Class in Java

In Java, a class is a blueprint or a template for creating objects. It serves as a fundamental concept in object-oriented programming and defines the structure and behavior of objects that can be instantiated from the class.

A class in Java is a user-defined data type that encapsulates data (fields) and behavior (methods). It acts as a template for creating objects, specifying the attributes and operations those objects can have. Classes are the building blocks of Java programs, enabling code organization, modularity, and the instantiation of objects with shared characteristics and functionality.

Syntax for creating a class –

```
class <class_name>{
    field;
    method;
    }
```

Objects in Java

An object is a specific instance of a class in Java. It is created based on the blueprint defined by the class and has its own unique set of data values for the class's attributes. Objects can be manipulated by invoking methods defined in the class, allowing them to perform actions and interact with other objects. Objects are

central to Java's OOP paradigm, enabling code reusability, modularity, and the modeling of real-world entities or concepts within a program.

An object has three characteristics:

- **State**: represents the data (value) of an object.
- **Behavior**: represents the behavior (functionality) of an object such as deposit, withdraw, etc.
- **Identity**: An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

Example –

```java
package p1;
class MobileDevices{
    1 usage
    void iPhone(){
        System.out.println("This is iphone 15 ....");
    }
    1 usage
    void iPad(){
        System.out.println("This is ipad air .....");
    }
    2 usages
    String iphone = "";
}
public class apple {
    public static void main(String[] args){
        MobileDevices m = new MobileDevices();
        m.iPhone();
        m.iPad();
        m.iphone ="Iphone 14";
        System.out.println("Name of the phone - "+ m.iphone);
    }
}
```

Output –

```
This is iphone 15 ....
This is ipad air .....
Name of the phone - Iphone 14
```

Constructor in Java

A constructor is a special type of method used to initialize and create objects of a class. Constructors have the same name as the class they belong to and are called automatically when an object of that class is created using the new keyword. Constructors are used to set the initial state of an object by initializing its instance variables or performing any necessary setup.

Example –

```java
package p1;
class Student{
    2 usages
    String name;
    2 usages
    int age;
    1 usage
    int height;
    1 usage
    public Student(String name,int age){
        this.name= name;
        this.age = age;
    }
    public Student(int height){
        this.height = height;
    }
}
public class constructorExample {
    public static void main(String[] args) {
        Student s= new Student( name: "Arjun", age: 23);
        System.out.println(s.name+" "+s.age);
    }
}
```

Control Statements in Java

n Java, control statements are used to control the flow of a program, making it possible to execute different parts of code under specific conditions or repeat code multiple times. Control statements can be broadly categorized into three main types:

Java provides three types of control flow statements.

1. Decision Making statements.
2. Loop statements.
3. Jump statements.

Decision Making statements:

Decision Making statements are statements decide which statement to execute and when. Decision-making statements evaluate the Boolean expression and control the program flow depending upon the result of the condition provided. There are two types of decision-making statements in Java, i.e., If statement and switch statement.

1. **If statements –**

   In Java, the "if" statement is used to evaluate a condition. The control of the program is diverted depending upon the specific condition. The condition of the If statement gives a Boolean value, either true or false. In Java, there are four types of if-statements given below.

   - **Simple if statement** - It is the most basic statement among all control flow statements in

Java. It evaluates a Boolean expression and enables the program to enter a block of code if the expression evaluates to be true.

- **if-else statement -** The if-else statement is an extension to the if-statement, which uses another block of code, i.e., else block. The else block is executed if the condition of the if-block is evaluated as false**.**
- **if-else-if ladder** - The if-else-if statement contains the if-statement followed by multiple else-if statements. In other words, we can say that it is the chain of if-else statements that create a decision tree where the program may enter in the block of code where the condition is true. We can also define an else statement at the end of the chain.
- **Nested if-statement** - In nested if-statements, the if statement can contain a if or if-else statement inside another if or else-if statement.

2. **Switch Statement –** In Java, Switch statements are similar to if-else-if statements. The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable which is being switched. The switch statement is easier to use instead of if-else-if statements. It also enhances the readability of the program.

Example –

```java
package p1;
import java.util.Scanner;
public class DayOfWeek {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a number (1-7) to find the day of the week: ");
        int dayNumber = scanner.nextInt();
        if (dayNumber >= 1 && dayNumber <= 7) {
            String day;
            if (dayNumber == 1) {
                day = "Monday";
            } else if (dayNumber == 2) {
                day = "Tuesday";
            } else if (dayNumber == 3) {
                day = "Wednesday";
            } else if (dayNumber == 4) {
                day = "Thursday";
            } else if (dayNumber == 5) {
                day = "Friday";
            } else if (dayNumber == 6) {
                day = "Saturday";
            } else {
                day = "Sunday";
            }
            System.out.println("Using if-else: The day of the week is " + day);
        }
        else {
            System.out.println("Invalid input. Please enter a number between 1 and 7.");
        }
```

```java
        switch (dayNumber) {
            case 1:
                System.out.println(" The day of the week is Monday");
                break;
            case 2:
                System.out.println("The day of the week is Tuesday");
                break;
            case 3:
                System.out.println("The day of the week is Wednesday");
                break;
            case 4:
                System.out.println("The day of the week is Thursday");
                break;
            case 5:
                System.out.println("The day of the week is Friday");
                break;
            case 6:
                System.out.println("The day of the week is Saturday");
                break;
            case 7:
                System.out.println("The day of the week is Sunday");
                break;
            default:
                System.out.println("Please enter a number between 1 and 7.");
        }
        scanner.close();
    }
}
```

Loop Statements

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

In Java, we have three types of loops that execute similarly. However, there are differences in their syntax and condition checking time.

- for loop – It enables us to initialize the loop variable, check the condition, and increment/decrement in a single line of code. We use the for loop only when we exactly know the number of times, we want to execute the block of code.
- while loop – The while loop is also used to iterate over the number of statements multiple times. However, if we don't know the number of iterations in advance, it is recommended to use a while loop. Unlike for loop, the initialization and increment/decrement doesn't take place inside the loop statement in while loop.
- do-while loop – The do-while loop checks the condition at the end of the loop after executing the loop statements. When the number of iteration is not known and we have to execute the loop at least once, we can use do-while loop.

Example –

```java
package p1;
public class loopExample {
    public static void main(String[] args){

        int arr[] ={2,1,4,5,7};
        int t =7;
        int i=0;
        for( i=0;i< arr.length;i++)
        {
            if(arr[i] == t ){
                System.out.println("Element found at index - " + i);
            }
        }
        int j=-4;

        do{
            System.out.println("Element found at index - "+ j);
            if(arr[j] == t){
                System.out.println("Element found at index - "+ j);
            }
            j--;
        }while(j>0);

    }
}
```

Jump Statements –

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

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

However, it breaks only the inner loop in the case of the nested loop.

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