# PRACTICAL 3

**Aim**: To understand the concepts of creating/calling Methods and usage of Classes and Objects inJava programming Language

**Prerequisite:**

* Knowledge of basic constructs in Java programming language.

**Outcome:** After successful completion of this experiment students will be able to,

* Define Classes and create Objects using different constructors
* To access an object’s data and methods using the object member access operator
* Pass parameters by value and reference to methods
* Implement overloaded functions

**Theory:**

**Classes:**

Classes and Objects are basic concepts of Object Oriented Programming which revolve around the real life entities. A class is a user defined blueprint or prototype from which objects are created.  It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order:

1. **Modifiers**: A class can be public or has default access
2. **Class name:** The name should begin with an initial letter (capitalized by convention).
3. **Superclass (if any):** The name of the class’s parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
4. **Interfaces (if any):** A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
5. **Body:** The class body surrounded by braces, { }.

Constructors are used for initializing new objects. Fields are variables that provides the state of the class and its objects, and methods are used to implement the behavior of the class and its objects.

**Objects:**

It is a basic unit of Object Oriented Programming and represents the real life entities.  A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of:

1. State: It is represented by attributes of an object. It also reflects the properties of an object.
2. Behaviour: It is represented by methods of an object. It also reflects the response of an object with other objects.
3. Identity: It gives a unique name to an object and enables one object to interact with other objects.

Objects correspond to things found in the real world. For example, a graphics program may have objects such as “circle”, “square”, “menu”. An online shopping system might have objects such as “shopping cart”, “customer”, and “product”.

When an object of a class is created, the class is said to be **instantiated**.

All the instances share the attributes and the behavior of the class. But the values of those attributes, i.e. the state are unique for each object. A single class may have any number of instances.

**Defining and Invoking methods:**

A **method** is a block of code, which only runs when it is called. You can pass data, known as parameters, into a method. Methods are used to perform certain actions, and they are known as **functions.**

A method must be declared within a class. It is defined with the name of the method, followed by parentheses **()**. Java provides some pre-defined methods, such as System.out.println(), but you can also create your own methods to perform certain actions:

public class MyClass {

static void myMethod () {

// code to be executed

}

}

* myMethod() is the name of the method
* static means that the method belongs to the MyClass class and not an object of the MyClass class.
* void means that this method does not have a return value.

To call a method in Java, write the method's name followed by two parentheses **()** and a semicolon**.** In the following example, myMethod () is used to print a text (the action), when it is called:

public class MyClass {

static void myMethod () {

System.out.println("I just got executed!");

}

public static void main(String[] args) {

myMethod ();

}

}

**Parameter Passing:**

Information can be passed to methods as parameter. Parameters act as variables inside themethod.Parameters are specified after the method name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.

public class MyClass {

static void myMethod (String fname) {

System.out.println(fname + " Refsnes");

}

public static void main(String[] args) {

myMethod("Liam");

myMethod("Jenny");

myMethod("Anja");

}

}

When a parameter is passed to the method, it is called an argument. So, from the example above: fname is a parameter, while Liam, Jenny and Anja are arguments.

The void keyword, indicates that the method should not return a value. If you want the method to return a value, you can use a primitive data type (such as int, char, etc.) instead of void, and use the return keyword inside the method:

**Method Overloading:**

With**method overloading**, multiple methods can have the same name with different parameters.

Example:

int myMethod (int x)

float myMethod (float x)

double myMethod (double x, double y)

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

**Examples:**

1. **Differ by a number of parameters in two methods.**

public class Sum {

       public int sum (int x, int y)

     {

         return (x + y);

     }

 public int sum (int x, int y, int z)

   {

         return (x + y + z);

     }

1. **Differ by datatypes of parameters in two methods.**

public class Sum {

       public int sum (int x, int y)

     {

         return (x + y);

     }

 public int sum(double x, double y)

   {

         return (x + y);

     }

1. **Differ by order of parameters in two methods.**

public class Sum {

       public int sum (int x, double y)

     {

         return (x + y);

     }

 public int sum (double x, int y)

   {

         return (x + y);

     }

(TO BE COMPLETED BY STUDENTS)

|  |  |
| --- | --- |
| Roll No.: K005 | Name: Jal Bafana |
| Class: Btech. Cyber Security (Sem-4) | Batch: K1 |
| Date of Practical: 27.01.2025 | Date of Submission: 22.02.2025 |
| Grade: |  |

1. Create a class with appropriate constructors (minimum 3) and a member function to print square of numbers from 1 to 15.

class Squareprinter

{

double num = 1.0;

void Square()

{

}

void Square(double newnum)

{

num = newnum;

}

void display()

{

System.out.println("Square of "+num+" is "+(num\*num));

}

public static void main(String args[])

{

Squareprinter obj = new Squareprinter();

for (int i = 0;i< 16;i++)

{

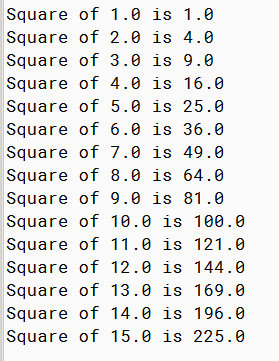
obj.Square(i);

obj.display();

}

}

}



1. Create a class with appropriate constructors and overloaded functions to print the area of a square and a rectangle. The two methods have the same name but different number of parameters. The method for printing area of rectangle has two parameters which are length and breadth respectively while the other method for printing area of square has one parameter which is side of square.

public class Squareandrectangle

{

double l=1.0;

double b=1.0;

void decArea(double nl, double nb)

{

l = nl;

b = nb;

}

public void area(double l, double b)

{

System.out.println("area of rectangle = " + (l\*b));

}

public void area(double l)

{

System.out.println("area of rectangle = " + (l\*l));

}

public static void main(String[]args)

{

Squareandrectangle obj = new Squareandrectangle();

obj.area(4.0,5.0);

obj.area(4.0);

}

}



1. Create a class with appropriate constructors and overloaded functions to multiply two numbers such that overloading is achieved

* By changing number of parameters in two methods
* By having different data types of the parameters in methods
* By changing order of the parameters in methods

public class Lab3Pgm3

{

int num1 = 1;

int num2 = 1;

void Multi()

{

}

public void multi(int num1, int num2)

{

System.out.println("Multiplication of two numbers = "+(num1\*num2));

}

public void multi(int num1, int num2, int num3)

{

System.out.println("Multiplication of three numbers = "+(num1\*num2\*num3));

}

public void multi(int num1, double num2)

{

System.out.println("Multiplication of two numbers = "+(num1\*num2));

}

public void multi(double num2, int num1)

{

System.out.println("Multiplication of two numbers = "+(num2\*num1));

}

public static void main(String args[])

{

Lab3Pgm3 obj = new Lab3Pgm3();

obj.multi(12, 4);

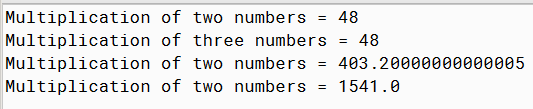
obj.multi(2, 4, 6);

obj.multi(12, 33.6);

obj.multi(23.0,67);

}

}



1. Create a class with appropriate constructors and a member function to print an emirp (prime spelled backward) number. It is a non-palindromic prime number whose reversal is also a prime. For example, 17 is a prime and 71 is a prime. So, 17 and 71 are emirps. Write a program that displays the first 100 emirps (13, 17, 31, 37, 71, 73, 79, and 97).

class Lab3Pgm4

{

Lab3Pgm4()

{

}

boolean isPrime(int num)

{

if(num<2)

return false;

for(int i = 2; i<= Math.sqrt(num); i++)

{

if(num % i == 0) return false;

}

return true;

}

int reverseNumber(int num)

{

int reverse = 0;

while(num != 0)

{

reverse = reverse \* 10 + (num % 10);

num = num / 10;

}

return reverse;

}

void printEmirp(int count)

{

int num = 10;

int found = 0;

while(found < count)

{

int reversed = reverseNumber(num);

if(isPrime(num) &&isPrime(reversed) &&num != reversed)

{

System.out.println(num + " ");

found ++;

if (found % 10 == 0) System.out.println();

}

num++;

}

}

public static void main(String args[])

{

Lab3Pgm4 obj = new Lab3Pgm4();

obj.printEmirp(100);

}

}

