# Swami Sahajanand College of Computer Science

## B.C.A. SEM-V[NEP]

## Subject: CORE JAVA
## Major (Core) - 25617

# UNIT 2

**JAVA PROGRAMMING**

- Classes, Objects and Methods.
- Polymorphism: Method Overloading.
- Constructor: Concept of Constructor, Types of Constructor, Constructor Overloading.
- Garbage Collection
- The 'this' keyword. 'static' and 'final' keyword.
- Access Control: Public, Private, Protected, Default.

## Classes, Objects and Methods

**Classes:**

***What is Class?***

- A class, in the context of Java, is templates that are used to create objects, and to define object data types and methods.
- Core properties include the data types and methods that may be used by the object. All class objects should have the basic class properties.
- Classes are categories, and objects are items within each category.
- Class can contain any of the following variable types.
  - **Local variables:** Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
  - **Instance variables**: Instance variables are variables within a class but outside any method. These variables are instantiated when the class is loaded. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.
  - **Class variables**: Class variables are variables declared with in a class, outside any method, with the static keyword.

The General Form of a Class:

```
class clsName
{
        // instance variable declaration
        type1 varName1 = value1;
        type2 varName2 = value2;
        :
        :
        typeN varNameN = valueN;
        // Methods
        Type1 methodName1(mParams1)
        {
        // body of method
        }
       :
       :
        TypeN methodNameN(mParamsN)
        {
        // body of method
        }
}
```

Example of a General Class

```
class clsSample
{
// Variables
        static int ctr;
        int i;
        int j;
```

```
        // Methods
        int addition()
        {
                ctr++;
                return i + j;
        }
    int NumberOfInstances()
    {
        return ctr;
    }
  }
```

**Objects:**

**What is an Object?**

- An Entity that has state and behavior is known as an object.
- For example, chair, bike, marker, pen, table, car etc.
- It can be physical or logical.
- The example of logical object is banking system.
- Object is an instance of a class. Class is a template or blueprint from which objects are created.
- An object has three characteristics:
  - **State:** represent data of an object. For example name of an object, colour etc.
  - **Behavior:** represent the behavior (functionality) of an object such as deposit, withdrawal etc..
  - **Identity:** object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. But, it is used internally by the JVM to identify each object uniquely.

**Declaring Objects:**

- You can create a class, you are creating a new data type. You can use this type of declare objects of that type.
- Obtaining of a class is a two step process.
- First you must declare a variable of the class type. Second you must acquire an actual, physical copy of the object and assign it to that variable.
- You can do this using the new operator. The new operator dynamically allocates memory for an object and returns a reference to it.
- Syntax to declare an object :
  Classname objectvariable = new classname ();

**Example :**

```
class objdecl
{
    class box
    {
        Int width, height, depth;
    }
    public static void main (String args[])
    {
```

```
        box  b = new box();
    }
}
```

## Polymorphism: Method Overloading

**Explain method overloading.**

- "Method name is same but method signature is different is known as method overloading." Another powerful object-oriented technique is method overloading.
- Method overloading enables you to specify different types of information (parameters) to send to a method.
- To overload a method, you declare another version with the same name but different parameters.
- The compiler keeps up with the parameters for each method along with the name.
- When a call to a method is encountered in a program, the compiler checks the name and the parameters to determine which overloaded method is being called.

Ex :-

```
class OverloadDemo
{
        // Overload test for no parameters. 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) a: " + a);
        }
        int test(int x,int y,int z)
        {
                return x*y*z;
        }

}

class MethodOverloading
{
        public static void main(String args[])
        {
                OverloadDemoob = new OverloadDemo(); ob.test();
                ob.test(10, 20);
                ob.test(20.5); // this will invoke test(double) int
                n;
```

```
            n=ob.test(5,10,15);
            System.out.println("multiplication="+n);
        }
    }
```

**Output :**

```
No parameters
a and b: 10 20
Inside test(double) a: 20.5 multiplication=750
```

## Constructor: Concept of Constructor, Types of Constructor, Constructor Overloading.

### What is Constructor? Explain its characteristics and all its types and also define about Constructor Overloading.

**"A constructor is used to initialize the objects of its class."**

- When you create an object, you typically want to initialize its member variables. It is special because its name is the same as the class name.
- The constructor is invoked automatically whenever an object of its associated class is created.
- It is called constructor because it construct the values of data member of the class.
- The constructor is a special method you can implement in all your classes; it allows you to initialize variables and perform any other operations when an object is created from the class.

**Characteristics of Constructor:**

- The constructor has always the same name as the class
- If we don't define any constructor than compiler automatically define it. There is no return type define in the constructor.
- There is no return statement require in the body of the constructor.
- We can use "this" to call another constructor in the same class. We can use "super" to call a constructor in a parent class.

**Types of Constructor:**

- There are three types of constructors
  - (1) Default
  - (2) Parameterized
  - (3) Copy
- The constructor which has no argument is called default constructor.
- The constructor which has arguments is called parameterized constructor. The constructor which has object as an argument is called copy constructor.

**Constructor overloading:**

- "When we declare more than one constructor in a class is known as constructor overloading or multiple constructor."

Ex :-

**// Multiple Constructor or constructor overloading**

```
class Box
{
    double width; double height; double depth;
    // construct clone of an object      //copy construct Box(Box ob)
    {
        // pass object to constructor width = ob.width;
```

```
                height = ob.height; depth = ob.depth;
        }
        // parameterized constructor Box(double w, double h, double d)
        {
                width = w; height = h; depth = d;
        }
        // default constructor Box()
        {
                width = -1; // use -1 to indicate height = -1; // an uninitialized depth = -1; // box
        }
        // parameterized constructor to assign same value to all the variables Box(double len)
        {
                width = height = depth = len;
        }
        // compute and return volume double volume()
        {
                return width * height * depth;
        }
}
class ConstructorOver
{
public static void main(String args[])
{
        // create boxes using the various constructors
        Box mybox1 = new Box(10, 20, 15); // 3 parameter constructor call Box mybox2 = new Box(); //default
        constructor call
        Box mycube = new Box(7); //1 parameterized constructor call
        Box myclone = new Box(mybox1);// object as parameter constructor call doublevol;

        // get volume of first box vol = mybox1.volume();
        System.out.println("Volume of mybox1 is " + vol);//3000.0

        // get volume of second box vol = mybox2.volume();
        System.out.println("Volume of mybox2 is " + vol);//-1.0

        //get volume of cube vol = mycube.volume();
        System.out.println("Volume of mycube is " + vol);//343.0

        //get volume of clone vol = myclone.volume();
        System.out.println("Volume of myclone is " + vol);//3000.0
}
}
```

**Output:**

```
Volume of mybox1 is 3000.0
Volume of mybox2 is -1.0
Volume of mycube is 343.0
Volume of myclone is 3000.0
```

## Garbage Collection, Finalized () Method.

**How can we destroy existing object in the Java? OR what is Garbage Collection? Can we use finalize method for it?**

- The mechanism to release memory automatically which is occupied by an object is known as Garbage Collection (GC).
- Java takes a different approach; it handles deallocation for you automatically. The technique that accomplishes this is called garbage collection.
- It works like this: when no references to an object exist, that object is assumed to be no longer needed, and the memory occupied by the object can be reclaimed.
- Java allows you to create as many objects as you want and you never have to worry about destroying them.
- You can ask the garbage collector to run at any time by calling system's gc method : System.gc( );

**finalize( ) method:**

- The mechanism to destroy reference file or external resources which are used in your program, is known as finalize( ) method.
- Sometimes an object will need to perform some action when it is destroyed.
- For example, if an object is holding some non-Java resource such as a file handle or window character font, then you might want to make sure these resources are freed before an object is destroyed.
- To handle such situations, Java provides a mechanism called finalization.
- By using finalization, you can define specific actions that will occur when an object is just about to be reclaimed by the garbage collector.
- To add a finalizer to a class, you simply define the finalize( ) method.
- Inside the finalize( ) method you will specify those actions that must be performed before an object is destroyed(before gc called).
- The finalize( ) method has this general form:

```
protected void finalize( )
{
// finalization code here
}
```

- Here, the keyword protected is a specifier that prevents access to finalize( ) by code defined outside its class.
- It is important to understand that finalize ( ) is only called just prior to garbage collection. For example,

## The "this" keyword.

- Describe about this keyword in detail. "this is refers current class object."
- "this can be used inside any method to refer to the current object." Sometimes a method will need to refer to the object that invoked it. To allow this, Java defines the this keyword.
- this is always a reference to the object on which the method was invoked.
- Ex :-

```
class Box
{
        double height,width,depth;
        Box(double height, double width, double depth)
```

```
                {
                this.height=height; this.width=width; this.depth=depth;
                }
        double volume()
        {
                return height*width*depth;
        }

}
class ThisDemo
{
        public static void main(String[] args)
        {
                Box b1=new Box(10,20,15); double vol; vol=b1.volume();
                System.out.println("volume is : "+vol);
        }
}
```

**Output**: **volume is : 3000.0**

## 'static' and 'final' keyword.

**Enlist the use of "static" keyword.**

- The static modifier specifies that a variable or method is the same for all objects of a particular class is known as static.
- When we want to maintain values common to the entire class at that time we used static data member.
- Typically, new variables are allocated for each instance of a class.
- When a variable is declared as being static, it is allocated only once regardless of how many objects are created.
- The result is that all instantiated objects share the same instance of the static variable. The static methods have access only to static variables.
- Following example of a static : static int refCount;

For example :

```
class student
{
        String name; int age; String id;
        static int noofst=0; student(String n, int a, String sid)
        {
                name=n; age=a; id=sid; noofst++;
        }
}
class staticprogram
{
        public static void main(String args[])
        {
                System.out.println("No of student at beginning is "+student.noofst); student s1=new
                student("xyz",100,"20111");
                student s2=new student("pqr",101,"36699"); intnum=student.noofst;
```

```
                System.out.println("No of student (Using class) is "+num); num=s1.noofst;
                System.out.println("No of student(Using first object) is "+num); num=s2.noofst;
                System.out.println("No of student(Using second object) is "+num);
        }
}
```

**Output :**

```
No of student at beginning is 0
No of student (Using class)is 2
No of student(Using first object) is 2 No
of student(Using second object) is 2
```

**Enlist the use of "final" keyword.**

- Final keyword is used in different contexts. First of all, final is a non-access modifier applicable only to a variable, a method or a class.
- **Final variables**
- When a variable is declared with final keyword, its value can't be modified. This also means that you must initialize a final variable.
- You can add or remove elements from final array or final collection.
- It is good practice to represent final variables in all uppercase, using underscore to separate words.

**Examples :**

```
// a final variable

final int THRESHOLD = 5;

// a blank final variable

final int THRESHOLD;


// a final static variable PI

static final double PI = 3.141592653589793;

// a blank final static variable

static final double PI;
```

**Initializing a final variable :**

- We must initialize a final variable, otherwise compiler will throw compile-time error.A final variable can only be initialized once, either via an initializer or an assignment statement. There are three ways to initialize a final variable :
  1. You can initialize a final variable when it is declared. This approach is the most common. A final variable is called blank final variable, if it is not initialized while declaration. Below are the two ways to initialize a blank final variable.
  2. A blank final variable can be initialized inside instance-initializer block or inside constructor. If you have more than one constructor in your class then it must be initialized in all of them, otherwise compile time error will be thrown.
  3. A blank final static variable can be initialized inside static block.

## Final methods

- When a method is declared with final keyword, it is called a final method. A final method cannot be overridden.
- We must declare methods with final keyword for which we required to follow the same implementation throughout all the derived classes.
- The following example explain final keyword with a method:

```
class A
        {
          final void m1()
          {
             System.out.println("This is a final method.");
          }
        }
        class B extends A
        {
          void m1()
          {
             // COMPILE-ERROR! Can't override.
             System.out.println("Illegal!");
          }
        }
```

## Final classes

- When a class is declared with final keyword, it is called a final class.
- A final class cannot be extended(inherited). There are two uses of a final class :
- One is definitely to prevent inheritance, as final classes cannot be extended. For example :

```
final class A

{

   // methods and fields

}

// The following class is
illegal. class B extends A
{

  // COMPILE-ERROR! Can't subclass A

}
```

- The other use of final with classes is to create an immutable class like the predefined String class.You can not make a class immutable without making it final.

## Access Control: Public, Private, Protected and Default.

**Explain access control used in java.**

- Access Control (or access specifiers or modifiers) are keywords in java that set the accessibility of classes, methods, and other members.
- Access Controls are a specific part of programming language used to facilitate the encapsulation of components.

### Types of access modifiers:

#### Private

- Methods, variables, and constructors that are declared private can only be accessed within the declared class itself.
- Private access modifier is the most restrictive access level. Class and interfaces cannot be private.
- Variables that are declared private can be accessed outside the class, if public getter methods are present in the class.
- Using the private modifier is the main way that an object encapsulates itself and hides data from the outside world.
  - Example

```
class A
{
  private int data=40;
  private void msg()
  {
        System.out.println("Hello java");
  }
}
public class Simple
{
  public static void main(String args[])
  {
        A obj=new A(); System.out.println(obj.data);//Compile
        Time Error obj.msg();//Compile Time Error
  }
}
```

**Default**

- If you don't use any modifier, it is treated as default bydefault. The default modifier is accessible only within package.
  - Example:

```
//save by A.java
package pack;
class A
{
  void msg(){System.out.println("Hello");}
}

//save by B.java
package mypack;
import pack.*;
class B
{
        public static void main(String args[]){  A
         obj = new A();//Compile Time Error
         obj.msg();//Compile Time Error
 }
}
```

**Protected**

- Variables, methods, and constructors, which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class.
- The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared protected, however methods and fields in a interface cannot be declared protected.
- Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it.
- Example:

```
//save by A.java
package pack;
public class A
{
  protected void msg(){System.out.println("Hello");}
}

//save by B.java
package mypack;
import pack.*;
class B extends A
{
  public static void main(String args[])
  {
        B obj = new B();
        obj.msg();
  }
}
```

**Public**

- A class, method, constructor, interface, etc. declared public can be accessed from any other class.
- Therefore, fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe.
- Example

```
//save by A.java
package pack;
public class A
{
  public void msg(){System.out.println("Hello");}
}

//save by B.java
package mypack;
import pack.*;
class B
{
   public static void main(String args[])
  {
            A obj = new A();
            obj.msg();
   }
}
```