Unit –2: Object Oriented Programming Concepts

1. Procedure-Oriented vs. Object Oriented Programming concept

No	Procedure oriented programming	Object oriented programming
1	Program is divided into small parts called functions.	Program is divided into parts called objects.
2	Importance is not given to data but to functions as well as sequence ofactions to be done.	Importance is given to the data rather than procedures or functions becauseit works as a real world.
3	POP follows Top Down approach.	OOP follows Bottom Up approach.
4	POP does not have any access specified.	OOP has access specified name Public, Private, Protected
5	In POP, Data can move freely fromfunction to function in the system.	In OOP, objects can move and communicate with each other throughmember functions.
6	To add new data and function in POP is not so easy.	OOP provides an easy way to add new data and function.
7	In POP, Most function uses Global data for sharing that can be accessed freely from function to function in the system.	In OOP, data cannot move easily from function to function, it can be kept public or private so we can control the access of data.
8	POP does not have any proper way for hiding data so it is less secure.	OOP provides Data Hiding so provides more security.
9	In POP, Overloading is not possible.	Overloading is possible in the form of Function Overloading and Operator Overloading.
10	Examples of POP are:C,VB,FORTRAN, and Pascal.	Example of OOP are : C++, JAVA, VB.NET, C#.NET.

2. Basics of OOP: Class, Object, Encapsulation, Polymorphism Abstraction, Inheritance

BASIC CONCEPT OF OBJECT ORIENTED PROGRAMMING

1) Object

- Objects are basic runtime entities in object oriented system. Object may represent a person, place, chair, or any item. Object can interact without having to know the details of each other's data or code. Object is a basically a set of data and code to′ perform operation on data.

2) CLASS

- Class is prototype that defines the variables and the methods common to all object of similar type. Classes are user defined data types and behave′ like built in type of programming language. Simply, Class is a collection of logically related′ data items which includes data (variable) and function (methods) for data.

3) Data abstraction and Encapsulation:

- Data abstraction refers to act of representing′ essential features without including background details or explanation. Data encapsulation means wrapping up of data′ and methods into a single unit. The data is not accessible to the outside and only′ these methods, which are wrapped in class access it this insulation of the data from direct access by program is called data hiding.

4) Inheritance

- Inheritance is a process by which object of one′ class use the properties of object of another class. It supports hierarchical classification. It provides Reusability of code. We can add an′ additional feature to an existing class without modifying it .This is possible by deriving a new class from the existing one. New class with additional features can be created,′ that new class with have combined features of both the new classes.

5) Polymorphism

- Polymorphism means the ability to take more than one form. Simply, it means that are the same operation have may behave different on different classes.

6) Dynamic Binding

- Binding refers to the linking of procedure call to the code to be executed in response to call. Dynamic binding means code associated with a given procedure call is not know until the time or call at runtime. It is associated with polymorphism and inheritance.

3. Defining classes, fields and methods, creating objects

Class

- Java is true object oriented language.
- Anything in java program must be encapsulated in class.
- Class defines state and behavior of objects.
- Class create objects and objects use method to communicate between them.
- Class provide convenient method for packing together a group of logically related data items and functions that work on them.
- Class is user defined data type with a template that serves to define its properties.
- Once class has been defined we can create variables of that type ,these variables are termed as instance of classes they are actual objects.
- Everything inside square brackets is optional
- Syntax: class classname[extends superclass name]
  {
  [variable declaration;] [methods declaration;]
  }
- Any class at least have, If class does not contain any properties and therefore can't do anything.but can compiled and even create object using it.

- Classname and super classname are any valid java identifiers.
- Keyword extends indicates ,the properties of super classname are extended to classname class.
- Example

```
class Rectangle
{
  int length; int width; // Instance variable Or Member variable
}
```

- Adding methods to class
- Class with only data fields (variables) or without method (that operates on data) has no life.
- Objects of that class can not respond to any message.
- Methods declared inside the body of class immediately after declaration of instance variable.
- Example

```
class Rectangle
{
 int length; //variable
int width; //variable
void getData ( int x , int y) //method definition
{
length=x;
width=y;
}
int rectArea ( ) //method definition
{
int area= length * width ;
return (area);
}
```

Creating objects

- Object is a essential block of memory that contains space to store all instance variable.
- Creating object is referred as instantiating an object.
- new operator creates objects in java. it creates object of specified class and returns reference to that object.
- Declaration:
  classname objectname;
- Instantiate:
  classname objectname = new classname( );
- Example
  Rectangle rect = new Rectangle( );

4. Accessing rules : public, private, protected, default

- There are four types of Java access modifiers:
  1. Private: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
  2. Default: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
  3. Protected: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.

4. Public: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

5. this keyword, static keyword, final keyword

this keyword

- this hides Instance variable.
- It is used in method body to refer to the members(variables) of current object. Current object is a object whose method is being called.
- this keyword used in constructor.
- this is used to make different variable name if one of argument / parameter has same name as other method. Circle (int x ,int y, int z) Circle (int x, int y)
- Method argument can have same name as one of class's member variable. • Typically with an method body you can just refer to directly to the member variable.
- Example

```
class Circle
{
 int x , y ;
public Circle (int x ,int y, int radius )
{
   this. x =x;
   this. y=y;
   this. radius=radius;
 }
}
class Circle_this
{
public static void main(String args[])
 {
Circle c1=new Circle( 50,50,25);
Circle c2=new Circle( 10,10,5);
System.out. println(c1.x + " " + c1.y +" "+ c1.radius);
System.out. println(c2.x + " " + c2.y +" "+ c2.radius);
}

}
```

Static keyword

- The static keyword in Java is used for memory management mainly. We can apply static keyword with variables, methods, blocks and nested classes. The static keyword belongs to the class than an instance of the class.
- The static can be:

1. Variable (also known as a class variable)
2. Method (also known as a class method)

3. Block
4. Nested class

1) Java static variable

- If we declare any variable as static, it is known as a static variable.
- The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
- The static variable gets memory only once in the class area at the time of class loading.
- Example

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

2) Java static method

- If we apply static keyword with any method, it is known as static method.
- A static method belongs to the class rather than the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data member and can change the value of it.
- Example

```
static void change()
{
    college = "VPMP";
  }
```

3) Java static block

- Is used to initialize the static data member.
- It is executed before the main method at the time of classloading.
- Example of static block

```
class A2
{
  static{System.out.println("static block is invoked");
}
  public static void main(String args[])
{
   System.out.println("Hello main");
  }
}
```

Final keyword

- final is a keyword.
- final means can not be changed or it is final.
- It is same as const in c++.
- It is used for efficiency.

- final can be used with

1. Variable (data)
2. Methods
3. Classes

1. final variables

- final specifies that the value of variable is final and must not be changed.
- final variable name must be in capital letters.
- All final variable must have an initial value.(it can't specify later).
- Ex: final int SIZE=100;
  final float PI= 3.14f;
- final variable behaves like class variable.
- They do not take any space on individual object.

2. final method

- All methods and variable can be overridden by default in subclasses.
- We can prevent overriding of methods of super class by declared as final.
- If we declare super class's method as final then it can not be override by its subclass.

```
class A
{
final void display( )
{
---------
}
}
class B extends A
{
void display() //invalid
{
--------
}
}
```

3. final class

- If class is declared with final modifier ,then class can not be extended and this will ensure that the exact properties and behavior of class.
- final class cannot be sub classed.
- If we declare class as final ,it prevents any unwanted extension to the class.
- It allows compiler to perform some optimization when method of class is invoked.

- Many classes of java .lang are declared as final.
- Methods of non abstract class can be declared as final.
- Example

```
final class A
{
//body
}
class B extends A //invalid ,it gives Error
{
//body
}
```

6. Constructors: Default constructors, Parameterized constructors, Copy constructors, Passing object as a parameter

Constructor

- Constructor enables an object to initialize itself when it is created. Characteristics :
- Constructor have the same name as the class name itself.
- Constructor do not specify a return type , not even void(because they return the instance of the class itself).
- Constructor can not be inherited , through a derived class can call the base class constructor.
- Constructor is a special type of method.
- Mainly two types of constructor
  1. Default constructor
  2. Parameterized constructor

1. Default constructor

- A constructor is called "Default Constructor" when it doesn't have any parameter.
- Default constructor automatically initialize instance variable to zero.
- Example

```
class Bike1
{
//creating a default constructor
Bike1()
{
System.out.println("Bike is created");
}
public static void main(String args[])
{
Bike1 b=new Bike1();
}
}
```

2. Parameterized constructor

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

```
class Rectangle
 {
int length, width;
Rectangle( int x , int y) //Parameter constructor
{
 length=x;
width=y;
 }
 int rectArea( )
{
 return ( length * width );
 }
 }
class ConstDemo1
{
 public static void main( String args[ ])
{
Rectangle rect1=new Rectangle ( 15 ,10 ); // calling constructor
int area1=rect1.rectArea( ) ;
 System . out .println( " Area= " + area1) ;
}
 }
```

Copy constructor

- Copy constructor in Java allows creating an exact copy on an existing object of the same class such that changes made on the existing object don't affect the new copy.
- To create a copy constructor, we need to take the existing object as an argument and initialize the values of instance variables with the values obtained in the object.
- We can assign a value to the final field using the copy constructor, but the same cannot be done using the clone() method.
- Example
- class Complex
- {

```
    private double re, im;
    public Complex(double re, double im)     // Parameterized constructor
    {

       this.re = re;            // this keyword refers to current instance itself
       this.im = im;
    }
```

```
    Complex(Complex c)              // Copy constructor
    {
       System.out.println("Copy constructor called");
       re = c.re;
       im = c.im;
    }

      public String toString()
    {

       return "(" + re + " + " + im + "i)";
    }
}
public class Main
{
    public static void main(String[] args)
    {

       Complex c1 = new Complex(10, 15);
       Complex c2 = new Complex(c1);
        Complex c3 = c2;
              System.out.println(c2);
    }
}
```

Passing object as a parameter

- A method can take an objects as a parameter. For example, in the following program, the method **setData( )** takes three parameter.
- The first parameter is an **Data** object. If you pass an object as an argument to a method, the mechanism that applies is called **pass-by-reference**, because a copy of the reference contained in the variable is transferred to the method, not a copy of the object itself.
- Example

```
class Data
{
    int data1;
   int data2;
}

class SetData
{
   void setData(Data da,int d1,int d2)
   {
      da.data1 = d1;
      da.data2 = d2;
   }
     void getData(Data da)
```

```
    {
        System.out.println("data1 : "+da.data1);
        System.out.println("data2 : "+da.data2);
    }
}
public class Javaapp
{
        public static void main(String[] args)
{
          Data da = new Data();
        SetData sd = new SetData();
        sd.setData(da,50,100);
        sd.getData(da);
    }
}
```

7. method overloading, constructor overloading

1. Method overloading

- In java ,it is possible to create methods that have same name but different definitions . this is called method overloading.
- Definition : multiple methods of same name with different arguments in a single class is called Method overloading.
- It is used when objects are required to perform similar tasks but using different input parameters.
- When we call up a method in object ,java matches up method name first then the number and type of parameters to decide which one of definitions to execute this process is polymorphism.
- In definition all method should have same name but different parameter lists (number of parameter or type of parameters). In java ,we can overload method but not variable or operator.
- Java uses type and number of argument to call a method at runtime.
- Example

```
class Demo
 {
  void display( )
{
        System .out .println( " method overloaded");
}
void display (int x)
{
     System .out .println( " x=" + x);
}
}
class Overload
{
  public static void main(String args[ ] )
{
     Demo obj= new Demo( );
```

```
        obj.display( ); //method without parameter
        obj.display(10); //method with parameter
    }
   }
```

2. **Constructor overloading**

- When multiple constructor of same name with different arguments in a single class is called Constructor overloading.
- Example

```
class A
{
A( ) // definition of default constructor
{
System.out.println( " Constructor demo " );
}
 A( int x) // definition of parameterized constructor
{
System.out.println (" value of x=" + x);
 }
 }
class ConstructorDemo
{
 public static void main( String args[ ])
{
A obj1=new A( ); // calling default constructor
 A obj2=new A(5); // calling parameterized constructor
}
}
```

Output: Constructor Demo Value of x=5

8. Wrapper Classes, String Class and its methods: chatAt(), contains(), format(), length(), split()

1. Wrapper Classes

- Vector class cannot handle primitive data types like int,′ float, long, char and double. Use:
- Primitive data types may be converted into objects types by using wrapper classes contained in java.lang;
- Wrapper classes for converting simple types.

No.	Primitive data type	Wrapper class
1	boolean	Boolean
2	char	Character
3	double	Double

4	float	Float
5	int	Integer
6	long	Long

- Wrapper classes have number of unique methods for handling primitive data type. Converting primitive numbers to object number Using constructor methods :

1	**Integer** intval= new **Integer**(i);	Primitive integer to Integerobject
2	**Float** floatval=new **Float**(f);	Primitive float to Float object
3	**Double** doubleval=new **Double**(d);	Primitive double to Double object
4	**Long** longVal=new **Long**(l);	Primitive long to Long object

- CONVERTING OBJECT NUMBERS TO PRIMITIVE NUMBER USING TYPEVALUE ( ) :

1	int i =intval.**intValue( );**	Object to Primitiveinteger number
2	float f = floatval.floatValue( );	Object to Primitive float number
3	double d= doubleval.doubleValue( );	Object to Primitivedouble number
4	Longl = longVal.longValue( );	Object to Primitive long number

- CONVERTING NUMERIC STRING TO PRIMITIVE NUMBER USING PARSING METHODS:

1	int i=Integer.parseInt (str)	Convert string to primitive integers
2	long l=Long.parseLong (str);	Convert string to primitive long

2. String Classes

- String represents a sequence of characters.
- The easiest way to represent a sequence of character in java is character array.
- E.g. : char charArray[ ]= new char[4];
  charArray[0]= 'J' ,charArray[1]='a' ,
  charArray[2]='v' ,charArray[3]='a'
- character array have the ability to query their length′ ,but they are not good enough to support the range of operations we want to perform on strings.(like to copy one character array into another array may require a lot of efforts)
- Java can handle these problem more efficiently using string.
- string is more reliable and predictable .
- No problem of bound checking same as in C.
- Java string is not character array and is not′ NULL terminated.
- STRING DECLARATION
- We can declare and create string as follows:

1) String stringname; stringname =new String("string");

E.g. : String str ; str= new String (“ VPMP”);

2) String stringname=new String(“string”);

E.g. : String str= new String (“ VPMP”);

String methods: chatAt(), contains(), format(), length(), split()

1). chatAt()

- The Java String class charAt() method returns a char value at the given index number.
- The index number starts from 0 and goes to n-1, where n is the length of the string. It returns StringIndexOutOfBoundsException, if the given index number is greater than or equal to this string length or a negative number.
- Example

```
String name="java";
char ch=name.charAt(4);        //returns the char value at the 4th index
```

2). contains()

- The **Java String class contains()** method searches the sequence of characters in this string. It returns *true* if the sequence of char values is found in this string otherwise returns *false*.
- Example

```
String name="VPMP Polytechnic";
System.out.println(name.contains(“vpmp”));
```

3). format()

- The java string format() method returns the formatted string by given locale, format and arguments.
- The format() method of java language is like sprintf() function in c language and printf() method of java language.
- Example

```
return new Formatter().format(format, args).toString();
```

4). length()

- The Java String class length() method finds the length of a string. The length of the Java string is the same as the Unicode code units of the string.
- Example

```
String str = "Welcome To Java";
int size = str.length();
```

5). split()

- The java string split() method splits this string against given regular expression and returns a char array.
- Example:

```
String text = "Java is a  programming language";

        // split string from space

String[] result = text.split(" ");
```

9. User Input: Scanner class and Command Line Arguments

1. User Input: Scanner class

- The Scanner class is used to get user input, and it is found in the java.util package.
- To use the Scanner class, create an object of the class and use any of the available methods found in the Scanner class documentation. In our example, we will use the nextLine() method, which is used to read Strings:
- Example

```
import java.util.Scanner;  // Import the Scanner class
class Example
{
  public static void main(String[] args)
{
    Scanner myObj = new Scanner(System.in);   // Create a Scanner object
    System.out.println("Enter username");

    String userName = myObj.nextLine();     // Read user input
    System.out.println("Username is: " + userName);   // Output user input
  }
}
```

2. Command Line Arguments

- If we want, our program to act in a particular way′ depending on the input provided at the time of execution. This is done in java programs by using command line′ arguments. Definition: command line arguments are′ parameters that are supplied to the application program at the time of on invoking it for execution. We can write java program that can receive and use′ the arguments provided in the command line. public static void main (String args[ ])
- args is declared as an array of strings(known as string object) . Any arguments provided in the command line are′ passed to the array args as its elements. We can simply access the array elements and use′ them in the program as we want. Command line arguments are passed when we run′ the program To compile: javac Test.java To run : java Test Basic C++ C Java This command line contains four arguments .These′ arguments are assigned to the array args as follows:

  Basic → args[0]
  C++ → args[1]
  C → args[2]
  Java → args[3]

- Example:

```
class Cmd1
 {
        public static void main (String args[ ])
        {
                int count=0;
                String str; count=args.length; //to find how many arguments are passed to command line
                 System.out.prinltn ( “total no. of arguments passed = “ + count );
```

```
            while ( i < count )
            {
                  str= args[ i ];
                  i=i+1;
                  System.out.println ( i + " : " + "Java is " + str + " !!!" );
            }
      }
}
```

- E: \java program \ javac Cmd1.java
- E: \java program \ java Cmd1 simple robust secure portable dynamic object_oriented
- Output : total no. Of arguments passed= 6

  1 : Java is simple !!!
  2 : Java is robust !!!
  3 : Java is secure !!!
  4 : Java is portable !!!
  5 : Java is dynamic !!!
  6: Java is object_oriented !!!