UNIT-II:

Classes and Objects: Classes and objects, Class declaration, Creating objects, Methods, Constructors and Constructor Overloading, Importance of Static Keyword and Examples, this Keyword, Arrays, Command Line Arguments, Nested Classes.

Classes and Objects:

Java achieves oops principles with the help of classes and objects.

Class: A class is a blueprint from which the individual objects are created. It represents the state and behaviour of an entity (anything that has existence with some properties and behaviour like car, student, laptop, bag, university, player, etc). A class is a logical entity

Object: An Object is the instance for a class (instance – initialization). An Object is a physical entity because it has memory allocated for all variables in the class. we can create any no of objects for the single class.

example:

let say I want to store 60 student's information like roll_num, name, age, marks. before knowing their property values, we will define a prototype suitable for them called as 'class'.

A class is declared using the keyword 'class'. all the properties and functions related to an entity are defined with in that class.

create class Student -

```
class Student
{
        String roll_num;
        String name;
        int age;
        int marks;
        void writeExams()
        {
        ::::::
        }
        void attendClasses()
        {
        ::::::
        }
}       // end of class
```

Now whenever we want to store a student information, we request memory for storing the property values, the allocated memory to store student information is called as 'object/instance'.

General Form of a Class:

A class generally contain three sections - variables, constructors and methods.

Variables represent its state/properties in form of fields. Class can have static and instance variables. functionality/behaviour will be implemented in methods under a class. Class can have static and instance methods. Constructors will initialize the instance variables of a class when an abject is created.

General form or Syntax of a Class

```
class NameOfClass
{
    // instance variable declaration
    type1 varName1 = value1;
    type2 varName2 = value2;
    :
    :
    typeN varNameN = valueN;

    //  Constructors
    // no argument constructor
    NameOfClass()
    {
        // body of constructor
    }
    :
    :
    // constructor with arguments, we can define any more than one constructor to same same
    NameOfClass (cparamN)
    {
        // body of constructor
    }
```

```
    // Methods
    static returnType1 methodName1(mParams1)
    {
        // body of static method
    }
    :
    :
    returnTypeN methodNameN(mParamsN)
    {
        // body of instance/non-static method
    }
}
```

class is keyword used to define the new class

Variables named varName1 through varNameN are declared inside the class and outside of all methods are called instance variables. Each variable must be assigned a type shown as type1 through typeN(any primitive data type like int,float,char,..) and may be initialized to a value shown as valueN. if a variable is declared by keyword static then they belong to all objects called as static or class variables (we will cover about static later, in this chapter)

Constructors are used to initialize the instance variables of an object, executed when an object is created. It always has the same name as the class. They will not return any value. A constructor can be defined with or without arguments. We can define more than one constructor for a class.

Methods named mthName1 through mthNameN should be defined with in the class. A static method can be called directly from main/any static method. a non-static method should be called through an object only.

when we want to run a class, a main method should be defined in it with following signature. Java program execution starts from main method. A main method is the starting point for execution by JVM.

```
    public static void main(String arguments[])
    {
    }
```

Declaring Objects using new

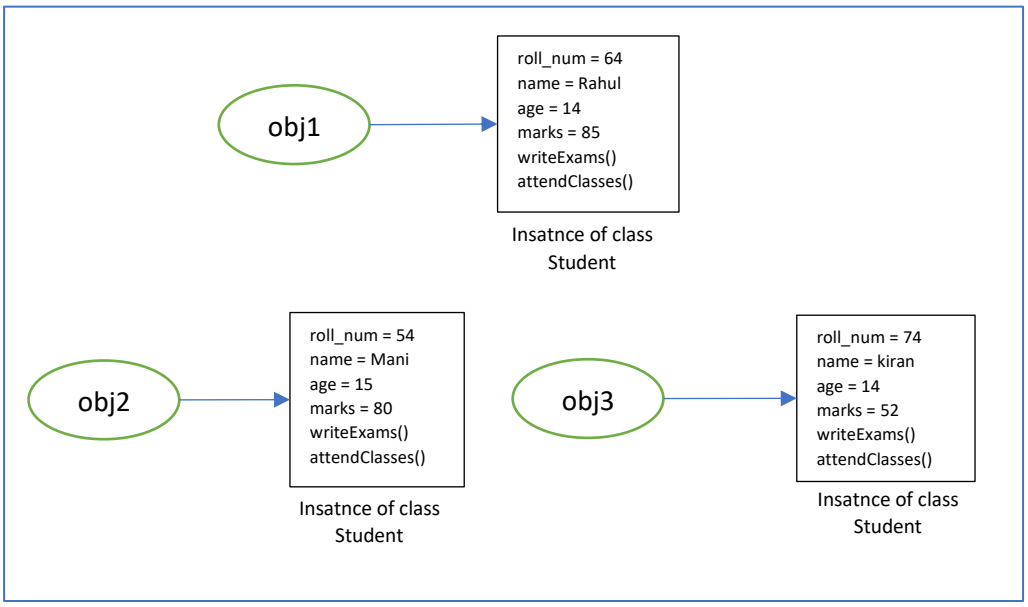
The *new* operator instantiates a class by dynamically allocating (i.e, allocation at run time) memory for a new object and returning a reference to that memory. This reference is then stored in the variable declared with type class name.

creating objects for class Student –

an object for class can be created using the key word **'new'**

```
Student obj1 = new Student ();
Student obj2 = new Student ();
Student obj3 = new Student ();
```

obj1.roll_num = 64;	obj1.roll_num = "Rahul";	obj1.age = 14;	obj1.age = 85;
obj2.roll_num = 54;	obj2.roll_num = "Mani";	obj2.age = 15;	obj2.age = 80;
obj3.roll_num = 74;	obj3.roll_num = "Kiran";	obj3.age = 14;	obj3.age = 52;



As shown in above diagram whenever a new object is created using new, an instance for class will be created, and can be accessed through class reference variables like obj1, obj2, obj3.

Write a program in single java file, which contains two classes Teacher and Student,

Student should have properties rollnum, name and marks. and functions increment, decrement which will increment and decrement marks by 10.

In class Teacher write a main method, in that create two student class objects initialize their properties and later call increment() on 1st student object and call decrement() on second Student object.

```
class Student
{
        int roll_num;
        String name;
        int marks;
```

```
        void increment()
        {
                marks = marks + 10;
        }
        void decrement()
        {
                marks = marks - 10;
        }
}

class Teacher
{
   public static void main(String args[])
   {
        Student s1 = new Student();
        Student s2 = new Student();

        s1.roll_num = 54;
        s1.name = "Rahul";
        s1.marks = 55;

        s2.roll_num = 64;
        s2.name = "Kiran";
        s2.marks = 80;

        System.out.println("intial marks of student "+s1.name+" are "+s1.marks);
        s1.increment();
        System.out.println("after increment marks of student "+s1.name+" are
"+s1.marks+"\n");
        System.out.println("intial marks of student "+s2.name+" are "+s2.marks);
        s2.decrement();
        System.out.println("after decrement marks of student "+s2.name+" are "+s2.marks);
    }

}
```

OutPut –
intial marks of student Rahul are 55
after increment marks of student Rahul are 65
intial marks of student Kiran are 80
after decrement marks of student Kiran are 70

Methods –

A method is a set of statements written with in the block to perform certain operation. Methods allow us to **reuse** the code, we can call a method any no of times to repeat the

execution of statements under it. A method must be declared within the class. Following is the syntax to define a method.

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

The components in a method include −

- **modifier** − It defines the access type of the method and it is optional to use.
- **returnType** − Method may return value of any type. if returns nothing type is void.
- **nameOfMethod** − This is name of the method.
- **Parameter** List − The list of parameters, also called as arguments. A method can have any no of arguments. These are optional
- **method body** − The method body defines statements for implementing the functionality.

Let's practice some programs on methods for better understanding.

Write a program with class name MethodExample, this class should contain a method called display, which prints "This is an example for method " and it returns nothing.

```
class MethodExample
{
        void display()
        {
                System.out.println("This is an example for method");
        }
        public static void main(String args[])
        {
                display (); // a non-static method cannot be called without an object
                MethodExample obj = new MethodExample();
                obj.display();
        }
}
```

OutPut:
This is an example for method

Write a program with class name MethodExample, this class should contain a method called display, it takes a String parameter of user name and prints "welcome to methods username" and it returns nothing.

```
class MethodExample
{
        void display(String userName)
        {
                System.out.println("Welcome to methods " + userName);
        }

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

```
	{
		MethodExample obj = new MethodExample ();
		obj.display("Rahul");		// string literal is passed directly

		String name = "kiran";
		obj.display(name);	// String literal is stored in a variable and then passed
	}
}
```

OutPut:
Welcome to methods Rahul
Welcome to methods kiran

Write a program with class name MyMethod, this class should contain a method called sum, it takes two integer parameters and prints their sum and it returns nothing.

```
class MyMethod
{
	void sum(int num1, int num2)
	{
		int sum;
		sum = num1+num2;
		System.out.println("sum of variables is " + sum);
	}

	public static void main(String args[])
	{
	MyMethod obj = new MyMethod ();
	obj.sum(2,4);		// integer literals are passed directly

	int i=12,j=4;
	obj.sum(i,j);		// integer literals are stored in variables and then passed
	}
}
```

OutPut:
sum of variables is 6
sum of variables is 16

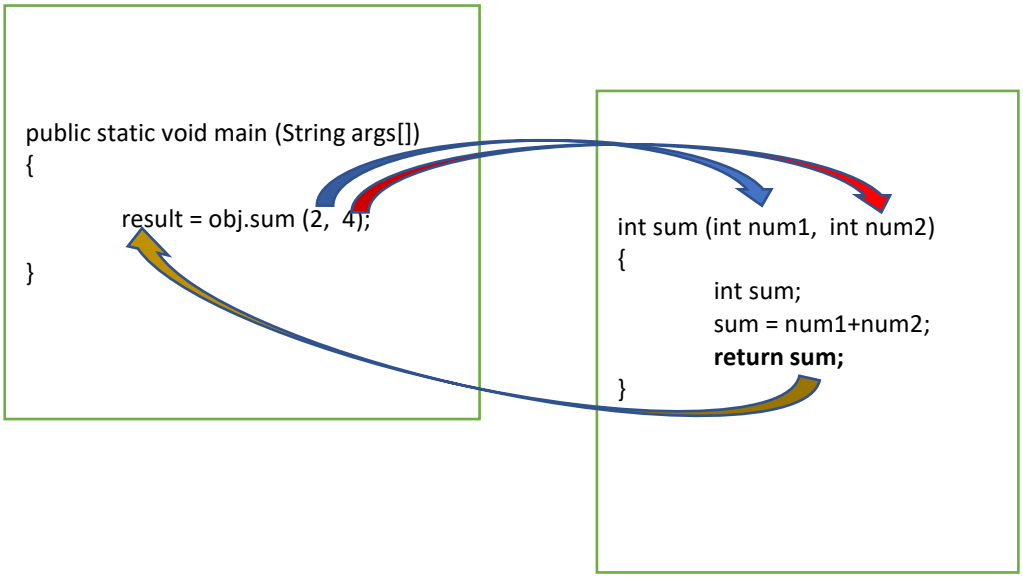
Write a program with class name MyMethod, this class should contain a method called sum, it takes two integer parameters and returns their sum, you should print sum value in main method after completing the function call.

```
class MyMethod
{
	int sum(int num1, int num2)
	{
		int sum;
		sum = num1+num2;
		return sum;
	}
	public static void main(String args[])
	{
	MyMethod obj = new MyMethod ();
	int result;
	result = obj.sum(2,4);
	System.out.println("received sum value from function "+result);
```

```
        int i=12,j=4;
        result = obj.sum(i,j);
        System.out.println("received sum value from function "+result);
        }
}
```

OutPut:
received sum value from function 6
received sum value from function 16

Note:
***** We can return a value or variable with keyword return.
* It is must to hold the returned value at function call. In our example the sum is returned and received value is captured under the variable result.



Constructors in Java

Constructors are used to initialize the instance variables of class, at the time of creating object.

rules to be followed for defining a constructor

1. Constructor name must be the same as its class name
2. Constructor **should not** be associated with any return type, not even void. A constructor will never return anything.

There are two types of constructors in Java:

1. Default constructor (constructor without any arguments)
2. Parameterized constructor (constructor with arguments)

Default Constructor (constructor without any arguments)

A constructor that has no parameter is known as default constructor. If we don't define a constructor in a class, then compiler creates **default constructor** for that class.

Write a program with class name MyDefaultConstructor, which have a default constructor that prints "My program using default constructor".

```
class MyDefaultConstructor
{
	MyDefaultConstructor ()
	{
		System.out.println(“My program using default constructor”);
	}

	public static void main(String args[ ])
	{
		MyDefaultConstructor obj = new MyDefaultConstructor();
	}

}
```

OutPut :

My program using default constructor.

Parameterized constructor (constructor with arguments)

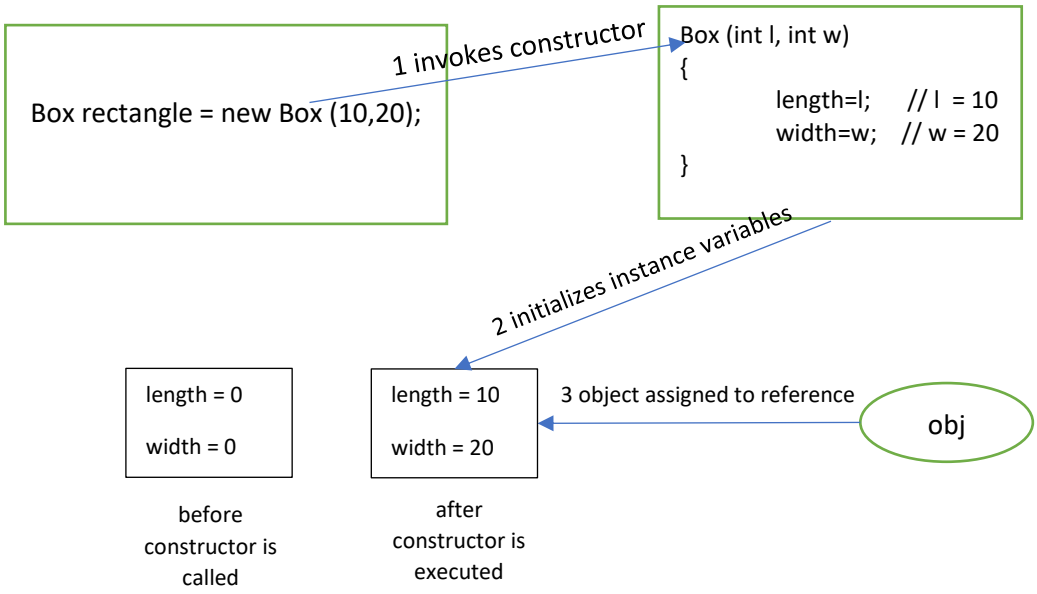
A constructor that has parameters is known as parameterized constructor. If we want to initialize variables of the class with your own values, then use a parameterized constructor.

Write a program with class name Box, define a parameterized constructor to initialize instance variables. write a method areRectangle to calculate area of rectangle.

```
class Box
{
	int length;
	int width;

	//Constructor for rectangle when two parameters are passed.
	Box (int l, int w)
	{
		length=l;
		width=w;
	}

	int areaRectangle()
	{
		return length*width;
	}

	public static void main(String args[])
	{
		Box obj = new Box (10,20);
		int area;
		area = obj.areaRectangle();
		System.out.println("Area of rectangle is " + area);
```

```
    }
}
```

OutPut:
Area of rectangle is 200



constructor overloading

Constructor overloading means defining two or more constructors for same class. The name of all constructors will be same, only number of arguments or type of arguments will be different.

when you pass arguments, the constructor which matches number of arguments and type of arguments will be executed automatically.

Write a program to find area of square and rectangle, with help of constructor overloading.

```
class Box
{
        double length;
        double width;

        //Constructor for rectangle when two parameters are passed.
        Box (double l,double w)         {
                length=l;
                width=w;
        }
        //Constructor for square when one value is initialized(area).
        Box (double l)  {
                length=l;
        }
        double AreaRectangle() {
                return length*width;
        }
```

```
        double AreaSquare()     {
                return length*length*length;
        }
}
        //Main class from where program execution begins
class Square_Rectangle_Const_Overloading      {
        public static void main (String args[])      {
        Box rectangle = new Box (10,20);
        Box square = new Box (10);
        System.out.println("Area of rectangle is " + rectangle.AreaRectangle());
        System.out.println("Area of square is " +square.AreaSquare());
        }
}
```

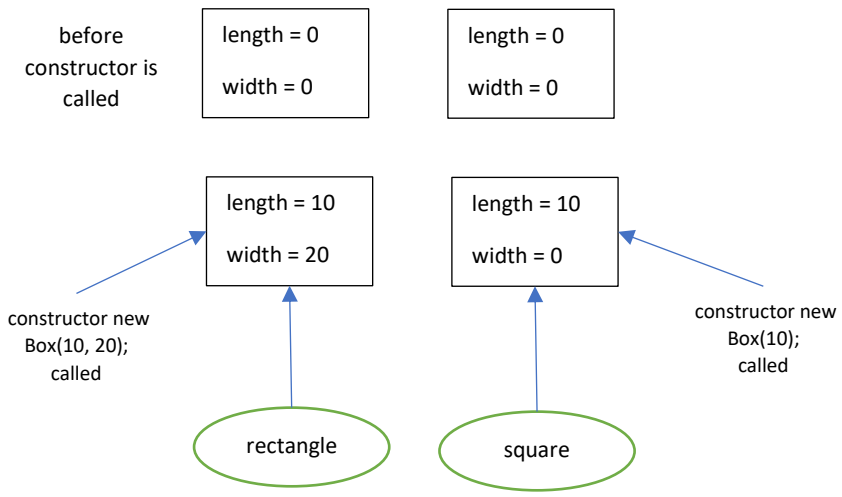
OutPut:

Area of rectangle is 200.0

Area of square is 1000.0

```
class Square_Rectangle_Const_Overloading
{
 public static void main (String args[])
 {
  Box rectangle = new Box (10,20);
  Box square = new Box (10);
  System.out.println("Area of rectangle is "
  + rectangle.AreaRectangle());
  System.out.println("Area of square is "
  +square.AreaSquare());
  }
}
```

```
class Box
{
  double length;
  double width;
//Constructor for rectangle of two parameters is called.
  Box (double l,double w)  {
            length=l;
            width=w;
    }
//Constructor for square of one value is initialized(area).
  Box (double l)  {
            length=l;
  }
  double AreaRectangle()  {
            return length*width;
  }

  double AreaSquare()  {
            return length*length*length;
  }
}
```



Method Overloading:

Method Overloading (or) static polymorphism allows a class to have more than one method to have the same name, if their argument lists are different.

A method is said to be overloaded, if two methods with same name differ by any of these

1. Number of parameters
   example:

```
add(int a, int b)          // function have two arguments
{
    :::::::
}

add(int, int, int)         // function have three arguments
{
    :::::::::
}
```

2. Data type of parameters
   example:

```
add(int a, int b)          // function have two int arguments
{
    :::::::
}

add(int a, float b)        // function have one int argument and one float argument
{
    :::::::::
}
```

3. Order of parameters
   example:

```
add(int a, float b, int c)          // function have two arguments
{
    :::::::
}

add(int a, int c, float b)          // function have three arguments
{
    :::::::::
}
```

Write a program to print sum of given numbers, implement it through method overloading.

```
class MethodOverloading
{

    void sum (int a, int b)
    {
```

```
		int sum;
		sum = a+b;
		System.out.println("method sum declared with two integer arguments");
		System.out.println("sum is "+sum);
	}



	void sum (int a, int b, int c)
	{
		int sum;
		sum = a+b+c;
		System.out.println("this method differs by number of arguments");
		System.out.println("sum is "+sum);
	}

	void sum (int a, float b)
	{
		float sum;
		sum = a+b;
		System.out.println("this method differs by type of arguments");
		System.out.println("sum is "+sum);
	}

	void sum (float b, int a)
	{
		float sum;
		sum = a+b;
		System.out.println("this method differs by order of arguments");
		System.out.println("sum is "+sum);
	}
	public static void main (String args[])
	{
		MethodOverloading obj = new MethodOverloading();
		obj.sum(2, 4);
		obj.sum(2, 4, 6);
		obj.sum(2, 3.4f);
		obj.sum(3.4f, 2);
	}

}
```

OutPut:

```
method sum declared with two integer arguments
sum is 6
this method differs by number of arguments
sum is 12
this method differs by type of arguments
sum is 5.4
this method differs by order of arguments
sum is 5.4
```

Note:

A method will not be overloaded by return type of that method, for example if two methods have same name, same parameters and have different return type, then this is not a valid method overloading. It will be a compilation error “method has been already defined”.

Static keyword:

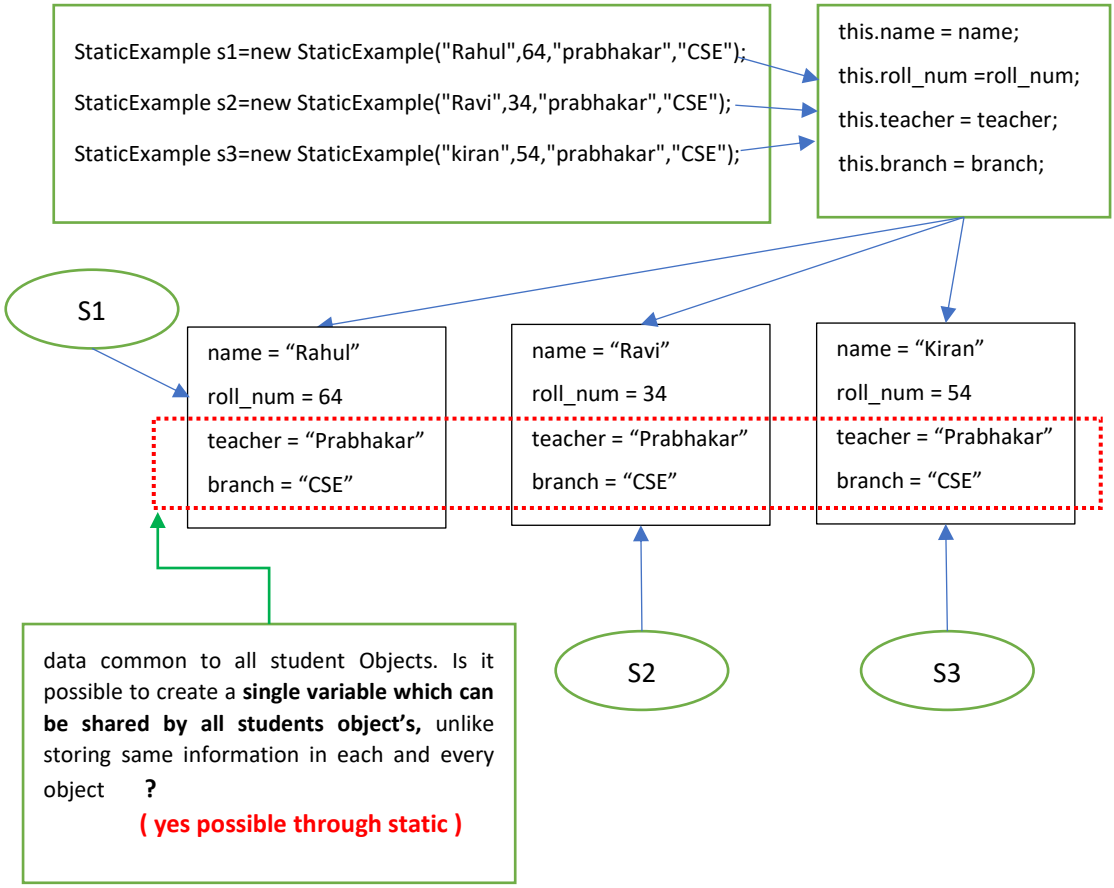
When a member is declared static, it can be accessed before any objects of its class are created. We can apply static keyword with variables, methods, blocks and nested class.

why static variables?

write a program to define student class with properties name, rollnum, teacher, branch. create 3 student objects with different properties initialized through a constructor. print all student details.

```
public class StaticExample
{

	String name;
	int roll_num;
	String teacher;
	String branch;

	StaticExample(String name, int roll_num, String teacher, String branch)
	{
		this.name = name;
		this.roll_num = roll_num;
		this.teacher = teacher;
		this.branch = branch;
	}

	void displayDetails()
	{
		System.out.println("name is "+name+" roll_num is "+roll_num+" teacher is "
					+teacher+" branch is "+branch);
	}

	public static void main(String args[])
	{
		StaticExample s1=new StaticExample("Rahul",64,"prabhakar","CSE");
		StaticExample s2=new StaticExample("Ravi",34,"prabhakar","CSE");
		StaticExample s3=new StaticExample("kiran",54,"prabhakar","CSE");


		s1.displayDetails();
		s2.displayDetails();
		s3.displayDetails();
	}
}
```

OutPut:
name is Rahul roll_num is 64 teacher is prabhakar branch is CSE
name is Ravi roll_num is 34 teacher is prabhakar branch is CSE
name is kiran roll_num is 54 teacher is prabhakar branch is CSE



Like this, if I'm going to create total of 195 student objects. then lot memory will be misused to store redundant(repeated) information. As a good programmer we should always care about memory and execution time. Either it may be 'program statements' (or) 'data', we should never repeat.

static variables:

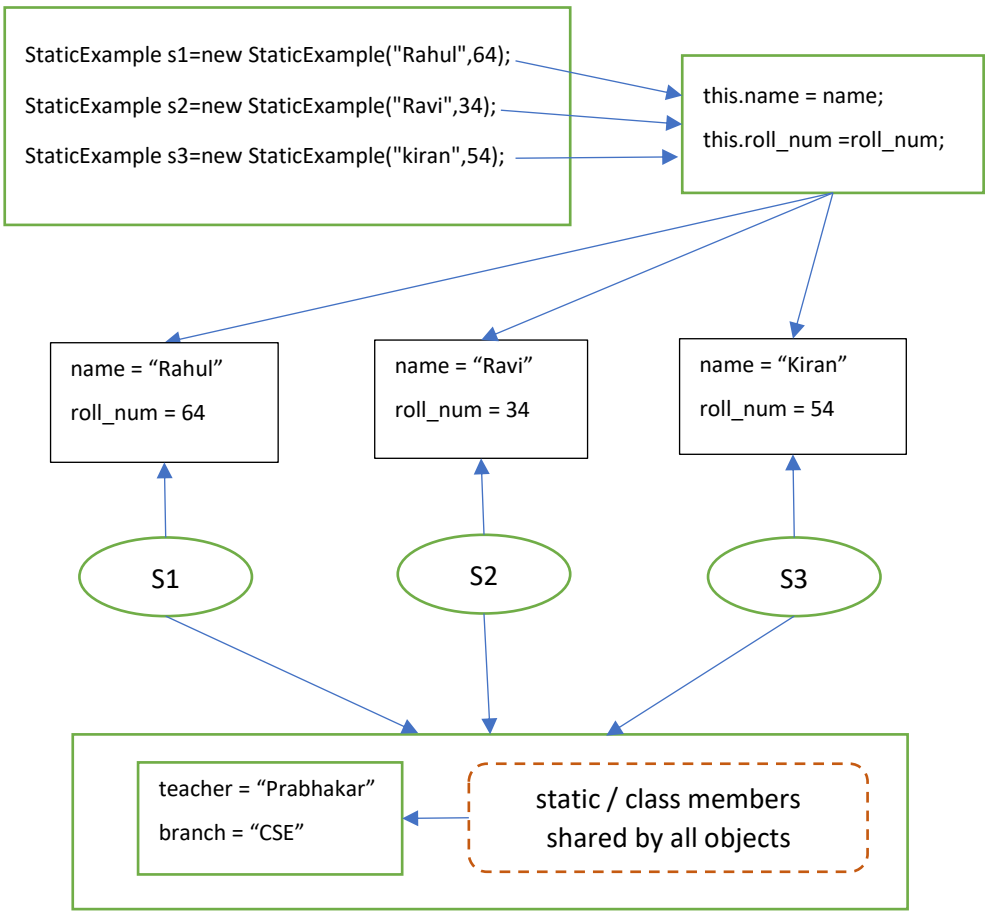
When a variable is declared as static, then a single copy of variable is created and shared among all objects.

For the above program, to store student details declare the properties teacher and branch as static and implement the same functionality

```
public class StaticExample
{

        String name;
        int roll_num;
        static String teacher = "Mr. Prabhakar";
        static String branch = "CSE";
```

```
        StaticExample(String name, int roll_num)
        {
                this.name = name;
                this.roll_num = roll_num;
        }

        void displayDetails()
        {
                System.out.println("name is "+name+" roll_num is "+roll_num+" teacher is
                        "+StaticExample.teacher+" branch is "+StaticExample.branch);
        }

        public static void main(String args[])
        {
                StaticExample s1=new StaticExample("Rahul",64);
                StaticExample s2=new StaticExample("Ravi",34);
                StaticExample s3=new StaticExample("kiran",54);

                s1.displayDetails();
                s2.displayDetails();
                s3.displayDetails();
        }
}
```

OutPut:
name is Rahul roll_num is 64 teacher is Mr. Prabhakar branch is CSE
name is Ravi roll_num is 34 teacher is Mr. Prabhakar branch is CSE
name is kiran roll_num is 54 teacher is Mr. Prabhakar branch is CSE



In the above program, member data teacher and branch are declared static, so only one variable for each teacher and branch will be created at class level. This static data will be shared by all student objects. it is programmer responsibility to identify which data (or) methods will be common to all objects and mark them as static, whereas data specific for individual objects should be non-static. here rollnum and name of student cannot be shared they are individual for each student object called as instance (or) object data.

Note:

1. static variables can be accessed from static blocks and static methods only.

static members can be accessed

i) directly without an object ii) with class name iii) using object

2. static members cannot invoke or access non static members.

static methods:

A static method can be invoked without creating an instance of class, they can be accessed directly or by class name.

A static method can access static data members only.

write a program with static method display, it accepts an integer number and print the square value of it, call the method display from main function by passing value 5.

```
class StaticExample
{
	static void display(int n)
	{
		int res;
		res = n*n;
		System.out.println("square value of "+ n +" is "+res);
	}

	public static void main(String args[])
	{
		display(5);
	}

}
```

OutPut:
square value of 5 is 25

static block:

whenever a java class file is executed, static block and static methods will be loaded. From static blocks and static methods, static block will be executed first. Later among the static methods JVM will start executing main method with following signature
public static void main(String args[ ])

static blocks are used to initialize the static variables.

Write a program with any print statement in static block.

```
class StaticExample
{
        static
        {
          System.out.println("static block will be executed first");
        }
        public static void main(String args[])
        {
                System.out.println("main will be executed after static block");
        }
}
```

OutPut:

static block will be executed first.
main will be executed after static block.

this keyword:

'this' is a reference, that refers to the current class object.

write a program to initialize instance variables in constructor.

```
class Example
{
        int a;
        float b;

        Example(int j, float k)
        {
                a = j;
                b = k;
        }

        public static void main(String args[ ])
        {
                Example obj = new Example(5,5.4f);
                System.out.println(obj.a);
                System.out.println(obj.b);

        }
}
```

OutPut:
5
5.4

what if variables declared inside constructor j, k are named with same name as instance variables i.e., a, b. then how to initialize instance variables ?

```
class Example
{
	int a;
	float b;

	Example(int a, float b)
	{
		a = a;
		b = b;
	}

	public static void main(String args[ ])
	{
		Example obj = new Example(5,5.4f);
		System.out.println(obj.a);
		System.out.println(obj.b);

	}
}
```

OutPut:
0
0.0
though constructor is defined to initialize instance variables, it has no effect because,

inside constructor
a = a;
b = b;

both at lhs and rhs a,b refers to same local variables that are declared inside constructor.

we can refer to instance variables using this operator, that refers to current object

```
class Example
{
	int a;
	float b;

	Example(int a, float b)
	{
		this.a = a;
		this.b = b;
	}

	public static void main(String args[ ])
	{
		Example obj = new Example(5,5.4f);
		System.out.println(obj.a);
		System.out.println(obj.b);

	}
}
```

OutPut:
5
5.4

this can also be used

1. to invoke current class method.
2. to invoke current class constructor.
3. this can be passed as an argument in the method call.

invoke current class method using this.

```
class ThisMethod
{
        void display()
        {
                System.out.println("from method display");
        }
        void n()
        {
                System.out.println("calling method using this");
                this.display();
        }

}
class ThisExample
{
        public static void main(String args[])
        {
                ThisMethod a=new ThisMethod();
                a.n();
        }
}
```

OutPut:
calling method using this
from method display

invoke current class constructor

The this() constructor call can be used to invoke another constructor of same class.

Program to invoke no argument constructor from parameterized constructor using this()

```
class ThisConstructor
{
        ThisConstructor()
        {
                System.out.println("From Constructor with no arguments.");
        }
        ThisConstructor(int x)
```

```
	{
		this();
		System.out.println("controller back to Parameterized constructor");
		System.out.println("value passed to constructor is "+x);
	}
}
class ThisExample
{
	public static void main(String args[])
	{
		ThisConstructor a = new ThisConstructor(10);
	}
}
```

OutPut:
From Constructor with no arguments.
controller back to Parameterized constructor
value passed to constructor is 10

Program to invoke parameterized constructor from, no argument constructor using this(argument)

```
class ThisConstructor
{
	ThisConstructor()
	{
		this(20);
		System.out.println("controller back to no argument constructor");

	}
	ThisConstructor(int x)
	{
		System.out.println("From Constructor with arguments.");
		System.out.println("value received is "+x);
	}
}
class ThisExample
{
	public static void main(String args[])
	{
		ThisConstructor a = new ThisConstructor();
	}
}
```

OutPut:

From Constructor with arguments.
value received is 20
controller back to no argument constructor

passed as an argument in the method call

The this can also be passed as an argument in function call. It is mainly used in the event handling. Let's see the example:

```
class ThisExample
{
   int a;
   int b;

   ThisExample()
   {
      a = 10;
      b = 20;
   }

   void display(ThisExample obj)
   {
      System.out.println("a = " + obj.a + "  b = " + obj.b);
   }

   void get()
   {
      display(this);
   }

   public static void main(String[] args)
   {
      ThisExample object = new ThisExample();
      object.get();
   }
}
```

OutPut:

a = 10 b = 20

Arrays one Dimensional and multidimensional

Array: An array is a data structure that stores elements of same data type in contiguous memory location. Once array size is defined it cannot be increased (or) decreased. The first element of an array starts with the index 0.

why arrays?

when you want store student marks, we will declare a variable and store value

int studentMarks; studentMarks = 92;

what if we need to store 90 students' marks, are we going to create 90 variables, oh it will be very difficult to program and maintain then. arrays will help us to store all these 90 student marks under same array name differed by array index (address location).

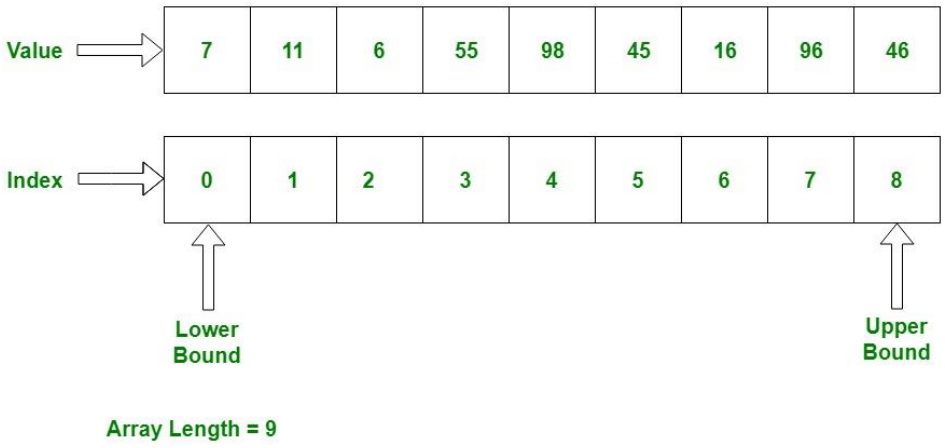
int studentMarks[ ] = new int[90];

studentMarks[0] = 80 // first element in array – marks of 1st student,

studentMarks[1] = 65 // second element in array – marks of 2nd student,

……

studentMarks[89] = 90 // last element in array – marks of 90th student



Declaring an Array –

datatype[ ] arrayname = new datatype[size];

(or)

datatype arrayname[ ] = new datatype[size];

Initializing array -

arrayname[index] = value;

Example –

//To create an array with name myarray of size 3.

int myarray[ ] = new int[3];

Initializing

myarray[0] = 1;

myarray[1] = 2;

myarray[2] = 3;

Note: If the array elements are known at the time of array declaration, we can initialize array elements along with the declaration itself as

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

an array, myarray will be created of size 5 with elements stored from index 0 to 4.

System.out.println(myarray[0]) // prints 1

System.out.println(myarray[4]) // prints 5

Write a program to declare an array with size n, then read n no of elements and store them in that array.

```
import java.util.Scanner;
class MyExample
```

```
{
	public static void main(String args[])
	{
		int n;
		Scanner sc = new Scanner(System.in);
	System.out.println("How many elements you want to store :")
		n = sc.nextInt();
		int ary[ ] = new int[n];
	// reads n elements from console, and stores in to array, ary
		System.out.println("Enter "+n+" no of elements");
		for(int i= 0; i<n;i++ )
		{
			ary[i] = sc.nextInt();
		}
	// print each value stored in the array ary,
		System.out.println("elements stored in array are");
		for(int i = 0; i< n; i++)
		{
			System.out.println(ary[i]);
		}
	}
}
```

OutPut:
How many elements you want to store : 4
Enter 4 no of elements: 2 4 6 8
elements stored in array are
2 4 6 8

Multidimensional arrays:

A multi dimensional array, is array for arrays. It has multiple levels. The simplest multi-dimensional array is the 2D array i.e., two-dimensional array

why multi-dimensional array?

when I want to store students' total marks, a one Dimensional array is enough. what if we want to store the 6 subject's marks of 90 students, we cannot accommodate them in one dimensional array, and we cannot declare 90*6 = 540 variables and memorize them.

In such a scenario, we can use 2-Dimensional array. one dimension to represent student number and other dimension to represent subject wise marks.

```
int studentSubjectWiseMarks[ ][ ] = new int[90][6];
```

A two-dimensional array can also be treated as rows and columns, in the above declaration studentSubjectWiseMarks we will have 90 rows and 6 columns for each row. where the row index and column index both starts from 0.

practice indexing,
let's assign first student, six subject marks with 80, 60,56,70,86,90

studentSubjectWiseMarks[0][0] = 80;

studentSubjectWiseMarks[0][1] = 60;

studentSubjectWiseMarks[0][2] = 56;

studentSubjectWiseMarks[0][3] = 70;

studentSubjectWiseMarks[0][4] = 86;

studentSubjectWiseMarks[0][5] = 90;

second student, six subject marks with 70, 64,58,74,86,70

studentSubjectWiseMarks[1][0] = 70;

studentSubjectWiseMarks[1][1] = 64;

studentSubjectWiseMarks[1][2] = 58;

studentSubjectWiseMarks[1][3] = 74;

studentSubjectWiseMarks[1][4] = 86;

studentSubjectWiseMarks[1][5] = 70;

……

90th student, six subject marks with 90, 90,90,80,86,70

studentSubjectWiseMarks[89][0] = 90;

studentSubjectWiseMarks[89][1] = 90;

studentSubjectWiseMarks[89][2] = 90;

studentSubjectWiseMarks[89][3] = 80;

studentSubjectWiseMarks[89][4] = 86;

studentSubjectWiseMarks[89][5] = 70;

Note:

The important thing that we should be careful is with indexing of an element. If you understand how to identify an element with its index you can simply play with arrays.

The frequent error you will observe while dealing with arrays is Array Index Out of Bounds Exception.

int a[ ] = new int[5]

here lower bound index is 0, upper bound index is 4, total 5 elements.

if we try to access an element out of these bounds, we will get, Array Index Out of Bounds Exception.

a[0] = 1; a[1] = 2; a[2] = 3; a[3] = 4; a[4] = 5; These are perfectly valid.

a[-1] = 3; a[5] = 4; a[56] = 7; these will cause an Array Index Out of Bounds Exception.

Declaring 2 – Dimensional array:

int matrix[ ][ ] = new int[4][3];

Initializing 2-D array:

matrix[0][0] = 1; matrix[0][1] = 2; matrix[0][2] = 3;

matrix[1][0] = 4; matrix[1][1] = 5; matrix[1][2] = 6;

matrix[2][0] = 7; matrix[2][1] = 8; matrix[2][2] = 9;

matrix[3][0] = 10; matrix[3][1] = 11; matrix[3][2] = 12;

rows ↓	columns →	0	1	2
0		matrix[0][0]	matrix[0][1]	matrix[0][2]
1		matrix[1][0]	matrix[1][1]	matrix[1][2]
2		matrix[2][0]	matrix[2][1]	matrix[2][2]
3		matrix[2][0]	matrix[3][1]	matrix[3][2]

write a program to store elements from 1 to 12, using 2-D array of order 4*3

```
public class MatrixExample
{
	public static void main(String args[])
	{
		int matrix[ ][ ] = new int[4][3];
		int k=1;
		for(int i = 0; i<4; i++)		// represents rows
		{
			for(int j = 0; j<3; j++)	// represents columns
			{
				matrix[i][j] = k;	// storing elements in array
				k++;
			}
		}
```

```
		// display all the elements in matrix array
		for(int i = 0; i<4; i++)
		{
			for(int j = 0; j<3; j++)
			{
				System.out.println(matrix[i][j]);
			}
		}
	}
}
```

OutPut :

1 2 3 4 5 6 7 8 9 10 11 12 (I know they will print line by line, don't want to make the document lengthy).

write a program that will ask the user for no of rows and no of columns, to declare 2-D array, and then ask user to enter elements for that matrix. finally print the elements in matrix format.

```
import java.util.*;
public class Example
{
	public static void main(String args[])
	{
		int rows, columns;
		Scanner sc = new Scanner(System.in);

		System.out.print("Enter no of rows: ");
		rows = sc.nextInt();				// reads no of rows

		System.out.print("Enter no of columns: ");
		columns = sc.nextInt();				// reads no of columns

	// Now declare 2-D array with specified rows and columns
		int matrix[][] = new int[rows][columns];
		System.out.print("Enter elements into matrix: ");
		for(int i = 0; i<rows; i++)			// represents rows
		{
			for(int j = 0; j<columns; j++)			// represents columns
			{
				matrix[i][j] = sc.nextInt();	// read element into the array
			}
		}

		// display all the elements in matrix array
		for(int i = 0; i<rows; i++)
		{
			for(int j = 0; j<columns; j++)
			{
```

```
                        System.out.print(matrix[i][j]+" ");
                    }
                    System.out.println();// line break, after printing each row elements
                }
        }
}
```

OutPut:

```
Enter no of rows: 3
Enter no of columns: 3
Enter elements into matrix: 1 2 3 4 5 6 7 8 9
1 2 3
4 5 6
7 8 9
```

*** Note : Finding length of an array, most important**
How to get no of elements/size of an array
arrayname.length will give us the size

example –
int a[] = new int[4];
System.out.println(a.length);
the output on console will be 4.

Program to print first and last elements of an array

```
public class ArrayExample
{
        public static void main(String args[])
        {
                int a[ ] = {1,2,3,4};
                display(a);
        }
        static void display(int b[])
        {
                int firstIndex = 0;
// b.length will give size 4, but last index is 3, as index starts from 0
                int lastIndex = b.length-1;
                System.out.println("First element is "+b[firstIndex]);
                System.out.println("Last Element is "+b[lastIndex]);
        }
}
```

OutPut :

```
First element is 1
Last Element is 4
```

Jagged Arrays:

Jagged array is a multidimensional array where member arrays are of different size. each row will have different no of elements in column .

Let say I want to store Student numbers of 3 sections, where each section have different number of students.

section 1 contains 64 students,
section 2 contains 65 students
section 3 contains 66 students

how to declare, first declare array with no of rows i.e., 3

```
int students[ ][ ] = new int[3][ ];
```

now declare size of each row, called member array.

```
students[0] = new int[64];
students[1] = new int[65];
students[2] = new int[66];
```

write a program that store elements if following manner
there should be 3 rows,
1st row stores elements 1,2
2nd row stores elements 3,4,5
3rd row stores elements 6,7,8,9

```
public class ArrayExample
{
	public static void main(String args[])
	{
		int rows = 3;
		int num = 1;

		int a[ ][ ] = new int[rows][ ];
		a[0] = new int[2];
		a[1] = new int[3];
		a[2] = new int[4];

		for(int i=0; i<a.length; i++)
		{
			for(int j = 0;j<a[i].length; j++)
			{
				a[i][j] = num;
				num++;
			}
		}

		for(int i=0; i<a.length; i++)
```

```
		{
			for(int j = 0;j<a[i].length; j++)
			{
				System.out.print(a[i][j]+" ");
			}
			System.out.println();
		}

	}
}
```

OutPut:
1 2
3 4 5
6 7 8 9

Command line arguments:

Command Line Arguments is the information passed to program, at the time of execution. The passed information is stored as a string array in the main method. Command line arguments can be retrieved from String args[ ] array declared in main function.

Write a program to print given name and branch details, passed at run time using command line arguments.

```
class CommandLineArguments
{
	public static void main (String args[])
	{
		System.out.println("name is "+args[0]);
		System.out.println("branch is "+args[1]);
	}
}
```

OutPut:

```
C:\Users\UMA SHANKAR\Desktop>javac StaticExample.java

C:\Users\UMA SHANKAR\Desktop>java StaticExample Ravi ECE
name is Ravi
branch is ECE
```

Garbage Collection:

In C, C++ languages, it is the programmer's responsibility to free the memory allocated dynamically, using free() function. If a programmer knowingly or unknowingly doesn't free the memory he consumed, it effects the application performance because of inefficient memory utilization. Whereas, Java takes care of memory management. java itself is responsible to free the unused (un referenced) object data.

In the Java, memory will be allocated to objects dynamically using the **new** operator. Once an object is created, the memory remains allocated till there are references for this object.

Garbage Collector is a program in java, which is responsible for de-allocating memory occupied by un referenced objects. Garbage collector will be running periodically to de allocate memory of unused objects.

When a java object is being destroyed, Garbage Collector calls finalize() method on the object to perform clean-up activities(free the resources accessed by that object). Once finalize() method is executed, Garbage Collector will destroy that object.

when an object is said to be ready for garbage collection ?

1. object reference variable is made null
2. when object reference variable points to another object

'to be specific when there is not at least one reference to that object'

Write a program implementing finalize() method with any print statement, and make it executed from garbage collector

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

OutPut:

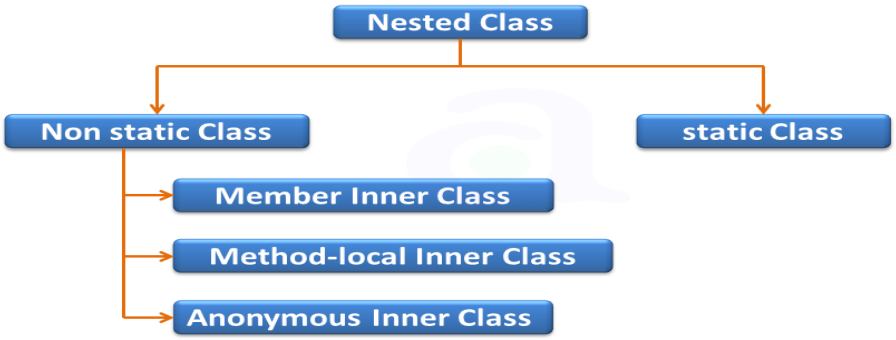
object is garbage collected.

Nested Classes

java allows us to define a class within another class, known as the **nested class**

```
class OuterClass {
   ...
   class NestedClass {
      ...
   }
}
```

Nested classes are divided into **static** and **non-static**. Nested classes that are declared as static are called ***static nested classes***. Non-static nested classes are called ***inner classes***.




```
class OuterClass {
        ...

        //static nested class
        static class StaticNestedClass {
             ...
        }

        //non-static nested class
        class InnerClass {
             ...
        }
}
```

Note:

Static nested classes do not have access to non-static members of the outer class.

As a member of the outer class, a nested class can be declared **private**, public, protected or can be left default.

Static Nested class :

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

```
class Outer
{
        static class StaticNestedClass
        {
                void innerClassMethod()
                {
                                System.out.println("from static Nested class");
                }
        }
        public static void main(String args[])
        {
                Outer.StaticNestedClass obj2 = new Outer.StaticNestedClass();
                obj2.innerClassMethod();
        }
}
```

Note : inner classes can access attributes and methods of the outer class

Inner Class:

A non-static class created within class and outside all methods

```
class Outer
{
	class InnerClass
	{
		void innerClassMethod()
		{
				System.out.println("from Inner class");
		}
	}

	public static void main(String args[])
	{
		Outer obj1	= new Outer();
		Outer.InnerClass obj2 = obj1.new InnerClass();
		obj2.innerClassMethod();
	}
}
```

Note: To create an object for inner class, We must create object for outer class first.

OuterClass.InnerClass ref= outerClassObject.new InnerClass();

Method Local Inner Class:

An inner class can be defined with in a method, just like local variables, the scope of the inner class is within the method. A method-local inner class can be instantiated only within the method.

```
class Outer
{
	void outerClassMethod()
	{
		class Inner
		{
			void innerClassMethod()
			{
				System.out.println("from method local inner class");
			}
		}
		Inner obj2 = new Inner();
		obj2.innerClassMethod();
	}
	public static void main(String args[])
	{
		Outer obj1 = new Outer();
		obj1.outerClassMethod();
	}
}
```

Anonymous InnerClass

Anonymous classes are declared at the time of instantiation/object creation. An inner class is declared without a class name that's why known to be anonymous inner class. Anonymous classes are mostly used to override methods of interfaces.

```
class Outer
{
	void display()
	{
		System.out.println("from outer class method");
	}
	public static void main(String args[])
	{
		Outer obj1 = new Outer();
		obj1.display();

		Outer obj2 = new Outer()
		{
			void display()
			{
				System.out.println("from anonymous inner class");
			}
		};
		obj2.display();
	}
}
```