**Java**

Java is a **programming language** and a **platform**.

**Platform**: Any hardware or software environment in which a program runs, is known as a platform.

Java Example

1. **class** Simple
2. {
3. **public** **static** **void** main(String args[])
4. {
5. System.out.println("Hello Java");
6. }
7. }

**For installing JAVA we need**

JDK

The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and [applets](https://www.javatpoint.com/java-applet)

. It physically exists. It contains JRE + development tools.

### JRE

The Java Runtime Environment is a set of software tools which are used for developing Java applications.It contains library files and convert datas into byte format.

### JVM

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.

**Data Types**

1. Premetive Data Type

2. Non Premetive Data types

**Premetive Data Type**

A primitive data type specifies the size and type of variable values.

There are eight primitive data types in Java

1 int Stores whole numbers

2 long Stores whole numbers

3short Stores whole numbers

4 byte Stores whole numbers

5 char Stores a single character

6 boolean Stores a true/false value

7 float Stores fractional numbers

8double Stores fractional numbers upto 15 decimal numbers.

Examples

### Byte

byte myNum = 100;

System.out.println(myNum);

### Short

short myNum = 5000;

System.out.println(myNum);

### Int

int myNum = 100000;

System.out.println(myNum);

### Long

long myNum = 15000000000L;

System.out.println(myNum);

### Float

### float f1=12.25468f;

### System.out.println("Value of f1=" +f1);

### Double

double myNum = 19.99d;

System.out.println(myNum);

Boolean

boolean a1=true,a2=false;

System.out.println("value of a1 is always " +a1+" value of a2 is always "+a2);

Char

char c1='a',c2='b';

System.out.println("characters are " +c1+c2);

**Non Premetive Type/Wrapper class/Predefined class**

Non-primitive data types are called **reference types** because they refer to objects. Non-primitive types are created by the programmer and is not defined by Java (except for String).

Examples are

1 String

2 Array

3 Integer

**Varibles used in JAVA**

1.Local Variables

2 Static Variables

3 Instance Variables

**Command Line**

The java command-line argument is an argument i.e. passed at the time of running the java program.

Example 1

public static void main(String args[])

{

System.out.println("My name is "+args[0]);

System.out.println("Name2 is "+args[1]);

}

}

Example 2

int n =Integer.parseInt(args[0]);

int n1=Integer.parseInt(args[1]);

int sum=n+n1;

System.out.println("sum= "+sum);

**Conditional Statements**

**IF Statement**

Types of IF statements are

1 simple if

2 else if

3 else if ladder

4 nested if

**Operators in JAVA**

**1 Urinary operator**

Used for increment/decrement function.

Eg: i++ , i— , i=i+1, i=i-1

**2 Arithmetical Operator**

Arithmetic operators are used to perform common mathematical operations.

Types are **+, - , / , \* , %**

**3 Assignment Operator**

Assignment operators are used to assign values to variables

**=**

Eg: int a=10

4 Shift Operator

<< , >>

5 Relational Operator

Used to compare two values

> ,< , <= , >=, ==, !=

6 Logical Operator

Logical operators are used to determine the logic between variables or values

AND (&&) OR (||)

**Simple IF Syntax**

if(condition)

{ }

**Else IF Syntax**

if(condition 1)

{ }

else

{statement}

**Else IF Ladder Syntax**

if(condition 1)

{}

else if

{condition 2}

……

else

{statement}

**Nested IF**

if(condition 1)

{

if

{condition 1}

else if

{condition 2}

……..

else

{condition n}

}

else

{statement}

Looping Statements

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.

1. For loop
2. While
3. Do While

**1 For Loop**

A looping statements consists of

1. **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.
2. **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.
3. **Increment/Decrement**: It increments or decrements the variable value. It is an optional condition.
4. **Statement**: The statement of the loop is executed each time until the second condition is false.

Syntax of for loop

**public** **class** ForExample {

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

    //Code of Java for loop

**for**(**int** i=1;i<=10;i++){

        System.out.println(i);

    }

}

**2 While Loop(Also known as Entry Controlled)**

*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.](https://www.javatpoint.com/programs-list)

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](https://www.javatpoint.com/java-for-loop).

**Syntax**

**while** (condition)

{

 Increment / decrement statement

}

**3 DO While Loop(Also known as Exit Controlled)**

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.

**Syntax**

**Do**

{

//code to be executed / loop body

//update statement

}

**while** (condition);

**Single line Command**

The single-line comment is used to comment only one line of the code.

Eg: //int a

**Multi line Command**

Multi line comments in Java **start with /\* and end with \*/**. You can comment multiple lines just by placing them between /\* and \*/.

# Switch Statement

The Java switch statement executes one statement from multiple conditions. It is like [if-else-if](https://www.javatpoint.com/java-if-else) ladder statement. The switch statement works with byte, short, int, long, enum types, String and some wrapper types like Byte, Short, Int, and Long.

Each case statement can have a break statement which is optional. When control reaches to the [break statement](https://www.javatpoint.com/java-break), it jumps the control after the switch expression. If a break statement is not found, it executes the next case.

**switch**(expression){

**case** value1:

 //code to be executed;

**break**;

**case** value2:

 //code to be executed;

**break**;

......

**default**:

  code to be executed **if** all cases are not matched;

}

# Break Statement

When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.

**Syntax**

break;

# Countinue Statement

The Java continue statement is used to continue the loop. The continue statement is used in loop control structure when you need to jump to the next iteration of the loop immediately.

**Syntax**

countine:

**Array**

**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. 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.

There are two types of array.

* Single Dimensional Array
* Multidimensional Array

**Single Dimensional Array**

Syntax

1. dataType[] arr; (or)
2. dataType []arr; (or)
3. dataType arr[];

## Declaration and Instantiation together

Syntax

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

**Multi Dimensional Array**

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

Syntax

int a[][];

Int a[][]={{1,2,3},{4,5,6},{7,8,9}}; //consist of 3 row and 3 column

**Methods in Java**

1.Static Method

2.Non static Method/Instance Method

**Static Method**

If you declare any variable as static, it is known as a static variable.To invoke a static method ,we use the syntax

**Class name.method name**

**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 within which it defined.

Syntax

**Classname obj=new classname()**

**Obj.methodname()**

**Difference between Static and Non static Methods**

**Static Method**

* Static methods are the methods in Java that can be called without creating an object of class
* Static method is declared with static keyword.
* Static methods can access the static variables and static methods directly.

**Non Static Method/Instance Method**

* Instance method are methods which require an object of its class to be created before it can be called
* Instance method is not with static keyword.
* Instance method can access the instance methods and instance variables directly.

**OverLoading**

Overloading **allows different methods to have the same name**, but different signatures where the signature can differ by the number of input parameters or type of input parameters or both.

## Advantage of method overloading

Method overloading increases the readability of the program.

### Different ways to overload the method

There are two ways to overload the method in java

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

Eg: public void sum(int a,int b)

public void sum(int a,int b,int c)

public void sum(int a,float b)

# Constructors in Java

In [Java](https://www.javatpoint.com/java-tutorial), a constructor is a block of codes similar to the method. It is called when an instance of the [class](https://www.javatpoint.com/object-and-class-in-java) is created. At the time of calling constructor, memory for the object is allocated in the memory.

There are two types of constructors in Java: **default constructor**, and **parameterized constructor.**

There are two rules defined for the constructor.

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

There are many differences between constructors and methods. They are given below.

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

Explicit means done by the programmer. **Implicit means done by the JVM** or the tool , not the Programmer. For Example: Java will provide us default constructor implicitly. Even if the programmer didn't write code for constructor, he can call default constructor

Eg of Constructor

Calss JavaTest //class creation

{

Public JavaTest() //constructor creation

{

}

# Inheritance in Java

**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](https://www.javatpoint.com/java-oops-concepts) (Object Oriented programming system). Inheritance represents the **IS-A relationship** which is also known as a parent-child relationship.

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:** As the name specifies, 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

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 in java

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



## Single Inheritance

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

**public** **class** Parent1

{

**public** **void** print()

{

System.***out***.println("i am aparent class");

}

**public** **class** Child1 **extends** Parent1

{

**public** **static** **void** main(String[] args)

{

Child1 obj=**new** Child1();

obj.print();

}

## Multilevel Inheritance

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

**class** Animal{   //parent class

**void** eat(){System.out.println("eating...");}

}

**class** Dog **extends** Animal{   //child class1

**void** bark(){System.out.println("barking...");}

}

**class** BabyDog **extends** Dog{   //child class2

**void** weep(){System.out.println("weeping...");}

}

**class** TestInheritance2{

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

BabyDog d=**new** BabyDog();

d.weep();

d.bark();

d.eat();   }

Output:

weeping...

barking...

eating...

## Hierarchical Inheritance

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

**class** Animal{

**void** eat(){System.out.println("eating...");}

}

**class** Dog **extends** Animal{

**void** bark(){System.out.println("barking...");}

}

**class** Cat **extends** Animal{

**void** meow(){System.out.println("meowing...");}

}

**class** TestInheritance3{

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

Cat c=**new** Cat();

c.meow();

c.eat();

//c.bark();//C.T.Error

}}

Output:

meowing...

eating...

# 

# Access Modifiers in Java

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. 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:

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **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.
3. **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.
4. **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.

# 

# Overloading

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

## Advantage of method overloading

Method overloading increases the readability of the program.

### Different ways to overload the method

There are two ways to overload the method in java

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

Example

**class** Adder{

**static** **int** add(**int** a,**int** b){**int c=** a+b;}

**static** **int** add(**int** a,**int** b,**int** c){**int d=** a+b+c;}

}

**class** TestOverloading1{

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

System.out.println(Adder.add(11,11));

System.out.println(Adder.add(11,11,11));

}}

# Super Keyword in Java

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

## Usage of Java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

# 

# 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**.

#### Rules for Java Method Overriding

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

# Aggregation

When a class access another class without a Parent-Child relationship is called aggregation.Here there is a HAS-A relationship.

# Encapsulation

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

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.

By providing only a setter or getter method, you can make the class **read-only or write-only**.

# Final Keyword In Java

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

1. variable
2. method
3. class

## final variable

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

## final method

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

## final class

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

Abstraction

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

Another way, it shows only essential things to the user and hides the internal details

There are two ways to achieve abstraction in java

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

Abstract class

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

* An abstract class must be declared with an abstract keyword.
* It can have abstract and non-abstract methods.
* It can have final methods which will force the subclass not to change the body of the method

# Interface in Java

An **interface in Java** is a blueprint of a class. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and **multiple**[**inheritance in Java**](https://www.javatpoint.com/inheritance-in-java)**.**

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.