Object Oriented Programming in Java

Laboratory Work
submitted to
Department of Computer Science and Engineering

By
SAI KIRAN NARAGAM
ID No: **N100638**
Class: **E4CSE-01**

Rajiv Gandhi University of Knowledge Technologies, Nuzvid,
Krishna District,
Andhra Pradesh.

Contents

Chapter 1

Introduction [13-08-2015]

1.1 Procedural vs. Object Oriented Programming Paradigm

We use programming languages for computation which most of the time involves manipulation of representation of `data`.It is an important facility to have in programming: convenient data manipulation.

In procedural paradigm we use `procedures` to manipulate. It requires sending data to procedures, as we do in The C Programming Language. Here data may be global to all procedures or local to a single procedure, where global data can be accessed/modified by any procedure and local data can be accessed/modified by any other procedure by sending data/location of data to it.

Object Oriented Programming Paradigm aims to create `objects` rather than procedures . Where objects are melding of data and procedures that manipulates the data. Only the procedures in the objects are allowed to manipulate the data of the objects. There by we can impose some restrictions in manipulating the data in procedural paradigm. The melding of data and functions as an object provides many features, those are encapsulation, data hiding, inheritance, polymorphism and programming interface. Languages that supports OOP are Java,C++,Python etc.

1.2 Important Points

- Most of the time it is strongly recommended to name the compilation-unit/translation unit with the name of one of the `class`es/`type`s declared in it. You need not to follow this unless One of the types(if the compilation unit has more than one type) or the only type
  - is referred to by code in other compilation units of the package in which the type is declared.
  - is declared public (and therefore is potentially accessible from code in other packages).

  We use the name of the referred type or public-declared type to name the compilation-unit.[3]
  The above scenario is observable during compile-time only.
- We no need to explicitly import types of the same package (It is an error if we do!).
- The term "constructor" is misleading since, as soon as you enter the constructor, the new object has actually been created for you. The job of the constructor is to ensure that the new object is in a valid state, usually by giving initial values to the instance variables of the object. So a "constructor" should really be called an "initializer."[1]
- Class constructor only initializes the data memebers and there is some `other`[1] mechanism by which an object is created.

[1] `https://docs.oracle.com/javase/tutorial/java/javaOO/objectcreation.html`

Chapter 2

Lab 2 [20-08-2015]

2.1 Matrix Multiplication

```
1  /*
2  Multiplication of matrices .....
3  */
4  import java.util.Scanner;
5
6  class Matrix{
7          int [][]mat;
8          Matrix(Scanner stdin){
9                  System.out.println("Rows : ");
10                 int row=stdin.nextInt();
11                 System.out.println("Columns : ");
12                 int col=stdin.nextInt();
13                 System.out.println("Elements :");
14                 mat=new int [row][col];
15                 for (int var=0;var<row;++var)
16                         for (int flag=0;flag<col;++flag)
17                                 mat[var][flag]=stdin.nextInt();
18                 }
19         }
20
21 class Multiplication{
22         int [][]mat1,mat2,result;
23         Multiplication(int m[][],int n[][]){
24                 mat1=m;
25                 mat2=n;
26                 }
27         void result(){
28                 if (checkMat()){
29                         //Multiplication of two matrices .
30                         System.out.println("Multiplication");
31                         int sum;
32                         result=new int[mat1.length][mat2[0].length];
33                         for (int var=0;var<mat1.length;++var)
34                                 for (int flag=0;flag<mat2[0].length;++flag){
35                                         sum=0;
36                                         for (int temp=0;temp<mat2.length;++temp)
37                                                 sum+=mat1[var][temp]*mat2[temp][flag];
38                                         result[var][flag]=sum;
39                                         }
40                         for (int var=0;var<mat1.length;++var)
41                                 for (int flag=0;flag<mat2[0].length;++flag)
42                                         System.out.println(result[var][flag]);
43                         }
44                 else
45                         System.out.println("Unable to perform multiplication");
46                 }
47         boolean checkMat(){
48                 if (mat1[0].length==mat2.length)
49                         return true;
50                 return false;
```

```
51                  }
52
53          }
54
55  class mainMatrixMultiplication{
56          public static void main(String args[]){
57                  Scanner stdin=new Scanner(System.in);
58                  Matrix mat1=new Matrix(stdin);
59                  Matrix mat2=new Matrix(stdin);
60                  Multiplication problem=new Multiplication(mat1.mat,mat2.mat);
61                  problem.result();
62                  }
63          }
```

2.2 Bubble Sort

2.2.1 BubbleSort.java

```
1  package MyPack;
2  public class BubbleSort {
3          public static void sort(int[] array){
4                  int temp;
5                  for (int var=0;var< array.length;++var){
6                          for (int flag=0;flag<array.length-1;++flag){
7                                  if (array[flag] > array[flag+1]){
8                                          temp=array[flag];
9                                          array[flag]=array[flag+1];
10                                         array[flag+1]=temp;
11                                         }
12                                 }
13                         }
14                 }
15
16         }
```

2.2.2 UseBubbleSort.java

```
1  import MyPack.BubbleSort;
2  public class UseBubbleSort{
3          public static void main(String[] args){
4                  int[] list={3,0,8,3,23,6,123,4,6,-1};
5                  BubbleSort.sort(list);
6                  for (int var=0;var<list.length;++var)
7                          System.out.println(list[var]);
8                  }
9          }
```

2.3 Operations on objects of String class

This program is to demonstrate operations on objects of String class, available in `java.lang`. Objects of String are immutable. We can use `StringBuffer` class to operate on mutable strings.

```
/*
Author: saikiran638
Description:
        This program demonstrate operations on String objects
        No object has been created for user-defined classes.
*/

class StringDemo{
        private static String str1,str2;
        public static void set( String str1,String str2 ) {
                //String object is created in String Constant Pool
                StringDemo.str1 = str1;
                // String object is created
                StringDemo.str2 = new String(str2);
                }
        public static void show() {
                System.out.println( str1 == str2 );
                System.out.println( str1.equals(str2) );
                System.out.println( str2.length() );
                System.out.println( str1.concat(str2) );
                System.out.println( str1.compareTo(str2) );
                System.out.println( str2.charAt(0) );
                System.out.println( str1.startsWith( "sai" ));
                System.out.println( str2.endsWith( "638" ) );
                System.out.println( str1.indexOf( "kiran" ) );

                set("saikiran638","SAIKIRAN638");

                System.out.println( str1.equals( str2 ) );
                System.out.println( str1.equalsIgnoreCase( str2 ) );
                System.out.println( str1.compareTo( str2 ) );
                System.out.println( str2.compareToIgnoreCase ( str1 ) );
                System.out.println( str2.toLowerCase() );
                System.out.println( str1.toUpperCase() );
                System.out.println( str1.substring(0,8) );

                // Split using a delimeter
                String[] strs = str1.split( "i" );
                for ( int index = 0 ; index < strs.length ; ++index )
                        System.out.println( strs[index] );
                // Changing contents of a string
                char[] chars = new char[ str1.length() ];
                str1.getChars(0,str1.length(), chars, 0 );
                System.out.println (chars);
                chars[8] ='1';
                chars[9] ='2';
                chars[10] = '3';

                set ( new String( chars ) , "saikiran" );
                System.out.println( str1 );

                //Trim the string
                set("  saikiran638  ","saikiran");
                System.out.println( str1.trim() );
                }
        }

public class Demo {
        public static void main( String args[] ) {
                StringDemo.set ( "saikiran638","saikiran638" );
                StringDemo.show();
                }
        }
```

2.4 Generic Types

2.4.1 Demo on `Vector` class

A *generic type* is a generic class or interface that is parameterized over *types*[2]. We can't use *primitive* data types as `type argument` to generic classes.

```
1  /*
2  Author: saikiran638
3  Description:
4          Vector is a part of Generic Types in Java much like C++'s STL.
5          Here in this program we are not specifying type of Vector,
6          Type will be determined by type inference.
7          So, we may get a note or warning from the compiler.
8  */
9  import java.util.*;
10
11 class VectorDemo{
12         public static void main(String args[]) {
13                 Vector vec = new Vector(4,8);
14                 //No. of elements that are there in Vector
15                 System.out.println("size:"+vec.size());
16                 //No. of elements + No. of empty blocks
17                 System.out.println("capacity:"+vec.capacity());
18                 //Add an object
19                 vec.addElement(new Integer(3));
20                 //Add another object
21                 vec.addElement(new String("saikiran638"));
22                 //Add another object
23                 vec.addElement(new Float(2.356));
24                 vec.addElement(new Boolean(true));
25                 vec.addElement(new Character('F'));
26                 //Print first object
27                 System.out.println("First element:"+vec.firstElement());
28                 //Print last object
29                 System.out.println("Last element:"+vec.lastElement());
30
31                 Enumeration vEnum = vec.elements();
32                 System.out.println("\nElements in vector:");
33                 while(vEnum.hasMoreElements())
34                         System.out.print(vEnum.nextElement() + " ");
35                 System.out.println();
36
37                 System.out.println("size of vector:"+ vec.size() );
38                 System.out.println("capacity of vector:"+ vec.capacity() );
39                 }
40         }
```

2.5 Wrapper Classes

2.5.1 WrapperClassDemo.java

```
1  /*
2  Author   : saikiran638
3  Description      : WrapperClasses Demo.
4  */
5
6  import java.util.*;
7
8  public class WrapperClassDemo {
9          public static float simpleInterest( Float p,Float r,Integer t){
10                 float principle = p.floatValue();
11                 float rate      = r.floatValue();
12                 int time        = t.intValue();
13                 float interest  = ( principle * rate * time) / 100.0f;
14                 return interest;
15                 }
16
17         public static void main(String args[]) {
18                 Scanner stdin   = new Scanner(System.in);
19                 System.out.println("Priciple Amount : ");
20                 float p         = stdin.nextFloat();
21                 System.out.println("Time :");
22                 int t           = stdin.nextInt();
23                 System.out.println("Rate :");
24                 float r         = stdin.nextFloat();
25                 Integer time    = new Integer(t);
26                 Float rate      = new Float(r);
27                 Float principle = new Float(p);
28                 System.out.println("Interest = "+ simpleInterest(principle,rate,time));
29                 }
30         }
```

Chapter 3

Lab 3 [25-08-2015]

3.1 static vs. instance

```
1
2  /*
3  Author: saikiran638
4  Description: To demonstrate instance,static variables and functions
5  */
6
7  class Car {
8          //class variable
9          private static int count = 0;
10         //instance variable
11         private int serial;
12         Car(){
13                 //one car is created
14                 Car.count++;
15                 //set serial number
16                 serial  = Car.count;
17                 }
18         int serial(){
19                 return serial;
20                 }
21         static int count(){
22                 return count;
23                 }
24         static void reset(int c){
25                 count   = c;
26                 }
27         }
28
29 class Manufacturer {
30         String name;
31         // Max 10 cars
32         private static Car cars[] = new Car[10];
33         Manufacturer(String name){
34                 this.name = name;
35                 }
36         void manufacture(){
37                 cars[0] = new Car();
38                 cars[1] = new Car();
39                 System.out.println("Cars manufactured : "+Car.count());
40                 cars[2] = new Car();
41                 }
42         static void statistics(){
43                 // Here cars.length = 10. But we didn't fill all of them
44                 for (int var = 0; var < cars.length;++var)
45                         if ( cars[var] != null )
46                                 System.out.println(cars[var].serial());
47                 }
48         }
49 /*Driver class*/
50 class Demo {
```

```
51          public static void main(String[] args){
52                  Manufacturer company;
53                  if ( args.length != 0 )
54                          company = new Manufacturer(args[0]);
55                  else    company = new Manufacturer("TATA");
56                  company.manufacture();
57                  company.statistics();
58                  }
59          }
```

3.2 Method and Constuctor overloading

```
/*
1.      We can't call a constructor from a method.
        But we can call a method from Constructor (it is not recommended).
2.      We can call one constructor from another and
        the call should be the first one in calling constructor.
*/

import java.util.Scanner;

class Box {
        double width;
        double height;
        double length;

        Box(double l,double w,double h){
                width = w;
                height = h;
                length = l;
                }

        Box(double len){
                width = length = height = len;
                }

        double volume(){
                return width*height*length;
                }
        /*Method overloading section starts */
        void set(double len,double h){
                // length = breadth = len
                length = width = len;
                height = h;
                }

        void set(double len){
                width = length = height = len;
                }

        void set(double l,double w,double h){
                length = l;
                width = w;
                height = h;
                }

        public static void main(String args[]){
                //Constructor Overloading
                /*
                Box cube        = new Box(4);
                Box box         = new Box(5,3,2);
                System.out.println(cube.volume());
                System.out.println(box.volume());
                */
                //Method overloading
                Box box = new Box(4);
                System.out.println(box.volume());
                box.set(5,3,2);
                System.out.println(box.volume());
                box.set(2,4);
                System.out.println(box.volume());
                }
        }
```

Chapter 4

Lab 4 [01-09-2015]

4.1 Abstract Class Demo

4.1.1 Year.java

```
1
2  package calendar;
3
4  public class Year{
5          private static int days[] = {0,31,28,31,30,31,30,31,31,30,31,30,31};
6          public static boolean isleap(int year){
7                  return (year%4 == 0 && (year%100 !=0 || year%400==0));
8                  }
9          public static int[] monthDays(int year){
10                 if ( isleap(year) )
11                         days[2] = 29;    //If leap.
12                 return days;
13                 }
14         }
```

4.1.2 Date.java

```
1  import calendar.Year;
2  import java.util.Scanner;
3
4  abstract class Day{
5          public static Scanner stdin = new Scanner(System.in);
6          int dd,mm,yyyy,days[];
7
8          Day(int day,int month,int year) {
9                  dd      = day;
10                 mm      = month;
11                 yyyy= year;
12                 days= Year.monthDays(yyyy);
13                 }
14
15         abstract void nextDay();
16         abstract void prevDay();
17
18         }
19
20 public class Date extends Day {
21
22         Date(int day,int month,int year){
23                 super(day,month,year);
24                 }
25
26         void prevDay(){
27                 if ( dd == 1) {
28                         if (mm==1){
29                                 mm = 12;
30                                 yyyy--;
31                                 dd=days[mm];
32                                 }
```

```
33                          else {
34                                  mm--;
35                                  dd = days[mm];
36                                  }
37                          }
38                  else dd--;
39                          System.out.println(dd+"-"+mm+"-"+yyyy);
40                  }
41
42          void nextDay(){
43                  if ( dd == days[mm] ) {
44                          if ( mm == 12 ) {
45                                  yyyy++;
46                                  dd = 1;
47                                  mm=1;
48                                  }
49                          else {
50                                  mm++;
51                                  dd = 1;
52                                  }
53                          }
54                  else
55                          dd++;
56                  System.out.println(dd+"-"+mm+"-"+yyyy);
57                  }
58
59          public static void main(String args[]) {
60                  int dd          = stdin.nextInt();
61                  int mm          = stdin.nextInt();
62                  int yyyy        = stdin.nextInt();
63                  Date date= new Date(dd,mm,yyyy);
64                  date.prevDay();
65                  date.nextDay();
66                  }
67          }
```

Chapter 5

Lab 5 [08-09-2015]

5.1 Multiple Inheritance using Interfaces

```
1  interface A{
2          void A1();
3          void A2();
4          }
5  interface B extends A{
6          void B1();
7          void B2();
8          }
9  interface C{
10         void C1();
11         void C2();
12         }
13
14 class InterfaceDemo implements B,C{
15         public void A1(){
16                 System.out.println("It's A1");
17                 }
18         public void A2(){
19                 System.out.println("It's A2");
20                 }
21         public void B1(){
22         System.out.println("It's B1");
23                 }
24         public void B2(){       System.out.println("It's B2");
25                 }
26         public void C1(){       System.out.println("It's C1");
27                 }
28         public void C2(){       System.out.println("It's C2");
29                 }
30         public static void main(String args[]){
31                 InterfaceDemo demo = new InterfaceDemo();
32                 demo.A1();
33                 demo.A2();
34                 demo.B1();
35                 demo.B2();
36                 demo.C1();
37                 demo.C2();
38                 }
39         }
```

5.2 User Defined Packages

5.2.1 Details.java

```
1  package MyPack;
2  import java.util.Scanner;
3
4  public class Details {
5          static String[] details;
6          public Details(String[] list){
7                  details = list;
8                  }
9          public void show() {
10                 for (int var=0;var<details.length;++var) {
11                         System.out.println(details[var]);
12                         }
13                 }
14         }
```

5.2.2 Usepack.java

```
1  import MyPack.Details;
2
3  public class Usepack{
4          static String[] det ={"SAI KIRAN","N","N100638"};
5          public static void main(String[] args){
6                  Details d = new Details(det);
7                  d.show();
8                  }
9          }
```

Chapter 6

Lab 6 [15-09-2015]

6.1 Threads

```
1  import java.util.*;
2
3  class Time extends Thread {
4          public void run() {
5                  try {
6                          for (int i=0;i<60;++i){
7                                  System.out.println(new Date());
8                                  Time.sleep(1000);
9                                  }
10                         }
11                 catch(InterruptedException e ){
12                         e.printStackTrace();
13                         }
14                 }
15         public static void main(String args[]) {
16                 new Time().start();
17                 }
18         }
```

Chapter 7

Mini Project [Movie Ticket Booking System]

Bibliography

[1] http://www.cis.upenn.edu/~matuszek/General/JavaSyntax/constructors.html

[2] https://docs.oracle.com/javase/tutorial/java/generics/why.html

[3] https://docs.oracle.com/javase/specs/jls/se8/html/jls-7.html#jls-7.6