UNIT II

Classes and Objects: Introduction, Class Declaration and Modifiers, Class Members, Declaration of Class Objects, Assigning One Object to Another, Access Control for Class Members, Accessing Private Members of Class, Constructor Methods for Class, Overloaded Constructor Methods, Nested Classes, Final Class and Methods, Passing Arguments by Value and by Reference, Keyword this.

Methods: Introduction, Defining Methods, Overloaded Methods, Overloaded Constructor Methods, Class Objects as Parameters in Methods, Access Control, Recursive Methods, Nesting of Methods, Overriding Methods, Attributes Final and Static.

Class:

- A class is the blueprint from which individual objects are created.
- This means the properties and actions of the objects are written in the class.

Class Declaration Syntax:

```
class ClassName
{
   VariableDeclaration-1;
   VariableDeclaration-2;
   VariableDeclaration-3;
    ------------------------
    ------------------------
   VariableDeclaration-n;

   returnType methodname-1([parameters list])
  {
      body of the method…
  }
  returnType methodname-2([parameters list])
  {
      body of the method…
  }
    ------------------------
    ------------------------
  returnType methodname-3([parameters list])
 {
      body of the method…
 }
}// end of class
```

Object:

- Object is an entity of a class.

Object Creation Syntax:

ClassName objectName=new className ();

Class Members:

All the variable declared and method defined inside a class are called class members.

Instance variables: The variables defined within a class are called instance Variables (data members).

Methods: The block in which code is written is called method (member functions).

The Java programming language defines the following kinds of variables:

There are 4 types of java variables

- instance variables
- class variables
- local variables
- Parameters

Instance variables:

Instance variables are declared inside a class. Instance variables are created when the objects are instantiated (created). They take different values for each object.

Class variables:

Class variables are also declared inside a class. These are global to a class. So these are accessed by all the objects of the same class. Only one memory location is created for a class variable.

Local variables:

Variables which are declared and used with in a method are called local variables.

Parameters:

Variables that are declared in the method parenthesis are called parameters.

Class and Object Example:

```
class Test
{       int a; //default access

        void setData(int i)
        {
            a=i;
         }
        int dispData()
        {
```

```
        return a;
       }
}
class AccessTest
{
 public static void main(String args[])
 {
  Test ob=new Test(); //object creation
  ob.setData(100);
  System.out.println(" value of a is:-"+ob.dispData());
}
}
```

Access specifiers (Or) Access Control (Or) access Modifiers or Access Control for Class Members (Or) Accessing Private Members of Class:

An access specifier determines which feature of a class (class itself, data members, methods) may be used by another classes.

Java supports four access specifiers:

1. The public access specifier
2. The private access specifier
3. The protected access specifier
4. The Default access specifier

Public:

If the members of a class are declared as public then the members (variables/methods) are accessed by out side of the class.

Private:

If the members of a class are declared as private then only the methods of same class can access private members (variables/methods).

Protected:

Discussed later at the time of inheritance.

Default access:

If the access specifier is not specified, then the scope is friendly. A class, variable, or method that has friendly access is accessible to all the classes of a package (A package is collection of classes).

Example:

```
class Test
{
        int a; //default access
        public int b; // public access
        private int c; // private access
        //methods to access c
```

```
        void setData(int i)
        {
                                c=i;
        }
        int dispData()
        {
                                 return c;
        }
}
class AccessTest
{
        public static void main(String args[])
        {
                                Test ob=new Test();
                                //a and b can be accessed directly
                                ob.a=10;
                                ob.b=20;
                                // c can not be accessed directly because it is private
                                //ob.c=100; // error
                                // private data must be accessed with methods of the same class
                                ob.setData(100);
                                System.out.println(" value of a, b and c are:"+ob.a+" "+ob.b+"
"+ob.dispData());
                                ob.dispData();
}
}
```

Assigning One Object to Another or Cloning of objects:

We can copy the values of one object to another using many ways like :

1. Using clone() method of an object class.
2. Using constructor.
3. By assigning the values of one object to another.
4. in this example, we copy the values of object to another by assigning the values of one object to another.

Example:

```
class Copy
{
        int a=10;
}
class CopyObject
{
        public static void main(String args[])
        {
                Copy c1=new Copy();
                Copy c2=c1;
                System.out.println("object c1 value-"+c1.a);
                System.out.println("object c2 value-"+c2.a);
        }
```

}

Constructors:

1. JAVA provides a special method called constructor which enables an object to initialize itself when it is created.
2. Constructor name and class name should be same.
3. Constructor is called automatically when the object is created.
4. Person p1=new Person() → invokes the constructor Person() and Initializes the Person object p1.
5. Constructor does not return any return type (not even void).

There are two types of constructors.

Default constructor: A constructor which does not accept any parameters.

Parameterized constructor:A constructor that accepts arguments is called parameterized constructor.

Default constructor Example:

```
class Rectangle
{
        int length,bredth;  // Declaration of variables
        Rectangle() // Default Constructor method
        {
                System.out.println("Constructing Rectangle..");
                length=10;
                bredth=20;
        }
        int rectArea()
        {
                return(length*bredth);
        }
}
class Rect_Defa
{
        public static void main(String args[])
        {
                Rectangle r1=new Rectangle();
                System.out.println("Area of rectangle="+r1.rectArea());
        }
}
```

Note: If the default constructor is not explicitly defined, then system default constructor automatically initializes all instance variables to zero.

Parameterized constructor:

```
class Rectangle
{
        int length,bredth;  // Declaration of variables
        Rectangle(int x,int y) // Constructor method
        {
                length=x;
                bredth=y;
        }
        int rectArea()
        {
                return(length*bredth);
```

```
        }
}
class Rect_Para
{
        public static void main(String args[])
        {
                Rectangle r1=new Rectangle(5,10);
                System.out.println("Area of rectangle="+r1.rectArea());
        }
}
```

Overloaded Constructor Methods:

Writing more than one constructor with in a same class with different parameters is called constructor overloading.

Example:

```
class Addition
{
        int a,b;
        Addition()
        {
          a=10;
          b=20;
        }
        Addition(int a1,int b1)
        {
           a=a1;
           b=b1;
        }
        void add()
        {
             System.out.println("Addition of "+a+" and "+b+" is "+(a+b));

        }
}
class ConsoverLoading
{
```

```
    public static void main(String args[])
    {
                        Addition obj=new Addition();
                        obj.add(); //output:30
                        Addition obj2=new Addition(2,3);
                        obj2.add(); // output: 5
    }
}
```

Nested Classes: nested class is a class that is declared inside the class or interface. it can access all the members of the outer class, including private data members and methods.

Advantages:

1. Nested classes can access all the members (data members and methods) of the outer class, including private.
2. to develop more readable and maintainable code
3. Code Optimization

Syntax of Inner class

```
class Java_Outer_class
{
     //code
     class Java_Inner_class
     {
          //code
     }
}
```

Types of Nested classes

There are two types of nested classes non-static and static nested classes. The non-static nested classes are also known as inner classes.

- Non-static nested class (inner class)
    1. Member inner class
    2. Anonymous inner class
    3. Local inner class
- Static nested class

Example: local inner class

```
public class localInner1
{
    private int data=30;//instance variable
    void display()
    {
        class Local
        {
          void msg()
```

```
            {
                System.out.println(data);
            }
        }
        Local l=new Local();
        l.msg();
    }
    public static void main(String args[]){
    localInner1 obj=new localInner1();
    obj.display();
    }
}
```

Example: member inner class

```
class Outer
{
    int a=10;
    class Inner // member inner class
    {
        int b=20;
    }
}
class MemberInnerClass
{
    public static void main(String args[])
    {
        Outer m=new Outer();
        Outer.Inner n=m.new Inner();//syntax to create inner class object
        System.out.println(m.a+n.b);
    }
}
```

Example: member inner class

```
class Outer
{
    static int a=10;
    static void sample()
    {
        System.out.println("Hello I am method of outer class");
    }
    static class Inner//static Inner Class
    {
        int b=20;//instance variable
        void display()
```

```
		{
			sample();
			System.out.println("I am outer class varible:"+a);
		}
	}
}
class StaticInnerClass
{
	public static void main(String args[])
	{
		Outer.Inner m=new Outer.Inner();
		System.out.println("I am Inner class varible:"+m.b);
		m.display();
	}
}
```

Final Class and Methods:

Final Class: Classes declared as final cannot be inherited.

Syntax:

```
 final class A
{
 ----
 ----
 }
 class B extends A // cannot be inherited
{ ---
  ----
}
```

Example: (Error Will get while executing following program)

```
final class A
{
  int a=10;
}
class B extends A //can't inherit because class-A is defined as final class
{
  int b=20;
}
class Final_Class
{
  public static void main(String args[])
  {
	B obj=new B();
	System.out.println(obj.a+obj.b);
   }
}
```

Final Method:

- To prevent method riding final keyword is used.
- Methods declared as final cannot be overridden.

Example: (Attempt to override final methods leads to compile time error.)

```
class A
{
	int a=10;
	final void display()
	{
		System.out.println("I am display() of class-A");
		System.out.println("my value is:"+a);
	}
}
class B extends A
{
	int b=20;
	void display()   //can't be overriden
	{
		System.out.println("I am display() of class-B");
		System.out.println("my value is:"+b);
	}
}
class Final_Methods_Classes
{
	public static void main(String args[])
	{
	B b=new B();
	b.display();
	}
}
```

Passing Arguments by Value and by Reference

- There is only call by value in java, not call by reference but we can pass non-primitive datatype to function to see the changes done by callee function in caller function.
- If we call a method passing a value, it is known as call by value. The changes being done in the called method, is not affected in the calling method.

Example 1: Passing Primitive datatype to function

```
class Example
{
	int a=10;
	void change(int a) //called or callee function
	{
		a=a+100;
	}
}
class CallByValue
```

```
{
        public static void main(String args[]) //calling function
        {
                Example e=new Example();
                System.out.println("a value before calling change() :"+e.a); //10
                e.change(10); //call by value(passing primitive data type)
                System.out.println("a value after calling change() :"+e.a);  //10
        }
}
```

Example 1: Passing Primitive datatype to function

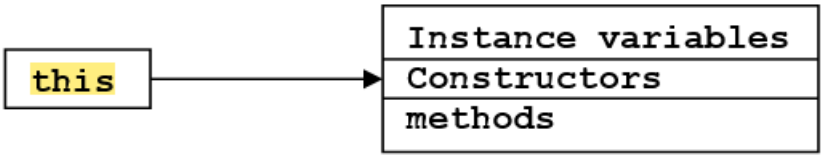
(Or)

Class Objects as Parameters in Methods:

```
class Example
{
        int a=10;
        void change(Example x) //called or callee function
        {
                x.a=x.a+100;
        }
}
class CallByValue
{
        public static void main(String args[]) //calling function
        {
                Example e=new Example();
                System.out.println("a value before calling change() :"+e.a); //10
                e.change(e); //call by value(passing non primitive data type) (or) Class Objects as Parameters in Methods
                System.out.println("a value after calling change() :"+e.a);  //110
        }
}
```

'this' keyword:

- 'this' is a keyword that referes to the object of the class where it is used.
- When an object is created to a class, a default reference is also created internally to the object.

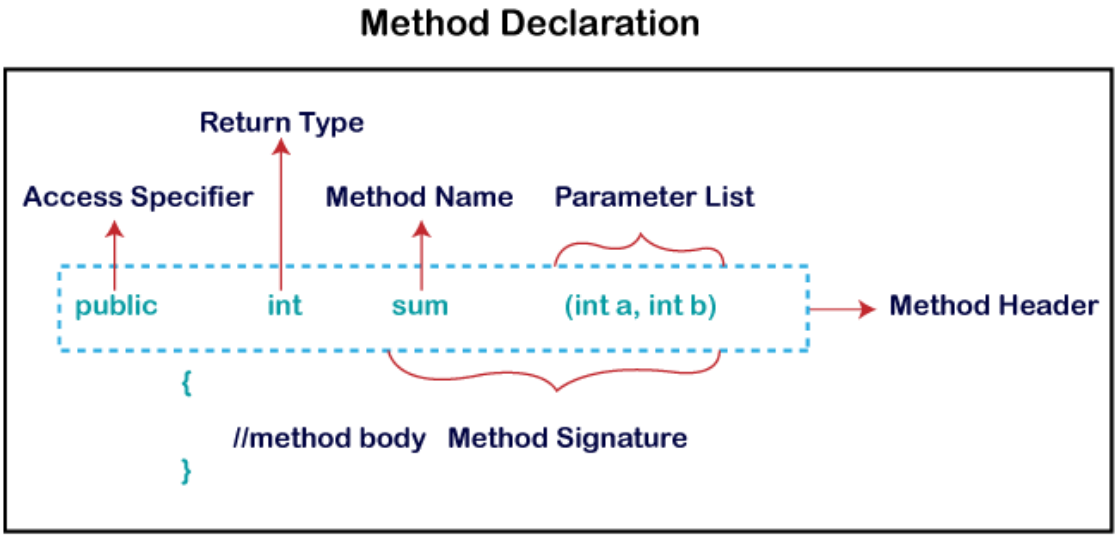


```
class Sample
{
  private int x;

  Sample()
   {
        this(10); // calls parameterized constructor and sends 10
        this.access(); //calls present class method
   }
  Sample(int x)
   {
        this.x=x; // referes present class reference variable
   }
  void access()
   {
        System.out.println("X ="+x);
   }
}
class ThisDemo
{
 public static void main(String[] args)
 {
         Sample s=new Sample();
 }
}
```

Methods in java:

- Method in Java is a collection of instructions that performs a specific task. It provides the reusability of code.
- Syntax of Defining a Method in java:



Types of Method

There are two types of methods in Java:

- Predefined Method
- User-defined Method

Predefined method:

In Java, predefined methods are the method that is already defined in the Java class libraries is known as predefined methods. It is also known as the standard library method or built-in method. We can directly use these methods just by calling them in the program at any point. Some pre-defined methods are length(), equals(), compareTo(), sqrt(), etc. When we call any of the predefined methods in our program, a series of codes related to the corresponding method runs in the background that is already stored in the library.

Each and every predefined method is defined inside a class. Such as print() method is defined in the java.io.PrintStream class. It prints the statement that we write inside the method. For example, print("Java"), it prints Java on the console.

Example:

```
public class Demo
{
	public static void main(String[] args)
	{
		// using the max() method of Math class
		System.out.print("The maximum number is: " + Math.max(9,7));
	}
}
```

User-defined Method:

The method written by the user or programmer is known as a user-defined method. These methods are modified according to the requirement.

Example:

```
import java.util.Scanner;
```

```
//user defined method
public static void findEvenOdd(int num)
{
	//method body
	if(num%2==0)
		System.out.println(num+" is even");
	else
	System.out.println(num+" is odd");
}
public class EvenOdd
{
	public static void main (String args[])
	{
		//creating Scanner class object
		Scanner scan=new Scanner(System.in);
		System.out.print("Enter the number: ");
		//reading value from the user
		int num=scan.nextInt();
		//method calling
		findEvenOdd(num);
	}
}
```

Overloaded Methods or Method Overloading:

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

Example:

```
class Method
{
	int add(int a,int b)
	{
		System.out.println("I am Integer method");
		return a+b;
	}
	float add(float a,float b,float c)
	{
		System.out.println("I am float method");
		return a+b+c;
	}
	int add(int a,int b,int c)
	{
		System.out.println("I am Integer method");
		return a+b+c;
	}
```

```
	float add(float a,float b)
	{
		System.out.println("I am float method");
		return a+b;
	}
}
class MethodOverLoad
{
	public static void main(String args[])
	{
		Method m=new Method();
		System.out.println(m.add(10,20));
		System.out.println(m.add(10.2f,20.4f,30.5f));
		System.out.println(m.add(10,20,30));
		System.out.println(m.add(10.2f,20.4f));

	}
}
```

Recursive Methods:

- A method in java that calls itself is called recursive method.
- It makes the code compact but complex to understand.

Syntax:

```
returntype methodname()
{
	//code to be executed
	methodname();//calling same method
}
```

Example:

```
public class Recursion
{
	static int count=0;
	static void p()
	{
		count++;
		if(count<=5)
		{
			System.out.println("hello "+count);
			p();
		}
	}
	public static void main(String[] args)
	{
		p();
	}
}
```

Nesting of Methods:

A method can be called by using only its name by another method of the same class that is called **Nesting of Methods**.

Syntax:

```
class Main
{
   method1()
  {

      // statements
   }

   method2()
   {
     // statements

    // calling method1() from method2()
     method1();
   }
   method3()
   {
    // statements

    // calling of method2() from method3()
    method2();
   }
}
```

Example:

```
public class NestingMethod
{
   public void a1()
  {
       System.out.println("****** Inside a1 method ******");
      // calling method a2() from a1() with parameters a
      a2();
   }
   public void a2()
   {
     System.out.println("****** Inside a2 method ******");
   }
   public void a3()
   {
     System.out.println("****** Inside a1 method ******");
      a1();
   }
   public static void main(String[] args)
   {
      // creating the object of class
```

```
        NestingMethod n=new NestingMethod();

        // calling method a3() from main() method
        n.a3(a, b);
    }
}
```

Overriding Method: If subclass has the same method as declared in the parent class, it is known as method overriding.

Uses: runtime polymorphism

Rules:

- There must be a IS-A relationship(inheritance).
- The method must have same method signature as in the parent class.

Example:

```
class SuperClass
{
        void calculate(double x)
        {
                        System.out.println("Square value of X is: "+(x*x));
        }
}
class SubClass extends SuperClass
{
        void calculate(double x)
        {
                        super();
                        System.out.println("Square root of X is: "+Math.sqrt(x));
        }
}
class MethodOver
{
        public static void main(String args[])
        {
                        SubClass s=new SubClass();
                        s.calculate(2.5);
        }
}
```

Note: when a super class method is overridden by the sub class method calls only the sub class method and never calls the super class method. We can say that sub class method is replacing super class method.