1. What is Java? • History • Features

2. JDK, JRE and JVM

3. How to set path?

4. Hello Java Program

5. Compile & Runtime errors

6. Command line arguments

7. Naming Convention

8. Data types

* Primitive
* Non Primitive(Wrapper classes)

9. . Variables & its types [Local  Instance  Static]

10. Decision Making in Java  Simple if Statement  if…else Statement  if...else if…else statement  Switch statement

11. Looping Statements in Java  For  While  Do…while

12.Explain about Arrays

13.Explain METHODS in Java

15. Methods

 Syntax of Methods

 Static

 Call static methods & variables

 Invoking Parameterized Methods & different Return types

16 Explain Constructors in java

17 Explain about java String17. String Class

 String class methods(length, equals, equalIgnoreCase, contains, charAt, etc)

 StringBuffer class

 StringBuilder class

18. this keyword and it‟s uses

19. Package

20. Access Modifiers

21. Inheritance(IS-A)  Types of Inheritance

22. super keyword and it‟s uses

23. this vs super

24. Polymorphism

 Method Overloading 

Method Overriding

 Method Overloading vs Method Overriding

25)Explain Aggregation

26) Explain Encapsulation

27)Explain Abstraction

Answers

1, 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.

Features of Java

1. Simple:- Java is very easy to learn, and its syntax is simple, clean and easy to understand.

2.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 behavior.
* Basic concepts of OOPs are:

Object ,Class ,Inheritance, Polymorphism, Abstraction ,Encapsulation

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

**Platform independent**:- Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.

java code can be executed on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc. Java code is compiled by the compiler and converted into bytecode

**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

**Architecture neutral**:- Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.

Interpreted

High Performance:- java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code.

Multithreaded:- 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.

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

**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++.

**JDK, JRE, and JVM**

**1,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 in which Java bytecode can be executed.**
* It can also run those programs which are written in other languages and compiled to Java bytecode.

The JVM performs the following main tasks:

* Loads code
* Verifies code
* Executes code
* Provides runtime environment

**JRE** (**Java Runtime Environment.)**

1. It is also written as Java RTE.
2. The Java Runtime Environment is a set of software tools which are used for developing Java applications.
3. It is used to provide the runtime environment.
4. **It is the implementation of JVM**. **It physically exists.**
5. **It contains a set of libraries + other files that JVM uses at runtime.**



**JDK(** **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.**

**Compile And Runtime Errors**

* A compile-time error generally refers to the errors that correspond to the semantics or syntax.
* A runtime error refers to the error that we encounter during the code execution during runtime. Eg :-logical Errors
* We can easily fix a compile-time error during the development of code.
* A compiler cannot identify a runtime error.

**Java Command Line Arguments**

* The java command-line argument is an argument i.e. passed at the time of running the java program.
* The arguments passed from the console can be received in the java program and it can be used as an input**.**
* it provides a convenient way to **check the behaviour of the program for the different values.**

**Naming Convention**

Java naming convention is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method, etc.

By using standard Java naming conventions, you make your code easier to read for yourself and other programmers.

Readability of Java program is very important.

It indicates that less time is spent to figure out what the code does.

**Class**

* It should start with the uppercase letter.
* It should be a noun such as Color, Button, System, Thread, etc.
* Use appropriate words, instead of acronyms.

Eg :-class Employee

**Method**

* It should start with lowercase letter.
* It should be a verb such as main(), print(), println().
* If the name contains multiple words, start it with a lowercase letter followed by an uppercase letter such as actionPerformed().

**Variable**

* It should start with a lowercase letter such as id, name.
* It should not start with the special characters like & (ampersand), $ (dollar), \_ (underscore).
* If the name contains multiple words, start it with the lowercase letter followed by an uppercase letter such as firstName, lastName.
* Avoid using one-character variables such as x, y, z.

**Interface**

* It should start with the uppercase letter.
* It should be an adjective such as Runnable, Remote, ActionListener.
* Use appropriate words, instead of acronyms.

**Package**

* It should be a lowercase letter such as java, lang.
* If the name contains multiple words, it should be separated by dots (.) such as java.util, java.lang.

**Constant**

* It should be in uppercase letters such as RED, YELLOW.
* If the name contains multiple words, it should be separated by an underscore(\_) such as MAX\_PRIORITY.
* It may contain digits but not as the first letter.

**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:

Primitive data types: The primitive data types include boolean, char, byte, short, int, long, float and double.

Non-primitive data types: The non-primitive data types include Classes, Interfaces, and Arrays.

**Java Primitive Data Types**

In Java language, **primitive data types are the building blocks of data manipulation.**

These are the most basic data types available in Java language.

**boolean data type**

The Boolean data type is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions.

The Boolean data type specifies one bit of information, but its "size" can't be defined precisely. Default Value :-false Default si**ze:-1bit**

**byte data type:-** The byte data type is an example of primitive data type.

* It is an **8-bit signed two's complement integer**.
* Its value-range lies between -128 to 127 (inclusive).
* Its minimum value is -128 and maximum value is 127.
* **Its default value is 0.**

**char data type**

* The char data type is a single 16-bit Unicode character.
* Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).
* The char data type is used to store characters.

**short data type**

* The short data type is a 16-bit signed two's complement integer.
* Its value-range lies between -32,768 to 32,767 (inclusive).
* Its minimum value is -32,768 and maximum value is 32,767.
* Its default value is 0.
* The short data type can also be used to save memory just like byte data type.
* A short data type is 2 times smaller than an integer.

**int data type**

* The int data type is a 32-bit signed two's complement integer.
* Its value-range lies between - 2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) (inclusive).
* Its minimum value is - 2,147,483,648and maximum value is 2,147,483,647.
* Its default value is 0.
* the int data type is generally used as a default data type for integral values unless if there is no problem about memory.

**long data type**

* The long data type is a 64-bit two's complement integer.
* Its value-range lies between -9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1)(inclusive).
* Its default value is 0.
* The long data type is used when you need a range of values more than those provided by int.

**float data type**

* The float data type is a single-precision 32-bit IEEE 754 floating point.
* Its value range is unlimited.
* It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers.
* The float data type should never be used for precise values, such as currency. Its default value is 0.0F.

**double data type**

* the double data type is a double-precision 64-bit IEEE 754 floating point.
* Its value range is unlimited.
* The double data type is generally used for decimal values just like float
* Its default value is 0.0d.

**Non-Primitive data types**

Non-Primitive data types in Java are **user-defined data types** (except for String class) and they can be **easily created or modified by the users.**

They can be used to store multiple values and invoke methods to perform certain operations.

**Class**

Class is a user-defined data type that is **used to create objects.** A class contains a set of properties and methods that are common and exhibited by all the objects of the class.

**Strings** in Java are designed in such a way that **they can hold a sequence of characters in a single variable**, unlike character arrays where there are separate char entities

**Arrays** are non-primitive data types in Java **that are used to store elements of the same data type in a contiguous manner.**

They are not pre-defined and users have to declare and initialize arrays by themselves.

Arrays have a unique reference name by which all their elements are accessed. Elements are stored in an indexed manner where the index starts from 0.

**Interace** in Java is a tool to achieve abstraction. Interface can contain non-implemented methods (without the method body) also known as abstract methods.

**Difference Between Primitive And Non-Primitive Data Types In Java**

In Java programming language, **Primitive data types are pre-defined in the system whereas Non-Primitive data types are user-defined and programmers** can easily create and modify them (except String class).

In Primitive data types**, variables can store only one value at a time whereas, in Non-Primitive data types, we can store multiple values of either the same data type or different data types in one variable.**

For Primitive data types, variables, or the data is stored on a stack whereas, for Non-Primitive data types, a stack holds a reference to the object on heap memory. This object is the data whose memory location is given as a reference to the variable.

Primitive data **types start with the lower-case letter** while in the case of Non-Primitive data types, they need to start with an **upper-case initial letter.**

**Wrapper classes in Java**

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

The automatic conversion of **primitive into an object** is known as **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.

The automatic conversion of object into an primitive **is known as unboxing.**

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

|  |  |
| --- | --- |
| **Primitive Type** | **Wrapper class** |
| boolean | [Boolean](https://www.javatpoint.com/java-boolean) |
| char | [Character](https://www.javatpoint.com/post/java-character) |
| byte | [Byte](https://www.javatpoint.com/java-byte) |
| short | [Short](https://www.javatpoint.com/java-short) |
| int | [Integer](https://www.javatpoint.com/java-integer) |
| long | [Long](https://www.javatpoint.com/java-long) |
| float | [Float](https://www.javatpoint.com/java-float) |
| double | Double |

**Variables & its types**

* 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**.
* A variable is the name of a reserved area allocated in memory.
* It is a combination of "vary + able" which means its **value can be changed.**

**Types of Variables**

There are three types of variables in Java:

* local variable
* instance variable
* static variable

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.
* It is called an instance variable because its value is instance-specific and is not shared among instances.

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.
* Memory allocation for static variables happens only once when the class is loaded in the memory.

10. Decision Making in Java

 Simple if Statement

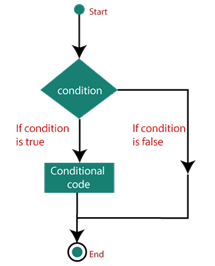
 if…else Statement

 if...else if…else statement

 Switch statement

**Conditional statements**

Conditional statements are used to perform different actions based on various conditions. The conditional statement evaluates a condition before the execution of instructions.



**Types of Conditional Statements**

The conditional statements in JavaScript are listed below:

* if statement
* if….else statement
* if….else if….statement
* switch statement

**The if statement**

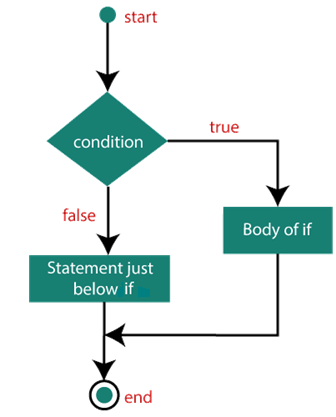
* It is one of the simplest decision-making statement which is used to decide whether a block of JavaScript code will execute if a certain condition is true.
* If the condition evaluates to true, the code within if statement will execute, but if the condition evaluates to false, then the code after the end of if statement (after the closing of curly braces) will execute.

**Syntax**

if (condition) {

// block of code will execute if the condition is true

}



**The if….else statement**

* An if….else statement includes two blocks that are if block and else block.
* It is the next form of the control statement, which allows the execution of JavaScript in a more controlled way.
* It is used when you require to check two different conditions and execute a different set of codes.
* The else statement is used for specifying the execution of a block of code if the condition is false.

if (condition)

{

// block of code will execute if the condition is true

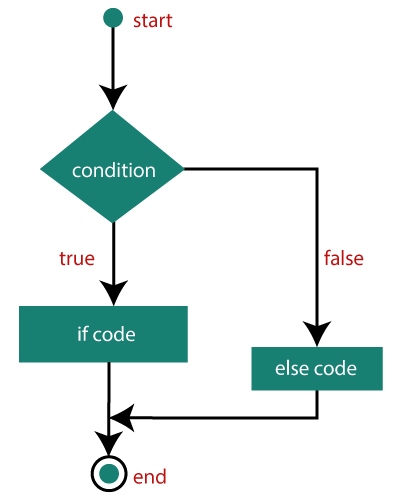
}

else

{

// block of code will execute if the condition is false

}



**The if….else if…..else statement**

* It is used to test multiple conditions.
* The if statement can have multiple or zero else if statements and they must be used before using the else statement.
* You should always be kept in mind that the else statement must come after the else if statements.

**Syntax**

if (condition1)

{

// block of code will execute if condition1 is true

}

else if (condition2)

{

// block of code will execute if the condition1 is false and condition2 is true

}

else

{

// block of code will execute if the condition1 is false and condition2 is false

}

**The switch statement**

* It is a multi-way branch statement that is also used for decision-making purposes.
* It is mainly used when all branches depend upon the value of a single variable.
* It executes a block of code depending upon the different cases.
* The switch statement uses the break or default keywords, but both of them are optional.

**Break:**

* It is used within the switch statement for terminating the sequence of a statement.
* It is optional to use.
* If it gets omitted, then the execution will continue on each statement. When it is used, then it will stop the execution within the block.

**Default:**

* It specifies some code to run when there is no case match.
* There can be only a single default keyword in a switch.
* It is also optional, but it is recommended to use it as it takes care of unexpected cases.
* If the condition passed to switch doesn't match with any value in cases, then the statement under the default will get executed.

**Loops in Java**

The Java for loop is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.

**There are three types of for loops** in Java.

**Java Simple for Loop**

A simple for loop is the same as C/C++. We can initialize the variable, check condition and increment/decrement value. It consists of four parts:

Initialization: It is the initial condition which is executed once when the loop starts. Here, we can initialize the variable, or we can use an already initialized variable. It is an optional condition.

Condition: It is the second condition which is executed each time to test the condition of the loop. It continues execution until the condition is false. It must return boolean value either true or false. It is an optional condition.

Increment/Decrement: It increments or decrements the variable value. It is an optional condition.

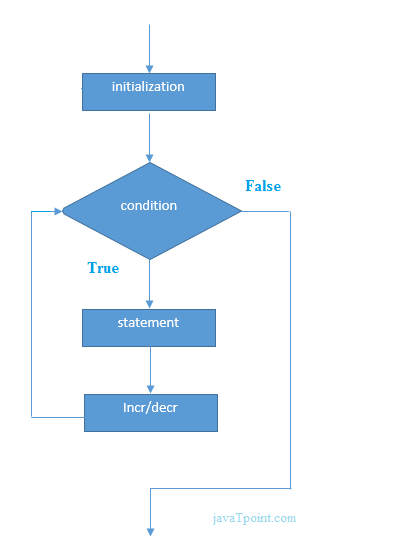
Statement: The statement of the loop is executed each time until the second condition is false.

**Syntax**

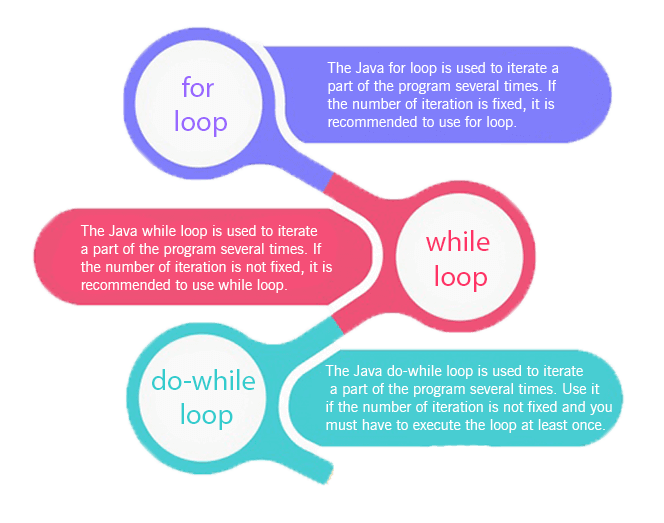
for(initialization; condition; increment/decrement){

//statement or code to be executed

}



:

.

**Java Nested for Loop**

If we have a for loop inside the another loop, it is known as nested for loop. The inner loop executes completely whenever outer loop executes

**Java for-each Loop**

The for-each loop is used to traverse array or collection in Java. It is easier to use than simple for loop because we don't need to increment value and use subscript notation.

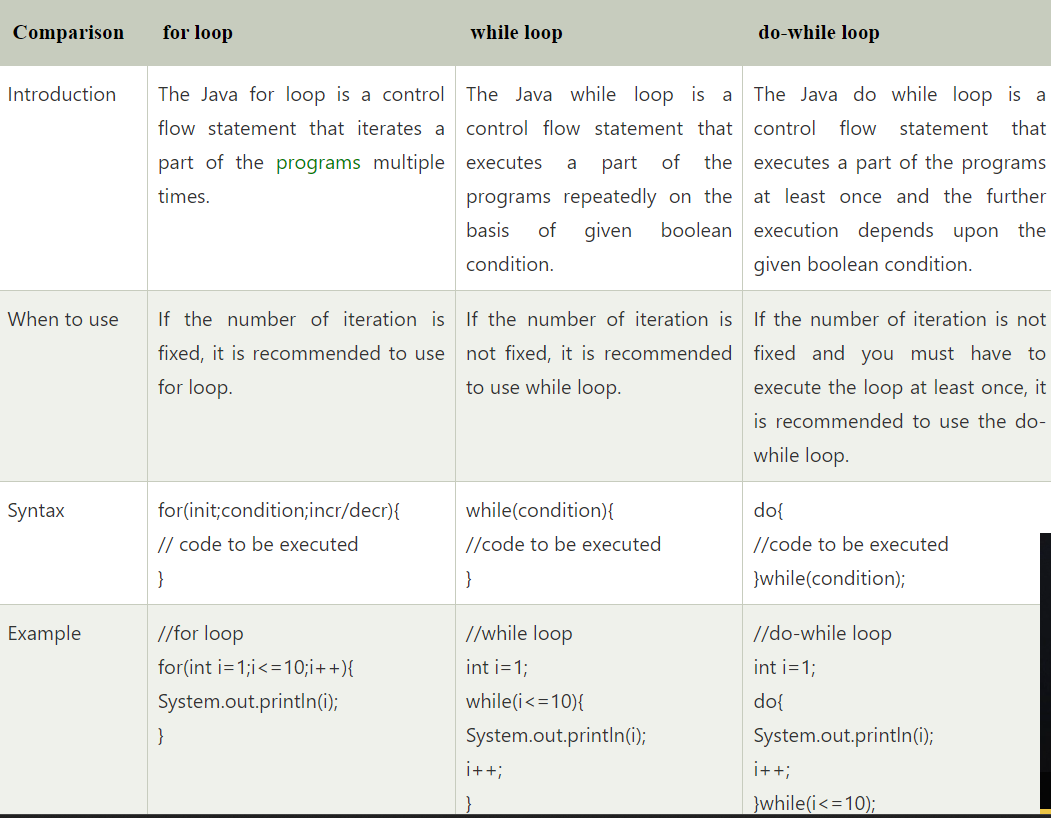
It works on the basis of elements and not the index. It returns element one by one in the defined variable.

Syntax:

for(data\_type variable : array\_name){

//code to be executed

}



Java While Loop

The Java while loop is used to iterate a part of the program repeatedly until the specified Boolean condition is true. As soon as the Boolean condition becomes false, the loop automatically stops.

The while loop is considered as a repeating if statement. If the number of iteration is not fixed, it is recommended to use the while loop.

Syntax:

while (condition){

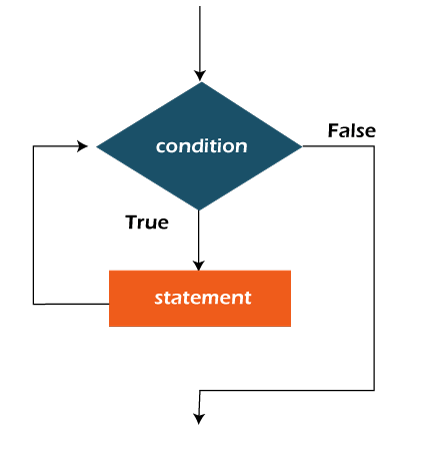
//code to be executed

I ncrement / decrement statement

}

1. Condition: It is an expression which is tested. If the condition is true, the loop body is executed and control goes to update expression. When the condition becomes false, we exit the while loop.

The Java do-while loop is used to iterate a part of the program repeatedly, until the specified condition is true. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use a do-while loop.



Java do-while loop

It is called an exit control loop. Therefore, unlike while loop and for loop, the do-while check the condition at the end of loop body. The Java do-while loop is executed at least once because condition is checked after loop body.

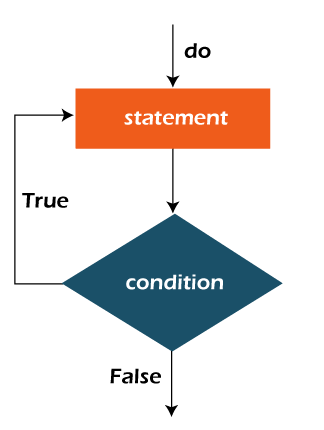
Syntax:

do{

//code to be executed / loop body

//update statement

}while (condition);



Jump Statements

* Jump statements are used to transfer the control of the program to the specific statements.
* jump statements transfer the execution control to the other part of the program.
* There are two types of jump statements in Java, i.e., break and continue.

Java break statement

* 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.
* It breaks only the inner loop in the case of the nested loop.
* The break statement cannot be used independently in the Java program, i.e., it can only be written inside the loop or switch statement.

Java continue 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.

**Array**

* An array is a collection of similar type of elements which has contiguous memory location.
* Java array is an object which contains elements of a similar data type.
* 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.



**Types of Array in java**

There are two types of array.

* Single Dimensional Array
* Multidimensional Array

Single Dimensional Array in Java

**Syntax to Declare an Array in Java**

dataType[] arr; (or)

dataType []arr; (or)

dataType arr[];

**Instantiation of an Array in Java**

arrayRefVar=new datatype[size];

**Declaration, Instantiation and Initialization of Java Array**

We can declare, instantiate and initialize the java array together by:

Int a[]={33,3,4,5};//declaration, instantiation and initialization

**Multidimensional Array in Java**

* In such case, data is stored in row and column based index (also known as matrix form).

**Syntax to Declare Multidimensional Array in Java**

dataType[][] arrayRefVar; (or)

dataType [][]arrayRefVar; (or)

dataType arrayRefVar[][]; (or)

dataType []arrayRefVar[];

**Example to instantiate Multidimensional Array in Java**

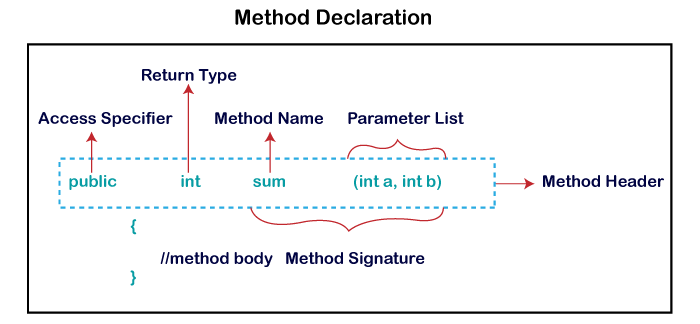
**int[][] arr=new int[3][3];//3 row and 3 column**

**Method in Java**

* The method in Java is a collection of instructions that performs a specific task.
* It provides the reusability of code.
* A method is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation.
* It is used to achieve the reusability of code.
* It also provides the easy modification and readability of code, just by adding or removing a chunk of code.
* The method is executed only when we call or invoke it.
* The most important method in Java is the main() method.

**Method Declaration/ Method Syntax**

The method declaration provides information about method attributes, such as visibility, return-type, name, and arguments. It has six components that are known as method header



**Syntax of Method**

<access\_modifier> <return\_type> <method\_name>( list\_of\_parameters)

{

//body

}

**Method Signature:**

* Every method has a method signature.
* It is a part of the method declaration. It includes the method name and parameter list.

**Access Specifier**

* Access specifier or modifier is the access type of the method.
* It specifies the visibility of the method. Java provides four types of access specifier:

**Public:** The method is accessible by all classes when we use public specifier in our application.

**Private**: When we use a private access specifier, the method is accessible only in the classes in which it is defined.

**Protected**: When we use protected access specifier, the method is accessible within the same package or subclasses in a different package.

**Default:** When we do not use any access specifier in the method declaration, Java uses default access specifier by default. It is visible only from the same package only.

**Return Type**:

Return type is a data type that the method returns.

It may have a primitive data type, object, collection, void, etc. If the method does not return anything, we use void keyword.

**Method Name**:

* It is a unique name that is used to define the name of a method.
* It must be corresponding to the functionality of the method. Suppose, if we are creating a method for subtraction of two numbers, the method name must be subtraction().
* A method is invoked by its name.

Parameter List: It is the list of parameters separated by a comma and enclosed in the pair of parentheses. It contains the data type and variable name. If the method has no parameter, left the parentheses blank.

Method Body: It is a part of the method declaration. It contains all the actions to be performed. It is enclosed within the pair of curly braces.

**Naming a Method**

* While defining a method, remember that the method name must be a verb and start with a lowercase letter.
* If the method name has more than two words, the first name must be a verb followed by adjective or noun.
* In the multi-word method name, the first letter of each word must be in uppercase except the first word.

For example

**Single-word method name: sum(), area()**

**Multi-word method name: areaOfCircle(), stringComparision()**

**Types of Method**

There are two types of methods in Java:

* Predefined Method
* User-defined Method

**Predefined Method**

* Predefined methods are the method that is already defined in the Java class libraries is known as predefined methods.
* It is also known as the standard library method or built-in method.
* We can directly use these methods just by calling them in the program at any point.
* Some pre-defined methods are length(), equals(), compareTo(), sqrt(), etc.
* When we call any of the predefined methods in our program, a series of codes related to the corresponding method runs in the background that is already stored in the library.

have used three predefined methods main(), print(), and max(). We have used these methods directly without declaration because they are predefined

The max() method is a method of the Math class that returns the greater of two numbers.

**User-defined Method**

* The method written by the user or programmer is known as a user-defined method.
* These methods are modified according to the requirement.

How to Create a User-defined Method

Let's create a user defined method that checks the number is even or odd. First, we will define the method.

Static Method

A method that has static keyword is known as static method. In other words,

a method that belongs to a class rather than an instance of a class is known as a static method. We can also create a static method by using the keyword static before the method name.

The main advantage of a static method is that we can call it **without creating an object.** It can access static data members and also change the value of it. It is used to create an instance method. It is invoked by using the class name. The best example of a static method is the main() method.

Instance Method

The method of the class is known as an instance method. It is a non-static method defined in the class. Before calling or invoking the instance method, it is necessary to create an object of its class. Let's see an example of an instance method.

here are two types of instance method:

Accessor Method

Mutator Method

Accessor Method: The method(s) that reads the instance variable(s) is known as the accessor method. We can easily identify it because the method is prefixed with the word get. It is also known as getters. It returns the value of the private field. It is used to get the value of the private field.

Mutator Method: The method(s) read the instance variable(s) and also modify the values. We can easily identify it because the method is prefixed with the word set. It is also known as setters or modifiers. It does not return anything. It accepts a parameter of the same data type that depends on the field. It is used to set the value of the private field

Abstract Method

The method that does not has method body is known as abstract method. In other words, without an implementation is known as abstract method. It always declares in the abstract class. It means the class itself must be abstract if it has abstract method. To create an abstract method, we use the keyword abstract.

Factory method

It is a method that returns an object to the class to which it belongs. All static methods are factory methods. For example, NumberFormat obj = NumberFormat.getNumberInstance();

A static method can be called directly from the class, without having to create an instance of the class. A static method can only access static variables; it cannot access instance variables. Since the static method refers to the class, the syntax to call or refer to a static method is: class name. method name

Static variables can be used in any type of methods: static or non-static. Non-static variables cannot be used inside static methods.

In Java, you can call a static method by using the class name followed by the method name. For example, if you have a class named MyClass with a static method named myStaticMethod, you can call it like this: MyClass.myStaticMethod();1

**Static methods vs Instance methods in Java**

**Instance Method**

Instance method are methods which require an object of its class to be created before it can be called. To invoke a instance method, we have to create an Object of the class in which the method is defined.

public void geek(String name)

{

// code to be executed....

}

// Return type can be int, float String or user defined data type.

**Static Method**

Static methods are the methods in Java that can be called without creating an object of class. They are referenced by the class name itself or reference to the Object of that class.

public static void geek(String name)

{

// code to be executed....

}

// Must have static modifier in their declaration.

// Return type can be int, float, String or user defined data type.

Static method(s) are associated with the class in which they reside i.e. they are called without creating an instance of the class i.e ClassName.methodName(args).

They are designed with the aim to be shared among all objects created from the same class.

When to use static methods?

When you have code that can be shared across all instances of the same class, put that portion of code into static method.

They are basically used to access static field(s) of the class.

Instance method vs Static method

* Instance method can access the instance methods and instance variables directly.
* Instance method can access static variables and static methods directly.
* Static methods can access the static variables and static methods directly.
* Static methods can’t access instance methods and instance variables directly. They must use reference to object. And static method can’t use this keyword as there is no instance for ‘this’ to refer to.

**return type in Java**

* A return statement causes the program control to transfer back to the caller of a method.
* Every method in Java is declared with a return type and it is mandatory for all java methods.
* A return type may be a primitive type like int, float, double, a reference type or void type(returns nothing).
* There are a few important things to understand about returning the values
* The type of data returned by a method must be compatible with the return type specified by the method. For instance, if the return type of some method is boolean, we can not return an integer.
* The variable receiving the value returned by a method must also be compatible with the return type specified for the method.
* The parameters can be passed in a sequence and they must be accepted by the method in the same sequence.

**Constructors 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.
* It is a special type of method which is used to initialize the object.
* Every time an object is created using the new() keyword, at least one constructor is called.
* It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

Note: It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

There are **two rules defined** for the constructor.

* Constructor name must be the same as its class name
* A Constructor must have no explicit return type
* A Java constructor cannot be abstract, static, final, and synchronized

Types of Java constructors

There are two types of constructors in Java:

Default constructor (no-arg constructor)

Parameterized constructor

**Java Default Constructor**

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor:

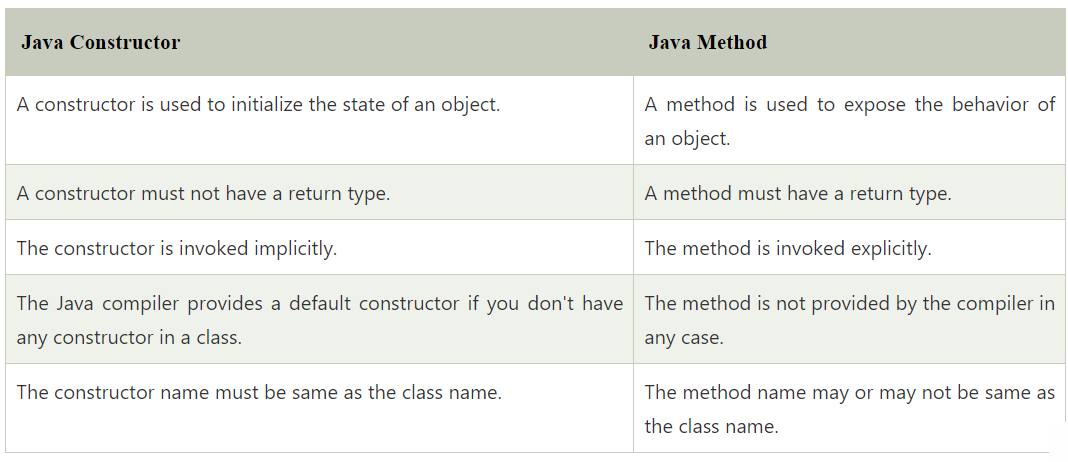
<class\_name>(){}

**Java Parameterized Constructor**

A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

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



**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.

dvantage of method overloading

Method overloading increases the readability of the program.

Different ways to overload the method

There are two ways to overload the method in java

* By changing number of arguments
* By changing the data type

**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);

Java String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

The java.lang.String class implements Serializable, Comparable and CharSequence interfaces.



CharSequence Interface

The CharSequence interface is used to represent the sequence of characters. String, StringBuffer and StringBuilder classes implement it. It means, **we can create strings in Java by using these three classes.**



* The Java String is immutable which means it cannot be changed. Whenever we change any string, a new instance is created
* . For mutable strings, you can use StringBuffer and StringBuilder classes.

What is String in Java?

in Java, string is an object that represents a sequence of characters. The java.lang.String class is used to create a string object.

create a string object

There are two ways to create String object:

* By string literal
* By new keyword

1) String Literal

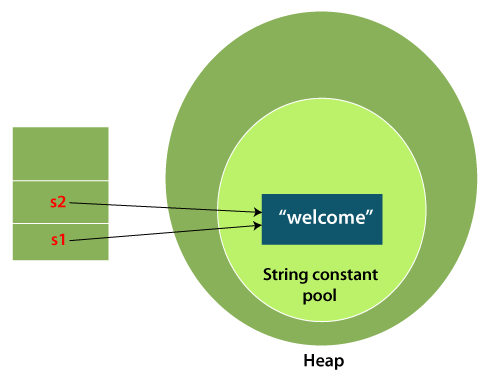
Java String literal is created by using double quotes. For Example:

String s="welcome";

Each time you create a string literal, the JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

String s1="Welcome";

String s2="Welcome";//It doesn't create a new instance



In the above example, only one object will be created. Firstly, JVM will not find any string object with the value "Welcome" in string constant pool that is why it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create a new object but will return the reference to the same instance.

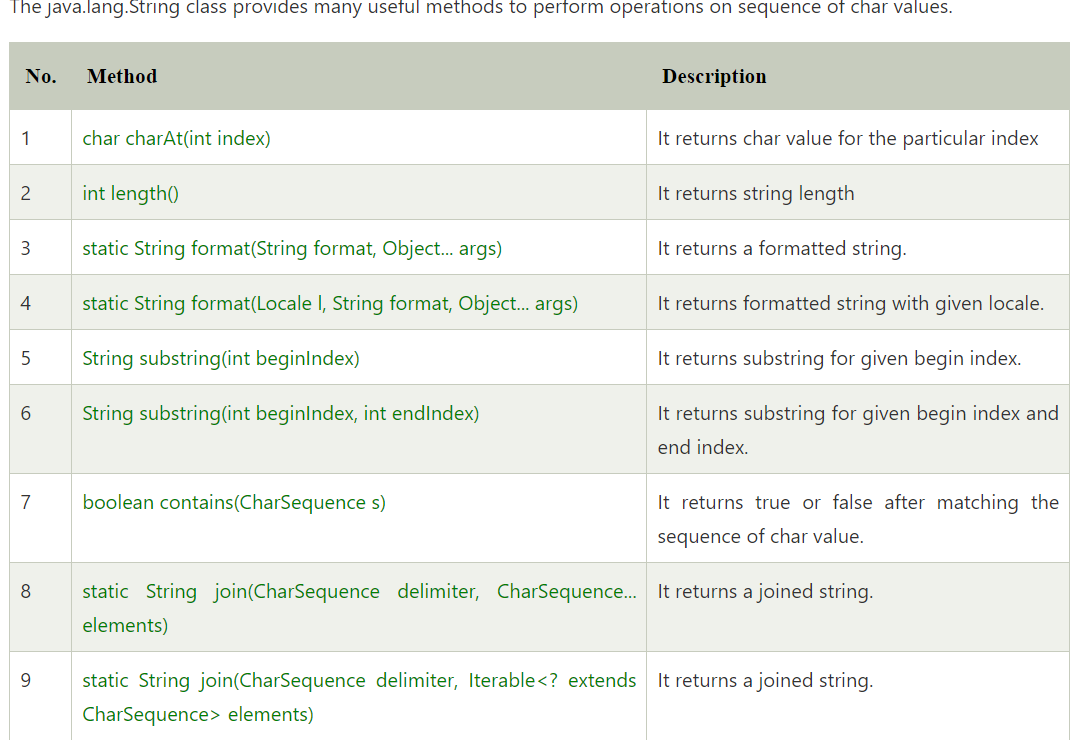
Java uses the concept of String literal

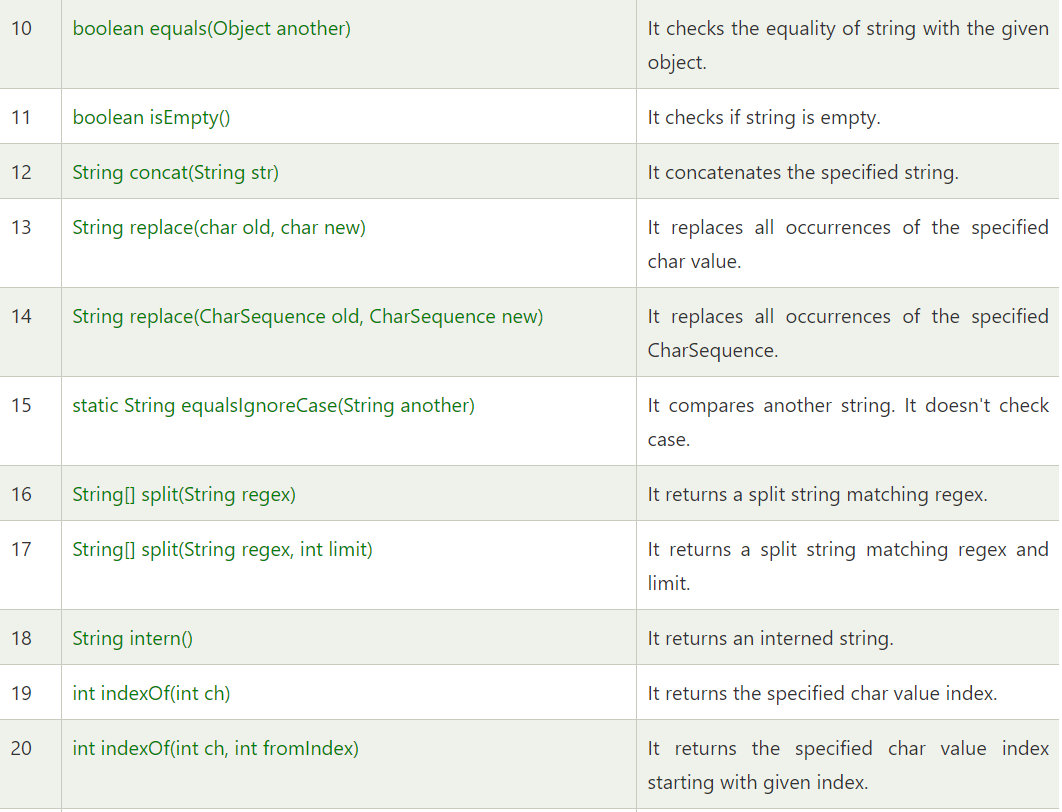
To make Java more memory efficient (because no new objects are created if it exists already in the string constant pool).

2) By new keyword

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

In such case, JVM will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in a heap (non-pool).



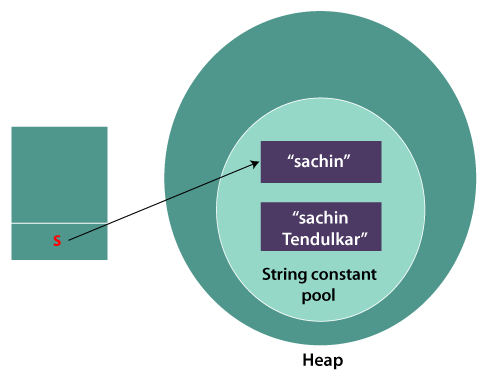




Immutable String in Java

In Java, String objects are immutable. Immutable simply means unmodifiable or unchangeable.

Once String object is created its data or state can't be changed but a new String object is created.



it can be understood by the diagram given below. Here Sachin is not changed but a new object is created with Sachin Tendulkar. That is why String is known as immutable.

String objects are immutable in Java

As Java uses the concept of String literal. Suppose there are 5 reference variables, all refer to one object "Sachin". If one reference variable changes the value of the object, it will be affected by all the reference variables. That is why String objects are immutable in Java.

String class is Final in Java

The reason behind the String class being final is because no one can override the methods of the String class. So that it can provide the same features to the new String objects as well as to the old ones.

Java String compare

We can compare String in Java on the basis of content and reference.

It is used in

* authentication (by equals() method),
* sorting (by compareTo() method),
* reference matching (by == operator) etc.

1) By Using equals() Method

The String class equals() method compares the original content of the string. It compares values of string for equality. String class provides the following two methods:

* public boolean equals(Object another) compares this string to the specified object.
* public boolean equalsIgnoreCase(String another) compares this string to another string, ignoring case.

the methods of String class are used. The equals() method returns true if String objects are matching and both strings are of same case. equalsIgnoreCase() returns true regardless of cases of strings.

2) By Using == operator

The == operator compares **references** not values.

) By Using compareTo() method

The String class compareTo() method compares values lexicographically and returns an integer value that describes if first string is less than, equal to or greater than second string.

Suppose s1 and s2 are two String objects. If:

s1 == s2 : The method returns 0.

s1 > s2 : The method returns a positive value.

s1 < s2 : The method returns a negative value.

this keyword in Java



1)this: to refer current class instance variable

* The this keyword can be used to refer current class instance variable.
* If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.
* If parameters (formal arguments) and instance variables are same. So, we are using this keyword to distinguish local variable and instance variable.
* If local variables(formal arguments) and instance variables are different, there is no need to use this keyword.

2) this: to invoke current class method

3) this() : to invoke current class constructor

4)this: to pass as an argument in the method

5) this: to pass as argument in the constructor call

6) this keyword can be used to return current class instance

19. 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

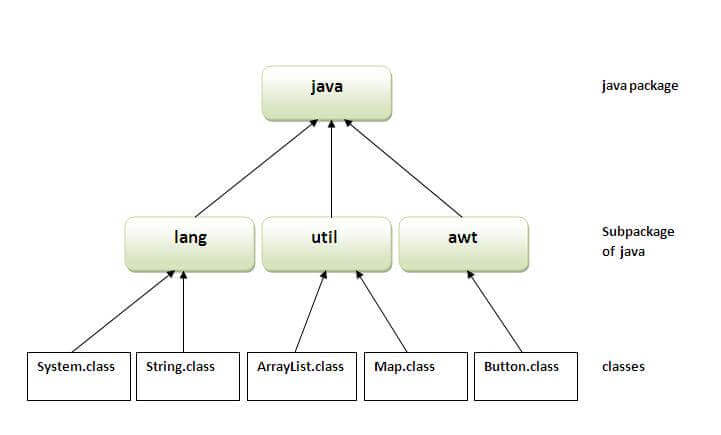
There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

**Advantage of Java Package**

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.



There are three ways to access the package from outside the package.

1. import package.\*;
2. import package.classname;
3. fully qualified name.

20. Access Modifiers

1. The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class.
2. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it

**There are four types of Java access modifiers**:

**Private**:

1. The access level of a private modifier is only within the class.
2. It cannot be accessed from outside the class.

**Default**:

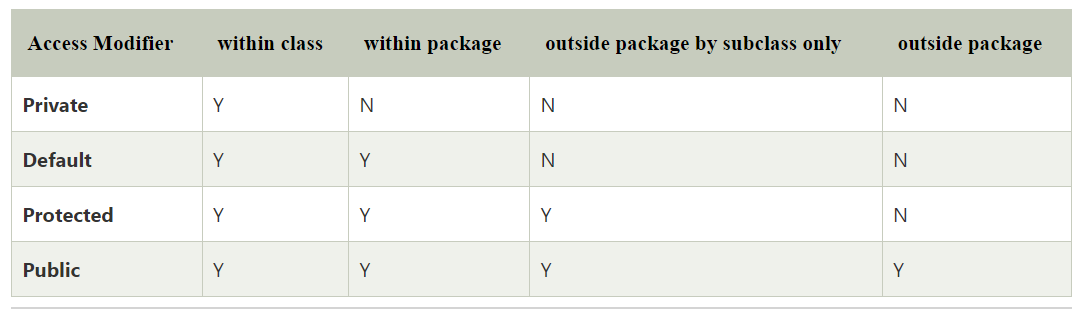
1. The access level of a default modifier is only within the package.
2. It cannot be accessed from outside the package.
3. If you do not specify any access level, it will be the default.

**Protected**:

1. The access level of a protected modifier is within the package and outside the package through child class.
2. If you do not make the child class, it cannot be accessed from outside the package.

**Public**:

1. The access level of a public modifier is everywhere.
2. It can be accessed from within the class, outside the class, within the package and outside the package.



21. Inheritance(IS-A) •

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

It is an important part of OOPs (Object Oriented programming system).

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.

**Terms used in Inheritance**

**Class:**

* A class is a group of objects which have common properties.
* It is a template or blueprint from which objects are created.

**Sub Class/Child Class:**

* Subclass is a class which inherits the other class.
* It is also called a derived class, extended class, or child class.

**Super Class/Parent Class:**

* Superclass is the class from where a subclass inherits the features.
* It is also called a base class or a parent class.

**Reusability:**

* reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class.
* You can use the same fields and methods already defined in the previous class.

The syntax of Java Inheritance

class Subclass-name extends Superclass-name

{

//methods and fields

}

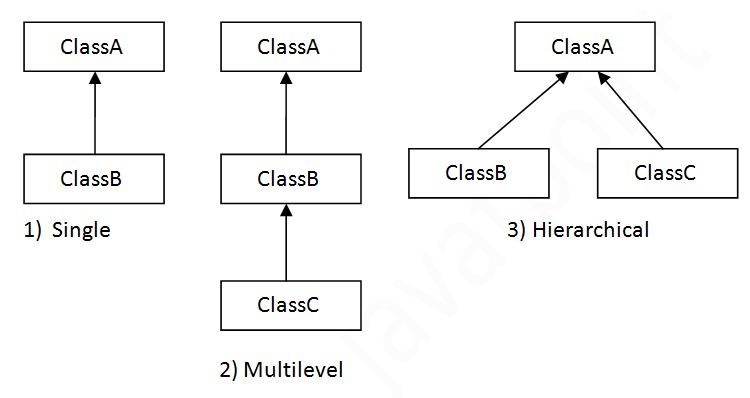
* The extends keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

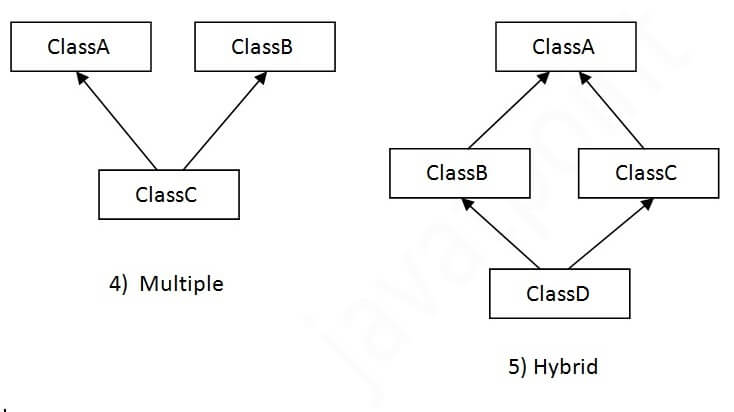
**Types of Inheritance**

On the basis of class, there can be three types of inheritance in java

: **single, multilevel and hierarchical.**

**In java programming, multiple and hybrid inheritance is supported through interface only**

****

****

**Single Inheritance**

* **When a class inherits another class, it is known as a single inheritance.**
* **Dog class inherits the Animal class, so there is the single inheritance.**

**Multilevel Inheritance**

* **When there is a chain of inheritance, it is known as multilevel inheritance**
* **BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.**

**Hierarchical Inheritance**

* **When two or more classes inherits a single class, it is known as hierarchical inheritance.**
* **Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.**

22. super keyword and it‟s uses

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

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

**Usage of Java super Keyword**

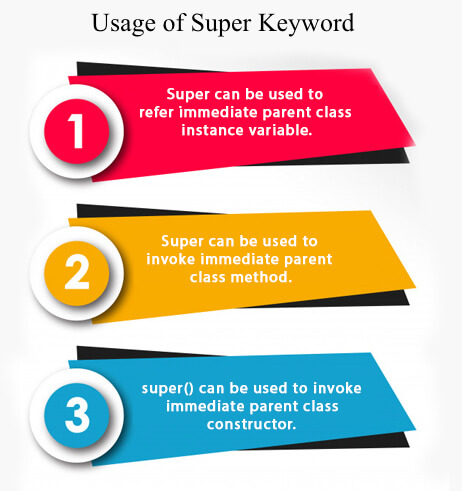
super can be used to refer immediate parent class instance variable.

* We can use super keyword to access the data member or field of parent class.
* It is used if parent class and child class have same fields.

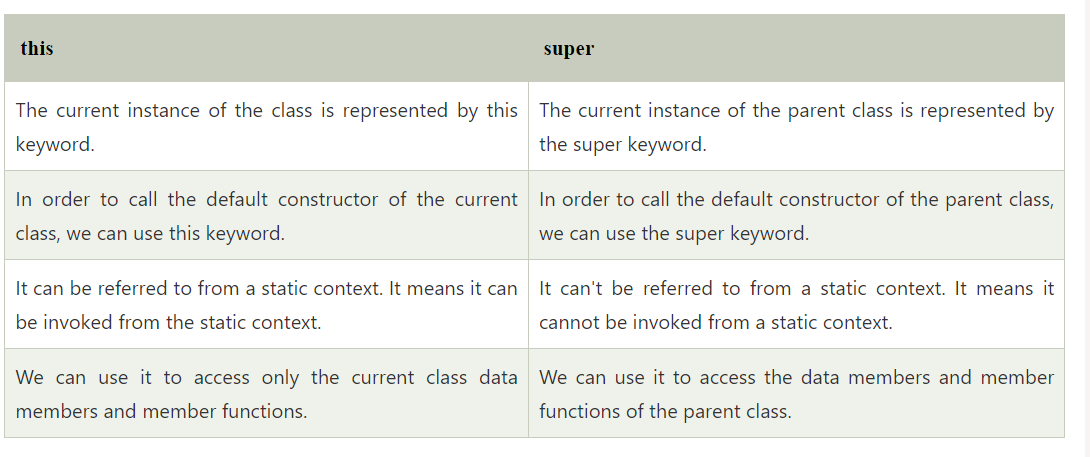
The super keyword can also be used to invoke parent class method.

* It should be used if subclass contains the same method as parent class.
* it is used if method is overridden.
* super can be used to invoke immediate parent class method.

super() can be used to invoke immediate parent class constructor.



23)Difference Between this and super keyword



**24 )Polymorphism**

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

Polymorphism in Java refers to an object’s capacity to take several forms

polymorphism means many forms.

There are two types of polymorphism in Java:

* compile-time polymorphism (method overloading) static method overloading
* runtime polymorphism. (method overriding)

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.
* In this process, an overridden method is called through the reference variable of a superclass.
* The determination of the method to be called is based on the object being referred to by the reference variable.

**Upcasting**

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



***• Method Overloading*** *•*

Compile-time Polymorphism/ Method Overloading

* + Compile-time polymorphism is also known as static polymorphism or early binding.
  + Compile-time polymorphism is a polymorphism that is resolved during the compilation process.
  + Overloading of methods is called through the reference variable of a class.
  + Compile-time polymorphism is achieved by method overloading and operator overloading.

• Method Overloading occurs

1. when a class has many methods with the same name but different parameters.

2. Two or more methods may have the same name if they have other numbers of parameters, different data types, or different numbers of parameters and different data types.

1,a). Method overloading by changing the number of parameters

show( char a )

show( char a ,char b )

1,b) Method overloading by changing Datatype of parameter

show( float a float b)

show( int a, int b )

(1,c). By changing the sequence of parameters

show( int a, float b )

show( float a, int b )

**Invalid cases of method overloading**

Method overloading does not allow changing the return type of method( function ); it occurs ambiguity.

2)Operator Overloading

* An operator is said to be overloaded if it can be used to perform more than one function.
* Operator overloading is an overloading method in which an existing operator is given a new meaning.
* In Java, the + operator is overloaded. Java, on the other hand, does not allow for user-defined operator overloading.
* To add integers, the + operator can be employed as an arithmetic addition operator.
* It can also be used to join strings together.

**Method Overriding**

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

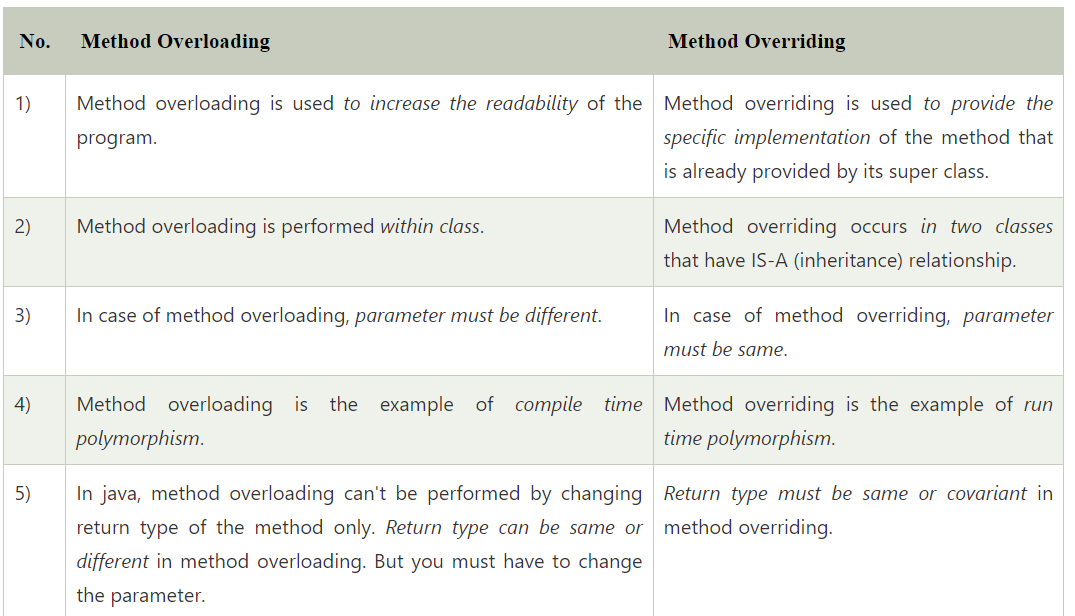
**Usage of Java Method Overriding**

* Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
* Method overriding is used for runtime polymorphism

**Rules for Java Method Overriding**

* The method must have the same name as in the parent class
* The method must have the same parameter as in the parent class.
* There must be an IS-A relationship (inheritance).

**• Method Overloading vs Method Overriding**



Aggregation in java

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

Consider a situation, Employee object contains many information such as id, name, emailId etc. It contains one more object named address, which contains its own informations such as city, state, country, zip code etc. as given below.

class Employee{

int id;

String name;

Address address;//Address is a class

...

}

In such case, Employee has an entity reference address, so relationship is **Employee HAS-A address.**

By using aggregation

* Code reuse is also best achieved by aggregation when there is no is-a relationship.
* Inheritance should be used only if the relationship is-a is maintained throughout the lifetime of the objects involved; otherwise, aggregation is the best choice.

Encapsulation in Java

Encapsulation in Java is a process of wrapping code and data together into a single unit

Eg: a capsule which is mixed of several medicines.

We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.

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.**
* 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.
* The encapsulate class is easy to test. So, it is better for unit testing.
* The standard IDE's are providing the facility to generate the getters and setters. So, it is easy and fast to create an encapsulated class in Java

**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.

it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

Abstraction lets you focus on what the object does instead of how it does it.

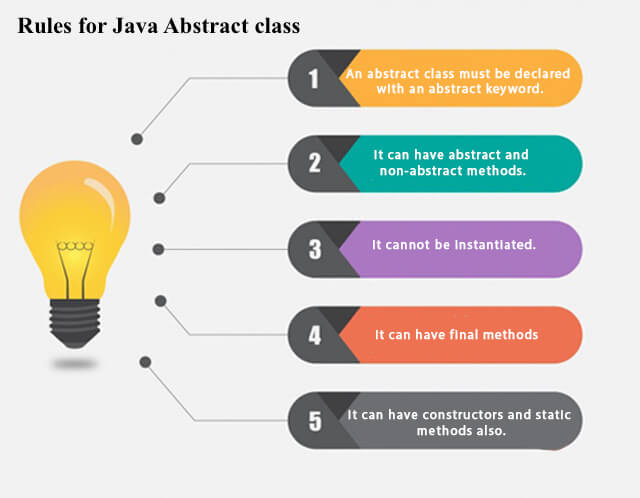
**Ways to achieve Abstraction**

There are two ways to achieve abstraction in java

* Abstract class (0 to 100%)
* Interface (100%)

**Abstract class in Java**

* A class which is declared as abstract is known as an abstract class.
* It can have abstract and non-abstract methods.
* It needs to be extended and its method implemented.
* It cannot be instantiated.



Eg:

abstract class A{}

**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.
* interfaces can have abstract methods and variables. It cannot have a method body.
* Java Interface also represents the IS-A relationship.
* It cannot be instantiated just like the abstract class.

Why use Java interface?

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



**declare an interface**

* An interface is declared by using the interface keyword.
* It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default.
* A class that implements an interface must implement all the methods declared in the interface.

Syntax:

interface <interface\_name>{

// declare constant fields

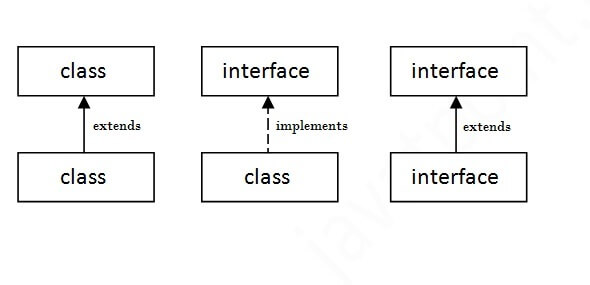
// declare methods that abstract

// by default.

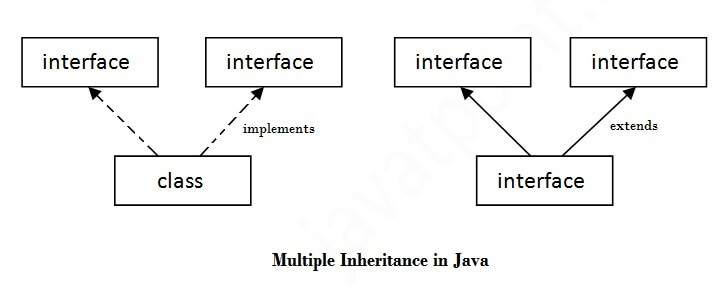
}

The relationship between classes and interfaces

a class extends another class, an interface extends another interface, but a class implements an interface.



**Multiple inheritance in Java by interface**



Multiple inheritance is not supported through class in java, but it is possible by an interface, because multiple inheritance is not supported in the case of class because of ambiguity. However, it is supported in case of an interface because there is no ambiguity. It is because its implementation is provided by the implementation class.

Difference between abstract class and interface

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

