UNIT – I

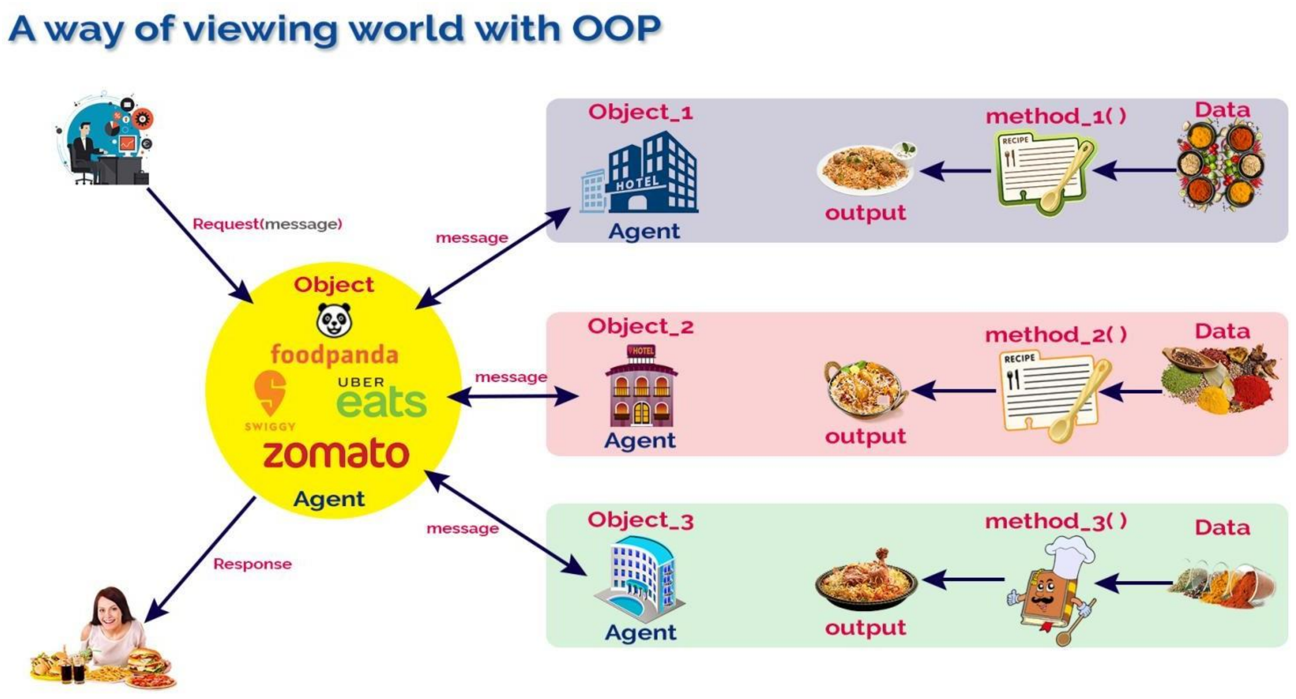
Object-Oriented Thinking

A way of viewing world

➔ A way of viewing the world is an idea to illustrate the object-oriented programming concept with an example of a real-world situation.

➔ Let us consider a situation, A person at office and he wants to get food to his family members who are at his home from a hotel. Because of the distance from his office to home, there is no possibility of getting food from a hotel . So, how do we solve the issue?

➔ To solve the problem, he calls zomato (an **agent** in food delevery community), tells them the variety and quantity of food and the hotel name from which he wish to deliver the food to hisfamily members. Look at the following image.



Agents and Communities

➔ To solve his food delivery problem, he used a solution by finding an appropriate agent (Zomato) and passes a message containing his request. It is the responsibility of the agent (Zomato) to satisfy his request. Here, the agent uses some method to do this. He does not need to know the method that the agent has used to solve his request. This is usually hidden from him.

➔ So, in object-oriented programming, problem-solving is the solution to our problem which requires the help of many individuals in the community.

➔ **An object-oriented program is structured as a community of interacting agents, called objects. Where each object provides a service (data and methods) that is used by other members of the community.**

➔ In our example, the online food delivery system is a community in which the agents are zomatoand set of hotels. Each hotel provides a variety of services that can be used by other memberslike zomato, himself, and his family in the community.

Messages and Methods

- ➔ To solve his problem, he started with a request to the agent zomato, which led to still more requests among the members of the community until his request has done. Here, the members of a community interact with one another by making requests until the problem has satisfied.
- ➔ **In object-oriented programming, every action is initiated by passing a message to an agent (object), which is responsible for the action. The receiver is the object to whom the message was sent. In response to the message, the receiver performs some method to carry out the request. Every message may include any additional information as arguments.**
- ➔ In our example, He send a request to zomato with a message that contains food items, the quantity of food, and the hotel details. The receiver uses a method to food get delivered to hishome.

Responsibilities

- ➔ In object-oriented programming, behaviors of an object described in terms of responsibilities.
- ➔ In our example, his request for action indicates only the desired outcome (food delivered to hisfamily). The agent (zomato) free to use any technique that solves my problem.
- ➔ By discussing a problem in terms of responsibilities increases the level of abstraction. Thisenables more independence between the objects in solving complex problems.

Classes and Instances

- ➔ In object-oriented programming, all objects are instances of a class. The method invoked by anobject in response to a message is decided by the class.
- ➔ All the objects of a class use the same method in response to a similar message.
- ➔ In our example, the zomato a class and all the hotels are sub-classes of it. For every request(message), the class creates an instance of it and uses a suitable method to solve the problem.

Classes Hierarchies

- ➔ A graphical representation is often used to illustrate the relationships among the classes(objects) of a community.
- ➔ This graphical representation shows classes listed in a hierarchical tree-like structure.
- ➔ In this more abstract class listed near the top of the tree, and more specific classes in the middleof the tree, and the individuals listed near the bottom.
- ➔ **In object-oriented programming, classes can be organized into a hierarchical inheritance structure. A child class inherits properties from the parent class that higher in the tree.**

Method Binding, Overriding, and Exception

- ➔ In the class hierarchy, both parent and child classes may have the same method whichimplemented individually.
- ➔ Here, the implementation of the parent is overridden by the child. Or a class may providemultiple definitions to a single method to work with different arguments (overloading).
- ➔ The search for the method to invoke in response to a request (message) begins with the class ofthis receiver.
- ➔ If no suitable method is found, the search is performed in the parent class of it.
- ➔ The search continues up the parent class chain until either a suitable method is found or theparent class chain is exhausted.
- ➔ If a suitable method is found, the method is executed. Otherwise, an error message is issued.

Summary of Object-Oriented concepts

- ➔ OOP stands for Object-Oriented Programming. OOP is a programming paradigm in which everyprogram is follows the concept of object.
- ➔ In other words, OOP is a way of writing programs based on the object concept.

The object-oriented programming paradigm has the following core concepts.

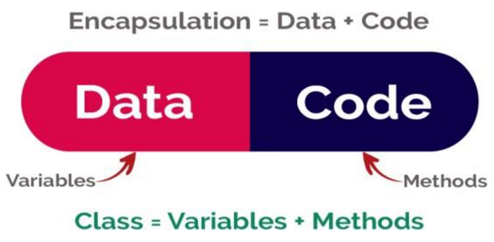
- **Encapsulation**
- **Inheritance**
- **Polymorphism**
- **Abstraction**

The popular object-oriented programming languages are

- Smalltalk,
- C++,
- Java,
- PHP,
- C#,
- Python, etc.

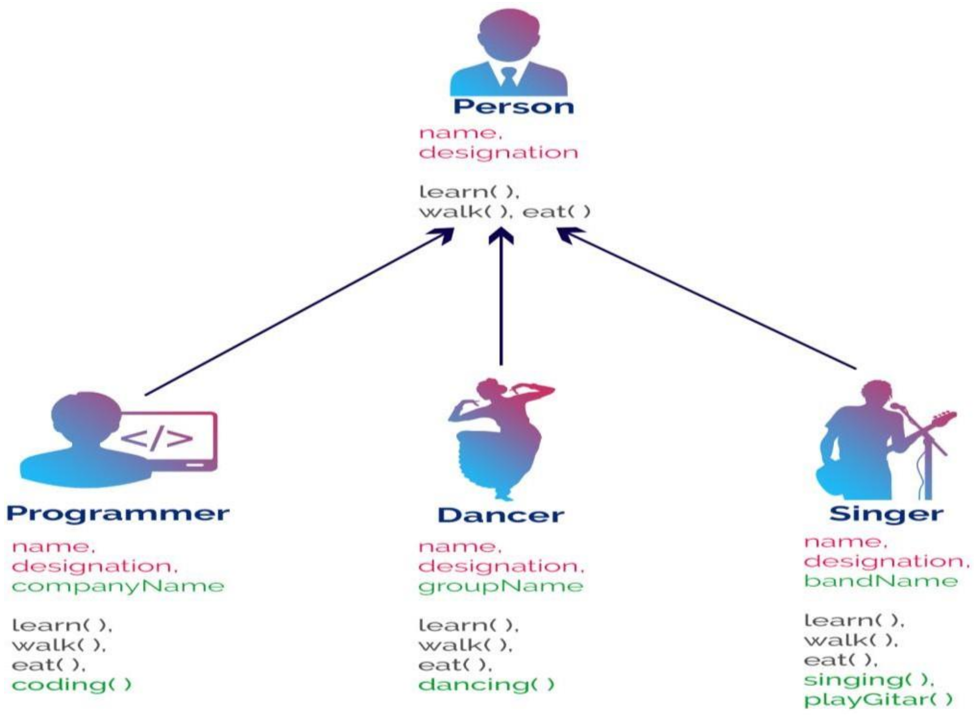
Encapsulation

- ➔ Encapsulation is the process of combining data and code into a single unit (object / class).
- ➔ In OOP, every object is associated with its data and code.
- ➔ In programming, data is defined as variables and code is defined as methods.
- ➔ The java programming language uses the class concept to implement encapsulation.



Inheritance

- ➔ Inheritance is the process of acquiring properties and behaviors from one object to anotherobject or one class to another class.
- ➔ In inheritance, we derive a new class from the existing class. Here, the new class acquires theproperties and behaviors from the existing class.
- ➔ In the inheritance concept, the class which provides properties is called as parent class and theclass which recieves the properties is called as child class.
- ➔ The parent class is also known as base class or supre class. The child class is also known asderived class or sub class.
- ➔ In the inheritance, the properties and behaviors of base class extended to its derived class, butthe base class never receive properties or behaviors from its derived class.
- ➔ In java programming language the keyword **extends** is used to implement inheritance.



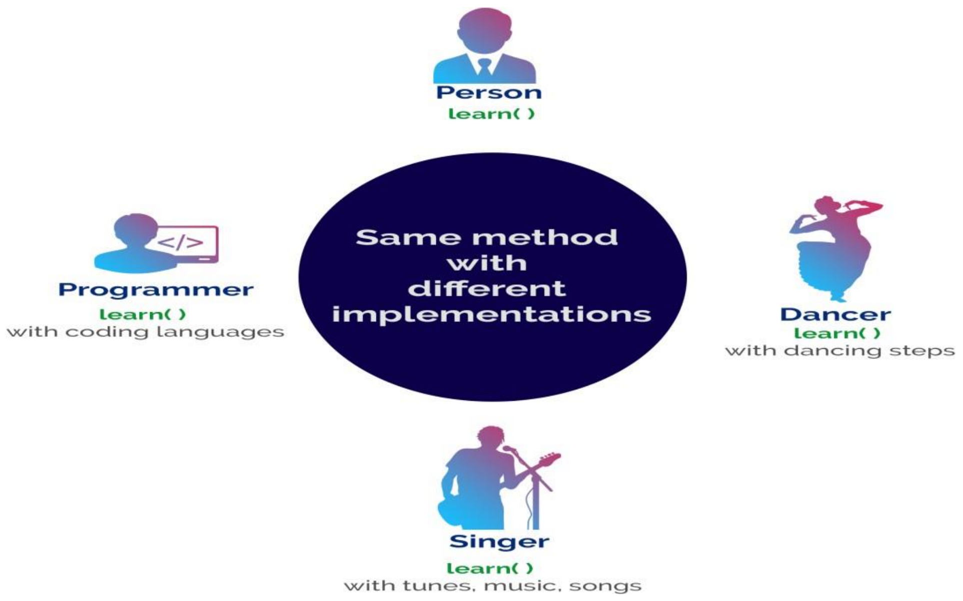
Polymorphism

Polymorphism is the process of defining same method with different implementation. That meanscreating multiple methods with different behaviors.

The java uses method overloading and method overriding to implement polymorphism.

Method overloading – Defining multiple methods with same name but with different parameters.

Method overriding - Defining multiple methods with same name and same parameters.



Abstraction

- Abstraction is hiding the internal details and showing only essential functionality.
- In the abstraction concept, we do not show the actual implementation to the end user, instead weprovide only essential things.
- For example, if we want to drive a car, we does not need to know about the internal functionalitylike how wheel system works? how brake system works? how music system works? etc.

Object
Level of abstraction increases
Becomes more specific

Java Buzz Words

Java is the most popular object-oriented programming language. Java has many advanced features, alist of key features is known as Java Buzz Words. Following are Java Buzz Words.

1. **Simple**
2. **Secure**
3. **Portable**
4. **Object-oriented**
5. **Robust**
6. **Architecture-neutral (or) Platform Independent**
7. **Multi-threaded**
8. **Interpreted**
9. **High performance**
10. **Distributed**
11. **Dynamic**

Simple

Java programming language is very simple and easy to learn, understand, and code. Most of the syntaxes in java follow basic programming language C and object-oriented programming concepts are similar to C++. In a java programming language, many complicated features like pointers, operator overloading, structures, unions, etc. have been removed. One of the most useful features is the garbage collector it makes java more simple.

Secure

Java is said to be more secure programming language because it does not have pointers concept, java provides a feature "applet" which can be embedded into a web application. The applet in java does not allow access to other parts of the computer, which keeps away from harmful programs like viruses and unauthorized access.

Portable

Portability is one of the core features of java which enables the java programs to run on any computer oroperating system. For example, an applet developed using java runs on a wide variety of CPUs, operating systems, and browsers connected to the Internet.

Object-oriented

Java is said to be a pure object-oriented programming language. In java, everything is an object. It supports all the features of the object-oriented programming paradigm. The primitive data types java also implemented as objects using wrapper classes, but still, it allows primitive data types to archive high-performance.

Robust

Java is more robust because the java code can be executed on a variety of environments, java has a strong memory management mechanism (garbage collector), java is a strictly typed language, it has a strong set of exception handling mechanism, and many more.

Architecture-neutral (or) Platform Independent

Java has invented to archive "write once; run anywhere, any time, forever". The java provides JVM(Java Virtual Machine) to to archive architectural-neutral or platform-independent. The JVM allows the java program created using one operating system can be executed on any other operating system.

Multi-threaded

Java supports multi-threading programming, which allows us to write programs that do multiple operations simultaneously.

Interpreted

Java enables the creation of cross-platform programs by compiling into an intermediate representation called Java bytecode. The byte code is interpreted to any machine code so that it runs on the native machine.

High performance

Java provides high performance with the help of features like JVM, interpretation, and its simplicity.

Distributed

Java programming language supports TCP/IP protocols which enable the java to support the distributed environment of the Internet. Java also supports Remote Method Invocation (RMI), this feature enables a program to invoke methods across a network.

Dynamic

Java is said to be dynamic because the java byte code may be dynamically updated on a runningsystem and it has a dynamic memory allocation and deallocation (objects and garbage collector).

An Overview of Java

- ➔ Java is a computer programming language. Java was created based on C and C++.
- ➔ Java uses C syntax and many of the object-oriented features are taken from C++.

- ➔ Before Java was invented there were other languages like COBOL, FORTRAN, C, C++, Small Talk, etc. These languages had few disadvantages which were corrected in Java.
- ➔ Java also innovated many new features to solve the fundamental problems which the previous languages could not solve.

History of Java:

- ➔ In 1990, Sun Micro Systems Inc. (US) was conceived a project to develop software for consumer electronic devices that could be controlled by a remote. This project was called Stealth Projectbut later its name was changed to Green Project.

- ➔ In January 1991, Project Manager James Gosling and his team members Patrick Naughton, Mike Sheridan, Chris Wrath, and Ed Frank met to discuss about this project.

- ➔ Gosling thought C and C++ would be used to develop the project. But the problem he faced with them is that they were system dependent languages. The trouble with C and C++ (and mostother languages) is that they are designed to be compiled for a specific target and could not be used on various processors, which the electronic devices might use.

- ➔ James Gosling with his team started developing a new language, which was completely system independent. This language was initially called **OAK**. Since this name was registered by some other company, later it was changed to **Java**.

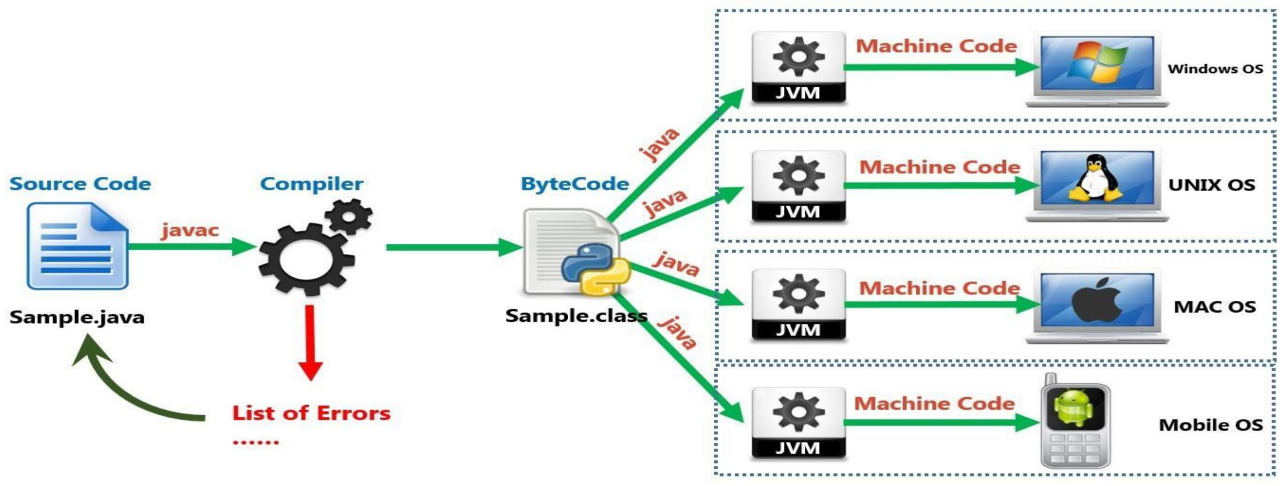
- ➔ James Gosling and his team members were consuming a lot of coffee while developing this language. Good quality of coffee was supplied from a place called "Java Island'. Hence they fixed the name of the language as Java. The symbol for Java language is cup and saucer.

- ➔ Sun formally announced Java at Sun World conference in 1995. On January 23rd 1996, JDK1.0 version was released.

Execution Process of Java Program

The following three steps are used to create and execute a java program.

1) Create a source code (.java file).
2) Compile the source code using javac command.
3) Run or execute .class file uisng java command.



Structure of the Java Program

//A Simple Java Program

```
import java.lang.System;
import java.lang.String;
class Sample
{
        public static void main(String args[])
        {
                System.out.print ("Hello world");
        }
}
```

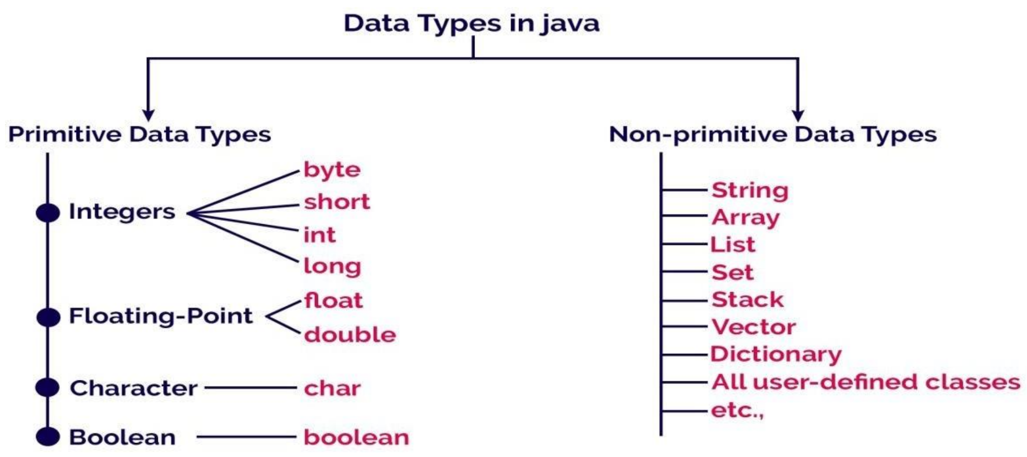
- ➔ As all other programming languages, Java also has a structure.
- ➔ The first line of the C/C++ program contains include statement. For example, <stdio.h> is the header file that contains functions, like printf (), scanf () etc. So if we want to use any of these functions, we should include this header file in C/ C++ program.
- ➔ Similarly in Java first we need to import the required packages. By default java.lang.* is imported.Java has several such packages in its library. A package is a kind of directory that contains a group of related classes and interfaces. A class or interface contains methods.
- ➔ Since Java is purely an Object Oriented Programming language, we cannot write a Java program without having at least one class or object. So, it is mandatory to write a class in Java program. We should use class keyword for this purpose and then write class name.
- ➔ In C/C++, program starts executing from main method similarly in Java, program starts executing from main method. The return type of main method is void because program starts executing from main method and it returns nothing.
- ➔ Since Java is purely an Object Oriented Programming language, without creating an object to a class it is not possible to access methods and members of a class. But main method is also a method inside a class, since program execution starts from main method we need to call main method without creating an object.
- ➔ Static methods are the methods, which can be called and executed without creating objects. Since we want to call main () method without using an object, we should declare main ()method as static. JVM calls main () method using its Classname.main () at the time of running the program.
- ➔ JVM is a program written by Java Soft people (Java development team) and main () is the method written by us. Since, main () method should be available to the JVM, it should be declared as public. If we don’t declare main () method as public, then it doesn’t make itself available to JVM and JVM cannot execute it.

- ➔ JVM always looks for main () method with String type array as parameter otherwise JVM cannot recognize the main () method, so we must provide String type array as parameter to main () method.
- ➔ A class code starts with a {and ends with a}. A class or an object contains variables and methods (functions). We can create any number of variables and methods inside the class. This is our first program, so we had written only one method called main ().
- ➔ Our aim of writing this program is just to display a string "Hello world". In Java, print () method is used to display something on the monitor.
- ➔ A method should be called by using objectname.methodname (). So, to call print () method,create an object to PrintStream class then call objectname.print () method.
- ➔ An alternative is given to create an object to PrintStream Class i.e. System.out. Here, System is the class name and out is a static variable in System class. out is called a field in System class. When we call this field a PrintStream class object will be created internally. So, we can call print()method as:

  System.out.print ("Hello world");
- ➔ println () is also a method belonging to PrintStream class. It throws the cursor to the next lineafter displaying the result.
- ➔ In the above Sample program System and String are the classes present in java.langpackage.

Java Data Types

The classification of data item is called data type. The data type is a category of data stored invariables. In java, data types are classified into two types and they are as follows.



The primitive data types are built-in data types and they specify the type of value stored in a variable andthe memory size. The primitive data types do not have any additional methods.

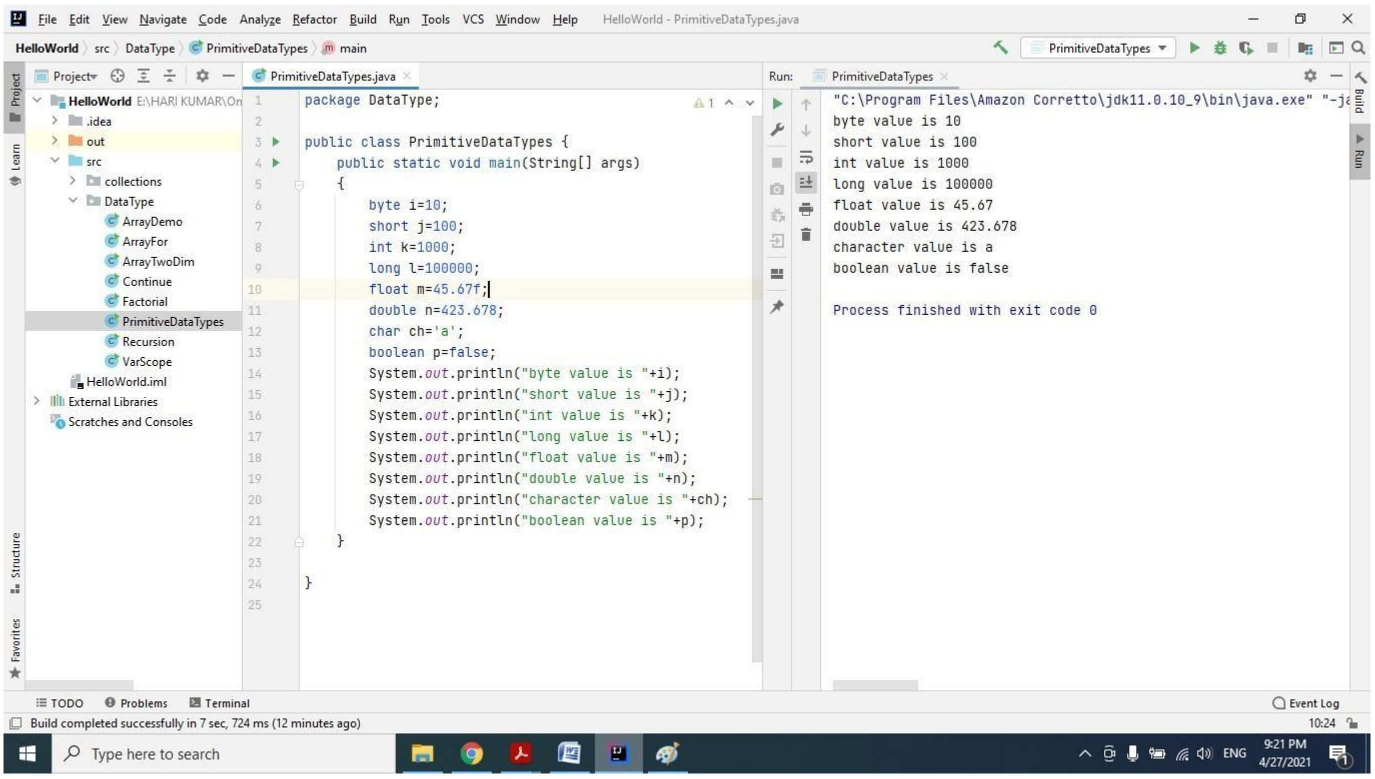
In java, primitive data types includes **byte**, **short**, **int**, **long**, **float**, **double**, **char**, and **boolean**.

Data type	Meaning	Memory size	Range	Default Value
byte	Whole numbers	1 byte	-128 to +127	0
short	Whole numbers	2 bytes	-32768 to +32767	0
int	Whole numbers	4 bytes	-2,147,483,648 to +2,147,483,647	0
long	Whole numbers	8 bytes	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807	0L
float	Fractional numbers	4 bytes	-	0.0f
double	Fractional numbers	8 bytes	-	0.0d
char	Single character	2 bytes	0 to 65535	\u0000
boolean	unsigned char	1 bit	0 or 1	0 (false)

// Java program to illustrate primitive data types in java and their default values.

```
public class PrimitiveDataTypes
{
        public static void main(String[] args)
        {
                byte i;
                short  j;
                int  k;
                long l;
                float m;
                double n;
                char ch;
                boolean p;
                System.out.println("byte value is ”+i);
                System.out.println("short value is ”+j);
                System.out.println("int value is ”+k);
                System.out.println("long value is ”+l);
                System.out.println("float value is ”+m);
                System.out.println("double value is ”+n);
                System.out.println("character value is”+ch);
                System.out.println("boolean value is"+p);
        }
}
```

OUTPUT



```
package DataType;

public class PrimitiveDataTypes {
    public static void main(String[] args)
    {
        byte i=10;
        short j=100;
        int k=1000;
        long l=100000;
        float m=45.67f;
        double n=423.678;
        char ch='a';
        boolean p=false;
        System.out.println("byte value is "+i);
        System.out.println("short value is "+j);
        System.out.println("int value is "+k);
        System.out.println("long value is "+l);
        System.out.println("float value is "+m);
        System.out.println("double value is "+n);
        System.out.println("character value is "+ch);
        System.out.println("boolean value is "+p);
    }

}
```

```
"C:\Program Files\Amazon Corretto\jdk11.0.10_9\bin\java.exe" "-ja
byte value is 10
short value is 100
int value is 1000
long value is 100000
float value is 45.67
double value is 423.678
character value is a
boolean value is false

Process finished with exit code 0
```

Build completed successfully in 7 sec, 724 ms (12 minutes ago)

Non-primitive Data Types

- ➔ In java, non-primitive data types are the reference data types or user-created data types.
- ➔ All non-primitive data types are implemented using object concepts.
- ➔ Every variable of the non-primitive data type is an object.
- ➔ The non-primitive data types may use additional methods to perform certain operations.
- ➔ The default value of non-primitive data type variable is null.

In java, examples of non-primitive data types are

String, **Array**, **List**, **Queue**, **Stack**, **Class**, **Interface**, etc.

Primitive Vs Non-primitive Data Types

Primitive Data Type	**Non-primitive Data Type**
These are built-in data types	These are created by the users
Does not support additional methods	Support additional methods
Always has a value	It can be null
Starts with lower-case letter	Starts with upper-case letter
Size depends on the data type	Same size for all

Java Variables

A variable is a named memory location used to store a data value. A variable can be defined as acontainer that holds a data value.

In java, we use the following syntax to create variables.

Syntax

data_type variable_name;
(or)
data_type variable_name_1, variable_name_2,...;(or)
data_type variable_name = value;(or)
data_type variable_name_1 = value, variable_name_2 = value,...;

In java programming language variables are clasiffied as follows.

1) **Local variables**
2) **Instance variables or Member variables or Global variables**
3) **Static variables or Class variables**
4) **Final variables**

Local variables

- The variables declared inside a method or a block are known as local variables.
- A local variable is visible within the method in which it is declared.
- The local variable is created when execution control enters into the method or block anddestroyed after the method or block execution completed.

Instance variables or member variables

- The variables declared inside a class and outside any method, constructor or block are known asinstance variables or member variables.
- These variables are visible to all the methods of the class.
- The changes made to these variables by method affects all the methods in the class.
- These variables are created separate copy for every object of that class.

Static variables or Class variables

- A static variable is a variable that declared using **static** keyword.
- The instance variables can be static variables but local variables can not.
- Static variables are initialized only once, at the start of the program execution.
- The static variable only has one copy per class irrespective of how many objects we create.

//java program to illustrate static variable in java.

```
public class StaticVariablesExample

{
        int x, y;  // Instance variables
        static int z;  // Static variable

        StaticVariablesExample(int x, int y){
                this.x = x;
                this.y = y;
        }

        public void show() {
         int a; // Local variables
                System.out.println("Inside show method,");
                System.out.println("x = " + x + ", y = " + y + ", z = " + z);
        }

        public static void main(String[] args) {
                StaticVariablesExample obj_1 = new StaticVariablesExample(10, 20);
                StaticVariablesExample obj_2 = new StaticVariablesExample(100, 200);
                obj_1.show();
                StaticVariablesExample.z = 1000;
                obj_2.show();
        }
}
```

final variables

- ➔ A final variable is a variable that declared using final keyword.
- ➔ The final variable is initialized only once, and does not allow any method to change it's valueagain.
- ➔ The variable created using final keyword acts as constant.
- ➔ All variables like local, instance, and static variables can be final variables.

// java program to illustrate final variable in java.

```
        public class FinalVariableExample
        {
                final int a = 10;

                void show()
                 {
                        System.out.println("a = " + a);
                        a = 20;         //Error due to final variable cann't be modified
                }

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

```
                FinalVariableExample obj = new FinalVariableExample();
                obj.show();
        }
}
```

Arrays

- ➔ An array is a collection of similar data values with a single name.
- ➔ An array can also be defined as, a special type of variable that holds multiple values of thesame data type at a time.
- ➔ In java, arrays are objects and they are created dynamically using **new** operator.
- ➔ Every array in java is organized using index values. The index value of an array starts with '0'and ends with 'zise-1'.
- ➔ We use the index value to access individual elements of an array.

In java, there are two types of arrays and they are as follows.

1) **One Dimensional Array**
2) **Multi Dimensional Array**

Creating an array

In the java programming language, an array must be created using new operator and with a specificsize. The size must be an integer value but not a byte, short, or long.

Syntax to create an array

data_type array_name[ ] = new data_type[size];
(or)
data_type[ ] array_name = new data_type[size];

// Example Program

```
public class ArrayExample
{
        public static void main(String[] args)
        {
                int list[] = new int[5];

                list[0] = 10;
                System.out.println("Value at index 0 - " + list[0]);
                System.out.println("Length of the array - " + list.length);
        }
}
```

OUTPUT

```
Value at index 0 - 10
Length of the array - 5
```

Looping through an array:

An entire array is accessed using either simple for statement or for-each statement.

```
public class ArrayDemo
{

    public static void main(String[] args)
    {
        String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
        System.out.println("lenth of array"+cars.length);
        for (int i = 0; i < cars.length; i++)
        {
            System.out.println(cars[i]);
        }
    }
}
```

OUTPUT

```
lenth of array4
Volvo
BMW
Ford
Mazda
```

Multidimensional Array

- Multidimensional arrays are arrays of arrays.
- To create a multidimensional array variable, specify each additional index using another setof square brackets.
- When we create a two-dimensional array, it created with a separate index for rows andcolumns.
- The individual element is accessed using the respective row index followed by the columnindex.
- When an array is initialized at the time of declaration, it need not specify the size of the arrayand use of the new operator.

Syntax

data_type array_name[ ][ ] = new data_type[rows][columns];(or)

data_type[ ][ ] array_name = new data_type[rows][columns];

int matrix_a[ ][ ] = {{1, 2},{3, 4},{5, 6}};

// Java program to demonstrate Multidimensional Array

```
public class ArrayTwoDim
{
    public static void main(String[] args)
    {
        int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
        System.out.println("row size:"+myNumbers.length);
        for (int i = 0; i < myNumbers.length; i++)
        {
            System.out.println("column size: "+myNumbers[i].length);
            for(int j = 0; j < myNumbers[i].length; j++)
            {
                System.out.print(myNumbers[i][j]+"\t");
            }
            System.out.println();
        }
    }

}
```

OUTPUT

```
row size:2
column size: 4
1	2	3	4
column size: 3
5	6	7
```

Operators

An operator is a symbol used to perform arithmetic and logical operations.In

java, operators are classified into the following four types.

- → Arithmetic Operators
- → Relational (or) Comparison Operators
- → Logical Operators
- → Assignment Operators
- → Bitwise Operators
- → Conditional Operators

Arithmetic Operators

- ➔ In java, arithmetic operators are used to performing basic mathematical operations like addition,subtraction, multiplication, division, modulus, increment, decrement, etc.,

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus - Remainder of the Division
++	Increment
--	Decrement

- The addition operator can be used with numerical data types and character or string data type. When it is used with numerical values, it performs mathematical addition and when it is used with character or string data type values, it performs concatination (appending).
- The modulus (remainder of the division) operator is used with integer data type only
- The increment and decrement operators are used as pre-increment or pre-decrement and post-increment or post-decrement.
- When they are used as pre, the value is get modified before it is used in the actual expresion and when it is used as post, the value is get modified after the the actual expression evaluation.

// Java program to demonstrate Arithmetic Operators

```
public class ArithmeticOperators
{
        public static void main(String[] args)
        {

         int a = 10, b = 20, result;

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

         result = a + b;
         System.out.println("Addition : " + a + " + " + b + " = " + result);

         result = a - b;
         System.out.println("Subtraction : " + a + " - " + b + " = " + result);

         result = a * b;
         System.out.println("Multiplucation : " + a + " * " + b + " = " + result);

         result = b / a;
         System.out.println("Division : " + b + " / " + a + " = " + result);

         result = b % a;
         System.out.println("Modulus : " + b + " % " + a + " = " + result);

         result = ++a;
         System.out.println("Pre-increment : ++a = " + result);

         result = b--;
         System.out.println("Post-decrement : b-- = " + result);
        }
}
```

OUTPUT

```
a = 10, b = 20
Addition : 10 + 20 = 30
Subtraction : 10 - 20 = -10
```

Multiplucation : 10 * 20 = 200
Division : 20 / 10 = 2
Modulus : 20 % 10 = 0
Pre-increment : ++a = 11
Post-decrement : b-- = 20

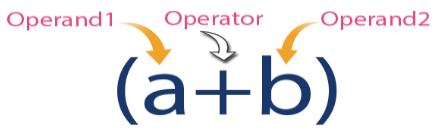
Expressions

- An expression is a collection of operators and operands that represents a specific value.
- An **operator** is a symbol that performs tasks like arithmetic operations, logical operations, andconditional operations, etc.
- **Operands** are the values on which the operators perform the task. Here operand can be a directvalue or variable or address of memory location.

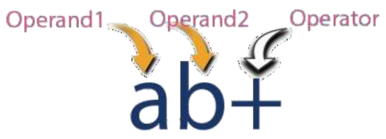
Expression Types

1. **Infix Expression**
2. **Postfix Expression**
3. **Prefix Expression**

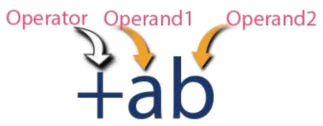
Infix Expression: The expression in which the operator is used between operands is called infixexpression.



Postfix Expression: The expression in which the operator is used after operands is called postfixexpression.



Prefix Expression:The expression in which the operator is used before operands is called a prefixexpression.

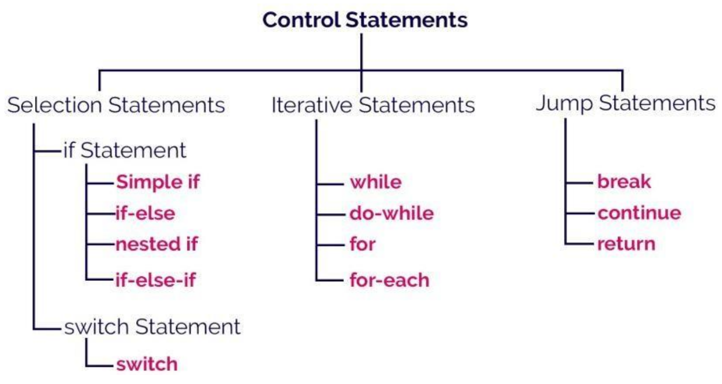


Control Statements

- The default execution flow of a program is a sequential order.
- Sometimes, we may want to jump from line to another line, we may want to skip a part of the program, or sometimes we may want to execute a part of the program again and again. To solvethis problem, java provides control statements.
- The control statements are used to control the order of execution according to our requirements.

Types of Control Statements

1. Selection Control Statements (Decision Making Statements)
2. Iterative Control Statements (Looping Statements)
3. Jump Statements



Selection Control Statements

- ➔ The selection statements are also known as decision making statements or branchingstatements.
- ➔ The selection statements are used to select a part of the program to be executed based on acondition.

Java provides the following selection statements.

1. if statement
2. if-else statement
3. if-elif statement
4. nested if statement
5. switch statement

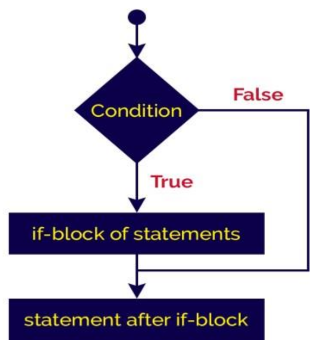
if statement

- ➔ if statement is used to test a condition and decide the execution of a block of statements basedon that condition result.
- ➔ The if statement checks, the given condition then decides the execution of a block ofstatements.
- ➔ If the condition is True, then the block of statements is executed and if it is False, then the blockof statements is ignored.

Syntax

```
if(condition){
        if-block of statements;
        ...
}
statement after if-block;
...
```

Flow of execution



// Java program to demonstrate If statement

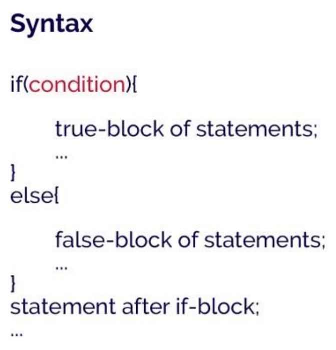
```
public class IfDemo
{
    public static void main(String[] args) {
        int x = 20;
        int y = 18;
        if (x > y) {
            System.out.println("x is greater than y");
        }
    }
}
```

Output

```
x is greater than y
```

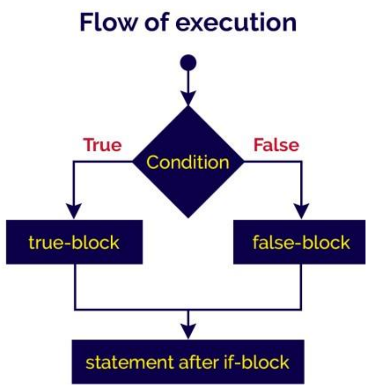
if-else statement

- if-else statement is used to test a condition and pick the execution of a block of statements out oftwo blocks based on that condition result.
- The if-else statement checks the given condition then decides which block of statements to beexecuted based on the condition result.
- If the condition is True, then the true block of statements is executed and if it is False, then thefalse block of statements is executed.



Syntax

```
if(condition){
        true-block of statements;
        ...
}
else{
        false-block of statements;
        ...
}
statement after if-block;
...
```



// Java program to demonstrate If-else statement

```
public class IfElseDem
{
    public static void main(String[] args) {
        int time = 20;
        if (time < 18) {
            System.out.println("Good day.");
        } else {
            System.out.println("Good  evening.");
        }
    }
}
```

Output

```
Good evening.
```

Nested if statement

Writing an if statement inside another if-statement is called nested if statement.The general syntax of the nested if-statement is as follows.

```
if(condition_1)
{
    if(condition_2)
{
        inner if-block of statements;
          }
    ...
}
```

// Java program to demonstrate nested If statement

```java
import java.util.Scanner;

public class NestedIfStatementTest
{
    public  static  void  main(String[]  args)  {

        Scanner read = new Scanner(System.in);
        System.out.print("Enter any number: ");
        int num = read.nextInt();

        if (num < 100) {
            System.out.println("\nGiven number is below  100");
            if (num % 2 == 0)
                System.out.println("And  it  is  EVEN");
            else
                System.out.println("And  it  is  ODD");
        } else
            System.out.println("Given number is not below 100");

        System.out.println("\nWe are outside the if-block!!!");

    }
}
```

Output

```
Enter any number: 95
Given number is below 100
And it is ODD
We are outside the if-block!!!
```

if-else if statement

Writing an if-statement inside else of an if statement is called if-else-if statement.The general **syntax** of the an if-else-if statement is as follows.

```
if (condition1)
{
  // block of code to be executed if condition1 is true
}
else if (condition2)
{
  // block of code to be executed if the condition1 is false and condition2 is true
}
else
{
  // block of code to be executed if the condition1 is false and condition2 is false
}
```

// Java program to demonstrate If-else if statement

```
import java.util.Scanner;

public class IfElseIf
{
    public static void main(String[] args)
    {
        Scanner s = new Scanner(System.in);
        System.out.println("Enter time");
        int time = s.nextInt();
        if (time < 10)
        {
            System.out.println("Good morning.");
        }
        else if (time < 20)
        {
            System.out.println("Good day.");
        }
        else
            {
            System.out.println("Good evening.");
            }
    }
}
```

Output

```
Enter time
21
Enter any three numbers: Good evening.
```

switch statement

- Using the switch statement, one can select only one option from more number of options veryeasily.
- In the switch statement, we provide a value that is to be compared with a value associated witheach option.
- Whenever the given value matches the value associated with an option, the execution starts fromthat option.
- In the switch statement, every option is defined as a **case**.

Syntax

```
switch(expression)
{
  case x:    // code block
    break;
  case y:    // code block
    break;
  default:    // code block
}
```

// Java program to demonstrate If-else if statement

```
import java.util.Scanner;

public class switchDemo
{
    public static void main(String[] args)
    {
        Scanner s = new Scanner(System.in);
        System.out.println("Enter a number");
        int day = s.nextInt();
        switch (day)
        {
            case 1:
                System.out.println("Monday");
                break;
            case 2:
                System.out.println("Tuesday");
                break;
            case 3:
                System.out.println("Wednesday");
                break;
            case 4:
                System.out.println("Thursday");
                break;
            case 5:
                System.out.println("Friday");
                break;
            case 6:
                System.out.println("Saturday");
                break;
            case 7:
                System.out.println("Sunday");
                break;
            default:
                System.out.println("Not Valid");
        }
    }
}
```

Output

```
Enter a number
7
Sunday
```

Iterative Control Statements

- ➔ The iterative statements are also known as looping statements or repetitive statements.
- ➔ The iterative statements are used to execute a part of the program repeatedly as long as thegiven condition is True.
- ➔ Using iterative statements reduces the size of the code, reduces the code complexity, makes itmore efficient, and increases the execution speed.

Java provides the following iterative statements.

1) while statement
2) do-while statement
3) for statement
4) for-each statement

while statement

- ➔ The while statement is used to execute a single statement or block of statements repeatedly aslong as the given condition is TRUE.
- ➔ The while statement is also known as Entry control looping statement since condition is tested atbeginning of the loop.

The **syntax** and execution flow of while statement is as follows.

```
variable initialization;
while (condition)
{
  // code block to be executed
Modification;
}
```

// Java program to demonstrate while loop

```
class WhileTest {

        public static void main(String[] args) {

                int num = 1;

                while(num <= 10) {
                        System.out.println(num);
                        num++;
                }

                System.out.println("Statement after while!");

        }
```

Output

```
1
2
3
4
5
```

```
6
7
8
9
10
Statement after while!
```

do-while statement

- The do-while statement is used to execute a single statement or block of statements repeatedlyas long as given the condition is TRUE.
- The do-while statement is also known as the **Exit control looping statement**. The do-whilestatement has the following syntax.

Syntax

```
do
{
  // code block to be executed
}
while (condition);
```

- The loop will always be executed at least once, even if the condition is false, because the codeblock is executed before the condition is tested:

// Java program to demonstrate do-while loop

```
public class DoWhileTest {

        public static void main(String[] args) {

                int num = 1;

                do {
                        System.out.println(num);
                        num++;
                }while(num <= 10); System.out.println("Statement

                after do-while!");

        }

}
```

Output

```
1
2
3
4
5
6
7
8
9
```

```
10
Statement after do-while!
```

for statement

- The for statement is used to execute a single statement or a block of statements repeatedly aslong as the given condition is TRUE. The for statement has the following syntax

Syntax

```
for (statement 1; statement 2; statement 3) {

  // code block to be executed

}
```

- **Statement 1** is executed (one time) before the execution of the code block.
- **Statement 2** defines the condition for executing the code block.
- **Statement 3** is executed (every time) after the code block has been executed**.**
- In for-statement, the execution begins with the **initialization** statement.
- After the initialization statement, it executes **Condition**. If the condition is evaluated to true, thenthe block of statements executed otherwise it terminates the for-statement.
- After the block of statements execution, the **modification** statement gets executed, followed bycondition again.

// Java program to demonstrate for loop

```
class ForTest {

        public static void main(String[] args) {

                for(int i = 0; i < 10; i++) {
                        System.out.println("i = " + i);
                }

                System.out.println("Statement after for!");
        }

}
```

Output

```
i = 0
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
Statement after for!
```

for-each statement

- ➔ The Java for-each statement was introduced since Java 5.0 version.
- ➔ It provides an approach to traverse through an array or collection in Java.
- ➔ The for-each statement also known as enhanced for statement
- ➔ The for-each statement executes the block of statements for each element of the given array orcollection.

Syntax

```
for (type variableName : arrayName) {

  // code block to be executed

}
```

// Java program to demonstrate for-each loop

```
public class ForEachTest
 {
  public static void main(String[] args) {
    String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
    for (String i : cars) {
      System.out.println(i);
    }
  }
}
```

Output

```
Volvo
BMW
Ford
Mazda
```

Jump Statements

The java programming language supports jump statements that used to transfer execution control fromone line to another line.

The java programming language provides the following jump statements.

- break statement
- continue statement
- labelled break and continue statements
- return statement

break statement

- ➔ The break statement in java is used to terminate a switch or looping statement.

- That means the break statement is used to come out of a switch statement and a loopingstatement like while, do-while, for, and for-each.
- Using the break statement outside the switch or loop statement is not allowed.

// Java program to demonstrate break statement

```
class JavaBreakStatement
 {
        public static void main(String[] args)
         {
                int list[] = {10, 20, 30, 40, 50};
                for(int i : list)
                {
                        if(i == 30)
                                break;
                        System.out.println(i);
                }
        }
}
```

Output

```
10
20
```

continue statement

- The continue statement is used to move the execution control to the beginning of the loopingstatement.
- When the continue statement is encountered in a looping statement, the execution control skipsthe rest of the statements in the looping block and directly jumps to the beginning of the loop.
- The continue statement can be used with looping statements like while, do-while, for, and for-each.
- When we use continue statement with while and do-while statements, the execution controldirectly jumps to the condition.
- When we use continue statement with for statement the execution control directly jumps to themodification portion (increment/decrement/any modification) of the for loop

// Java program to demonstrate continue statement

```
class ContinueStatement {

        public static void main(String[] args) {

                int list[] = {10, 20, 30, 40, 50};

                for(int i : list) {
                        if(i == 30)
                                continue;
                        System.out.println(i);
                }
        }
}
```

Output

```
10
20
40
50
```

Labelled break and continue statement

- ➔ The java programming langauge does not support **goto** statement, alternatively, the break andcontinue statements can be used with label.
- ➔ The labelled break statement terminates the block with specified label. The labbeled contonuestatement takes the execution control to the beginning of a loop with specified label.

// Java program to demonstrate labeled break and continue statement

// Java program to demonstrate labeled break and continue statement

```
import  java.util.Scanner;

class JavaLabelledStatement {
        public static void main(String args[]) {
                Scanner read = new Scanner(System.in);
                reading: for (int i = 1; i <= 3; i++) {
                        System.out.print("Enter a even number: ");
                        int value = read.nextInt();
                        verify: if (value % 2 == 0) {
                                System.out.println("\nYou won!!!");
                                System.out.println("Your score is " + i*10 + " out of 30.");
                                break reading;
                        } else {
                                System.out.println("\nSorry try again!!!");
                                System.out.println("You left with " + (3-i) + " more options...");
                                continue reading;
                        }
                }
        }
}
```

Output

```
Enter a even number
Sorry try again!!!
You left with " + (3-i) + " more options...
Enter a even number
You won!!!
Your score is 20  out of 30
```

return statement

- ➔ return statement is useful to terminate a method and come back to the calling method.

- ➔ return statement in main method terminates the application.
- ➔ return statement can be used to return some value from a method to a calling method

Syntax:

```
    return;
    (or)
    return value; // value may be of any type
```

// Java program to demonstrate return statement

```
class ReturnDemo {
        public static void main(String args[]) {
                boolean t = true;
                System.out.println (“Before the return”);if
                (t)
                return;
                System.out.println (“This won’t execute”);
        }
}
```

Output

```
Before the return
```

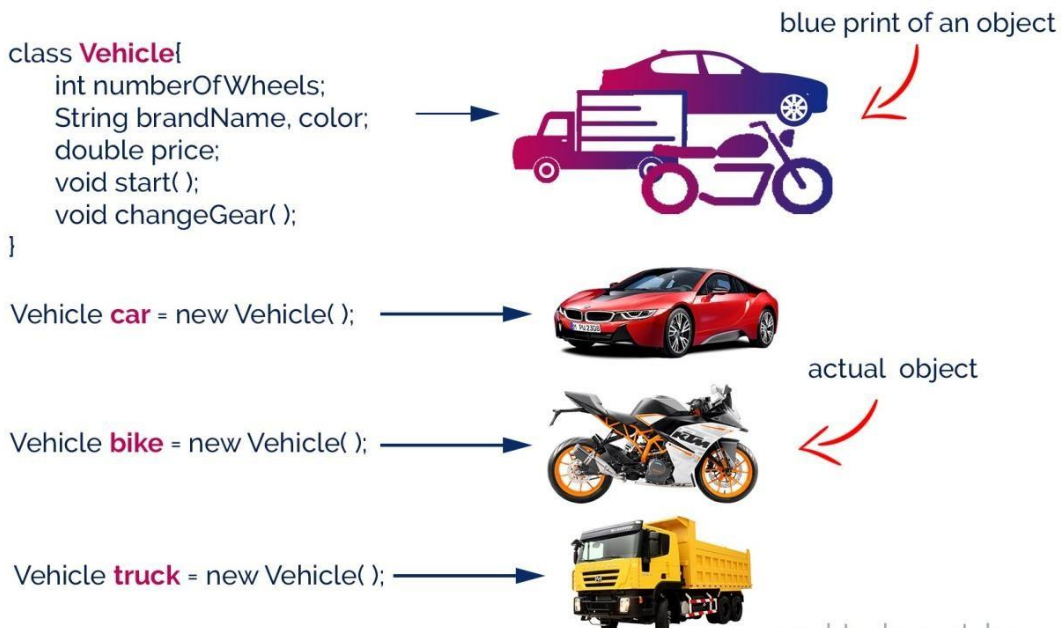
Java Classes

- ➔ Java is an object-oriented programming language, so everything in java program must be basedon the object concept.
- ➔ In a java programming language, the class concept defines the skeleton of an object.
- ➔ The java class is a template of an object.
- ➔ The class defines the blueprint of an object.
- ➔ Every class in java forms a new data type.
- ➔ Once a class got created, we can generate as many objects as we want.
- ➔ Every class defines the properties and behaviors of an object.
- ➔ All the objects of a class have the same properties and behaviors that were defined in the class.

Every class of java programming language has the following characteristics.

- **Identity** - It is the name given to the class.
- **State** - Represents data values that are associated with an object.
- **Behavior** - Represents actions can be performed by an object.

Look at the following picture to understand the class and object concept



Class

- ➔ In java, we use the keyword class to create a class. A class in java contains properties asvariables and behaviors as methods. Following is the syntax of class in the java.
- ➔ The ClassName must begin with an alphabet, and the Upper-case letter is preferred.
- ➔ The ClassName must follow all naming rules.

Syntax

```
class <ClassName>
{
    data members declaration;
    methods defination;
}
```

Object

- ➔ An Object is a real time entity.
- ➔ An object is an instance of a class. Instance means physically happening. An object will havesome properties and it can perform some actions.
- ➔ When an object of a class is created, the class is said to be instantiated.
- ➔ All the objects that are created using a single class have the same properties and methods. But the value of properties is different for every object.
- ➔ Object contains variables and methods. The objects which exhibit similar properties and actions are grouped under one class.
- ➔ "To give a real world analogy, a house is constructed according to a specification. Here, the specification is a blueprint that represents a class, and the constructed house represents the object".

Syntax

<ClassName> <objectName> = new <ClassName>( );Student s

= new Student ();

Now we can access the properties and methods of a class by using object with dot operator as:s.rollNo, s.name, s.display ()

String Handling

Java String

string is basically an object that represents sequence of char values. An array of characters workssame as java string. For example:

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

s same as:

```
String s="javatpoint";
```

1. **Java String** class provides a lot of methods to perform operations on string such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.
2. The java.lang.String class implements *Serializable*, *Comparable* and *CharSequence* interfaces.CharSequence

The CharSequence interface is used to represent sequence of characters. It is implemented by String, StringBuffer and StringBuilder classes. It means, we can create string in java by using these 3 classes.

The java String is immutable i.e. it cannot be changed. Whenever we change any string, a new instance is created. For mutable string, you can use StringBuffer and StringBuilderclasses.

➔ In Java, string is basically an object that represents sequence of char values.
➔ An array of characters works same as Java string.
➔ A String represents group of characters

Creating Strings:

➔ We can declare a String variable and directly store a String literal using assignment operator.

String str = "Hello";

➔ We can create String object using new operator with some data.

String s1 = new String ("Java");

➔ We can create a String by using character array also.char

arr[] = { 'p','r','o',’g’,’r’,’a’,’m’};

➔ We can create a String by passing array name to it, as:String s2 =

new String (arr);

➔ We can create a String by passing array name and specifying which characters we need:String s3 =

new String (str, 2, 3);

➔ Here starting from 2nd character a total of 3 characters are copied into String s3.

String Class Methods

Method	Description	Return
charAt(int)	Finds the character at given index	char
length()	Finds the length of given string	int
compareTo(String)	Compares two strings	int
compareToIgnoreCase(String)	Compares two strings, ignoring case	int
concat(String)	Concatenates the object string with argument string.	String
contains(String)	Checks whether a string contains sub-string	boolean
contentEquals(String)	Checks whether two strings are same	boolean
equals(String)	Checks whether two strings are same	boolean
equalsIgnoreCase(String)	Checks whether two strings are same, ignoring case	boolean
startsWith(String)	Checks whether a string starts with the specified string	boolean
Method	**Description**	**Return**
endsWith(String)	Checks whether a string ends with the specified string	boolean
getBytes()	Converts string value to bytes	byte[]
hashCode()	Finds the hash code of a string	int
indexOf(String)	Finds the first index of argument string in object string	int
lastIndexOf(String)	Finds the last index of argument string in object string	int
isEmpty()	Checks whether a string is empty or not	boolean
replace(String, String)	Replaces the first string with second string	String
replaceAll(String, String)	Replaces the first string with second string at all	String
substring(int, int)	Extracts a sub-string from specified start and end	String
toLowerCase()	Converts a string to lower case letters	String
toUpperCase()	Converts a string to upper case letters	String
trim()	Removes whitespace from both ends	String

toString(int)	Converts the value to a String object	String
split(String)	splits the string matching argument string	String[]
intern()	returns string from the pool	String
join(String, String, ...)	Joins all strings, first string as delimiter.	String

// Java program to demonstrate string functions

```
class JavaStringExample {

        public static void main(String[] args) {
                String title = "Java Tutorials";
                String siteName = "Java Programming";
                System.out.println("Length of title: " + title.length());
                System.out.println("Char at index 3: " + title.charAt(3));
                System.out.println("Index of 'T': " + title.indexOf('T'));
                System.out.println("Last index of 'a': " + title.lastIndexOf('a'));
                System.out.println("Empty: " + title.isEmpty());
                System.out.println("Ends with '.com': " + siteName.endsWith(".com"));
                System.out.println("Equals: " + siteName.equals(title));
                System.out.println("Sub-string: " + siteName.substring(9, 14));
                System.out.println("Upper case: " + siteName.toUpperCase());
        }
}
```

Output

```
Length of title: 14
Char at index 3: a
Index of 'T': 5
Last index of 'a': 11
Empty: false
Ends with '.com': false
Equals: false
Sub-string: rammi
Upper case: JAVA PROGRAMMING
```

StringBuffer

- ➔ We can divide objects broadly as mutable and immutable objects.
- ➔ Mutable objects are those objects whose contents can be modified.
- ➔ Immutable objects are those objects, once created can not be modified.
- ➔ String objects are immutable.

- ➔ The methods that directly manipulate data of the object are not available in String class.
- ➔ StringBuffer objects are mutable, so they can be modified.

Creating StringBuffer

- We can create a StringBuffer object by using new operator and pass the string to the object,as:

StringBuffer sb = new StringBuffer ("Kiran");

- We can create a StringBuffer object by first allotting memory to the StringBuffer objectusing new operator and later storing the String into it as:

StringBuffer sb = new StringBuffer (30);

- ➔ In general a StringBuffer object will be created with a default capacity of 16 characters.
- ➔ Even if we declare the capacity as 30, it is possible to store more than 30 characters into StringBuffer.

// Java program using StringBuffer class methods

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Mutable {
   public static void main(String[] args) throws IOException
   {
      // to accept data from keyboard
      BufferedReader br=new BufferedReader (new InputStreamReader (System.in));
      System.out.print("Enter sur name : ");
      String sur=br.readLine( );
      System.out.print("Enter mid name : ");
      String mid=br.readLine ( );
      System.out.print("Enter last name : ");
      String last=br.readLine( );
      // create String Buffer object
      StringBuffer sb=new StringBuffer( );
      // append sur, last to
      sbsb.append(sur);
      sb.append(last);
      //insert mid after
      surint n=sur.length (
      ); sb.insert(n, mid);
      //display full name
      System.out.println ("Full name = "+sb);
      System.out.println ("In reverse ="+sb.reverse ( ));
   }
}
```

Output

```
Enter sur name : Azaad
Enter mid name :
ChandraEnter last name :
Shekar
Full name =
AzaadChandraShekarIn reverse
=rakehSardnahCdaazA
```

Constructors

- ➔ A constructor is similar to a method that initializes the instance variables of a class.
- ➔ A constructor name and classname must be same.
- ➔ A constructor may have or may not have parameters. Parameters are local variables to
- ➔ receive data.
- ➔ A constructor does not return any value, not even void.
- ➔ A constructor without any parameters is called default constructor.

e.g. class Student

```
        {
                int rollNo;
                String name;
                Student ()
                {
                        rollNo = 101;
                        name = "Kiran";
                }
}
```

- ➔ A constructor with one or more parameters is called parameterized constructor.

e.g.

```
class Student
{       int rollNo;
        String name;
        Student (int r, String n)
        {
                rollNo = r;
                name = n;
        }
}
```

- ➔ A constructor is called and executed at the time of creating an object.
- ➔ A constructor is called only once per object.
- ➔ Default constructor is used to initialize every object with same data where as parameterizedconstructor is used to initialize each object with different data.
- ➔ If no constructor is written in a class then java compiler will provide default values.
- ➔ The parameterized constructor of parent class must be called explicitly usingthe super keyword.

// Java program to initialize student details using <u>default constructor</u> and display the same.

```
class Student
{
            int rollNo;
            String name;
            Student ()
            {
                         rollNo = 101;
                        name = "Suresh";
            }
            void display ()
            {
                        System.out.println ("Student Roll Number is: " + rollNo);
                        System.out.println ("Student Name is: " + name);
            }
            }
            class StudentDemo
            {
            public static void main(String args[])
            {
                        Student s1 = new Student ();
                        System.out.println ("s1 object contains: ");
                        s1.display ();
                        Student s2 = new Student ();
                        System.out.println ("s2 object contains: ");
                        s2.display ();
            }
}
```

OUTPUT

```
s1 object contains:
Student Roll Number is: 101
Student Name is: Suresh
s2 object contains:
Student Roll Number is: 101
Student Name is: Suresh
```

// Java program to initialize student details using Parameterized constructor and display thesame.

```
class Student
{
             int rollNo;
            String
            name;
            Student (int r, String n)
            {
                        rollNo =
                        r;name =
                        n;
            }
            void display ()
            {
                System.out.println ("Student Roll Number is: " + rollNo);
                System.out.println ("Student Name is: " + name);
        }
        }
        class StudentDemo
        {
        public static void main(String args[])
        {
                Student s1 = new Student (101, “Suresh”);
                System.out.println (“s1 object contains: “
                );s1.display ();
                Student   s2   =   new   Student   (102,
                “Ramesh”); System.out.println (“s2 object
                contains: “ );s2.display ();
        }
}
```

OUTPUT

```
s1 object contains:
Student Roll Number is: 101
Student Name is: Suresh
s2 object contains:
Student Roll Number is: 102
Student Name is: Ramesh
```

“final” keyword

- ➔ final keyword before a class prevents inheritance.
  e.g.: final class A
  class B extends A //invalid
- ➔ final keyword before a method prevents overriding.
- ➔ final keyword before a variable makes that variable as a constant.
  e.g.: final double PI = 3.14159; //PI is a constant.

final with variables

- ➔ When a variable defined with the **final** keyword, it becomes a constant, and it does not allow usto modify the value.
- ➔ The variable defined with the final keyword allows only a one-time assignment, once a valueassigned to it, never allows us to change it again.

```
public class FinalVariableExample
 {
        public static void main(String[] args)
        {
                final int a = 10;
                System.out.println("a = " +
                a);
                a = 100;            // Can't be modified
        }
}
```

OUTPUT

```
Main.java:9:    error: cannot assign a value to final variable
                aa = 100;          // Can't be modified
```

Java -Methods

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

Now you will learn how to create your own methods with or without return values, invoke amethod with or without parameters, and apply method abstraction in the program design.

Creating Method

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

Syntax

```
public static int methodName(int a, int b) {
   // body
}
```

Here,

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

Method definition consists of a method header and a method body. The same is shown in thefollowing syntax −

Syntax

```
modifier returnType nameOfMethod (Parameter List) {
   // method body
}
```

The syntax shown above includes −

- **modifier** − It defines the access type of the method and it is optional to use.
- **returnType** − Method may return a value.

- **nameOfMethod** − This is the method name. The method signature consists of the method name and the parameter list.
- **Parameter List** − The list of parameters, it is the type, order, and number of parameters of a method. These are optional, method may contain zero parameters.
- **method body** − The method body defines what the method does with the statements.

Call by Value and Call by Reference in Java

There is only call by value in java, not call by reference. If we call a method passing a value, it is known as call by value. The changes being done in the called method, is not affected in the calling method.

Example of call by value in java

In case of call by value original value is not changed. Let's take a simple example:

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

```
Output:before change 50
       after change 50
```

In Java, parameters are always passed by value. For example, following program printsi = 10, j = 20.

```
// Test.java
class Test {
   // swap() doesn't swap i and j
   public static void swap(Integer i, Integer j) {
      Integer temp = new Integer(i);
      i = j;
      j = temp;
   }
   public static void main(String[] args) {
      Integer i = new Integer(10);
      Integer j = new Integer(20);
      swap(i, j);
      System.out.println("i = " + i + ", j = " + j);
```

```
    }
  }
```

Static Fields and Methods

The **static keyword** in java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than instance of the class.

The static can be:

1. variable (also known as class variable)
2. method (also known as class method)
3. block
4. nested class

Java static variable

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

- The static variable can be used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees,college name of students etc.
- The static variable gets memory only once in class area at the time of class loading.

Advantage of static variable

It makes your program **memory efficient** (i.e it saves memory).

Understanding problem without static variable

```
class Student{
    int rollno;
    String name;
    String college="ITS";
}
```

Example of static variable

```
//Program of static variable
class Student8{
  int rollno;
  String name;
  static String college ="ITS";
  Student8(int r,String n){
  rollno = r;
  name = n;
  }
 void display (){System.out.println(rollno+" "+name+" "+college);}
 public static void main(String args[]){
 Student8 s1 = new Student8(111,"Karan");
 Student8 s2 = new Student8(222,"Aryan");

 s1.display();
 s2.display();
 } }
```

```
Output:111 Karan ITS
       222 Aryan ITS
```

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

Example of static method

```
//Program of changing the common property of all objects(static field).

class Student9{
   int rollno;
   String name;
   static String college = "ITS";
   static void change(){
   college = "BBDIT";
   }
   Student9(int r, String n){
   rollno = r;
   name = n;
```

```
 }
 void display (){System.out.println(rollno+" "+name+" "+college);}
public static void main(String args[]){
Student9.change();
Student9  s1  =  new  Student9  (111,"Karan");
Student9  s2  =  new  Student9  (222,"Aryan");
Student9  s3  =  new  Student9  (333,"Sonoo");
s1.display();
s2.display();
s3.display();
} }
```

```
Output:111 Karan BBDIT
      222 Aryan BBDIT
       333 Sonoo BBDIT
```

Java static block

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

Example of static blockclass A2{

```
static{System.out.println("static block is invoked");}public
static void main(String args[]){ System.out.println("Hello
main");
} }
```

```
Output: static block is invoked
       Hello main
```

Access Control

Access Modifiers in java

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

The access modifiers in java specifies accessibility (scope) of a data member, method, constructoror class.

There are 4 types of java access modifiers:

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

private access modifier

The private access modifier is accessible only within class.

Simple example of private access modifier

In this example, we have created two classes A and Simple. A class contains private data member and private method. We are accessing these private members from outside the class, so there is compile time error.

```
class A{
private int data=40;
private void msg(){System.out.println("Hello java");} }
public class Simple{
 public static void main(String args[]){
   A obj=new A();
   System.out.println(obj.data);//Compile Time Error
   obj.msg();//Compile Time Error
   } }
```

2) default access modifier

If you don't use any modifier, it is treated as **default** bydefault. The default modifier is accessible only within package.

Example of default access modifier

In this example, we have created two packages pack and mypack. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

```
//save by A.java
package pack;
class A{
  void msg(){System.out.println("Hello");}
}

//save by B.java
package mypack;
import pack.*;
```

```
class B{
  public static void main(String args[]){
   A obj = new A();//Compile Time Error
   obj.msg();//Compile Time Error }  }
```

In the above example, the scope of class A and its method msg() is default so it cannot beaccessed from outside the package.

3) protected access modifier

The **protected access modifier** is accessible within package and outside the package but through inheritance only.

The protected access modifier can be applied on the data member, method and constructor. It can'tbe applied on the class.

Example of protected access modifier

In this example, we have created the two packages pack and mypack. The A class of pack package is public, so can be accessed from outside the package. But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

```
//save by A.java
package pack;
public class A{
protected void msg(){System.out.println("Hello");} }
//save by B.java
package mypack;
import pack.*;
class B extends A{
  public static void main(String args[]){
   B obj = new B();
   obj.msg();
  } }
```

```
Output:Hello
```

4) public access modifier

The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

Example of public access modifier

```
//save by A.java
package pack;
public class A{
public void msg(){System.out.println("Hello");} }
//save by B.java
package mypack;
import pack.*;
class B{
 public static void main(String args[]){
  A obj = new A();
  obj.msg();
 } }
```

```
Output:Hello
```

Understanding all java access modifiers

Let's understand the access modifiers by a simple table.

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

this keyword in java

Usage of java this keyword

Here is given the 6 usage of java this keyword.

1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.

4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this can be used to return the current class instance from the method.

```
class Student{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee){
this.rollno=rollno;
this.name=name;
this.fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}
class TestThis2{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

Output:

```
111 ankit 5000
112 sumit 6000
```

Difference between constructor and method in java

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

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

Method Overloading in java

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

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

Method Overloading: changing no. of arguments

In this example, we have created two methods, first add() method performs addition of twonumbers and second add method performs addition of three numbers.

In this example, we are creating static methods so that we don't need to create instance for callingmethods.

```
class Adder{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return 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));
}}
```

Output:

```
22
33
```

Method Overloading: changing data type of arguments

In this example, we have created two methods that differs in data type. The first add methodreceives two integer arguments and second add method receives two double arguments.

Recursion in Java

Recursion in java is a process in which a method calls itself continuously. A method in java that calls itself is called recursive method.

Java Recursion Example 1: Factorial Number

```
public class RecursionExample3 {
  static int factorial(int n){
   if (n == 1)
     return 1;
     else
    return(n * factorial(n-1));
    } }
public static void main(String[] args) {
System.out.println("Factorial of 5 is: "+factorial(5));
} }
```

Output:

```
Factorial of 5 is: 120
```

Java Garbage Collection

In java, garbage means unreferenced objects.

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

To do so, we were using free() function in C language and delete() in C++. But, in java it isperformed automatically. So, java provides better memory management.

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.

gc() method

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

```
public static void gc(){}
```

Simple Example of garbage collection in java

```
public class TestGarbage1{
 public void finalize(){System.out.println("object is garbage collected");}
 public static void main(String args[]){
  TestGarbage1 s1=new TestGarbage1();
  TestGarbage1 s2=new TestGarbage1();
  s1=null;
  s2=null;
  System.gc();
 } }
```

```
object is garbage collected
object is garbage collected
```

Java Nested and Inner Class

In this tutorial, you will learn about the nested class in Java and its types with the help of examples.
Java Inner Classes
In Java, it is also possible to nest classes (a class within a class). The purpose of nested classes is to group classes that belong together, which makes your code more readable and maintainable.

To access the inner class, create an object of the outer class, and then create an object of the inner class:

Example

```
class OuterClass {
  int x = 10;

  class InnerClass {
    int y = 5;
  }
}

public class Main {
  public static void main(String[] args) {
    OuterClass myOuter = new OuterClass();
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
    System.out.println(myInner.y + myOuter.x);
  }
}

// Outputs 15 (5 + 10)
```

Private Inner Class
Unlike a "regular" class, an inner class can be private or protected. If you don't want outside objects to access the inner class, declare the class as private:

Example

```
class OuterClass {
  int x = 10;

  private class InnerClass {
    int y = 5;
  }
}

public class Main {
  public static void main(String[] args) {
    OuterClass myOuter = new OuterClass();
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
    System.out.println(myInner.y + myOuter.x);
  }
}
```

If you try to access a private inner class from an outside class, an error occurs:

```
Main.java:13: error: OuterClass.InnerClass has private access in OuterClass
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
```

Static Inner Class

An inner class can also be static, which means that you can access it without creating an object of the outer class:

Example

```
class OuterClass {
  int x = 10;

  static class InnerClass {
    int y = 5;
  }
}

public class Main {
  public static void main(String[] args) {
    OuterClass.InnerClass myInner = new OuterClass.InnerClass();
    System.out.println(myInner.y);
  }
}

// Output
5
```

Note: just like static attributes and methods, a static inner class does not have access to members of the outer class.

Access Outer Class From Inner Class

One advantage of inner classes, is that they can access attributes and methods of the outer class:

Example

```
class OuterClass {
  int x = 10;

  class InnerClass {
    public int myInnerMethod() {
      return x;
    }
  }
}

public class Main {
  public static void main(String[] args) {
    OuterClass myOuter = new OuterClass();
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
    System.out.println(myInner.myInnerMethod());
  }
}

// Output
 10
```