**Experiment: 3**

PART A

(PART A: TO BE REFERRED BY STUDENTS)

**Aim : To study Class, object, methods and constructor.**

**Learning Outcomes: Learner would be able to**

1. Understand the classes, Objects and methods in JAVA
2. Understand the Constructors in JAVA

**Theory:**

**What is Class ?**

Java is an *object oriented programming* language. Classes and objects are the key concepts to understand *object oriented programming*. Java programming is based on these two terms.

**Syntax :**

class ClassName

{

// variables, constructors,

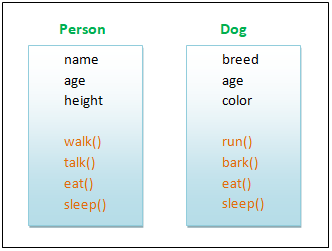
// method declarations

}

Here class is a keyword, used to define a class in java. *ClassName* is the name of class, given by the programmer. After the class name, it's the class body given inside { }. Variables, methods, constructors etc defined inside the balanced { } after the class name are the part of that class.

**Real time example of class:**

Some of the real world examples of class are Person, Vehicle, House, Tree, Dog etc. For an example a Person class have properties like name, age, height etc. and behaviours like walk, talk, eat, sleep etc. Now if you create an object of this class, it will have these properties and behaviours inside it. Once you created a class, you can create multiple objects of that class.



**Fig 3.1 : Real time example of class**

**What is Object ?**

**Object** is a real-world entity or an instance of a class that encapsulates both data (fields or attributes) and behaviour (methods or functions). It is the fundamental building block of object-oriented programming (OOP).

**Syntax :**

ClassName objName = new ClassName();

Here *ClassName* is the name of class whose object need to be created and *objName* is the name of object, given by the programmer. The object name should be unique and should follow the convention given in [identifier naming convention](https://www.refreshjava.com/java/java-identifier-naming-convention). The keyword new is used to create object in java. Using the () with *ClassName* calls the constructor of that class to create and initialize the object in memory.

**Methods in java:**

A method is a block of instructions (one or more line of codes) given in { } inside a class. This block is referred by a name which is basically the method name. The name of method is used to call(invoke) that method within or outside a class. Once a method is called, the code given inside that method is executed. After execution a method may or may not return a value. The most basic or minimal **Syntax** of declaring a method is:

// Method without parameter

Return\_Type methodName()

{

// one or more line of code.

return some\_value;

}

// Method with parameter

Return\_Type methodName(DataType param1, DataType param2 ...)

{

// one or more line of code.

return some\_value;

}

A method must have a return type which basically tells what type of value this method returns. The **Return\_Type** of a method can be [primitive or non-primitive data type](https://www.refreshjava.com/java/primitive-and-non-primitive-data-type). If a method doesn't return a value, it's return type must be void or in other way if the return type of a method is void, it means that method doesn't return any value.

The name of the method is given by the programmer. As per the convention the method name should be in small letters while in multiWord method name, the first letter of any subsequent word should be in capital letter. For example method names like add(), calculateArea(), getName() etc. are some examples of this convention.

After method name, it's the parameters given inside () which are basically variables. Parameters are optional which means a method may or may not have parameters. You can access these parameters within the method only, not outside the method. The data type of parameters can be primitive or non-primitive. After parameters it's method body given inside { }. Everything given inside { } after method name are the part of that method.

int add(int num1, int num2)

{

// one or more line of code.

return value; // returns integer value.

}

void calculateArea()

{

// one or more line of code.

}

**Static Variables :**

* A static variable is shared among all instances of the class.
* It is initialized only once, at the start of the program.
* It can be accessed directly through the class without creating an object.

**Example :**

class MyClass

{

static int count = 0; // Static variable

MyClass()

{

count++;

}

}

public class Main

{

public static void main(String[] args)

{

MyClass obj1 = new MyClass();

MyClass obj2 = new MyClass();

System.out.println(MyClass.count); // Output will be 2

}

}

**Static Classes :**

* An inner class can be marked as static, making it a static nested class.
* A static inner class can access only static members of the outer class.

**Example :**

class OuterClass {

static int outerStaticVar = 10;

static class InnerClass {

void display() {

System.out.println("Outer static variable: " + outerStaticVar);

}

}

}

public class Main {

public static void main(String[] args) {

OuterClass.InnerClass obj = new OuterClass.InnerClass();

obj.display();

}

}

**Static and Non Static Method:**

A method declared with static keyword is known as static method. Static method belongs to class not object which means you don't need to create object to access such methods. You can access static methods with class name itself. Methods defined without static keyword is known as non static method, also known as instance method. You need object or instance of the class to access such methods. For example in below program calculateArea() is a static method while add()is non static methods.

Static method can access only static variables inside it's body while non static method can access both static and non static variables.

class Main

{

public static void main(String [] args)

{

Main md = new Main();

int sum = md.add(20,30); // calling add() method

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

Main.calculateArea(100,50); // calling calculateArea method using className

}

int add(int num1, int num2)

{

int sum = num1 + num2;

return sum;

}

static void calculateArea(int length, int width)

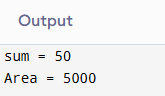
{

int area = length\*width;

System.out.println("Area = "+area);

}

}



**Constructors in Java:**

Constructor in java is used in creation and initialization of object of a class inside memory. It's the constructor that provides the initial values of instance variables for an object inside memory. Constructors declarations are similar as method declarations except that constructors don't have any return type and their name is same as class name. The **syntax** of declaring a constructor is :

class MyClassName {

// No-argument constructor

Access\_Modifier MyClassName() {

// Initialization code

}

// Parameterized constructor

Access\_Modifier MyClassName(DataType param1, DataType param2 ...) {

// Initialization code

}

}

**Example** :

class Person {

Person() {

// Initialization code

}

public Person(int param1, String param2) {

// Initialization code

}

}

Access modifier of a constructor can be public, private, protected or default(no modifier). These modifiers with constructors decides the accessibility(visibility) of the constructor within or outside the class. After access modifier it's the name of constructor which must be same as the class name.

Parameters in constructors are **optional**, a constructor may or may not have parameters. A constructor without parameter is also known as **no-argument**constructor while a constructor with parameter is also known as **parameterized** constructor. The data type of parameters can be [primitive or non-primitive](https://www.refreshjava.com/java/primitive-and-non-primitive-data-type).

In constructor body generally we initialize instance variables with some values but it's not limited to that only. You can use other statements as well as you do in methods.

You can define multiple constructors with different argument lists. Java differentiates constructors on the basis of the number of arguments in the list and their types. This is also known as **constructor overloading**. You cannot write two constructors that have the same number and type of arguments for the same class, because java compiler would not be able to differentiate them. In this case compiler will throw error.

**Types of Constructors in Java:**

Constructor can be divided in two types.

1. Default constructor / No-argument constructor
2. Parameterized constructor
3. **Default constructor / No-argument constructor:**

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

If we don’t define a constructor in a class, then the compiler creates a default constructor(with no arguments) for the class. And if we write a constructor with arguments then the compiler does not create a default constructor.

Default constructor provides the default values to the object like 0, null, etc. depending on the type.

**Syntax of default constructor:**

class\_name()

{

}

class Main

{

// constructor

Main()

{

System.out.println("Default Constructor Called");

}

public static void main(String[] args)

{

// constructor is invoked while

// creating an object of the Main class

Main obj = new Main();

}

}

**Output :**

**Default Constructor Called**

1. **Parameterized Constructor :**

A Constructor which has **parameters** in it called as **Parameterized Constructors**, the Parameterized constructor is used to assign different values for the different objects.

**Syntax:**

Class\_name object\_name=new class\_name(parameter list);

**Eg. :**

Student4 s1 = new Student4(111,"Karan");

**Example :**

public class paramC

{

paramC(int a, int b)

{

System.out.print("Parameterized Constructor");

System.out.println(" having Two parameters");

}

paramC(int a, int b, int c)

{

System.out.print("Parameterized Constructor");

System.out.println(" having Three parameters");

}

public static void main(String args[])

{

paramC pc1 = new paramC(12, 12);

paramC pc2 = new paramC(1, 2, 13);

}

}