Module-1 (Part-II): Class and Object

by:

Dr. Soumya Priyadarsini Panda

Sr. Assistant Professor

Dept. of CSE

Silicon Institute of Technology, Bhubaneswar

Class Fundamentals

- Class defines a new data type.

- Once a class is defined, it can be used to create objects of that type.

- A class is a template for an object and object is an instance of a class.
- It is a logical entity.

- A class is defined by use of the ***class*** keyword

The General Form of a Class

```
class classname{
                type   instance-variable1;
                type   instance-variable2;
                . . .
                type   instance-variableN;

                type   methodname1(parameter-list) {
                       //body of method
                       }
                 type   methodname2(parameter-list){
                       //body of method
                       }
                . . .
                type   methodnameN(parameter-list){
                       //body of method
                }
        }
```

Instance Variables

- The data or variables defined within a class but outside the method are called **instance variables.**

- Instance variable doesn't get memory at compile time. It gets memory at run time when object (instance) is created.

Methods

- A method is like function used to expose behaviour of an object.

- The code is contained within methods.

- The main advantage of the method is code reusability.

Cont…

- Both variables and methods within a class are called members of the class.

- Each instance (object) of the class contains its own copy of these variables.

- Thus the data of one object is separate and unique from the data for other.

Example

```
class Box{
        double width;
        double height;
        double depth;
  }
```

- A class declaration only creates a template, it does not create an actual object.

- To create an object of Box class:

```
        Box mybox = new Box();
```

Cont…

- Every Box object will contain its own copies of the instance variables width, height and depth of each instance variable defined by the class.

- To access the instance variables and methods within an object the dot(.) operator is needed

Example:

```
mybox.width
mybox.height
mybox.depth
```

Example-1

```
class Box {
    double width;
    double height;
    double depth;
    }
class BoxDemo {
    public static void main(String args[]) {
            Box b1 = new Box();
            double  vol;

            b1.width = 10;
            b1.height = 20;
            b1.depth = 30;

            vol = b1.width * b1.height * b1.depth;
            System.out.println("Volume is  " +vol);
    }
 }
```

Output:

```
            Volume is 6000.0
```

Example-2

```
class BoxDemo{
    public static void main(String args[]){
            Box b1 = new Box();
            Box b2 = new Box();
            double vol;

            b1.width = 10;
            b1.height = 20;
            b1.depth = 30;

            b2.width = 2;
            b2.height = 3;
            b2.depth = 6;

            vol = b1.width * b1.height * b1.depth;
            System.out.println("Volume of first box  " +vol);

            vol = b2.width * b2.height * b2.depth;
            System.out.println("Volume of  second box " +vol);
    }
 }
```

Output:

```
Volume of first box 6000.0
Volume of second box 36.0
```

Objects

- Object is an instance of a class.
- Class is a template or blueprint from which objects are created. So object is the instance (result) of a class.

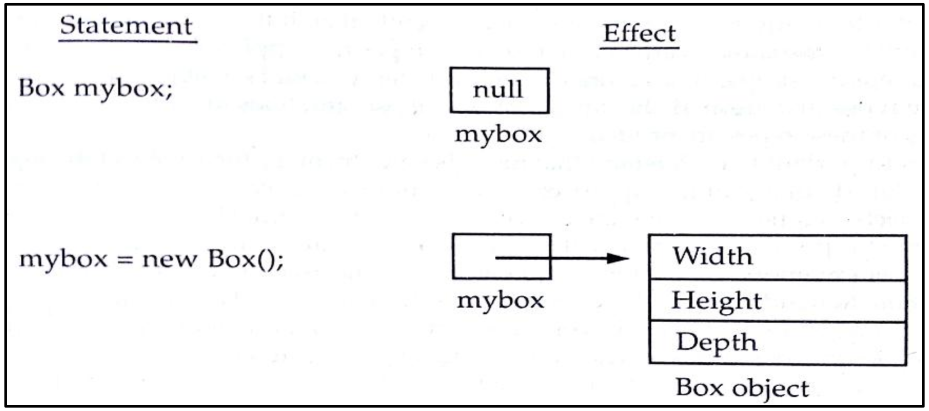
Declare a variable of the class type:

Box mybox;

Acquire an actual, physical copy of the object and assign it to that variable:

mybox = new Box();

- The **new** operator dynamically (at run time) allocates memory for an object and returns a reference (address) to it.



Methods

- A method is a function that is written in a class.
- Whenever a function is written in Java it should be written inside the class only.

The general form of a method:

```
          type name (parameter-list)
               {
               //body of method
               }
```

Example-1:

```
class Box
{
    double width;
    double height;
    double depth;

    //display volume of a box
    void volume()
            {
            System.out.println(“ Volume  is “ + (width *height *depth));
            }
}
```

Using Scanner class to read data at runtime

```
class BoxDemo
{
    public static void main(String args[])
            {
            Box b1 = new Box();
            Scanner s1=new Scanner(System.in);

            System.out.println("Enter the width height and depth for the box");
            b1.width = s1.nextDouble();
            b1.height = s1.nextDouble();
            b1.depth = s1.nextDouble();
            b1.volume();
            }
    }
```

Cont…

```
class BoxDemo {
    public static void main(String args[]){
            Box  b1 = new Box();
            Box b2 = new Box();

            b1.width = 10;
            b1.height = 20;
            b1.depth = 30;

            b2.width = 3;
            b2.height = 2;
            b2.depth = 6;
            b1.volume();
            b2.volume();
    }
 }
```

```
Output:
    Volume is 6000.0
    Volume is 36.0
```

Example-2: Method with return type

```
class Box{
    double width;
    double height;
    double depth;

    //display volume of a box
    double volume()
             {
             return width *height *depth;
             }
}
```

Cont…

```
class BoxDemo{
    public static void main(String args[]){
            Box b1 = new Box();
            Box b2 = new Box();
            double vol;
            b1.width = 10;
            b1.height = 20;
            b1.depth = 30;
            b2.width = 3;
            b2.height = 2;
            b2.depth = 6;
            vol = b1.volume();
            System.out.println(“ Volume is “ + vol);
            vol = b2.volume();
            System.out.println(“ Volume is “ + vol);
            }
 }
```

Output:

```
    Volume is 6000.0
     Volume is 36.0
```

Note:

/*In well-designed Java programs, instance variable should be accessed only through methods defined by their class.

In that way the behaviour of a method can be changed easily if required */

Example-3

```
class Box{
    double width,
    double height;
    double depth;
    //display volume of a box
    double volume(){
             return width *height *depth;
             }
    void setDim(double w, double h, double d) {
             width = w;
             height = h;
             depth = d;
             }
    }
```

Cont…

```
class BoxDemo{
    public static void main(String args[]){
            Box b1 = new Box();
            Box b2 = new Box();
            double vol;

            b1.setDim(10, 20, 30);
            b2.setDim(3, 2, 6);

            vol = b1.volume();
            System.out.println(“ Volume is “ + vol);
            vol = b2.volume();
            System.out.println(“ Volume is “ + vol);
    }
 }
```

```
Output:
            Volume is 6000.0
            Volume is 36.0
```

Question for Homework

- Define a class **Rectangle**, having the data members length and width. The class should have a method which can return the area of the Rectangle. Create another **RectangleDemo** class which create an object of the Rectangle class and test the functionalities.

Constructors

- Java allows objects to initialize themselves when they are created through the use of a constructor.

- A constructor initializes the instance variables immediately upon the creation of objects.

Cont…

- A constructor has the following characteristics:
  - It has the same name as the class in which it resides and is syntactically similar to a method.
  - A constructor does not return any value, not even void.
    - This is because the implicit return type of a class' constructor is the class type itself.
  - Once we define a constructor, during the object creation, it is automatically called and finishes its execution before the new operator completes its work.
  - A constructor is called and executed only once per each object creation.

Example-1: Default Constructor

```
class Box
{
    double width;
    double height;
    double depth;

        //Constructor
        Box() {
                width = 20;
                height = 20;
                depth = 20;
        }
        //method
        double volume() {
        return width *height *depth;
        }
```

Example-1 Cont…

```
class BoxDemo {
    public static void main(String args[]) {
            Box b1 = new Box();
            Box b2 = new Box();
            double vol;

            vol = b1.volume();
            System.out.println(“ Volume is “ + vol);
            vol = b2.volume();
            System.out.println(“ Volume is “ + vol);
    }
}
```

Output:

```
Volume is 8000.0
Volume is 8000.0
```

Constructors Cont…

- In the line Box b1 = new **Box()**;

  Box() constructor is called.

- It is not necessary to write a constructor for a class.
  - It is because java compiler creates a default constructor if your class doesn't have any.

Types of Constructors

- Default Constructor
- Parameterized Constructor

Default Constructor

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

  syntax of default constructor:

  ```
  class-name(){ }
  ```

  Example:

  Box() is a default constructor in previous example

Cont…

- Default constructor is used to provide the default values to the object like 0, null etc. depending on the type.

- It is useful to initialize all objects with the same data.

- Once we define our own constructor, the default constructor is no longer used.

- When data is not passed at the time of creating an object, default constructor is called.

Parameterized Constructor

- A constructor which has a specific number of parameters is called parameterized constructor.

- It is useful to initialize each object with different data.

- When data is passed at the time of creating an object, parameterized constructor is called.

Example-2: Parameterized Constructor

```
class Box
{
    double width;
    double height;
    double depth;
    // Parameterized Constructor
    Box(double w, double h, double d){
                        width = w;
                        height = h;
                        depth = d;
    }
    //display volume of a box
    double volume(){
              return width *height *depth;
              }
    }
```

Example-2 Cont…

```
class BoxDemo {
     public static void main(String args[]) {
               Box b1 = new Box(10, 20, 30);
               Box b2 = new Box(3, 2, 6);
               double vol;

               vol = b1.volume();
               System.out.println(“ Volume is “ + vol);
               vol = b2.volume();
               System.out.println(“ Volume is “ + vol);
               }
 }
```

Output:

```
Volume is 6000.0
Volume is 36.0
```

Difference between Constructor and Method

Java Constructor	Java Method
Constructor is used to initialize the state of an object.	Method is used to expose behaviour of 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 in any case.
Constructor name must be same as the class name.	Method name may or may not be same as class name.

Example-3

Define a class Stack, which perform the basic operation of stack. Define another driver class to demonstrate the basic operations.

```
class Stack
   {
            int arr[];
            int top;
            int size;

            Stack(int s)
                     {
                     size=s;
                     top = -1;
                     arr=new int[size];
                     }
```

```
void push(int item)
  {
      if(top==size-1)
            System.out.println("Overflow");
      else
            {
            top++;
            arr[top]=item;
            }
  }
```

```
int pop()
  {
        if(top<0)
        {
                System.out.println("Underflow");
                return -1;
        }
        else
                return arr[top--];
  }
}
```

```
class TestStack
{
    public static void main(String args[])
            {
            Stack s1 = new Stack(5);  //for a stack with size 5
            for(int i=1;i<=5;i++)
                    s1.push(i);

            System.out.print("Elements in stack-1:");
            for(int i=0;i<5;i++)
                    System.out.print("  "+ s1.pop());
            }
}
```

Output:

```
            Elements in stack-1: 5 4 3 2 1
```

```
//for implementing 2 stacks
class TestStack
{
    public static void main(String args[])
            {
            Stack s1 = new Stack(5);                              //  stack1 size=5
            Stack s2 = new Stack(10);                             //stack2 size =10

            for(int i=1;i<=5;i++)
                        s1.push(i);

            for(int i=11;i<=20;i++)
                        s2.push(i);
```

```
        System.out.print("Elements in stack-1:");
        for(int i=0;i<5;i++)
                System.out.print("  "+ s1.pop());

        System.out.println("");
        System.out.print("Elements in stack-2:");
        for(int i=0;i<10;i++)
                System.out.print("  "+ s2.pop());
        }

    }
```

Output:

```
        Elements in stack-1: 5 4 3 2 1
        Elements in stack-2: 20 19 18 17 16 15 14 13 12 11
```

Constructor Overloading

- Constructor overloading in Java is a technique of having more than one constructor with different parameter lists.

- Each constructor performs a different task.

- They are differentiated by the compiler by the number of parameters in the list and their types.

Example

```
class Box
{
    double width;
    double height;
    double depth;


    // constructor used when all dimensions specified
    Box(double w, double h, double d)
    {
        width = w; height = h; depth = d;
    }

// constructor used when no dimensions specified
    Box()
    {
        width = -1;
        height = -1;
        depth = -1;
    }
    // constructor used when cube is created
    Box(double len)
            {
            width = height = depth = len;
            }

    // compute and return volume
    double volume()
            {
            return width * height * depth;
            }
} //end of Box class
```

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

            Box box1 = new Box(10, 20, 15);
            Box box2 = new Box();
            Box box3 = new Box(7);
            double vol;

            vol = box1.volume();
            System.out.println("Volume of box1 is " + vol);


            vol = box2.volume();
            System.out.println("Volume of box2 is " + vol);


            vol = box3.volume();
            System.out.println("Volume of box3 is " + vol);
    }
}
```

Output:

```
Volume of box1 is 3000.0
Volume of box2 is -1.0
Volume of box3 is 343.0
```

Passing Object to Constructor

```
class Box
{
    double width;
    double height;
    double depth;
    …….
    Box()
    {
            width = -1;
            height = -1;
            depth = -1;
    }
    …….
    // passing object to constructor
    Box(Box ob)
            {
            width = ob.width;
            height = ob.height;
            depth = ob.depth;
    }
    …..
    …..
```

```
class OverloadCons {
    public static void main(String args[])  {
            Box b1 = new Box(10, 20, 15);
            Box b2 = new Box();
            Box b3 = new Box(7);
            Box b4 = new Box(b1); // create copy of b1 in b4

            double vol;
            vol = b1.volume();
            System.out.println("Volume of box1 is " + vol);

            vol = b2.volume();
            System.out.println("Volume of box2 is " + vol);

            vol = b3.volume();
            System.out.println("Volume of box3 is " + vol);

            vol = b4.volume();
            System.out.println("Volume of clone box4 is " + vol);
    }
}
```

Output:

```
Volume of box1 is 3000.0
Volume of box2 is -1.0
Volume of box3 is 343.0
Volume of clone box4 is 3000.0
```

Question for Homework

- Define a class Rectangle that uses a parameterized constructor to initialize the dimensions of a Rectangle, having the data members length, and width,. The class should have a method which can return the area of the Rectangle. Create an object of the Rectangle class and test the functionalities.

Example: Array of Objects

Write a java program to define a class Employee with data members empId, empName and a method to display the employee details. Define a constructor to initialize the members of the class. Read and display the **details of 5 employees using an array of Employee class object**.

```
class Employee{
    int empid;
    String name;

       Employee(int id, String n,){
       empid=id;
       name=n;
           }

    void disp()
           {
           System.out.println(empid+ "\t\t"+name);
           }
}
```

For one Employee

```
class Test
   {
    public static void main(String args[])
       {
        Employee e1=new Employee(1,“Amit");
        System.out.println("EMPID: \t\t NAME:);
        e1.disp();
       }
    }
```

Output:

```
EMPID:       NAME:
    1          Amit
```

For Three Employees

```
class Test
          {
      public static void main(String args[])
          {
          Employee e1=new Employee(1,“Amit");
          Employee e2=new Employee(2,“Sumit");
          Employee e3=new Employee(3,“Rohit");
          System.out.println("EMPID: \t\t NAME:);
          e1.disp();
          e2.disp();
          e3.disp();
      }
   }
```

Output:

```
EMPID:      NAME:
     1     Amit
     2     Sumit
     3     Rohit
```

For large number of Employees-
Use array of Objects

```
class Employee{
    int empid;
    String name;
   void disp() {
           System.out.println(empid+ "\t\t"+name);
           }
//method to read data
void getData()
     {
    Scanner sc=new Scanner(System.in);
    System.out.println("enter id");
    empid=sc.nextInt();
    System.out.println("enter name");
    name=sc.nextLine();
    }
    }
```

```
class Test{
        public static void main(String args[]){
        int i;
        Employee e[]=new Employee[3];
        for(i=0;i<3;i++){
                e[i]=new Employee();
                }
        System.out.println("enter the details");
        for(i=0;i<3;i++){
                e[i].getData();
                }
        System.out.println("EMPID: \t\tNAME:);
        for(i=0;i<3;i++){
                e[i].disp();
        }
}
}
```

Questions

1. Define a class **Rectangle**, having the data members length and width. The class should have a **method which can return the area** of the Rectangle. Include **both default and parameterized constructors** in the class. Create another **RectangleDemo** class which create an object of the Rectangle class and test the functionalities.
2. Define a class **Student** having the attribute **rollNo, name, branch, cgpa** and a **method to display** the student details. Define **default and parameterized constructors** for the above class. Read the details of **5 students** using an **array of Student class object**. Display the student detail who has **secured the highest cgpa**.

Method Overloading

- If a class has multiple methods having same name but different in parameters, it is known as Method Overloading.

- In Java, two or more methods can have same name if they differ in parameters (different number of parameters, different types of parameters, or both).

- Method overloading increases the readability of the program.

- When an overloaded method is invoked, Java uses the type and/or number of arguments as its guide to determine which version of the overloaded method to actually called.

- Overloaded methods must differ in the type and/or number of their parameters.

Cont…

- Two ways to overload the method in java-
  - By changing number of arguments
  - By changing the data type

Example-1: Method Overloading

```
class OverloadDemo {
    void test() {
            System.out.println("No parameters");
            }
    void test(int a) {
            System.out.println("a: " + a);
            }
    void test(int a, int b) {
            System.out.println("a and b: " + a + " " + b);
            }
    double test(double a) {
            System.out.println("double a: " + a);
            return a*a;
    }}
```

```
class Overload
    {
    public static void main(String args[])
            {
            OverloadDemo ob1 = new OverloadDemo();

            double result;
            ob1.test();
            ob1.test(10);
            ob1.test(10, 20);
            result = ob1.test(123.25);

            System.out.println(double method returned result: “+result);
    }
}
```

Output:

```
No parameters
a: 10
a and b: 10 20
double a: 123.25
ouble method returned result: 15190.5625
```

Example-2

```
class OverloadDemo {
   void test() {
            System.out.println("No parameters");
            }

   // Overload test for two integer parameters.
   void test(int a, int b) {
            System.out.println("a and b: " + a + " " + b);
            }
   // overload test for a double parameter
   void test(double a) {
            System.out.println("Inside test double method a: " + a);
            }
```

```
class Overload
{
    public static void main(String args[])
    {
            OverloadDemo ob = new OverloadDemo();
            int i=88;

            ob.test();
            ob.test(10, 20);
            ob.test(i);
            ob.test(123.2);
    }
}
```

Output:

```
No parameters
a and b: 10 20
Inside test(double) a: 88.0
Inside test double method a: 123.2
```

Question for Homework

Write a java program to find out the area of rectangle, square and circle using method overloading and constructor overloading.

static Keyword

- The ‘static’ keyword in java is used for memory management mainly.

- When a member is declared static, it can be accessed before any objects of its class are created, and without reference to any object.

- ‘static’ keyword can be used to declare-
  - Instance variables
  - Methods
  - Blocks
  - Nested class

static Instance Variables

- Any variable declared as static, it is known static variable.

- Static instance variable are global variables which are used to refer the common property of all objects
  - e.g. company name of employees, college name of students etc.

- When objects of its class are declared, no copy of the static variable is made.
  - All instances of the class share the same static variable.

- The static variable gets memory only once in class area at the time of class loading.

- Advantage of static variable: It makes program memory efficient (i.e saves memory).

Example-1

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

- If there are 1000 students in a college, all instance data members will get memory each time when an object is created

Example-2

```
class Student{
         int rollno;
        String name;
        static String college="Silicon";

        Student(int r, String n)
                {
                rollno = r;
                name = n;
                }
  void display ()
                {
                System.out.println(rollno + “ ” + name + “ ” + college );
                }
        }
```

Cont…

```
class Test{
    public static void main(String args[]){
            Student s1 = new Student(1,“Amit");
            Student s2 = new Student (2,“Sumit");

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

Output:

```
            1 Amit Silicon
            2 Sumit Silicon
```

Example-3

- static variable will get the memory only once, if any object changes the value of the static variable, it will retain its value.

```
class Counter{
        static int count=0; //will get memory only once and retain its value
        Counter(){
                count++;
                System.out.println(count);
                }  }
Class Test{
        public static void main(String args[]) {
        Counter c1=new Counter();
        Counter c2=new Counter();
        Counter c3=new Counter();
    }
}
```

Output:

```
1
2
3
```

static Method

- Any method declared as static 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.

- A static method can only directly access static data member and can change the value of it.

- A static method cannot call non-static method directly it can call static methods only.

- A non-static method can call both static and non-static methods.

Example

```
class Student{
        int rollno;
        String name;
        static String college ="Silicon";
        Student(int r, String n)
                {
                rollno = r;
                name = n;
                }
   static void change(){
        college = "SIT";
        }
   void display (){
   System.out.println(rollno+" "+name+" "+college);
   }
}
```

Cont…

```
class Test{
        public static void main(String args[])
                {
                Student s1 = new Student(1, “Amit”);
                Student s2 = new Student(2,“Sumit”);

                s1.display();
                s2.display();
                Student.change();   //calling a static method

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

```
Output:
1 Amit Silicon
2 Sumit Silicon
1 Amit SIT
2 Sumit SIT
```

Why java main method is static?

- Because object is not required to call static methods.
- If we would make main() method as non-static then, JVM would need to create object of that class first to call the main() method
  - Which would lead the problem of extra memory allocation.

static Block

- A static block is a block of statements declared as static.

```
static{
        statements;
        }
```

- JVM executes the static block before executing main( ) method.
- Static block is used for initializing the static variables.
- This block gets executed when the class is loaded in the memory.

Example-1

```
class JavaExample
        {
        static int num;
        static String mystr;
        static
                {
                num = 97;
                mystr = "Static keyword in Java";
                }
        public static void main(String args[])
                {
                System.out.println("Value of num: "+num);
                System.out.println("Value of mystr: "+mystr);
                }
        }
```

Output:

```
Value of num: 97
Value of mystr: Static keyword in Java
```

Example-2

- A class can have multiple Static blocks, which will execute in the same sequence in which they have been written into the program.

```
class Example2{
            static int num;
             static String mystr;

             //First Static block
              Static {
                                       System.out.println("Static Block 1");
                                       num = 68;
                                       mystr = "Block1";
                         }
```

Cont…

```
 //Second static block
  Static {
      System.out.println("Static Block 2");
      num = 98;
      mystr = "Block2";
  }
public static void main(String args[])
  {
      System.out.println("Value of num: "+num);
      System.out.println("Value of mystr: "+mystr);
   }
}
```

Output:

```
Static Block 1
Static Block 2
Value of num: 98
Value of mystr: Block2
```

static Class

- A class can be made static only if it is a nested class.
- Nested static class doesn’t need reference of Outer class.
- A static class cannot access non-static members of the Outer class

Example

```
class Example{
                private static String str = "BeginnersBook";

                //Static class
                static class MyNestedClass
                {
                //non-static method
                public void disp() {

                /* If you make the str variable of outer class non-static then you
   will get compilation error because: a nested static class cannot access non-
   static members of the outer class.
     */
    System.out.println(str);
          }
```

Cont…

```
public static void main(String args[])
    {
            /* To create instance of nested class we didn't need the outer
            class instance but for a regular nested class you would need
            to create an instance of outer class first*/

             MyNestedClass ob1 = new MyNestedClass();
                        ob1.disp();
    }
}
```

Output:

```
    BeginnersBook
```

this keyword

- Sometimes a method may need to refer to the object that invoked it.
- To allow this, Java defines the ‘this’ keyword.
- ‘this’ can be used inside any method to **refer to the current object**.
- That is, ‘this’ is always a reference to the object on which the method was invoked.
- You can use this anywhere a reference to an object of the current class’ type is permitted.

Use of this Keyword

1. **to refer to the current class instance variable.**

2. to invoke current class constructor

3. to pass as an argument in the method

4. to pass as argument in the constructor call

5. this keyword can be used to return current class instance

Example-1:

```
class Student{
    int rollno;
    String name;
    float fee;
    Student(int rollno,String name,float fee)
              {
              rollno=rollno;
              name=name;
              fee=fee;
              }
              void display(){
              System.out.println(rollno+" "+name+" "+fee);
              }
    }
```

Cont…

```
class Test{
          public static void main(String args[])
                    {
                    Student s1=new Student(1,“Amit",4000f);
                    Student s2=new Student(2,“Sumit",7000f);
s1.display();
s2.display();
}
}
```

Output:

```
0 null 0.0
0 null 0.0
```

Cont…

- It is illegal in Java to declare two local variables with the same name inside the same or enclosing scopes.

- There can be local variables, including formal parameters to methods, which overlap with the names of the class' instance variables.

- When a local variable has the same name as an instance variable, the local variable hides the instance variable.

1. to refer to the current class instance variable

```
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);
       }
  }
```

Cont…

```
class Test{
        public static void main(String args[])
                {
                Student s1=new Student(1,“Amit",4000f);
                Student s2=new Student(2,“Sumit",7000f);
s1.display();
s2.display();
}
}
```

Output:

```
  1  Amit 4000.0
  2 Sumit 7000.0
```

Example-3:

```
// A redundant use of this in Box example.
class Box{
        double width, height, depth;
        ……
        Box(double w, double h, double d)
                {
        this.width = w;
        this.height = h;
        this.depth = d;
        }
  …..
}
```

Example-4

```
// Use this to resolve name-space collisions.
class Box{
        double width, height, depth;
        ……

        Box(double width, double height, double depth)
                {
                this.width = width;
                this.height = height;
                this.depth = depth;
                }
        …….
}
```

2. ‘this’ to invoke current class constructor

- The this () constructor call can be used to invoke the current class constructor. It is used to reuse the constructor.

```
class A{
            A(){
            System.out.println("Hello a");
            }
            A(int x){
                    this();
            System.out.println(x);
            }
    }
class Test{
            public static void main(String args[]){
            A a=new A(10);
            }
    }
```

Output:

```
Hello a
10
```

3. ‘this’ to pass as an argument in the method

- The this keyword can also be passed as an argument in the method. It is used to reuse one object in many methods.

```
class A{
                    void m(A obj){
                                 System.out.println("method is invoked");
                     }
                     void p(){
                                 m(this);
                                 }
}
class Test{
                     public static void main(String args[]){
                                 A a = new A();
                                 a.p();
                     }
}
```

Output:

```
method is invoked
```

Cont…

(4) this: to pass as argument in the constructor call

- The ‘this’ keyword can be passed in the constructor also.
- It is useful for using one object in multiple classes.

(5) this keyword can be used to return current class instance

- The ‘this’ keyword can be returned as a statement from the method.
- In such case, return type of the method must be the class type (non-primitive).

Access Modifiers

- Encapsulation provides an important attribute: **access control.**

- Using encapsulation we can control what parts of a program can access which members of a class.

- By controlling access, we can prevent misuse.

- There are 4 types of java access specifiers:
  - Public
  - Private
  - Protected
  - default (no specifier)

public

- Public access modifier achieves the highest level of accessibility.

- Classes, methods, and fields declared as public can be accessed from any class in the Java program, whether these classes are in the same package or in another package.

Example-1: use of ‘Public’

```
class Hello
{
    public int a=20;
    public void show()
            {
            System.out.println("Hello World");
            }
    }
class Demo
    {
    public static void main(String args[])
            {
            Hello h1=new Hello();
            System.out.println(h1.a);      // No error
            h1.show();                     //No error
    }
}
```

Output:

```
20
Hello World
```

Private

- Private access modifier achieves the lowest level of accessibility.
- Private methods and fields can only be accessed **within the same class** to which the methods and fields belong.

- Private methods and fields are
  - not visible within subclasses and are not inherited by subclasses.
- So, the private access modifier is opposite to the public access modifier.

- Using private modifier we can achieve encapsulation and hide data from the outside world.

- The private access modifier cannot be applied to class because if we do that it will not be available to Java compiler.
- However, inner classes can be private.

Example-2: use of ‘private’

```
class Hello{
    private int a=20;
    private void show()
            {
            System.out.println("Hello World");
            }
    }
class Demo{
    public static void main(String args[])
            {
            Hello h1=new Hello();

            System.out.println(h1.a);      //Compile Error, you can't access private data here
            h1.show();          //Compile Time Error, you can't access private methods here
            }
    }
```

protected

- protected members of the class are accessible
  - within the same class
  - another class of the same package
  - inherited class of another package

- The protected access modifier **cannot be applied to class and interfaces**.

Example-3: use of ‘protected’

```
// save A.java
package pack1;
public class A {
    protected void show() {
      System.out.println("Hello Java");
              }
    }
```

```
//save B.java
package pack2;
import pack1.*;
class B extends A {
    public static void main(String args[]){
              B b1 = new B();
              b1.show();
              }
    }
```

```
Output:          Hello Java
```

default (no modifier)

- When you don't set access modifier for the element, it will follow the default accessibility level.

- There is **no default modifier keyword**.

- Classes, variables, and methods can be default accessed.

- Using default modifier we can access class, method, or field which
  - belongs to same package, but not from outside this package.

Example-4: use of no Specifier

```
// save A.java
package pack1;
class A{
   void show(){
         System.out.println("Hello Java");
         }
}
//save B.java
package pack2;
import pack1.*;
class B{
   public static void main(String args[]){
         A a1 = new A();  //Compile Time Error, can't access outside the package
         a1.show();    //Compile Time Error, can't access outside the package
         }
   }
```

Access Modifier

Access Specifier (AS)	within class	within package	outside package by subclass only	outside package
Public (Universal AS)	Yes	Yes	Yes	Yes
Private (Class Level AS)	Yes	No	No	No
Protected (Derived Level AS)	Yes	Yes	Yes	No
Default (Package Level AS)	Yes	Yes	No	No

Command Line Arguments

- The command line argument is the argument passed to a program at the time when the program is run.
- To access the command-line argument inside a java program is quite easy, they are stored as string in String array passed to the args parameter of main() method.

Example:

```
class cmd {
        public static void main(String args[]){
                System.out.println(“No.of arguments:”+ args.length);
                for(int i=0;i< args.length;i++){
                        System.out.println(args[i]);
                        }
                }
        }
```

```
//  To run: java cmd 10 20 30
```

Output:

```
No. of arguments: 3
10
20
30
```