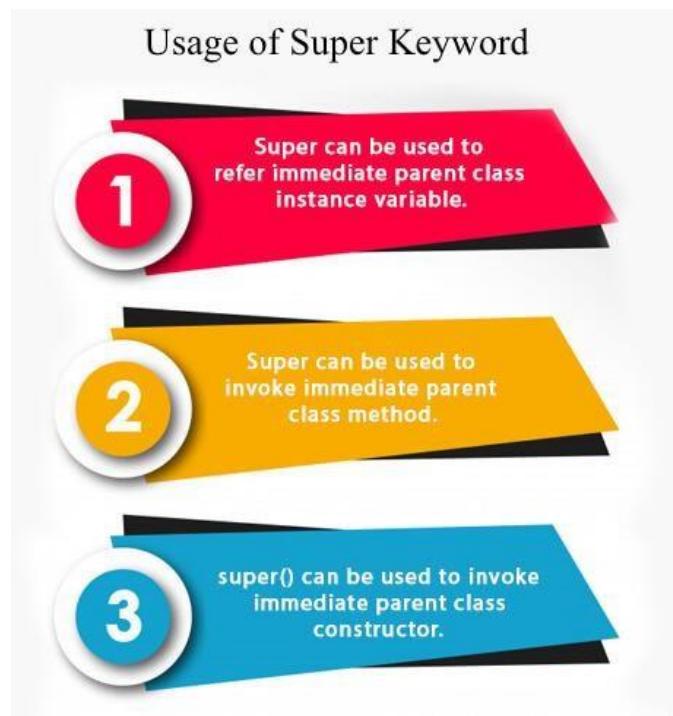


Covariant Return Type

The covariant return type specifies that the return type may vary in the same direction as the subclass.

Super Keyword in Java

The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.



Instance initializer block

Instance Initializer block is used to initialize the instance data member. It runs each time when object of the class is created.

Final Keyword in Java

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many contexts. Final can be:

1. variable
2. method

3. class

1) Java final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).



2) Java final method

If you make any method as final, you cannot override it.

3) Java final class

If you make any class as final, you cannot extend it.

Q) Is final method inherited?

Ans) Yes, final method is inherited but you cannot override it.

Q) What is blank or uninitialized final variable?

A final variable that is not initialized at the time of declaration is known as blank final variable.

Q) Can we initialize blank final variable?

Yes, but only in constructor.

Q) What is final parameter?

If you declare any parameter as final, you cannot change the value of it.

Q) Can we declare a constructor final?

No, because constructor is never inherited.

Polymorphism in Java

Polymorphism in Java is a concept by which we can perform a single action in different ways.

There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

Runtime Polymorphism 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 compile-time.

Upcasting

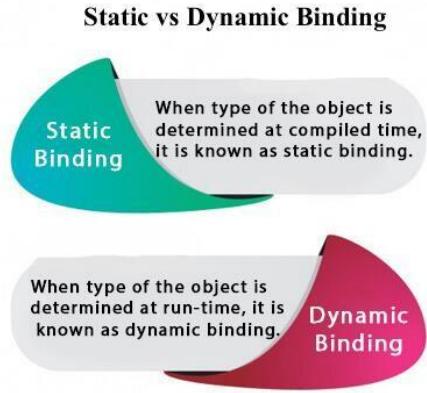
If the reference variable of Parent class refers to the object of Child class, it is known as upcasting.

Static Binding and Dynamic Binding

Connecting a method call to the method body is known as binding.

There are two types of binding

1. Static Binding (also known as Early Binding).
2. Dynamic Binding (also known as Late Binding).



static binding

When type of the object is determined at compiled time (by the compiler), it is known as static binding.

Dynamic binding

When type of the object is determined at run-time, it is known as dynamic binding.

Java instanceof

The **java instanceof operator** is used to test whether the object is an instance of the specified type (class or subclass or interface).

The instanceof in java is also known as type comparison operator because it compares the instance with type. It returns either true or false.

Java Abstraction

Abstract class in Java

A class which is declared with the abstract keyword is known as an abstract class in [Java](#). It can have abstract and non-abstract methods (method with the body).

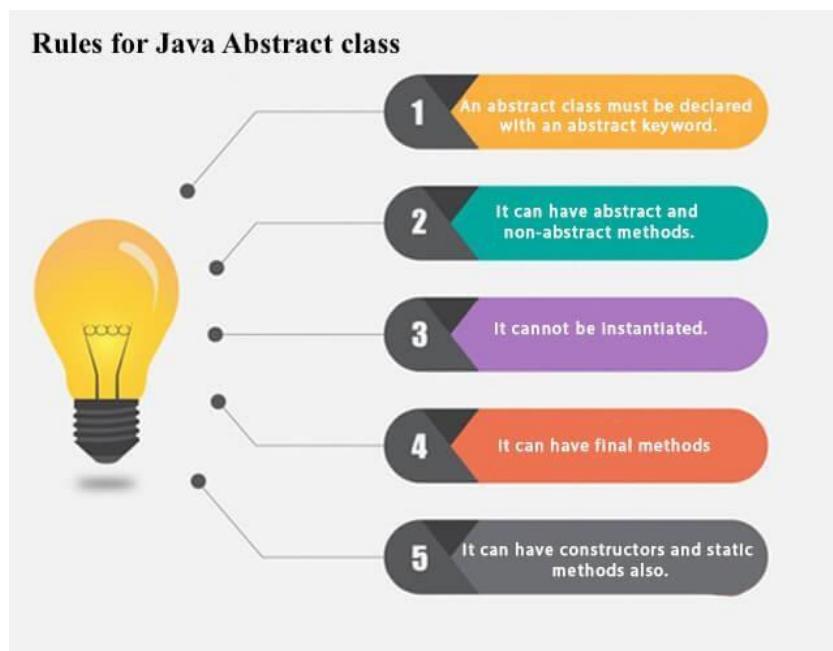
Abstraction in Java

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)



Abstract Method in Java

A method which is declared as abstract and does not have implementation is known as an abstract method.

Interface in Java

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

Why use Java interface?

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

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

Syntax:

```
interface <interface_name>{
```

```
    // declare constant fields  
    // declare methods that abstract  
    // by default.  
}
```

Q) What is marker or tagged interface?

An interface which has no member is known as a marker or tagged interface, for example, Serializable, Cloneable, Remote, etc. They are used to provide some essential information to the JVM so that JVM may perform some useful operation.

Difference between abstract class and interface

Abstract class	Interface
1) Abstract class can have abstract and non-abstract methods.	Interface can have only abstract methods. Since Java 8, it can have default and static methods also.
2) Abstract class doesn't support multiple inheritance .	Interface supports multiple inheritance .

3) Abstract class can have final, non-final, static and non-static variables.	Interface has only static and final variables.
4) Abstract class can provide the implementation of interface.	Interface can't provide the implementation of abstract class.
5) The abstract keyword is used to declare abstract class.	The interface keyword is used to declare interface.
6) An abstract class can extend another Java class and implement multiple Java interfaces.	An interface can extend another Java interface only.
7) An abstract class can be extended using keyword "extends".	An interface can be implemented using keyword "implements".
8) A Java abstract class can have class members like private, protected, etc.	Members of a Java interface are public by default.
9) Example: <pre>public abstract class Shape{ public abstract void draw(); }</pre>	Example: <pre>public interface Drawable{ void draw(); }</pre>

Simply, abstract class achieves partial abstraction (0 to 100%) whereas interface achieves fully abstraction (100%).

Java Encapsulation

Java Package

A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, SQL etc.

Sub package in java

Package inside the package is called the sub package. It should be created to categorize the package further.

Access Modifiers in Java

There are two types of modifiers in Java: **access modifiers** and **non-access modifiers**.

There are four types of Java access modifiers:

- Private:** The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
- Default:** The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
- Protected:** The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
- Public:** The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Encapsulation in Java

Encapsulation in Java is a process of wrapping code and data together into a single unit, for example, a capsule which is mixed of several medicines.

Advantage of Encapsulation in Java

By providing only a setter or getter method, you can make the class **read-only or write-only**. In other words, you can skip the getter or setter methods.

It provides you the **control over the data**. Suppose you want to set the value of id which should be greater than 100 only, you can write the logic inside the setter method.

It is a way to achieve **data hiding** in Java because other class will not be able to access the data through the private data members.

Java Array

Java Arrays

Java array is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

Advantages

- **Code Optimization:** It makes the code optimized, we can retrieve or sort the data efficiently.
- **Random access:** We can get any data located at an index position.

Disadvantages

- **Size Limit:** We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.

Types of Array in java

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

Java OOPs Misc.

Object class in Java

The **Object class** is the parent class of all the classes in java by default. In other words, it is the topmost class of java.

The Object class is beneficial if you want to refer any object whose type you don't know. Notice that parent class reference variable can refer the child class object, known as upcasting.

1. `Object obj= getObject(); //we don't know what object will be returned from this method`

Object Cloning in Java

The object cloning is a way to create exact copy of an object. The `clone()` method of Object class is used to clone an object.

The `java.lang.Cloneable` interface must be implemented by the class whose object clone we want to create. If we don't implement `Cloneable` interface, `clone()` method generates `CloneNotSupportedException`.

Why use `clone()` method ?

The **clone() method** saves the extra processing task for creating the exact copy of an object. If we perform it by using the `new` keyword, it will take a lot of processing time to be performed that is why we use object cloning.

Advantage of Object cloning

- You don't need to write lengthy and repetitive codes. Just use an abstract class with a 4- or 5-line long clone() method.
- It is the easiest and most efficient way for copying objects, especially if we are applying it to an already developed or an old project.
- Clone() is the fastest way to copy array.

Disadvantage of Object cloning

- To use the Object.clone() method, we have to change a lot of syntaxes to our code, like implementing a Cloneable interface, defining the clone() method and handling CloneNotSupportedException, and finally, calling Object.clone() etc.
- We have to implement cloneable interface while it doesn't have any methods in it.

Java Math class

Java Math class provides several methods to work on math calculations like min(), max(), avg(), sin(), cos(), tan(), round(), ceil(), floor(), abs() etc.

Unlike some of the Strict Math class numeric methods, all implementations of the equivalent function of Math class can't define to return the bit-for-bit same results.

Wrapper classes in Java

The **wrapper class in Java** provides the mechanism to convert primitive into object and object into primitive.

Since J2SE 5.0, **autoboxing** and **unboxing** feature convert primitives into objects and objects into primitives automatically. The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing.

Autoboxing

The automatic conversion of primitive data type into its corresponding wrapper class is known as autoboxing, for example, byte to Byte, char to

Character, int to Integer, long to Long, float to Float, boolean to Boolean, double to Double, and short to Short.

Unboxing

The automatic conversion of wrapper type into its corresponding primitive type is known as unboxing. It is the reverse process of autoboxing.

Java Strictfp Keyword

Java strictfp keyword ensures that you will get the same result on every platform if you perform operations in the floating-point variable.

Difference between object and class

No.	Object	Class
1)	Object is an instance of a class.	Class is a blueprint or template from which objects are created.
2)	Object is a real world entity such as pen, laptop, mobile, bed, keyboard, mouse, chair etc.	Class is a group of similar objects .
3)	Object is a physical entity .	Class is a logical entity .
4)	Object is created through new keyword mainly e.g. Student s1=new Student();	Class is declared using class keyword e.g. class Student{ }
5)	Object is created many times as per requirement.	Class is declared once .
6)	Object allocates memory when it is created .	Class doesn't allocated memory when it is created .
7)	There are many ways to create object in java such as new keyword, newInstance() method, clone() method, factory method and deserialization.	There is only one way to define class in java using class keyword.

Difference between method overloading and method overriding in java

No.	Method Overloading	Method Overriding
1)	Method overloading is used to increase the readability of the program.	Method overriding is used to provide the specific implementation of the method that is already provided by its super class.
2)	Method overloading is performed within class.	Method overriding occurs in two classes that have IS-A (inheritance) relationship.
3)	In case of method overloading, parameter must be different.	In case of method overriding, parameter must be same.
4)	Method overloading is the example of compile time polymorphism.	Method overriding is the example of run time polymorphism.
5)	In java, method overloading can't be performed by changing return type of the method only. Return type can be same or different in method overloading. But you must have to change the parameter.	Return type must be same or covariant in method overriding.

Java String

Java String

In [Java](#), string is basically an object that represents sequence of char values. An [array](#) of characters works same as Java string. For example:

```
char[] ch={'j','a','v','a','t','p','o','i','n','t'};  
String s=new String(ch);
```

How to create a string object?

There are two ways to create String object:

1. By string literal
2. By new keyword

1) String Literal

Java String literal is created by using double quotes. For Example:

1. String s="welcome";

2) By new keyword

1. String s=new String("Welcome");//creates two objects and one reference variable

Immutable String in Java

A String is an unavoidable type of variable while writing any application program. String references are used to store various attributes like username, password, etc. In Java, **String objects are immutable**. Immutable simply means unmodifiable or unchangeable.

Java String Buffer Class

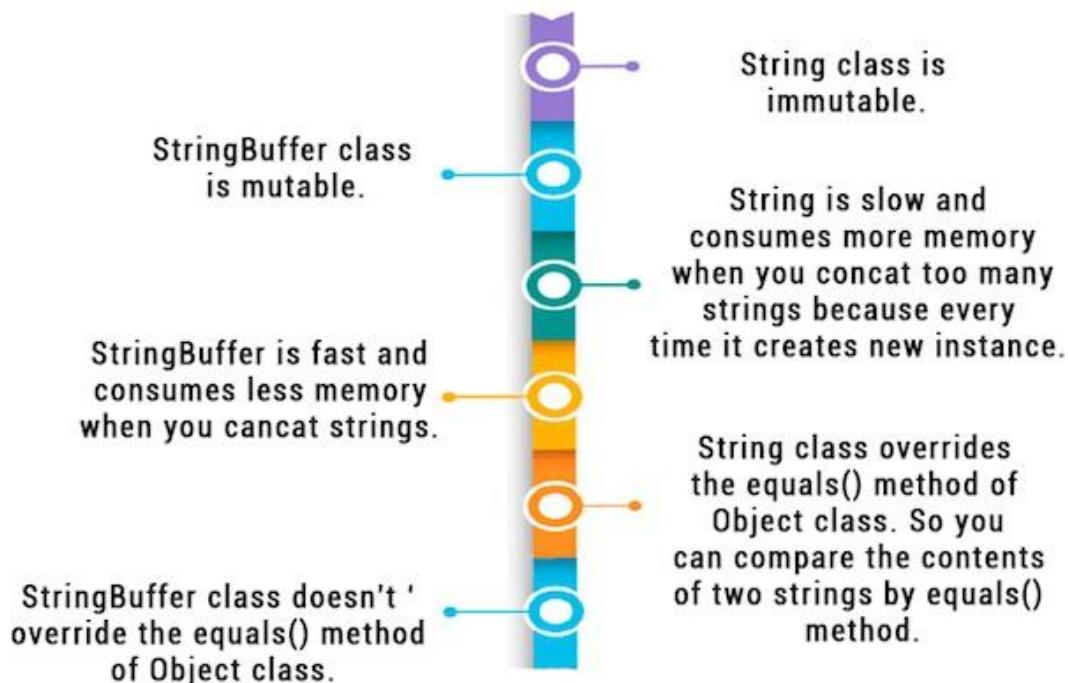
Java String Buffer class is used to create mutable (modifiable) String objects. The String Buffer class in Java is the same as String class except it is mutable i.e. it can be changed.

Java StringBuilder Class

Java StringBuilder class is used to create mutable (modifiable) String. The Java StringBuilder class is same as StringBuffer class except that it is non-synchronized. It is available since JDK 1.5.

Difference between String and String Buffer

StringBuffer vs String



Difference between StringBuffer and StringBuilder

StringBuffer Class	StringBuilder Class
StringBuffer is present in Java.	StringBuilder was introduced in Java 5.
StringBuffer is synchronized. This means that multiple threads cannot call the methods of StringBuffer simultaneously.	StringBuilder is asynchronous. This means that multiple threads can call the methods of StringBuilder simultaneously.

StringBuffer Class	StringBuilder Class
Due to synchronization, StringBuffer is called a thread safe class.	Due to its asynchronous nature, StringBuilder is not a thread safe class.
Due to synchronization, StringBuffer is lot slower than StringBuilder.	Since there is no preliminary check for multiple threads, StringBuilder is a lot faster than StringBuffer.

How to create Immutable class?

There are many immutable classes like String, Boolean, Byte, Short, Integer, Long, Float, Double etc. In short, all the wrapper classes and String class is immutable.

Java **toString()** Method

If you want to represent any object as a string, **toString() method** comes into existence.

The **toString()** method returns the String representation of the object.

If you print any object, Java compiler internally invokes the **toString()** method on the object.

Advantage of Java **toString()** method

By overriding the **toString()** method of the Object class, we can return values of the object, so we don't need to write much code.

StringTokenizer in Java

The **java.util.StringTokenizer** class allows you to break a String into tokens. It is simple way to break a String. It is a legacy class of Java.

Exception Handling

Exception Handling in Java

The **Exception Handling in Java** is one of the powerful mechanism to handle the runtime errors so that the normal flow of the application can be maintained.

What is Exception in Java?

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

Types of Java Exceptions

There are mainly two types of exceptions: checked and unchecked. An error is considered as the unchecked exception. However, according to Oracle, there are three types of exceptions namely:

1. Checked Exception
2. Unchecked Exception
3. Error

Difference between Checked and Unchecked Exceptions

1) Checked Exception

The classes that directly inherit the `Throwable` class except `Runtime Exception` and `Error` are known as checked exceptions. For example, `IOException`, `SQLException`, etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes that inherit the `Runtime Exception` are known as unchecked exceptions. For example, `Arithmetic Exception`, `NullPointerException`,

`ArrayIndexOutOfBoundsException`, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

3) Error

Error is irrecoverable. Some example of errors are `OutOfMemoryError`, `VirtualMachineError`, `AssertionError` etc.

Java try-catch block

Java try block

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

Syntax of Java try-catch

```
try{  
//code that may throw an exception  
}catch(Exception_class_Name ref){ }
```

Java catch block

Java catch block is used to handle the Exception by declaring the type of exception within the parameter.

The catch block must be used after the try block only. You can use multiple catch block with a single try block.

Java Catch Multiple Exceptions

Java Multi-catch block

A try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler. So, if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.

Java Nested try block

In Java, using a try block inside another try block is permitted. It is called as nested try block.

Why use nested try block

Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.

Java finally block

Java finally block is a block used to execute important code such as closing the connection, etc.

Java finally block is always executed whether an exception is handled or not.

Why use Java finally block?

- finally block in Java can be used to put "**cleanup**" code such as closing a file, closing connection, etc.
- The important statements to be printed can be placed in the finally block.

Usage of Java finally

Let's see the different cases where Java finally block can be used.

Java throw keyword

The Java throw keyword is used to throw an exception explicitly.

We can throw either checked or unchecked exceptions in Java by throw keyword. It is mainly used to throw a custom exception.

Java Exception Propagation

Exception propagation in Java occurs when an exception thrown from the top of the stack.

Java throws keyword

The **Java throws keyword** is used to declare an exception. It gives an information to the programmer that there may occur an exception.

Syntax of Java throws

```
return_type method_name() throws exception_class_name{  
//method code  
}
```

Difference between throw and throws in Java

Throw vs Throws

Throw	Throws
Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare an exception.
Checked exception cannot be propagated using 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 exceptions.	You can declare multiple exceptions e.g. public void method() throws IOException,SQLException.

Difference between final, finally and finalize

Sr. no.	Key	final	finally	finalize
1.	Definition	final is the keyword and access modifier which is used to apply restrictions on a class, method or variable.	finally is the block in Java Exception Handling to execute the important code whether the exception occurs or not.	finalize is the method in Java which is used to perform clean up processing just before object is garbage collected.
2.	Applicable to	Final keyword is used with the classes, methods and variables.	Finally block is always related to the try and catch block in exception handling.	finalize() method is used with the objects.
3.	Functionality	(1) Once declared, final variable becomes constant and cannot be modified. (2) final method cannot be overridden by sub class. (3) final class cannot be inherited.	(1) finally block runs the important code even if exception occurs or not. (2) finally block cleans up all the resources used in try block	finalize method performs the cleaning activities with respect to the object before its destruction.
4.	Execution	Final method is executed only when we call it.	Finally block is executed as soon as the try-catch block is executed. It's execution is not dependant on the exception.	finalize method is executed just before the object is destroyed.

Java Custom Exception

Creating our own Exception is known as custom exception or user-defined exception. Basically, Java custom exceptions are used to customize the exception according to user need.

Why use custom exceptions?

Java exceptions cover almost all the general type of exceptions that may occur in the programming.

However, we sometimes need to create custom exceptions.

Java Multithreading

Multithreading in Java

Multithreading in Java is a process of executing multiple threads simultaneously.

A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

Multitasking

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved in two ways:

- Process-based Multitasking (Multiprocessing)
- Thread-based Multitasking (Multithreading)

1) Process-based Multitasking (Multiprocessing)

- Each process has an address in memory. In other words, each process allocates a separate memory area.
- A process is heavyweight.
- Cost of communication between the process is high.

2) Thread-based Multitasking (Multithreading)

- Threads share the same address space.
- A thread is lightweight.
- Cost of communication between the thread is low.

What is Thread in java

A thread is a lightweight subprocess, the smallest unit of processing. It is a separate path of execution.

Threads are independent. If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area.

Life cycle of a Thread (Thread States)

In Java, a thread always exists in any one of the following states. These states are:

1. New
2. Active
3. Blocked / Waiting
4. Timed Waiting
5. Terminated

Explanation of Different Thread States

New: Whenever a new thread is created, it is always in the new state.

Active: When a thread invokes the start() method, it moves from the new state to the active state. The active state contains two states within it: one is **Runnable**, and the other is **running**.

Runnable: A thread, that is ready to run is then moved to the runnable state. In the runnable state, the thread may be running or may be ready to run at any given instant of time.

Running: When the thread gets the CPU, it moves from the runnable to the running state.

Blocked or Waiting: Whenever a thread is inactive for a span of time (not permanently) then, either the thread is in the blocked state or is in the waiting state.

Terminated: A thread reaches the termination state because of the following reasons:

- When a thread has finished its job, then it exists or terminates normally.

- **Abnormal termination:** It occurs when some unusual events such as an unhandled exception or segmentation fault.

Java Threads | How to create a thread in Java

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

Thread class:

Thread class provide constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface.

Commonly used Constructors of Thread class:

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r, String name)

Commonly used methods of Thread class:

1. **public void run():** is used to perform action for a thread.
2. **public void start():** starts the execution of the thread. JVM calls the run() method on the thread.
3. **public void sleep(long miliseconds):** Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
4. **public void join():** waits for a thread to die.
5. **public void join(long miliseconds):** waits for a thread to die for the specified miliseconds.
6. **public int getPriority():** returns the priority of the thread.
7. **public int setPriority(int priority):** changes the priority of the thread.

8. **public String getName():** returns the name of the thread.
9. **public void setName(String name):** changes the name of the thread.
10. **public Thread currentThread():** returns the reference of currently executing thread.
11. **public int getId():** returns the id of the thread.
12. **public Thread.State getState():** returns the state of the thread.
13. **public boolean isAlive():** tests if the thread is alive.
14. **public void yield():** causes the currently executing thread object to temporarily pause and allow other threads to execute.
15. **public void suspend():** is used to suspend the thread(deprecated).
16. **public void resume():** is used to resume the suspended thread(deprecated).
17. **public void stop():** is used to stop the thread(deprecated).
18. **public boolean isDaemon():** tests if the thread is a daemon thread.
19. **public void setDaemon(boolean b):** marks the thread as daemon or user thread.
20. **public void interrupt():** interrupts the thread.
21. **public boolean isInterrupted():** tests if the thread has been interrupted.
22. **public static boolean interrupted():** tests if the current thread has been interrupted.

Runnable interface:

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().

1. **public void run():** is used to perform action for a thread.

Starting a thread:

The **start()** method of Thread class is used to start a newly created thread. It performs the following tasks:

- A new thread starts (with new callstack).
- The thread moves from New state to the Runnable state.
- When the thread gets a chance to execute, its target run() method will run.

Thread Scheduler in Java

A component of Java that decides which thread to run or execute and which thread to wait is called a **thread scheduler in Java**. In Java, a thread is only chosen by a thread scheduler if it is in the runnable state. However, if there is more than one thread in the runnable state, it is up to the thread scheduler to pick one of the threads and ignore the other ones.

Priority: Priority of each thread lies between 1 to 10. If a thread has a higher priority, it means that thread has got a better chance of getting picked up by the thread scheduler.

Time of Arrival: Suppose two threads of the same priority enter the runnable state, then priority cannot be the factor to pick a thread from these two threads. In such a case, **arrival time** of thread is considered by the thread scheduler.

Thread Scheduler Algorithms

On the basis of the above-mentioned factors, the scheduling algorithm is followed by a Java thread scheduler.

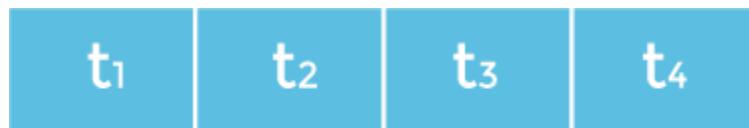
First Come First Serve Scheduling:

In this scheduling algorithm, the scheduler picks the threads that arrive first in the runnable queue. Observe the following table:

Threads	Time of Arrival
t1	0
t2	1

t3	2
t4	3

In the above table, we can see that Thread t1 has arrived first, then Thread t2, then t3, and at last t4, and the order in which the threads will be processed is according to the time of arrival of threads.

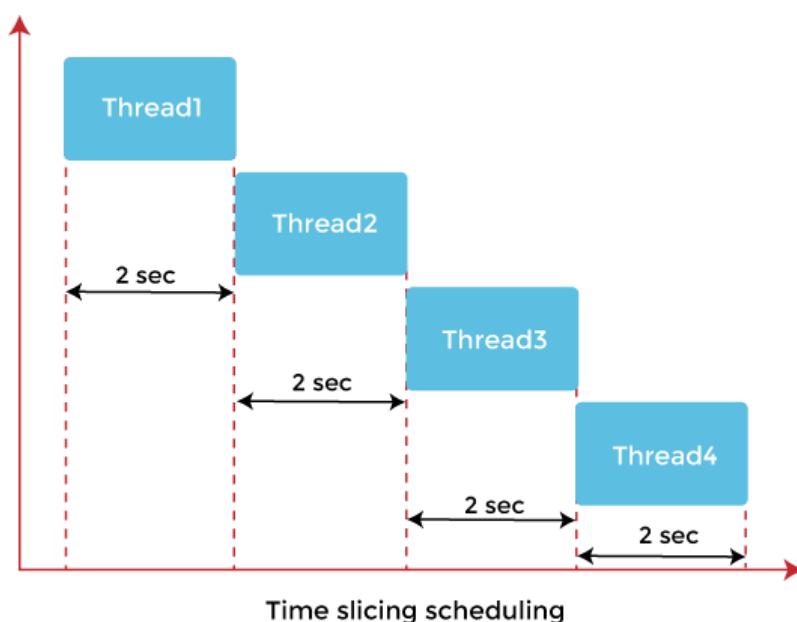


First Come First Serve Scheduling

Hence, Thread t1 will be processed first, and Thread t4 will be processed last.

Time-slicing scheduling:

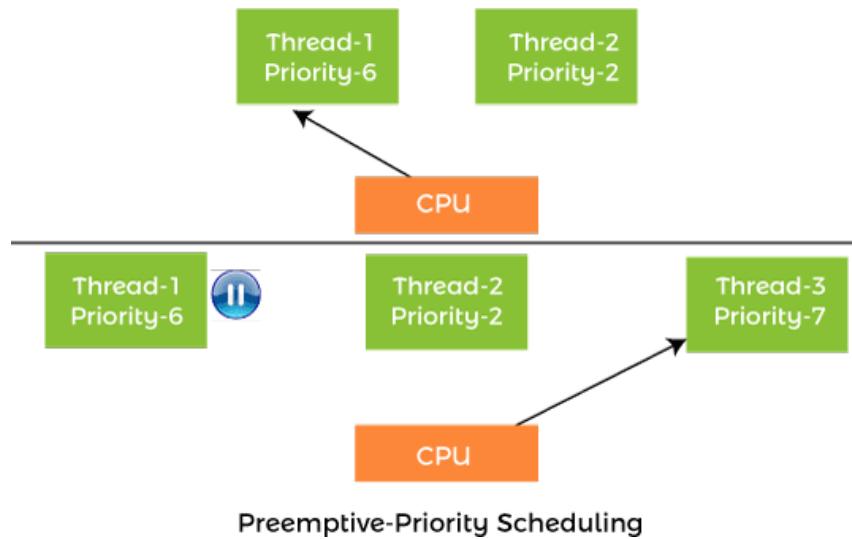
Usually, the First Come First Serve algorithm is non-preemptive, which is bad as it may lead to infinite blocking (also known as starvation). To avoid that, some time-slices are provided to the threads so that after some time, the running thread has to give up the CPU. Thus, the other waiting threads also get time to run their job.



In the above diagram, each thread is given a time slice of 2 seconds. Thus, after 2 seconds, the first thread leaves the CPU, and the CPU is then captured by Thread2. The same process repeats for the other threads too.

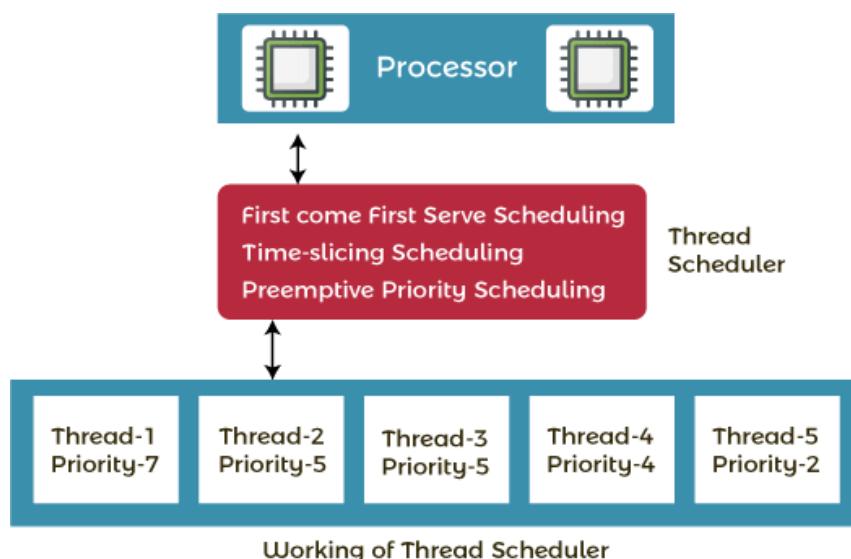
Preemptive-Priority Scheduling:

The name of the scheduling algorithm denotes that the algorithm is related to the priority of the threads.



Suppose there are multiple threads available in the runnable state. The thread scheduler picks that thread that has the highest priority.

Working of the Java Thread Scheduler



Let's understand the working of the Java thread scheduler. Suppose, there are five threads that have different arrival times and different priorities. Now, it is the responsibility of the thread scheduler to decide which thread will get the CPU first.

The thread scheduler selects the thread that has the highest priority, and the thread begins the execution of the job. If a thread is already in runnable state and another thread (that has higher priority) reaches in the runnable state, then the current thread is pre-empted from the processor, and the arrived thread with higher priority gets the CPU time.

When two threads (Thread 2 and Thread 3) having the same priorities and arrival time, the scheduling will be decided on the basis of FCFS algorithm. Thus, the thread that arrives first gets the opportunity to execute first.

Thread. Sleep() in Java

The Java Thread class provides the two variant of the sleep() method. First one accepts only an arguments, whereas the other variant accepts two arguments. The method sleep() is being used to halt the working of a thread for a given amount of time. The time up to which the thread remains in the sleeping state is known as the sleeping time of the thread. After the sleeping time is over, the thread starts its execution from where it has left.

The sleep() Method Syntax:

Following are the syntax of the sleep() method.

1. **public static void sleep(`long` mls) throws InterruptedException**
2. **public static void sleep(`long` mls, `int` n) throws InterruptedException**

Can we start a thread twice

No. After starting a thread, it can never be started again. If you does so, an *IllegalThreadStateException* is thrown. In such case, thread will run once but for second time, it will throw exception.

What if we call Java run() method directly instead start() method?

- Each thread starts in a separate call stack.
- Invoking the run() method from the main thread, the run() method goes onto the current call stack rather than at the beginning of a new call stack.

Java join() method

The join() method in Java is provided by the `java.lang.Thread` class that permits one thread to wait until the other thread to finish its execution. Suppose `th` be the object the class Thread whose thread is doing its execution currently, then the `th.join();` statement ensures that `th` is finished before the program does the execution of the next statement. When there are more than one thread invoking the join() method, then it leads to overloading on the join() method that permits the developer or programmer to mention the waiting period.

Description of The Overloaded join() Method

join(): When the join() method is invoked, the current thread stops its execution and the thread goes into the wait state. The current thread remains in the wait state until the thread on which the join() method is invoked has achieved its dead state.

Syntax:

1. **public final void** join() **throws** InterruptedException

Naming Thread and Current Thread

Naming Thread

The Thread class provides methods to change and get the name of a thread. By default, each thread has a name, i.e. `thread-0`, `thread-1` and so on. By we can change the name of the thread by using the `setName()` method. The syntax of `setName()` and `getName()` methods are given below:

1. **public** String getName(): is used to **return** the name of a thread.
2. **public void** setName(String name): is used to change the name of a thread.

Current Thread

The currentThread() method returns a reference of the currently executing thread.

```
public static Thread currentThread();
```

Priority of a Thread (Thread Priority)

Each thread has a priority. Priorities are represented by a number between 1 and 10. In most cases, the thread scheduler schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.

Setter & Getter Method of Thread Priority

public final int getPriority(): The java.lang.Thread.getPriority() method returns the priority of the given thread.

public final void setPriority(int newPriority): The java.lang.Thread.setPriority() method updates or assign the priority of the thread to newPriority. The method throws IllegalArgumentException if the value newPriority goes out of the range, which is 1 (minimum) to 10 (maximum).

3 constants defined in Thread class:

1. public static int MIN_PRIORITY
2. public static int NORM_PRIORITY
3. public static int MAX_PRIORITY

Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

Daemon Thread in Java

Daemon thread in Java is a service provider thread that provides services to the user thread. Its life depends on the mercy of user threads i.e. when all the user threads die, JVM terminates this thread automatically.

There are many java daemon threads running automatically e.g. gc, finalizer etc.

Points to remember for Daemon Thread in Java

- It provides services to user threads for background supporting tasks. It has no role in life than to serve user threads.
- Its life depends on user threads.
- It is a low priority thread.

Why JVM terminates the daemon thread if there is no user thread?

The sole purpose of the daemon thread is that it provides services to user thread for background supporting task. If there is no user thread, why should JVM keep running this thread. That is why JVM terminates the daemon thread if there is no user thread.

Java Thread Pool

Java Thread pool represents a group of worker threads that are waiting for the job and reused many times. A thread pool **reuses previously created threads to execute current tasks**

In the case of a thread pool, a group of fixed-size threads is created. A thread from the thread pool is pulled out and assigned a job by the service provider.

Thread Pool Methods

newFixedThreadPool(int s): The method creates a thread pool of the fixed size s.

newCachedThreadPool(): The method creates a new thread pool that creates the new threads when needed but will still use the previously created thread whenever they are available to use.

newSingleThreadExecutor(): The method creates a new thread.

Advantage of Java Thread Pool

Better performance It saves time because there is no need to create a new thread.

Real time usage

It is used in Servlet and JSP where the container creates a thread pool to process the request.

ThreadGroup in Java

Java provides a convenient way to group multiple threads in a single object. In such a way, we can suspend, resume or interrupt a group of threads by a single method call.

Java thread group is implemented by `java.lang.ThreadGroup` class.

A `ThreadGroup` represents a set of threads. A thread group can also include the other thread group. The thread group creates a tree in which every thread group except the initial thread group has a parent.

A thread is allowed to access information about its own thread group, but it cannot access the information about its thread group's parent thread group or any other thread groups.

Constructors of ThreadGroup class

There are only two constructors of ThreadGroup class.

No.	Constructor	Description
1)	ThreadGroup (String name)	creates a thread group with given name.
2)	ThreadGroup(ThreadGroup parent, String name)	creates a thread group with a given parent group and name.

Java Shutdown Hook

A special construct that facilitates the developers to add some code that has to be run when the Java Virtual Machine (JVM) is shutting down is known as the **Java shutdown hook**. The Java shutdown hook comes in very handy in the cases where one needs to perform some special cleanup work when the JVM is shutting down.

When does the JVM shut down?

The JVM shuts down when:

- user presses ctrl+c on the command prompt
- System.exit(int) method is invoked
- user logoff
- user shutdown etc.

The addShutdownHook (Thread hook) method

The addShutdownHook () method of the Runtime class is used to register the thread with the Virtual Machine.

Syntax:

1. **public void addShutdownHook(Thread hook){ }**

How to perform single task by multiple threads in Java?

If you have to perform a single task by many threads, have only one run() method.

How to perform multiple tasks by multiple threads (multitasking in multithreading)?

If you have to perform multiple tasks by multiple threads , have multiple run() methods.

Java Garbage Collection

In java, garbage means unreferenced objects.

Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

Advantage of Garbage Collection

- It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.
- It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

How can an object be unreferenced?

There are many ways:

- By nulling the reference
- By assigning a reference to another
- By anonymous object etc.

1) By nulling a reference:

1. Employee e=**new** Employee();
2. e=**null**;

2) By assigning a reference to another:

1. Employee e1=**new** Employee();
2. Employee e2=**new** Employee();
3. e1=e2;**//now the first object referred by e1 is available for garbage collection**

3) By anonymous object:

1. **new** Employee();

finalize() method

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:

1. **protected void** finalize(){}

gc() method

The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

1. **public static void** gc(){}

Java Runtime class

Java Runtime class is used *to interact with java runtime environment*. Java Runtime class provides methods to execute a process, invoke GC, get total and free memory etc. There is only one instance of java.lang.Runtime class is available for one java application.

Important methods of Java Runtime class

No.	Method	Description
1)	public static Runtime getRuntime()	returns the instance of Runtime class.
2)	public void exit(int status)	terminates the current virtual machine.
3)	public void addShutdownHook(Thread hook)	registers new hook thread.
4)	public Process exec(String command) throws IOException	executes given command in a separate process.
5)	public int availableProcessors()	returns no. of available processors.
6)	public long freeMemory()	returns amount of free memory in JVM.
7)	public long totalMemory()	returns amount of total memory in JVM.

Java Collections

Collections in Java

The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects.

Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

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

What is Collection in Java

A Collection represents a single unit of objects, i.e., a group.

What is a framework in Java

- It provides readymade architecture.
- It represents a set of classes and interfaces.
- It is optional.

What is Collection framework

The Collection framework represents a unified architecture for storing and manipulating a group of objects. It has:

1. Interfaces and its implementations, i.e., classes
2. Algorithm

1) Java ArrayList

Java **ArrayList** class uses a *dynamic array* for storing the elements. It is like an array, but there is *no size limit*. We can add or remove elements anytime. So, it is much more flexible than the

traditional array. It is found in the *java.util* package. It is like the Vector in C++.

- We can not create an array list of the primitive types, such as int, float, char, etc. It is required to use the required wrapper class in such cases. For example:
 1. `ArrayList<int> al = ArrayList<int>(); // does not work`
 2. `ArrayList<Integer> al = new ArrayList<Integer>(); // works fine`

ArrayList class declaration

Let's see the declaration for `java.util.ArrayList` class.

1. `public class ArrayList<E> extends AbstractList<E> implements List<E>, RandomAccess, Cloneable, Ser`

Java Non-generic Vs. Generic Collection

Java collection framework was non-generic before JDK 1.5. Since 1.5, it is generic.

Java new generic collection allows you to have only one type of object in a collection. Now it is type-safe, so typecasting is not required at runtime.

Let's see the old non-generic example of creating a Java collection.

1. `ArrayList list=new ArrayList(); //creating old non-generic arraylist`

Let's see the new generic example of creating java collection.

1. `ArrayList<String> list=new ArrayList<String>(); //creating new generic arraylist`

2) Java LinkedList class

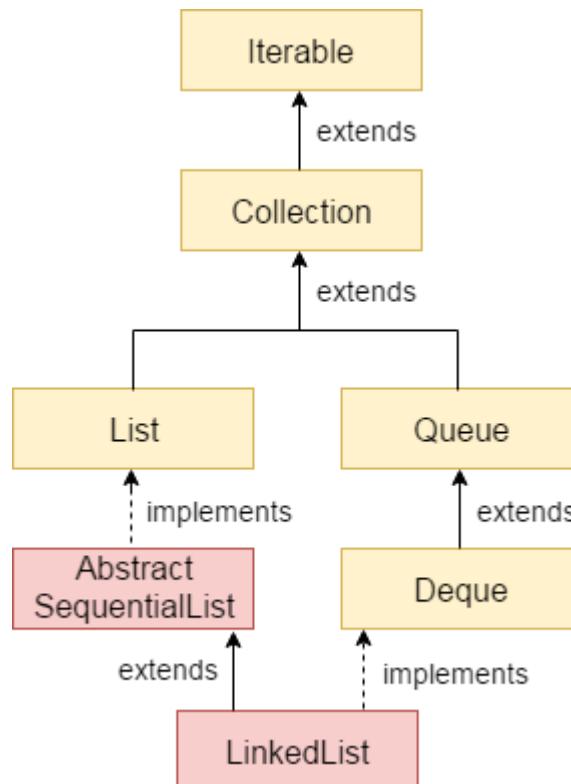
Java LinkedList class uses a doubly linked list to store the elements. It provides a linked-list data structure. It inherits the Abstract List class and implements List and Deque interfaces.

The important points about Java LinkedList are:

- Java LinkedList class can contain duplicate elements.
- Java LinkedList class maintains insertion order.
- Java LinkedList class is non synchronized.
- In Java LinkedList class, manipulation is fast because no shifting needs to occur.
- Java LinkedList class can be used as a list, stack or queue.

Hierarchy of LinkedList class

As shown in the above diagram, Java LinkedList class extends AbstractSequentialList class and implements List and Deque interfaces.



Doubly Linked List

In the case of a doubly linked list, we can add or remove elements from both sides.

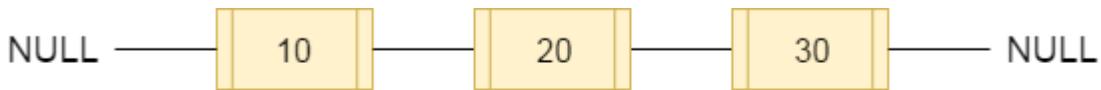


fig- doubly linked list

LinkedList class declaration

Let's see the declaration for `java.util.LinkedList` class.

1. `public class LinkedList<E> extends AbstractSequentialList<E> implements List<E>, Deque<E>, Cloneable`

Difference Between ArrayList and LinkedList

ArrayList	LinkedList
1) ArrayList internally uses a dynamic array to store the elements.	LinkedList internally uses a doubly linked list to store the elements.
2) Manipulation with ArrayList is slow because it internally uses an array. If any element is removed from the array, all the other elements are shifted in memory.	Manipulation with LinkedList is faster than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory.
3) An ArrayList class can act as a list only because it implements List only.	LinkedList class can act as a list and queue both because it implements List and Deque interfaces.
4) ArrayList is better for storing and accessing data.	LinkedList is better for manipulating data.

5) The memory location for the elements of an ArrayList is contiguous.	The location for the elements of a linked list is not contagious.
6) Generally, when an ArrayList is initialized, a default capacity of 10 is assigned to the ArrayList.	There is no case of default capacity in a LinkedList. In LinkedList, an empty list is created when a LinkedList is initialized.
7) To be precise, an ArrayList is a resizable array.	LinkedList implements the doubly linked list of the list interface.

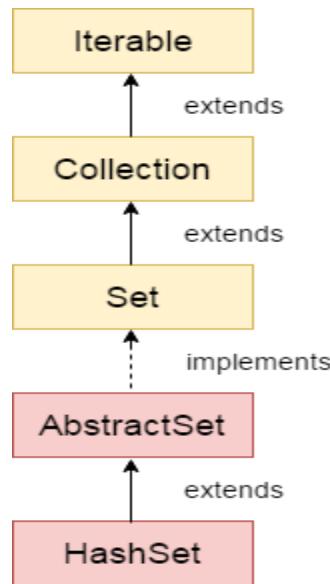
Java List

List in Java provides the facility to maintain the *ordered collection*. It contains the index-based methods to insert, update, delete and search the elements. It can have the duplicate elements also. We can also store the null elements in the list.

The List interface is found in the `java.util` package and inherits the Collection interface. It is a factory of ListIterator interface. Through the ListIterator, we can iterate the list in forward and backward directions. The implementation classes of List interface are `ArrayList`, `LinkedList`, Stack and Vector. The `ArrayList` and `LinkedList` are widely used in Java programming. The `Vector` class is deprecated since Java 5.

Java HashSet

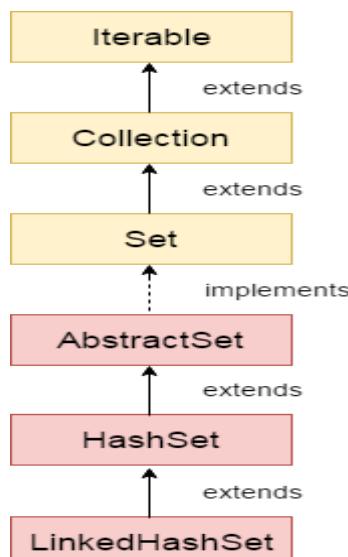
Java HashSet class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface.



The important points about Java HashSet class are:

- HashSet stores the elements by using a mechanism called **hashing**.
- HashSet contains unique elements only.
- HashSet allows null value.
- HashSet class is non synchronized.
- HashSet doesn't maintain the insertion order. Here, elements are inserted on the basis of their hashCode.
- HashSet is the best approach for search operations.
- The initial default capacity of HashSet is 16, and the load factor is 0.75.

Java LinkedHashSet Class

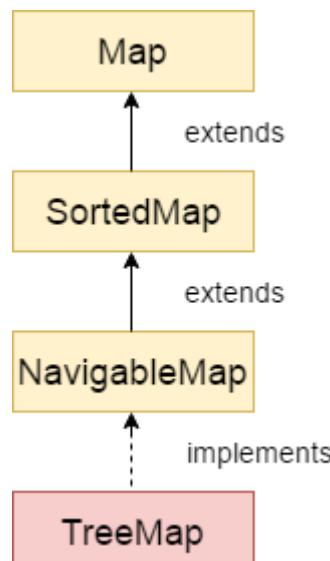


Java LinkedHashSet class is a Hashtable and Linked list implementation of the Set interface. It inherits the HashSet class and implements the Set interface.

The important points about the Java LinkedHashSet class are:

- Java LinkedHashSet class contains unique elements only like HashSet.
- Java LinkedHashSet class provides all optional set operations and permits null elements.
- Java LinkedHashSet class is non-synchronized.
- Java LinkedHashSet class maintains insertion order.

Java TreeMap class



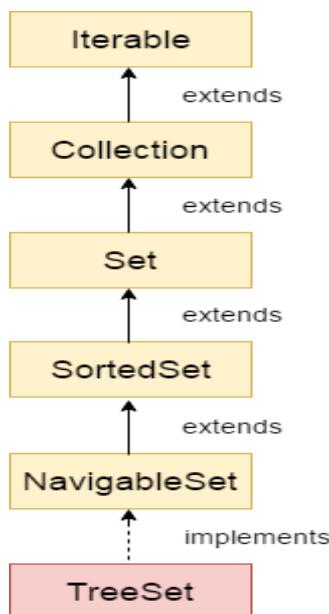
Java TreeMap class is a red-black tree based implementation. It provides an efficient means of storing key-value pairs in sorted order.

The important points about Java TreeMap class are:

- Java TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class.
- Java TreeMap contains only unique elements.

- Java TreeMap cannot have a null key but can have multiple null values.
- Java TreeMap is non synchronized.
- Java TreeMap maintains ascending order.

Java TreeSet class



Java TreeSet class implements the Set interface that uses a tree for storage. It inherits AbstractSet class and implements the NavigableSet interface. The objects of the TreeSet class are stored in ascending order.

The important points about the Java TreeSet class are:

- Java TreeSet class contains unique elements only like HashSet.
- Java TreeSet class access and retrieval times are quiet fast.
- Java TreeSet class doesn't allow null element.
- Java TreeSet class is non synchronized.
- Java TreeSet class maintains ascending order.

Java Queue Interface

The interface Queue is available in the `java.util` package and does extend the Collection interface. It is used to keep the elements that are processed in the First In First Out (FIFO) manner. It is an ordered list of objects, where insertion of elements occurs at the end of the list, and removal of elements occur at the beginning of the list.

Being an interface, the queue requires, for the declaration, a concrete class, and the most common classes are the `LinkedList` and `PriorityQueue` in Java. Implementations done by these classes are not thread safe. If it is required to have a thread safe implementation, `PriorityBlockingQueue` is an available option.

Features of a Queue

The following are some important features of a queue.

- As discussed earlier, FIFO concept is used for insertion and deletion of elements from a queue.
- The Java Queue provides support for all of the methods of the Collection interface including deletion, insertion, etc.
- `PriorityQueue`, `ArrayBlockingQueue` and `LinkedList` are the implementations that are used most frequently.
- The `NullPointerException` is raised, if any null operation is done on the `BlockingQueues`.
- Those Queues that are present in the `util` package are known as Unbounded Queues.
- Those Queues that are present in the `util.concurrent` package are known as bounded Queues.
- All Queues barring the Deques facilitates removal and insertion at the head and tail of the queue; respectively. In fact, deques support element insertion and removal at both ends.

PriorityQueue Class

PriorityQueue is also class that is defined in the collection framework that gives us a way for processing the objects on the basis of priority. It is already described that the insertion and deletion of objects follows FIFO pattern in the Java queue. However, sometimes the elements of the queue are needed to be processed according to the priority, that's where a PriorityQueue comes into action.

Java Deque Interface

The interface called Deque is present in `java.util` package. It is the subtype of the interface queue. The Deque supports the addition as well as the removal of elements from both ends of the data structure. Therefore, a deque can be used as a stack or a queue. We know that the stack supports the Last In First Out (LIFO) operation, and the operation First In First Out is supported by a queue. As a deque supports both, either of the mentioned operations can be performed on it. Deque is an acronym for "**double ended queue**".

Java Hashtable class

Java Hashtable class implements a hashtable, which maps keys to values. It inherits Dictionary class and implements the Map interface.

Points to remember

- A Hashtable is an array of a list. Each list is known as a bucket. The position of the bucket is identified by calling the `hashcode()` method. A Hashtable contains values based on the key.
- Java Hashtable class contains unique elements.
- Java Hashtable class doesn't allow null key or value.
- Java Hashtable class is synchronized.

The initial default capacity of Hashtable class is 11 whereas loadFactor is 0.75

Java Map Interface

A map contains values on the basis of key, i.e. key and value pair. Each key and value pair is known as an entry. A Map contains unique keys.

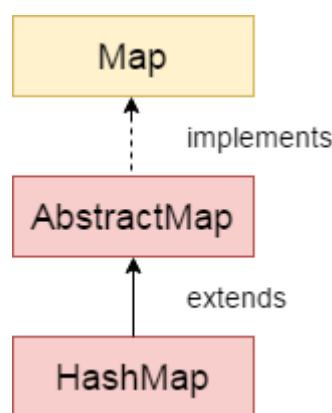
A Map is useful if you have to search, update or delete elements on the basis of a key.

Java Map Hierarchy

There are two interfaces for implementing Map in java: Map and SortedMap, and three classes: HashMap, LinkedHashMap, and TreeMap. A Map doesn't allow duplicate keys, but you can have duplicate values. HashMap and LinkedHashMap allow null keys and values, but TreeMap doesn't allow any null key or value.

A Map can't be traversed, so you need to convert it into Set using *keySet()* or *entrySet()* method.

Java HashMap



Java **HashMap** class implements the Map interface which allows us *to store key and value pair*, where keys should be unique. If you try to insert the duplicate key, it will replace the element of the corresponding key. It is easy

to perform operations using the key index like updation, deletion, etc. HashMap class is found in the `java.util` package.

HashMap in Java is like the legacy Hashtable class, but it is not synchronized. It allows us to store the null elements as well, but there should be only one null key. Since Java 5, it is denoted as `HashMap<K,V>`, where K stands for key and V for value. It inherits the AbstractMap class and implements the Map interface.

Points to remember

- Java HashMap contains values based on the key.
- Java HashMap contains only unique keys.
- Java HashMap may have one null key and multiple null values.
- Java HashMap is non synchronized.
- Java HashMap maintains no order.
- The initial default capacity of Java HashMap class is 16 with a load factor of 0.75.

Java EnumSet class

Java EnumSet class is the specialized Set implementation for use with enum types. It inherits AbstractSet class and implements the Set interface.

EnumSet class declaration

Let's see the declaration for `java.util.EnumSet` class.

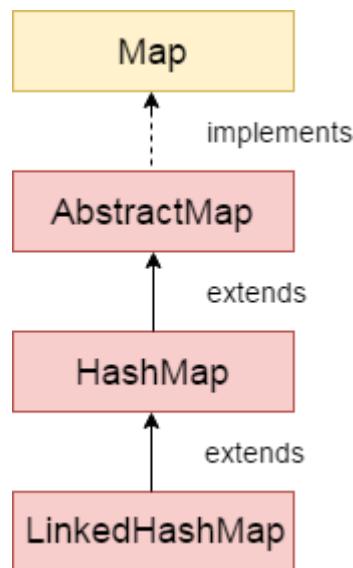
1. **public abstract class** `EnumSet<E extends Enum<E>> extends AbstractSet<E> implements Cloneable, Serializable`

Difference between HashMap and Hashtable

HashMap and Hashtable both are used to store data in key and value form. Both are using hashing technique to store unique keys.

HashMap	Hashtable
1) HashMap is non synchronized . It is not-thread safe and can't be shared between many threads without proper synchronization code.	Hashtable is synchronized . It is thread-safe and can be shared with many threads.
2) HashMap allows one null key and multiple null values.	Hashtable doesn't allow any null key or value .
3) HashMap is a new class introduced in JDK 1.2 .	Hashtable is a legacy class .
4) HashMap is fast .	Hashtable is slow .
5) We can make the HashMap as synchronized by calling this code <pre>Map m = Collections.synchronizedMap(hashMap);</pre>	Hashtable is internally synchronized and can't be unsynchronized.
6) HashMap is traversed by Iterator .	Hashtable is traversed by Enumerator and Iterator .
7) Iterator in HashMap is fail-fast .	Enumerator in Hashtable is not fail-fast .
8) HashMap inherits AbstractMap class.	Hashtable inherits Dictionary class.

Java LinkedHashMap class



Java LinkedHashMap class is Hashtable and Linked list implementation of the Map interface, with predictable iteration order. It inherits HashMap class and implements the Map interface.

Points to remember

- Java LinkedHashMap contains values based on the key.
- Java LinkedHashMap contains unique elements.
- Java LinkedHashMap may have one null key and multiple null values.
- Java LinkedHashMap is non synchronized.
- Java LinkedHashMap maintains insertion order.
- The initial default capacity of Java HashMap class is 16 with a load factor of 0.75.

Java Collections class

Java collection class is used exclusively with static methods that operate on or return collections. It inherits Object class.

The important points about Java Collections class are:

- Java Collection class supports the **polymorphic algorithms** that operate on collections.
- Java Collection class throws a **NullPointerException** if the collections or class objects provided to them are null.

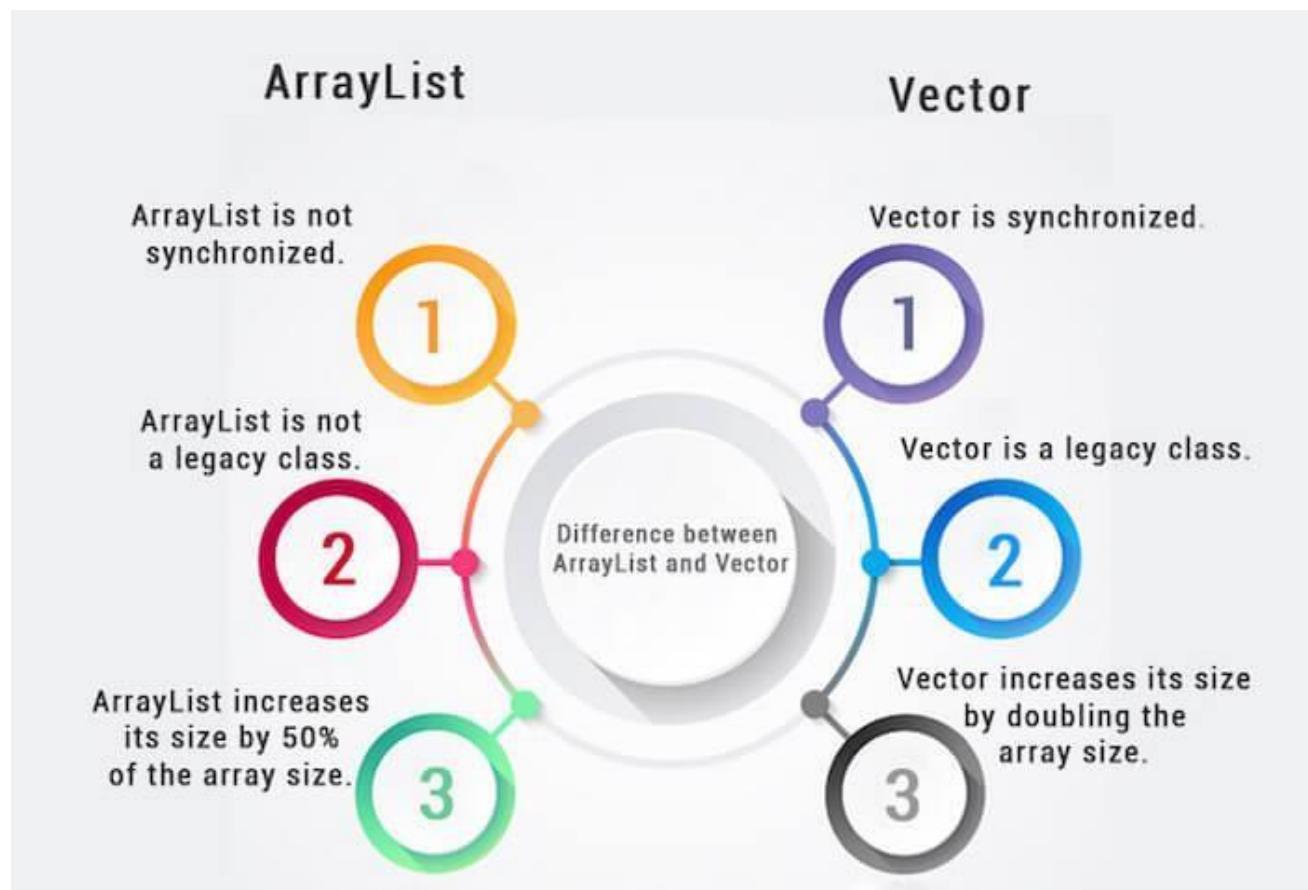
Difference between Comparable and Comparator

- Comparable and Comparator both are interfaces and can be used to sort collection elements.
- However, there are many differences between Comparable and Comparator interfaces that are given below.

Comparable	Comparator
1) Comparable provides a single sorting sequence . In other words, we can sort the collection on the basis of a single element such as id, name, and price.	The Comparator provides multiple sorting sequences . In other words, we can sort the collection on the basis of multiple elements such as id, name, and price etc.
2) Comparable affects the original class , i.e., the actual class is modified.	Comparator doesn't affect the original class , i.e., the actual class is not modified.

3) Comparable provides compareTo() method to sort elements.	Comparator provides compare() method to sort elements.
4) Comparable is present in java.lang package.	A Comparator is present in the java.util package.
5) We can sort the list elements of Comparable type by Collections.sort(List) method.	We can sort the list elements of Comparator type by Collections.sort(List, Comparator) method.

Difference between ArrayList and Vector



Java Vector

Vector is like the *dynamic array* which can grow or shrink its size. Unlike array, we can store n-number of elements in it as there is no size limit. It is a part of Java Collection framework since Java 1.2. It is found in the `java.util` package and implements the *List* interface, so we can use all the methods of List interface here.

It is recommended to use the Vector class in the thread-safe implementation only. If you don't need to use the thread-safe implementation, you should use the ArrayList, the ArrayList will perform better in such case.

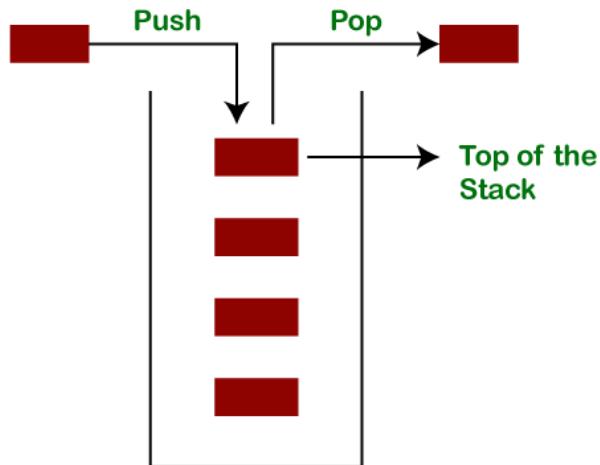
The Iterators returned by the Vector class are *fail-fast*. In case of concurrent modification, it fails and throws the ConcurrentModificationException.

Java Stack

The **stack** is a linear data structure that is used to store the collection of objects. It is based on **Last-In-First-Out** (LIFO). [Java collection](#) framework provides many interfaces and classes to store the collection of objects. One of them is the **Stack class** that provides different operations such as push, pop, search, etc.

In this section, we will discuss the **Java Stack class**, its **methods**, and **implement** the stack data structure in a [Java program](#). But before moving to the Java Stack class have a quick view of how the stack works.

The stack data structure has the two most important operations that are **push** and **pop**. The push operation inserts an element into the stack and pop operation removes an element from the top of the stack. Let's see how they work on stack.



Java JDBC

JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:

- JDBC-ODBC Bridge Driver,
- Native Driver,
- Network Protocol Driver, and
- Thin Driver

The **java.sql** package contains classes and interfaces for JDBC API. A list of popular *interfaces* of JDBC API are given below:

- Driver interface
- Connection interface
- Statement interface
- PreparedStatement interface
- CallableStatement interface
- ResultSet interface
- ResultSetMetaData interface
- DatabaseMetaData interface
- RowSet interface

A list of popular *classes* of JDBC API are given below:

- DriverManager class
- Blob class
- Clob class
- Types class

Why Should We Use JDBC

Before JDBC, ODBC API was the database API to connect and execute the query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured).

Java Reflection API

Java Reflection is a *process of examining or modifying the run time behavior of a class at run time.*

The **java.lang.Class** class provides many methods that can be used to get metadata, examine and change the run time behavior of a class.

The **java.lang** and **java.lang.reflect** packages provide classes for java reflection.

JDBC Driver

JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

1) JDBC-ODBC bridge driver

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls.

Advantages:

- easy to use.
- can be easily connected to any database.

Disadvantages:

- Performance degraded because JDBC method call is converted into the ODBC function calls.
- The ODBC driver needs to be installed on the client machine.

2) Native-API driver

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

Advantage:

- performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

- The Native driver needs to be installed on the each client machine.
- The Vendor client library needs to be installed on client machine.

3) Network Protocol driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

Advantage:

- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

4) Thin driver

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

Advantage:

- Better performance than all other drivers.
- No software is required at client side or server side.

Disadvantage:

- Drivers depend on the Database.

Java Database Connectivity with 5 Steps



1) Register the driver class

The **forName()** method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

Syntax of forName() method

1. **public static void** forName(String className)**throws** ClassNotFoundException

2) get connection object

The **getConnection()** method of DriverManager class is used to establish connection with the database.

Syntax of getConnection() method

- 1) **public static** Connection getConnection(String url)**throws** SQLException
- 2) **public static** Connection getConnection(String url, String name, String password)
3. **throws** SQLException

3) Create the Statement object

The **createStatement()** method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

Syntax of createStatement() method

1. **public** Statement createStatement()**throws** SQLException

4) Execute the query

The **executeQuery()** method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

Syntax of executeQuery() method

1. **public** ResultSet executeQuery(String sql)**throws** SQLException



Follow for more useful content



Telegram
@codehype

Join Telegram for
FREE Coding
Resources

CodeHype

