Pillai College of Engineering

Dr. K. M. Vasudevan Pillai Campus, Plot No. 10, Sector 16, New Panvel, Navi Mumbai, Maharashtra 410206

DEPARTMENT OF INFORMATION TECHNOLOGY

JAVA PROGRAMMING LAB

LAB MANUAL

2019 – 2020

Semester : III **Class :** SE - IT

Prepared By

Prof. Deepti S. Lawand

LIST OF EXPERIMENTS

Sem III 2019-20

MODULE 1 - Fundamental of Java Programming

1. Write a Java program to understand how to accept input using Scanner and print output using System.out.println statement.

2. Write a Java program to display the default value and size of all primitive data types in Java.

3. Write a program to implement a menu driven Calculator to perform addition, subtraction, multiplication and division functions.

MODULE 2 - Class, Object, Arrays and Recursion

4. Write a program to demonstrate Command Line Arguments.

5. Write a java program to demonstrate default constructors, Parameterized Constructors and Constructor Overloading

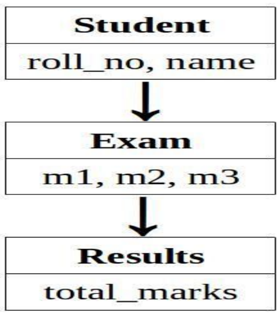
6. Write a program using recursive function 'power' to compute x^n, power(x,n) = 1 if n=0, power(x,n) = x if n=1, power(x,n) = x*power(x, n-1) otherwise

7. Write a program to take input for 'N' integers in an array and display only those integers that are greater than the average of all integers.

8. Write a program to create Vector objects with Student names. Program should perform following operations based on choice :
    - Add Student name - to add new student name in the Vector
    - Remove Student Name - Removes student name if already exists else display appropriate message
    - Search a student by index
    - Display - Display contents of vector

MODULE 3 - Inheritance, Interface and Package

9. Consider the class network given below for multilevel inheritance. Write a program to display 'Results' object.



10. Write a java program to create user defined package. Define interface for Area and Volume, and implement the required interfaces in Circle class and Sphere class.

MODULE 4 - Exception Handling and Multithreading

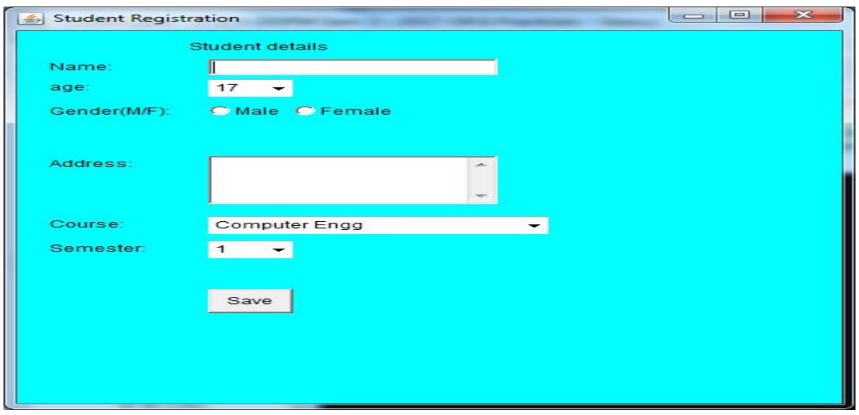
11. Write a program to accept and display the month number. Throw an exception if improper month number is entered. Make your own exception class to handle this exception.

12. Write a program to print 1A2B3C4D5E6F7G8H9I10J using two child threads.

MODULE 5 - Applet Programming, GUI development using AWT and Event handling

13. Write an applet to display a smiley face. Also display a welcome message by passing parameters to the applet.

14. Write a java program to display the following GUI Form to accept Student's Information using AWT components (TextField, TextArea Button, Label, etc) and creating frame to add all the components and event handling



MODULE 6 - Java Swings

15. Write a program to create a window with four text fields for the name, street, city and zipcode with suitable labels. Also windows contains a button MyInfo. When the user types the name, his street, city and pincode and then clicks the button, the types details must appear in Arial Font with Size 32, Italics.

Experiment 1

Title: Write a Java program to understand how to accept input using Scanner and print output using System.out.println statement.

Theory:

Java programming language was originally developed by Sun Microsystems which was initiated by James Gosling and released in 1995 as core component of Sun Microsystems' Java platform (Java 1.0 [J2SE]). The latest release of the Java Standard Edition is Java SE 8. With the advancement of Java and its widespread popularity, multiple configurations were built to suite various types of platforms. Ex: J2EE for Enterprise Applications, J2ME for Mobile Applications. The new J2 versions were renamed as Java SE, Java EE and Java ME respectively. Java is guaranteed to be Write Once, Run Anywhere.

Features of Java:

- Object Oriented: In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
- Platform independent: Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent bytecode. This byte code is distributed over the web and interpreted by virtual Machine (JVM) on whichever platform it is being run.
- Simple: Java is designed to be easy to learn. If you understand the basic concept of OOP Java would be easy to master.
- Secure: With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
- Portable: Being architectural-neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler in Java is written in ANSI C with a clean portability boundary which is a POSIX subset.
- Robust: Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.
- Multithreaded: With Java multithreaded feature it is possible to write programs that can do many tasks simultaneously. This design feature allows developers to construct smoothly running interactive applications.

- Interpreted: Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light weight process.
- High Performance: With the use of Just-In-Time compilers, Java enables high performance.
- Distributed: Java is designed for the distributed environment of the internet.
- Dynamic: Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

Simple Java Program

```
class Simple{
   public static void main(String args[]){
    System.out.println("Hello Java");
   }
}
```

To Save	Simple.java
To compile	javac Simple.java
To execute	java Simple

Output: Hello Java

Understanding first java program:

- **class** keyword is used to declare a class in java.
- **public** keyword is an access modifier which represents visibility, it means it is visible to all.
- **static** is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method. The main method is executed by the JVM, so it doesn't require to create object to invoke the main method. So it saves memory.
- **void** is the return type of the method, it means it doesn't return any value.
- **main** represents startup of the program.
- **String[] args** is used for command line argument.
- **System.out.println()** is used print/display message on output screen.

Input using Scanner Class

There are various ways to read input from the keyboard, the java.util.Scanner class is one of them. The **Java Scanner** class breaks the input into tokens using a delimiter that is whitespace by default. It provides many methods to read and parse various primitive values.

Java Scanner class is widely used to parse text for string and primitive types using regular expression. Java Scanner class extends Object class and implements Iterator and Closeable interfaces. Commonly used methods of Scanner class are:

Method	**Description**
public String next()	it returns the next token from the scanner, it can read the input only till the space
public String nextLine()	it moves the scanner position to the next line and returns the value as a string. It reads input including space between the words (that is, it reads till the end of line n)
public byte nextByte()	it scans the next token as a byte.
public short nextShort()	it scans the next token as a short value.
public int nextInt()	it scans the next token as an int value.
public long nextLong()	it scans the next token as a long value.
public float nextFloat()	it scans the next token as a float value.
public double nextDouble()	it scans the next token as a double value.

Input using BufferedReader

In Java, console input is accomplished by reading from System.in. To obtain a character based stream that is attached to the console, wrap System.in in a BufferedReader object. BufferedReader supports a buffered input stream. Its most commonly used constructor is shown here:

BufferedReader(Reader inputReader)

Here, inputReader is the stream that is linked to the instance of BufferedReader that is being created. Reader is an abstract class. One of its concrete subclasses is InputStreamReader,

which converts bytes to characters. To obtain an InputStreamReader object that is linked to System.in, use the following constructor:

InputStreamReader(InputStream inputStream)

Because System.in refers to an object of type InputStream, it can be used for inputStream. Putting it all together, the following line of code creates a BufferedReader that is connected to the keyboard:

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

After this statement executes, br is a character-based stream that is linked to the console through System.in.

Conclusion: Thus we have implemented a java program to understand how to accept input using Scanner and BufferedReader and print output using System.out.println statement.

Program using Scanner

```
import java.util.*;

class scanner{

  public static void main(String[] args) throws Exception{

    Scanner s1 = new Scanner(System.in);

    System.out.print("Enter a String:");

    String s= s1.nextLine();

    System.out.println("The entered String is: " +s)

    System.out.print("Enter a character:");

    char c= (char)System.in.read();

    System.out.println("The entered character is: " +c);

    System.out.print("Enter a Byte:");

    int b= s1.nextByte();
    System.out.println("The entered Byte is: " +b);

    System.out.print("Enter a Short:");

    int sh= s1.nextShort();

    System.out.println("The entered Short is: " +sh);

    System.out.print("Enter an Integer:");

    int i= s1.nextInt();

    System.out.println("The entered Integer is: " +i);



    System.out.print("Enter a Long:");

    long l= s1.nextLong();

    System.out.println("The entered Long is: " +l);



    System.out.print("Enter a Float no:");

    float f= s1.nextFloat();

    System.out.println("The entered Float is: " +f);



    System.out.print("Enter a Double:");

    double d= s1.nextDouble();

    System.out.println("The entered Double is: " +d);

  }

}
```

Program using BufferedReader

```
import java.io.*;

class bufferedreader

{

  public static void main(String[] args) throws Exception

   {

    BufferedReader b= new BufferedReader(new InputStreamReader(System.in));

    System.out.print("Enter a String: ");

    String s=b.readLine();

    System.out.println("The entered STRING  is: " + s);

   }

}
```

Output

```
Enter a String: JavaWorld

The entered STRING  is: JavaWorld
```

Experiment 2

Title: Write a Java program to display the default value and size of all primitive data types in Java.

Theory:

Java defines eight primitive types of data: byte, short, int, long, char, float, double, and boolean. The primitive types are also commonly referred to as simple types, and both terms will be used in this book. These can be put in four groups:

- Integers This group includes byte, short, int, and long, which are for whole-valued signed numbers.
- Floating-point numbers This group includes float and double, which represent numbers with fractional precision.
- Characters This group includes char, which represents symbols in a character set, like letters and numbers.
- Boolean This group includes boolean, which is a special type for representing true/false values.

Integers

Java defines four integer types: byte, short, int, and long. All of these are signed, positive and negative values. Java does not support unsigned, positive-only integers byte

Name	**Width**	**Range**
long	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
int	32	–2,147,483,648 to 2,147,483,647
short	16	–32,768 to 32,767
byte	8	–128 to 127

The smallest integer type is byte. This is a signed 8-bit type that has a range from –128 to 127. Variables of type byte are especially useful when you're working with a stream of data from a network or file. They are also useful when you're working with raw binary data that may not be directly compatible with Java's other built-in types. Byte variables are declared by use of the byte keyword. For example, the following declares two byte variables called b and c:

```
byte b, c;
```

short

short is a signed 16-bit type. It has a range from –32,768 to 32,767. It is probably the least-used Java type. Here are some examples of short variable declarations:

```
short s;   short t;
```

int

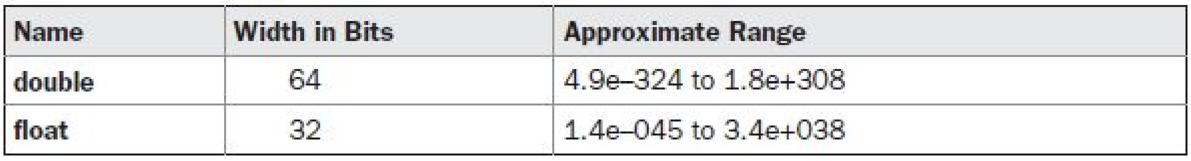
The most commonly used integer type is int. It is a signed 32-bit type that has a range from –2,147,483,648 to 2,147,483,647.

long

long is a signed 64-bit type and is useful for those occasions where an int type is not large enough to hold the desired value. The range of a long is quite large. This makes it useful when big, whole numbers are needed.

Floating-Point Types

Floating-point numbers, also known as real numbers, are used when evaluating expressions that require fractional precision. For example, calculations such as square root, or transcendentals such as sine and cosine, result in a value whose precision requires a floating-point type. Java implements the standard (IEEE–754) set of floating-point types and operators. There are two kinds of floating-point types, float and double, which represent single- and double-precision numbers, respectively. Their width and ranges are shown here:



Name	Width in Bits	Approximate Range
double	64	4.9e–324 to 1.8e+308
float	32	1.4e–045 to 3.4e+038

float

The type float specifies a single-precision value that uses 32 bits of storage. Single precision is faster on some processors and takes half as much space as double precision, but will become imprecise when the values are either very large or very small. Variables of type float are useful when you need a fractional component, but don't require a large degree of precision. For example, float can be useful when representing dollars and cents. Here are some example float variable declarations:

```
float hightemp, lowtemp;
```

double

Double precision, as denoted by the double keyword, uses 64 bits to store a value. Double precision is actually faster than single precision on some modern processors that have been optimized for high-speed mathematical calculations. All transcendental math functions, such as sin( ), cos( ), and sqrt( ), return double values. When you need to maintain accuracy over many iterative calculations, or are manipulating large-valued numbers, double is the best choice.

```
double d;
```

Characters

In Java, the data type used to store characters is char. In Java char is a 16-bit type. The range of a char is 0 to 65,536. There are no negative chars. Here is a program that demonstrates char variables:

Booleans

Java has a primitive type, called boolean, for logical values. It can have only one of two possible values, true or false. This is the type returned by all relational operators, as in the case of a < b. boolean is also the type required by the conditional expressions that govern the control statements such as if and for.

Conclusion: Thus we have implemented a Java program to display the default value of all primitive data types in Java.

Program

/*Java has no sizeof operator to find the size of primitive data types but all Java primitive wrappers except Boolean provide a SIZE constant in bits that could be divided by eight to get the size of a data type in bytes. Moreover, since Java 8, all primitive wrapper classes (except Boolean) have a BYTES constant, which gives data type's size in bytes.So you can use that also once you have been moved to Java 8. */

```
class defaultvalues

{

	byte b; short s; static int i;
```

```
        static long l; static float f; static double d; static char c; boolean b1;

        void display()

        {

           System.out.println("Default Value of byte: " +b);

        }

public static void main(String[] args)

{

defaultvalues dv= new defaultvalues();

System.out.println("Default Value of byte: " +dv.b);

System.out.println("Default Value of short: "+dv.s);

System.out.println("Default Value of int: "+i);

System.out.println("Default Value of long: "+l);

System.out.println("Default Value of float: "+f);

System.out.println("Default Value of double: "+d);

System.out.println("Default Value of char:"+c);

System.out.println("Default Value of boolean:"+dv.b1);



    System.out.println("Size of char: " + (Character.SIZE/8) + " bytes.");

    System.out.println("Size of float: " + (Float.SIZE/8) + " bytes.");

    System.out.println("Size of double: " + (Double.SIZE/8) + " bytes.");

    System.out.println("Size of byte: " + (Byte.BYTES) + " bytes.");

    System.out.println("Size of short: " + (Short.BYTES) + " bytes.");

    System.out.println("Size of Integer: " + (Integer.BYTES) + " bytes.");
    System.out.println("Size of Long: " + (Long.BYTES) + " bytes.");

dv.display();

}

}

/*output

Default Value of byte: 0

Default Value of short: 0

Default Value of int: 0

Default Value of long: 0

Default Value of float: 0.0

Default Value of double: 0.0

Default Value of char:

Default Value of boolean:false

Size of char: 2 bytes.

Size of float: 4 bytes.

Size of double: 8 bytes.

Size of byte: 1 bytes.

Size of short: 2 bytes.

Size of Integer: 4 bytes.

Size of Long: 8 bytes.

Default Value of byte: 0

*/
```

Experiment 3

Title: Write a java program to implement a menu driven Calculator to perform addition, subtraction, multiplication and division functions.

Theory:

When a program breaks the sequential flow and jumps to another part of the code, it is called branching. If the branching is based upon a condition, it is called conditional branching, and when the branching takes place without any decision, it is called as unconditional branching.

Control Statements

The Java if statement is used to test the condition. It returns true or false. There are various types of if statement in java:

1. if statement
2. ifelse statement
3. nested if statement
4. ifelseif ladder

IF

```
if(Boolean_expression)
{
 //Statements will execute if the Boolean expression is true
 }
```

IF ELSE

```
if(Boolean_expression)
{
  //Executes when the Boolean expression is true
}
 else
{
 //Executes when the Boolean expression is false
 }
```

Nested IF ELSE

```
if(Boolean_expression 1)
{
  //Executes when the Boolean expression 1 is true
  if(Boolean_expression 2)
  {
    //Executes when the Boolean expression 2 is true
  }
  }
```

IF ELSE if ladder

```
if(Boolean_expression 1){
  //Executes when the Boolean expression 1 is true} else
if(Boolean_expression 2){
  //Executes when the Boolean expression 2 is true} else
if(Boolean_expression 3){
  //Executes when the Boolean expression 3 is true} else
{
  //Executes when the none of the above condition is true.}
```

SWITCH

```
switch(expression)
{
            case value1 :
              //Statements break;
              //optional
            case value2 :
              //Statements break;
              //optional
  //You can have any number of case statements.
          default : //Optional
```

Break

The break statement in Java programming language has the following two usages −

- When the break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
- It can be used to terminate a case in the switch statement (covered in the next chapter).

Syntax

The syntax of a break is a single statement inside any loop −

```
break;
```

Iterative Statements

- **for**

```
for(initialization; Boolean_expression; updation)
{
 //Statements
 }
```

- **while**

When executing, if the *boolean_expression* result is true then the actions inside the loop will be executed. This will continue as long as the expression result is true.

```
 while(Boolean_expression
)
 {
 //Statements
  }
```

- **dowhile**

The Boolean expression appears at the end of the loop, so the statements in the loop execute once before the Boolean is tested. If the Boolean expression is true, the flow of control jumps back up to do, and the statements in the loop execute again. This process repeats until the Boolean expression is false.

```
do{

//Statements

 }

while(Boolean_expression);
```

Conclusion: Thus we have implemented a java program to implement a menu driven Calculator to perform addition, subtraction, multiplication and division functions.

Program:Write a program to implement a menu driven Calculator to perform addition, subtraction, multiplication and division functions.

```
import java.util.*;
class Mycal1
{
public static void main(String args[])
{
int choice,a=0,b=0;
do
{
System.out.println("select ur choice\n 1.addition\n 2.subtraction\n 3.multiplication\n 4.division\n 5.Exit");
Scanner sc=new Scanner(System.in);
choice=sc.nextInt();
switch(choice)
{
case 1:  System.out.println("enter two numbers");
                a=sc.nextInt();
                b=sc.nextInt();
                System.out.println("addition="+(a+b));
break;

case 2: System.out.println("enter two numbers");
                a=sc.nextInt();
                b=sc.nextInt();
                System.out.println("subtraction="+(a-b));
break;

case 3: System.out.println("enter two numbers");
                a=sc.nextInt();
                b=sc.nextInt();
                System.out.println("multiplication="+(a*b));
break;

case 4: System.out.println("enter two numbers");
                a=sc.nextInt();
                b=sc.nextInt();
```

```
                    System.out.println("division="+(a/b));
break;

case 5 : System.exit(0);
default : System.out.println("Invalid Input");
}
}while(choice !=5);
}
}
```

Output:

```
select ur choice

 1.addition

 2.subtraction

 3.multiplication

 4.division

 5.Exit

1

enter two numbers

5

6

adition=11

select ur choice

 1.addition

 2.subtraction

 3.multiplication

 4.division

 5.Exit
```

```
2
enter two numbers
8
1
subtraction=7
select ur choice
 1.addition
 2.subtraction
 3.multiplication
 4.division
 5.Exit
3
enter two numbers
4
5
multiplication=20
select ur choice
 1.addition
 2.subtraction
 3.multiplication
 4.division
 5.Exit
4
```

```
enter two numbers

10

5

division=2

select ur choice

 1.addition

 2.subtraction

 3.multiplication

 4.division

 5.Exit

5

------------------

(program exited with code: 0)

Press any key to continue . . .
```

Experiment 4

Title: Write a java program to demonstrate Command Line Arguments.

Theory: Command-Line Arguments

Sometimes you will want to pass information into a program when you run it. This is accomplished by passing command-line arguments to main( ). A command-line argument is the information that directly follows the program's name on the command line when it is executed. To access the command-line arguments inside a Java program is quite easy— they are stored as strings in a String array passed to the args parameter of main( ). The first command-line argument is stored at args[0], the second at args[1], and so on.

Conclusion: Thus we have implemented a java program to demonstrate Command Line Arguments.

Program 1

```
// Display all command-line arguments.
class CommandLine {
public static void main(String args[]) {
for(int i=0; i<args.length; i++)
System.out.println("args[" + i + "]: " +args[i]);
}
}
```

OUTPUT:

```
C:\Users\admin\Desktop\oopm notes\programs>javac CommandLine.java

C:\Users\admin\Desktop\oopm notes\programs>java CommandLine these are command line arguments 2 4.0 -7

args[0]: these

args[1]: are

args[2]: command

args[3]: line

args[4]: arguments

args[5]: 2

args[6]: 4.0

args[7]: -7
```

Program 2

```
public class CommandLine2 {

   public static void main(String[] args) {

      int sum = 0;

      for (int i = 0; i < args.length; i++) {

         sum = sum + Integer.parseInt(args[i]);

      }

      System.out.println("The sum of the arguments passed is " + sum);

   }

}
```

OUTPUT:

```
C:\Users\admin\Desktop\oopm notes\programs>javac CommandLine2.java

C:\Users\admin\Desktop\oopm notes\programs>java CommandLine2 4 3 1

The sum of the arguments passed is 8
```

Experiment 5

Title: Write a java program to demonstrate Constructors, Parameterized Constructors and Constructor Overloading

Theory:

Constructor in java is a special type of method that is used to initialize the object. Java constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object that is why it is known as constructor.

There are basically two rules defined for the constructor.

1. Constructor name must be same as its class name.
2. Constructor must have no explicit return type.

There are two types of constructors:

1. Default constructor

A constructor that has no parameter is known as default constructor.

Note: If there is no constructor in a class, compiler *automatically* creates a default constructor.

2. **Parameterized constructor**

A constructor that have parameters is known as parameterized constructor.

Constructor overloading

Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

Conclusion: Thus we have implemented a java program to demonstrate Constructors, Parameterized Constructors and Constructor Overloading

Programs

Program for constructor

```
class Bike
```

```
{        Bike()
           {        System.out.println("Bike is created");
           }
           public static void main(String args[])
           {        Bike b=new Bike();
           }
}
```

Output:

```
Bike is created
```

Program for Parameterized constructor

```
class Student
{
           int   id;   String
           name;

           Student(int i,String n){
           id = i; name = n;
           }
           void display()
           {
                    System.out.println(id+" "+name);
           }

           public static void main(String args[])
           {
                    Student s1 = new Student(111,"Karan"); Student s2 =
                    new Student(222,"Aryan"); s1.display();
                    s2.display();
      }
}
```

OUTPUT :

```
111 Karan
222 Aryan
```

Program for Constructor overloading

```
class Student
{
int   id;   String
name; int age;
Student(int i,String n)
{        id = i; name = n;
}
```

```
            Student(int i,String n,int a)
            {    id = i;
                name = n;
                age=a;
            }
            void display()
            {        System.out.println(id+" "+name+" "+age);
            }

            public static void main(String args[])
                    {          Student  s1  =  new  Student(111,"Karan");  Student
                               s2 = new Student(222,"Aryan",25); s1.display();
                               s2.display();
            }
}
```

OUTPUT :

```
111 Karan 0
222 Aryan 25
```

Experiment 6

Title: Write a java program using recursive function 'power' to compute x^n, power(x,n) = 1 if n=0, power(x,n) = x if n=1, power(x,n) = x*power(x, n-1) otherwise

Theory:

Recursion

Java supports recursion. Recursion is the process of defining something in terms of itself. As it relates to Java programming, recursion is the attribute that allows a method to call itself. A method that calls itself is said to be recursive. The classic example of recursion is the computation of the factorial of a number. The factorial of a number N is the product of all the whole numbers between 1 and N. For example, 3 factorial is 1 × 2 × 3, or 6. Here is how a factorial can be computed by the use of a recursive method:

```
// A simple example of recursion.
class Factorial {
// this is a recursive method
int fact(int n)
{
int result;
if(n==1) return 1;
result = fact(n-1) * n;
return result;
}
}
class Recursion {
public static void main(String args[]) {
Factorial f = new Factorial();
System.out.println("Factorial of 3 is " + f.fact(3));
System.out.println("Factorial of 4 is " + f.fact(4));
System.out.println("Factorial of 5 is " + f.fact(5));
}
}
```

The output from this program is shown here:

```
Factorial of 3 is 6
Factorial of 4 is 24
Factorial of 5 is 120
```

When fact( ) is called with an argument of 1, the function returns 1; otherwise, it returns the product of fact(n–1)*n. To evaluate this expression, fact( ) is called with n–1. This process repeats until n equals 1 and the calls to the method begin Returning. To better understand how the fact( ) method works, let's go through a short example. When you compute the factorial of 3, the first call to fact( ) will cause a second call to be made with an argument of 2. This invocation will cause fact( ) to be called a third time with an argument of 1. This call will return 1, which is then multiplied by 2 (the value of n in the second invocation). This result (which is 2) is then returned to the original invocation of fact( ) and multiplied by 3 (the original value of n). This yields the answer, 6. You might find it interesting to insert println( ) statements into fact( ), which will show at what level each call is and what the intermediate answers are.

Conclusion:Thus we have implemented a java program using recursive function 'power' to compute x^n, power(x,n) = 1 if n=0, power(x,n) = x if n=1, power(x,n) = x*power(x, n-1) otherwise

Program: Write a program using recursive function 'power' to compute x^n
power(x,n) = 1 if n=0, power(x,n) = x if n=1, power(x,n) = x*power(x, n-1) otherwise*/

```
import java.util.Scanner;
class power
{
public static void main(String args[])
{
Scanner s=new Scanner(System.in);
int a,b;
 System.out.println("Enter base value");
 a=s.nextInt();
 System.out.println("Enter power value");
 b=s.nextInt();
power p=new power();
System.out.println("Ans="+p.cal(a,b));
}

int cal(int x,int y)
{

if(y==0)
{
return 1;
}

else
{
return(x*cal(x,y-1));

}
}
}
```

```
OUTPUT:
Enter base value
5
Enter power value
3
Ans=125
```

Experiment 7

Title: Write a java program to take input for 'N' integers in an array and display only those integers that are greater than the average of all integers.

Theory:

Java provides a data structure, the **array**, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables.

There are two types of array:

- Single Dimensional Array
- Multidimensional Array

Declaring Array Variables:

Syntax to Declare a Single Dimensional Array in java:

To use an array in a program, you must declare a variable to reference the array, and you must specify the type of array the variable can reference. Here is the syntax for declaring an array variable:

```
dataType[] arrayRefVar;   // preferred way.

or

dataType arrayRefVar[];  //  works but not preferred way.
```

Example:

```
double[] myList;         // preferred way.

or

double myList[];         //  works but not preferred way.
```

Creating Arrays:

You can create an array using the new operator with the following syntax:

arrayRefVar = new dataType[arraySize];

The above statement does two things:

- It creates an array using new dataType[arraySize];
- It assigns the reference of the newly created array to the variable arrayRefVar.

Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement, as shown below:

dataType[] arrayRefVar = new dataType[arraySize];

Alternatively you can create arrays as follows:

dataType[] arrayRefVar = {value0, value1, ..., valuek};

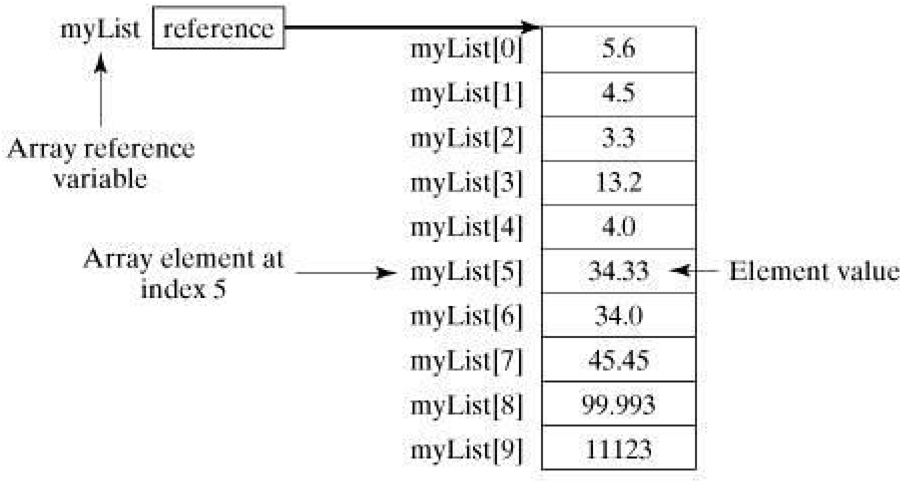
The array elements are accessed through the **index**. Array indices are 0-based; that is, they start from 0 to **arrayRefVar.length-1**.

Example:

Following statement declares an array variable, myList, creates an array of 10 elements of double type and assigns its reference to myList:

double[] myList = new double[10];

Following picture represents array myList. Here, myList holds ten double values and the indices are from 0 to 9.



Syntax to Declare a Multidimensional Array in java:

```
dataType[][] arr; OR

dataType   [][]arr; OR

dataType   arr[][];  OR
dataType   []arr[];  OR
```

Creating a Multidimensional Array in Java

```
arr = new dataType[row][col];
```

Note: Both can be combined into one statement as

```
dataType[][]    arr = new dataType[row][col];
```

Processing Arrays: When processing array elements, we often use either for loop or foreach loop because all of the elements in an array are of the same type and the size of the array is known.

Conclusion: Thus we have implemented a java program to take input for 'N' integers in an array and display only those integers that are greater than the average of all integers.

Program

```
import java.util.Scanner;
class Array
{       public static void main(String args[])
        {       int n, i;
                double sum=0, avg;

                Scanner sc = new Scanner (System.in);
                System.out.print("Enter array size: ");
                n = sc.nextInt();
                int a[] = new int[n];
                System.out.println("Enter array elements: ");
                for (i=0 ; i<n ; i++)
                {       a[i] = sc.nextInt();
                }
```

```
                    for (i=0 ; i<n ; i++)
                    {         sum += a[i];
                    }
                    avg = sum/n;
                    System.out.println("Average = " + avg);
                    System.out.println("Elements greater than average are:");
                    for ( i=0 ; i<n ; i++)
                    {         if ( a[i] > avg)
                              {         System.out.println(a[i]);
                              }
                    }
          }
}
```

OUTPUT:

```
Enter array size: 4

Enter array elements:

3

6

4

2

Average = 3.75

Elements greater than average are:

6

4
```

Experiment 8

Title: Write a java program to create Vector objects with Student names. Program should perform following operations based on choice :

- **Add Student name - to add new student name in the Vector**
- **Remove Student Name - Removes student name if already exists else display appropriate message**
- **Search a student by index**
- **Display - Display contents of vector**

Theory

Vector implements a dynamic array. It is similar to ArrayList, but with two differences: Vector is synchronized, and it contains many legacy methods that are not part of the Collections Framework. With the advent of collections, Vector was reengineered to extend AbstractList and to implement the List interface. With the release of JDK 5, it was retrofitted for generics and reengineered to implement Iterable. This means that Vector is fully compatible with collections, and a Vector can have its contents iterated by the enhanced for loop. Vector is declared like this:

class Vector<E>

Here, E specifies the type of element that will be stored. Here are the Vector constructors:

Vector( )

Vector(int size)

Vector(int size, int incr)

Vector(Collection c)

The first form creates a default vector, which has an initial size of 10. The second form creates a vector whose initial capacity is specified by size. The third form creates a vector whose initial capacity is specified by size and whose increment is specified by incr. The increment specifies the number of elements to allocate each time that a vector is resized upward. The fourth form creates a vector that contains the elements of collection c. All vectors start with an initial capacity. After this initial capacity is reached, the next time that you attempt to store an object in the vector, the vector automatically allocates space for that object plus extra room for additional objects. By allocating more than just the required memory, the vector reduces the number of allocations that must take place. This reduction is important, because allocations are costly in terms of time. The amount of extra space allocated during each reallocation is determined by the increment that you specify when you create the vector. If you don’t specify an increment, the vector’s size is doubled by each allocation cycle. Vector defines these protected data members:

```
int capacityIncrement;

int elementCount;

Object[ ] elementData;
```

The increment value is stored in capacityIncrement. The number of elements currently in the vector is stored in elementCount. The array that holds the vector is stored in elementData

The following program uses a vector to store various types of numeric objects. It demonstrates several of the legacy methods defined by Vector. It also demonstrates the Enumeration interface.

Conclusion: Thus we have implemented a java program to create Vector objects with Student names and perform vector operations based on choice.

Program

```
import java.util.*;

class Student

{

    public static void main(String args[])

    {

        Scanner sc=new Scanner(System.in);

        String str,s1;

        Vector v=new Vector(3,2);//3 is size and 2 is increment

        System.out.println("Initial size="+v.size());// gives how many elements are present in vector

        System.out.println("Initial capacity="+v.capacity());// gives how many elements u can add or
vector can hold

        int i,choice=0;

        /*System.out.println("how many student names do u want to add:");

        int n=sc.nextInt();

        System.out.println("Enter names of students :");

        for(i=0;i<n;i++)
```

```
	{
		str=sc.next();
		v.addElement(str);
	}
	System.out.println("size after adding elements="+v.size());
	System.out.println("capacity size after adding elements="+v.capacity());
	*/
	do
	{
		System.out.println();
		System.out.println("1.Add Student name\n2.Remove student name\n3.Search Student 
by Index");
		System.out.println("4.Display content of vector\n5.firstElement()\n6.lastElement()\n7.
contains()");
		System.out.println("8.To exit Program");
		System.out.println("------------------------------");
		System.out.print("Enter your choice :");
		choice=sc.nextInt();
		switch(choice)
		{
			case 1:
				System.out.println("------------------------------");
				System.out.println("how many student names do u want to add:");
				int n=sc.nextInt();
```

```
                System.out.println("Enter names of students :");
                for(i=0;i<n;i++)
                {
                str=sc.next();
                v.addElement(str);
                }
                System.out.println("size after adding elements="+v.size());
                System.out.println("capacity size after adding elements="+v.capacity());
                break;
            case 2:
                System.out.println("------------------------------");
                System.out.print("Enter Name :");
                str=sc.next();
                v.removeElement(str);
                System.out.println("size after removing elements="+v.size());
                System.out.println("capacity size after removing
elements="+v.capacity());
                break;
            case 3:
                try
                {
                System.out.println("------------------------------");
                System.out.print("Enter Index :");
```

```
		i=sc.nextInt();

		System.out.println("Name "+v.elementAt(i));

		break;

		}

		catch(ArrayIndexOutOfBoundsException e)

		{

			System.out.println("Wrong index value provided");

			break;

		}




		case 4:

		System.out.println("------------------------------");

		System.out.println("Current List:");

		for(i=0;i<v.size();i++)

		{

			System.out.println(v.elementAt(i));

		}

		break;


	case 5:

		System.out.println("First element in vector="+v.firstElement());

		break;
```

```
        case 6:

            System.out.println("Last element in vector="+v.lastElement());

            break;


        case 7:

            System.out.println("enter the element to check whether vector contains
it=");

            s1=sc.next();

            if(v.contains(s1))

            {

            System.out.println("vector contains "+s1);

            break;

            }

            else

            {

            System.out.println("vector does not contain "+s1);

            break;

            }


        case 8:

            {System.exit(0);

            break;}
```

```
                    default:{System.out.println("Invalid choice");
                            break;}
                }
        }while(choice>=1 && choice<=8);
    }
}
```

OUTPUT

```
Enter names of three students :
ram
sam
tam

1.Add Student name
2.Remove student name
3.Search Student by Index
4.Display content of vector
5.To exit Program
------------------------------
```

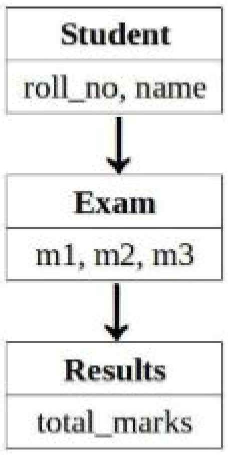
```
Enter your choice :1

------------------------------

Enter Name :tom



1.Add Student name

2.Remove student name

3.Search Student by Index

4.Display content of vector

5.To exit Program

------------------------------

Enter your choice :4

------------------------------

Current List:

ram

sam

tam

tom



1.Add Student name

2.Remove student name

3.Search Student by Index

4.Display content of vector

5.To exit Program
```

```
------------------------------

Enter your choice :2

------------------------------

Enter Name :tam



1.Add Student name

2.Remove student name

3.Search Student by Index

4.Display content of vector

5.To exit Program

------------------------------

Enter your choice :4

------------------------------

Current List:

ram

sam

tom



1.Add Student name

2.Remove student name

3.Search Student by Index

4.Display content of vector

5.To exit Program
```

```
------------------------------

Enter your choice :3

------------------------------

Enter Index :2

Name tom

1.Add Student name

2.Remove student name

3.Search Student by Index

4.Display content of vector

5.To exit Program
```

Experiment 9

Title: Consider the class network given below for multilevel inheritance. Write a java program to display 'Results' object.



Theory:

Inheritance

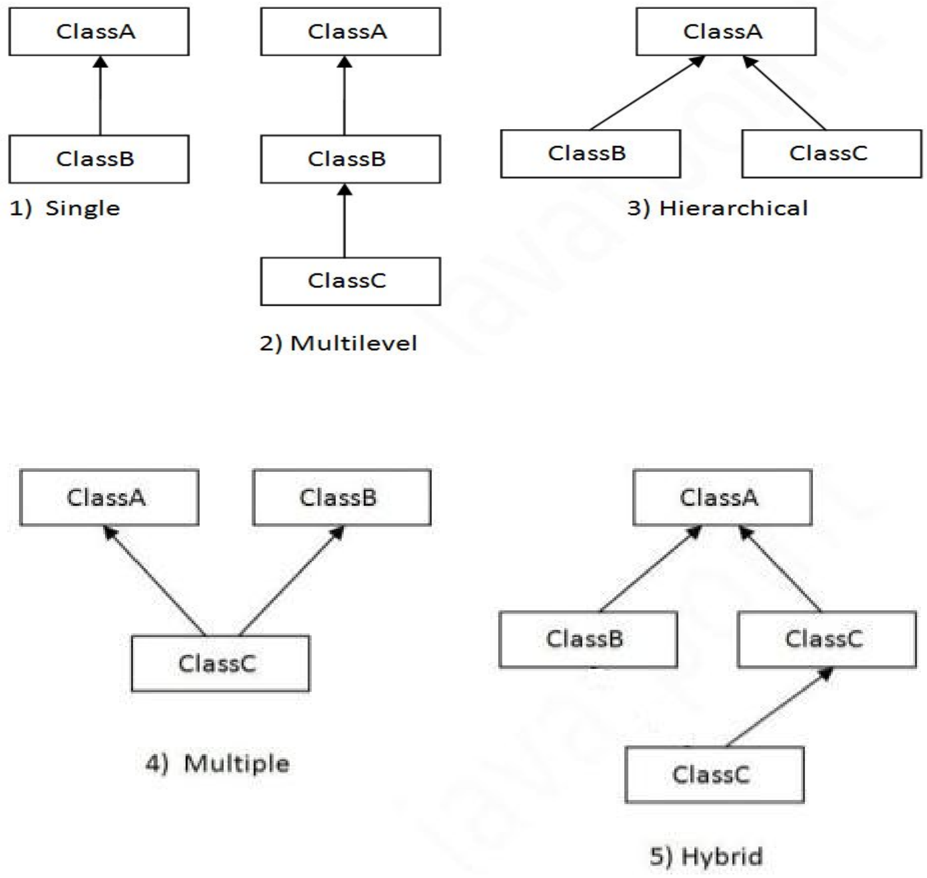
Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of parent object. The idea behind inheritance in java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of parent class, and you can add new methods and fields also. Inheritance represents *parent-child* relationship. The extends keyword indicates that a new class is derived from an existing class. In Java, a class that is inherited is called a super class. The new class is called a subclass.

Why use inheritance in java?

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical. In java programming, multiple and hybrid inheritance is supported through interface only.

Note: Multiple inheritance is not supported in java through class.



<u>**Abstraction in Java:**</u>

Abstraction is a process of hiding the implementation details and showing only functionality to the user. Another way, it shows only important things to the user and hides the internal details for example sending sms, you just type the text and send the message.

You don't know the internal processing about the message delivery. Abstraction lets you focus on what the object does instead of how it does it.

<u>Ways to achieve Abstraction</u>

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

<u>Abstract class in Java</u>

A class that is declared as abstract is known as an abstract **class**. It can have abstract and non-abstract methods (method with body). It needs to be extended and its method implemented. It cannot be instantiated.

Example

abstract class A{}

Abstract method

A method that is declared as abstract and does not have implementation is known as abstract method.

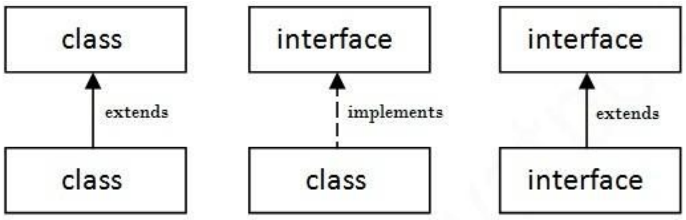
Example

abstract void printStatus();//no body and abstract

Interfaces

An **interface in java** is a blueprint of a class. It has static constants and abstract methods only. The interface in java is **a mechanism to achieve fully abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve fully abstraction and multiple inheritance in Java.

As shown in the figure given below, a class extends another class, an interface extends another interface but a **class implements an interface**.



Conclusion:Thus we have implemented a java program for the class network given below for multilevel inheritance to display 'Results' object.

Program

```
class Student     //super class

{
        protected String name;
        protected int roll_no;
        Student(String n, int r)  //parameterized constructor
        {
                name = n;
                roll_no = r;
        }

        public void display()
        {
                System.out.println("Name of Student is: "+name+"\nRoll No. is: "+roll_no);
        }
}
```

```
class Exam extends Student        //intermediate super class
{
        protected int marks1,marks2,marks3;
        Exam(String n, int r, int m1, int m2, int m3)     //parameterized constructor
        {
                super(n,r);
                marks1 = m1;
                marks2 = m2;
                marks3 = m3;
        }

        public void display()
        {
                super.display();
                System.out.println("Marks in Subject1 : "+marks1);
                System.out.println("Marks in Subject2 : "+marks2);
                System.out.println("Marks in Subject3 : "+marks3);
        }
}
class Result extends Exam                  //sub class
{
        protected int total;
        Result(String n,int r,int m1, int m2, int m3) //parameterized constructor
        {
                super(n,r,m1,m2,m3);
                total = m1+m2+m3;
        }

        public void display()
        {
                super.display();
                System.out.println("Total marks : "+total);
        }
}

class StudentResult
{
        public static void main( String args[])
                {
                        Result m=new Result("ABC",1,78,65,72);
                        m.display();
                }
}
```

```
/*OUTPUT:

Name of Student is: ABC

Roll No. is: 1

Marks in Subject1 : 78

Marks in Subject2 : 65
```

Marks in Subject3 : 72

Total marks : 215

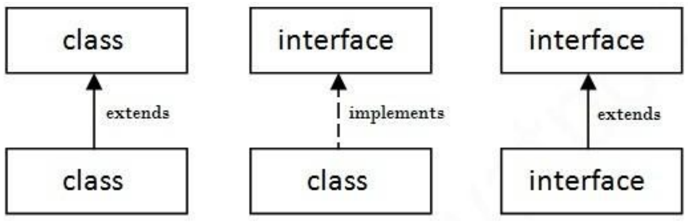
Experiment No. 10

Title: Write a java program to create user defined package. Define interface for Area and Volume, and implement the required interfaces in Circle class and Sphere class.

Theory:

Interfaces

An **interface in java** is a blueprint of a class. It has static constants and abstract methods only. The interface in java is **a mechanism to achieve fully abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve fully abstraction and multiple inheritance in Java. As shown in the figure given below, a class extends another class, an interface extends another interface but a **class implements an interface**.

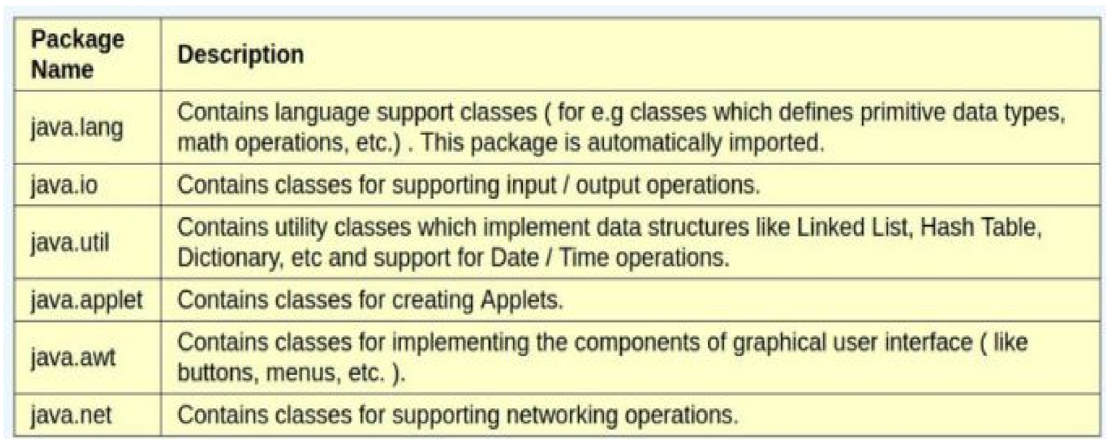


Packages

A package is a collection of related classes. It helps Organize your classes into a folder structure and make it easy to locate and use them. More importantly, it helps improve reusability.

Types of package:

1. **Built in** package: The already defined package like java.io.*, java.lang.* etc are known as built in packages.



Package Name	Description
java.lang	Contains language support classes (for e.g classes which defines primitive data types, math operations, etc.) . This package is automatically imported.
java.io	Contains classes for supporting input / output operations.
java.util	Contains utility classes which implement data structures like Linked List, Hash Table, Dictionary, etc and support for Date / Time operations.
java.applet	Contains classes for creating Applets.
java.awt	Contains classes for implementing the components of graphical user interface (like buttons, menus, etc.).
java.net	Contains classes for supporting networking operations.

2. User defined package: The package we create is called user defined package.

<u>Creating packages</u>

Include the ‘package’ keyword followed by the name of the package as the first statement in java source file.

Syntax: package package_name;

<u>import keyword</u>

import keyword is used to import builtin and user defined packages into your java source file. So that your class can refer to a class that is in another package by directly using its name.

There are **3 different ways** to refer to class that is present in different package:

1. Using fully qualified name ***(But this is not a good practice.)***

Example :

```
class MyDate extends java.util.Date
{       //statement;
}
```

2. import the **only class** you want to use. Example :

```
  import java.util.Date;
class MyDate extends Date
          {       //statement.
          }
```

3. import **all the classes** from the particular package

Example :

```
import java.util.*;
class MyDate extends Date
{         //statement;
}
```

Note: import statement is kept after the package statement.

```
Example : package mypack;
          Import java.util.*;
```

But if you are not creating any package then import statement will be the first statement of your java source file.

Conclusion: Thus we have implemented a java program to create user defined package and defined interface for Area and Volume, and implemented the required interfaces in Circle class and Sphere class.

```
/*import java.util.Scanner;

interface Area

{    double pi=3.142;

     void compute();

}


interface Volume

{    double c1 = 4.0;

     double c2 = 4.0/3;

     void compute();

}


class Circle implements Area

{    double r;

public void compute()

     {
            System.out.println("Area of circle  = "+(pi*r*r));
      }
}


class Sphere implements Area, Volume
{     double r;
      public void compute()
      {     System.out.println("Area of sphere  = "+(c1*pi*r*r));
            System.out.println("Volume of sphere  = "+(c2*pi*r*r*r));
      }
}


class InterfaceDemo1
{      public static void main(String args[])
       {    Scanner sc = new Scanner(System.in);
            Circle c = new Circle();
            Sphere s = new Sphere();

            System.out.println("Enter the radius of circle : ");
            c.r = sc.nextDouble();
            c.compute();

            System.out.println("Enter the radius of sphere : ");
            s.r = sc.nextDouble();
            s.compute();


      }
}


*/
```

Program

```
package myPackage;
interface Area
{           double pi = 3.142;// by default public static and final
            void compute();
}
public class Circle implements Area
{           public double r;
            public void compute()
            {           System.out.println("Area of circle  = "+(pi*r*r));
            }
}
package myPackage;
interface Area
{           double pi = 3.142;
            void compute();
```

```
}


interface Volume
{           void compute();
}
public class Sphere implements Area, Volume
{           public double r;
            public void compute()
            {           System.out.println("Area of sphere  = "+(4.0*pi*r*r));
                        System.out.println("Volume of sphere  = "+(4.0/3*pi*r*r*r));
            }
}


import java.util.Scanner;
import myPackage.*;
class InterfaceDemo
{           public static void main(String args[])
            {           Scanner sc = new Scanner(System.in);
                        Circle c = new Circle();
                        Sphere s = new Sphere();

                        System.out.println("Enter the radius of circle : ");
                        c.r = sc.nextDouble();
                        c.compute();
```

```
                System.out.println("Enter the radius of sphere : ");

                s.r = sc.nextDouble();

                s.compute();


        }

}
```

Output:

```
Enter the radius of circle :

2

Area of circle  = 12.568

Enter the radius of sphere :

3

Area of sphere  = 113.112

Volume of sphere  = 113.1119999999999
```

Experiment No. 11

Title: Write a program to accept and display the month number. Throw an exception if improper month number is entered. Make your own exception class to handle this exception.

Theory:

Exception Handling

Exception handling in java is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained.

What is an exception?

-Dictionary Meaning: Exception is an abnormal condition.
-In java, exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

What is exception handling?

-Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, etc.
-The core advantage of exception handling is to maintain the normal flow of the application. Exception normally disrupts the normal flow of the application that is why we use exception handling.

Types of Exception

There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception. The sun microsystem says there are three types of exceptions:

1. Checked Exception - The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g.IOException, SQLException etc. Checked exceptions are checked at compile-time.

2. Unchecked Exception - The classes that extend RuntimeException are known as unchecked exceptions. E.g. ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

3. Error - It is irrecoverable e.g. OutOfMemoryError, AssertionError etc.

There are 5 keywords used in java exception handling:

1. try - Java try block is used to enclose the code that might throw an exception. It must be used within the method. Java try block must be followed by either catch or finally block.
2. catch - Java catch block is used to handle the Exception. It must be used after the try block only. You can use multiple catch blocks with a single try.
3. finally
4. throw
5. throws

Conclusion: Thus we have implemented a java program to accept and display the month number and throw an exception if improper month number is entered and created our own exception class to handle this exception.

Program

```
import java.util.Scanner;
class MonthException extends Exception
{
        String msg;
        MonthException()
        {       msg = new String("MonthException : Enter a number between 1 and 12");
        }
}

class MonthExceptionDemo
{
  public static void main (String args [])
  {     Scanner sc = new Scanner(System.in);
        System.out.print("Enter month number : ");
        int x = sc.nextInt();
```

```
            try{
                  if (x >= 1 && x <=12)
                        System.out.println(x + " is a valid month number");
                  else
                        throw new MonthException();
             }
            catch (MonthException e)
            {
                  System.out.println (e.msg);
            }
      }
}
```

Output:

```
Enter month number : 4
4 is a valid month number


Enter month number : 15
MonthException : Enter a number between 1 and 12
```

Experiment No. 12

Title: Write a java program to print 1A2B3C4D5E6F7G8H9I10J using two child threads.

Theory:

Multithreading

Multithreading in java is a process of executing multiple threads simultaneously. Thread is basically a lightweight sub-process, a smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking. But we use multithreading than multiprocessing because threads share a common memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Advantage of Java Multithreading

1) It doesn't block the user because threads are independent.

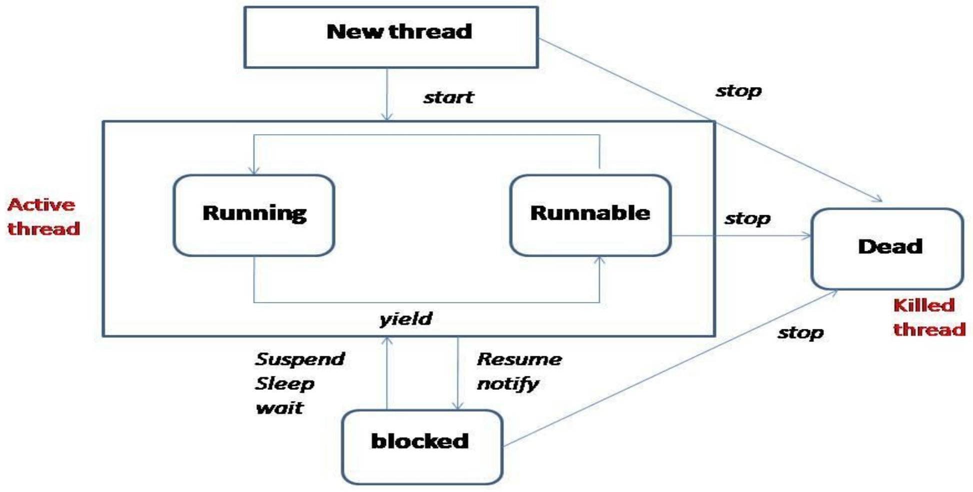
2) You can perform many operations together so it saves time.

3) Threads are independent so it doesn't affect other threads if exception occurs in a single thread.

Life Cycle of a Thread

The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:

1. New
2. Runnable
3. Running
4. Non-Runnable (Blocked)
5. Terminated



There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

Thread Class vs Runnable Interface

1. If we extend the Thread class, our class cannot extend any other class because Java doesn't support multiple inheritance. But, if we implement the Runnable interface, our class can still extend other base classes.

2. We can achieve basic functionality of a thread by extending Thread class because it provides some inbuilt methods like yield(), interrupt() etc. that are not available in Runnable interface.

Thread Methods:

Following is the list of important methods available in the Thread class.

SN	Methods with Description
1	**public void start()** Starts the thread in a separate path of execution, then invokes the run() method on this Thread object.
2	**public void run()** If this Thread object was instantiated using a separate Runnable target, the run() method is invoked on that Runnable object.
3	**public final void setName(String name)** Changes the name of the Thread object. There is also a getName() method for retrieving the name.
4	**public final void setPriority(int priority)** Sets the priority of this Thread object. The possible values are between 1 and 10.
6	**public final void join(long millisec)** The current thread invokes this method on a second thread, causing the current thread to block until the second thread terminates or the specified number of milliseconds passes.
7	**public void interrupt()** Interrupts this thread, causing it to continue execution if it was blocked for any reason.
8	**public final boolean isAlive()** Returns true if the thread is alive, which is any time after the thread has been started but before it runs to completion.

The previous methods are invoked on a particular Thread object. The following methods in the Thread class are static. Invoking one of the static methods performs the operation on the currently running thread.

SN	Methods with Description
1	**public static void yield()** Causes the currently running thread to yield to any other threads of the same priority that are waiting to be scheduled.
2	**public static void sleep(long millisec)** Causes the currently running thread to block for at least the specified number of milliseconds.

Conclusion: **Thus we have implemented a java program to print 1A2B3C4D5E6F7G8H9I10J using two child threads.**

Program

```
// Creating a thread by extending Thread class
class A extends Thread
{
        public void run()
        {       for(int i=1;i<=28;i++)
                {
                        System.out.print(i+" ");
                        try
                        {       Thread.sleep(500);
                        }
                        catch(Exception e)
                        {}
                }
        }
}

// Creating a thread by implementing Runnable interface
class B implements Runnable
{
        public void run()
        {       for(int i=1;i<=26;i++)
```

```
                {
                        System.out.print((char)(i+64)+" ");
                        try
                        {       Thread.sleep(550);
                        }
                        catch(Exception e)
                        {}
                }
        }
}
class ThreadDemo
{       public static void main(String args[])
        {       A a =new A();
                a.start();

                B b =new B();
                Thread t=new Thread(b);
                t.start();
        }
}
```

Output:

```
1 A 2 B 3 C 4 D 5 E 6 F 7 G 8 H 9 I 10 J 11 K 12 13 L 14 M 15 N 16 O 17 P 18 Q 19 R 20 S 21 T 22
U 23 V 24 25 W 26 X 27 Y 28 Z
```

Experiment 13

Title: Write an applet to display a smiley face. Also display a welcome message by passing parameters to the applet.

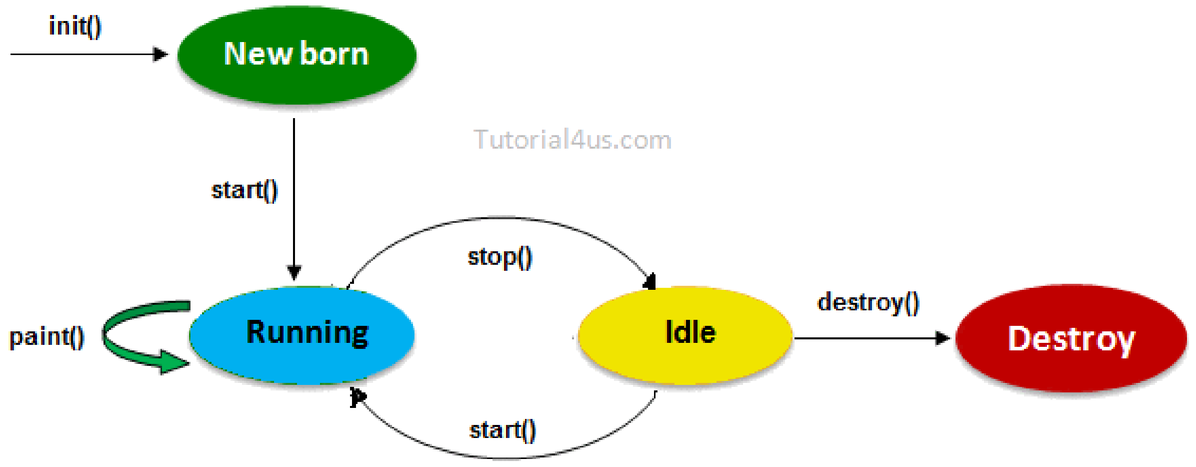
Theory:

Applets

An applet is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal. There are some important differences between an applet and a standalone Java application, including the following:

- An applet is a Java class that extends the java.applet.Applet class.
- A main() method is not invoked on an applet, and an applet class will not define main().
- Applets are designed to be embedded within an HTML page.
- When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.
- A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.
- The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime.
- Applets have strict security rules that are enforced by the Web browser. The security of an applet is often referred to as sandbox security, comparing the applet to a child playing in a sandbox with various rules that must be followed.
- Other classes that the applet needs can be downloaded in a single Java Archive (JAR) file.

Life Cycle of an Applet:



Four methods in the Applet class give you the framework on which you build any serious applet:

- **init:** This method is intended for whatever initialization is needed for your applet. It is called after the param tags inside the applet tag have been processed.
- **start:** This method is automatically called after the browser calls the init method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages.
- **stop:** This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.
- **destroy:** This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet.
- **paint:** Invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt.

The java.applet.Applet class 4 life cycle methods and java.awt.Component class provides 1 life cycle methods for an applet.

<u>java.applet.Applet class</u>

For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.

1. **public void init():** is used to initialized the Applet. It is invoked only once.
2. **public void start():** is invoked after the init() method or browser is maximized. It is used to start the Applet.
3. **public void stop():** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
4. **public void destroy():** is used to destroy the Applet. It is invoked only once.

There are two ways to run an applet

1. By html file.
2. By appletViewer tool (for testing purpose).

Conclusion:Thus we have implemented an applet to display a smiley face. Also displayed a welcome message by passing parameters to the applet.

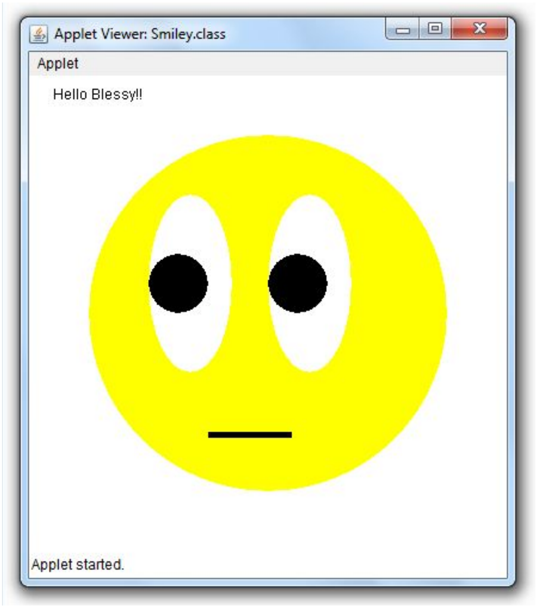
Program:

Write an applet to display a smiley face. Also display a welcome message by passing parameters to the applet.

```
import java.applet.*;
import java.awt.*;
public class Smiley extends Applet
{         String str;
          public void init()
          {         str = getParameter("text");
                    if(str == null)
                    {         str = "User";
                    }
                    str = "Hello "+ str +"!!";
          }
          public void paint (Graphics g)
          {         g.drawString(str, 20, 20);
                    g.setColor(Color.yellow);
                    g.fillOval(50,50,300,300);      /*face*/
                    g.setColor(Color.white);
                    g.fillOval(100,100,70,150);    /*border left*/
                    g.setColor(Color.black);
                    g.fillOval(100,150,50,50);      /*eyeleft*/
                    g.setColor(Color.white);
                    g.fillOval(200,100,70,150);    /*border right*/
                    g.setColor(Color.black);
                    g.fillOval(200,150,50,50);      /*eyeright*/
                    g.fillRect(150,300,70,5);
          }
}

 <applet CODE = "Smiley.class"      WIDTH = 400    HEIGHT = 400>
  <param name="text" value="Welcome to the world of Applets">
  </applet>
```

OUTPUT:

```
D:\bles>javac Smiley.java
D:\bles>appletviewer Smiley.java
```

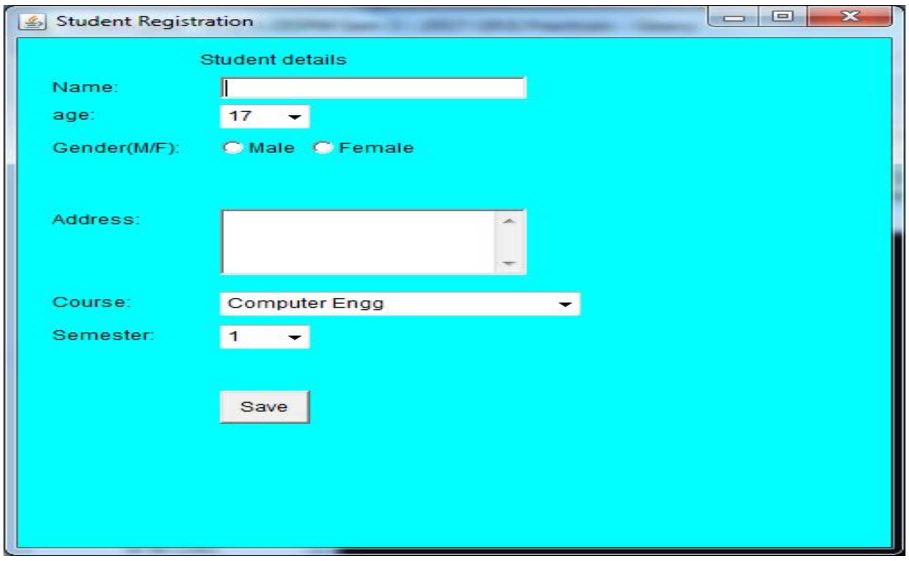


```
Hello Blessy!!
```

```
Applet started.
```

Experiment 14

Title: Write a java program to display the following GUI Form to accept Student's Information using AWT components (TextField, TextArea Button, Label, etc) and creating frame to add all the components and event handling.


Student Registration

Student details

Name:

age: 17

Gender(M/F): Male Female

Address:

Course: Computer Engg

Semester: 1

Save

java.awt.Component class

The Component class provides 1 life cycle method of applet.

1. **public void paint(Graphics g):** is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

Every user interface considers the following three main aspects:

- **UI elements** : These are the core visual elements the user eventually sees and interacts with.
- **Layouts:** They define how UI elements should be organized on the screen and provide a final look and feel to the GUI.
- **Behavior:** These are events which occur when the user interacts with UI elements.

Component

A Component is an abstract super class for GUI controls and it represents an object with graphical representation.

AWT UI Elements:

Following is the list of commonly used controls while designed GUI using AWT.

- **Label**

A Label object is a component for placing text in a container.

- **Button**

This class creates a labeled button.

- **Check Box**

A check box is a graphical component that can be in either an **on**(true) or **off** (false) state.

- **List**

The List component presents the user with a scrolling list of text items.

- **Text Field**

A TextField object is a text component that allows for the editing of a single line of text.

Text Area

A TextArea object is a text component that allows for the editing of a multiple lines of text.

- **Canvas**

A Canvas control represents a rectangular area where application can draw something or can receive inputs created by user.

- **Image**

An Image control is superclass for all image classes representing graphical images.

- **Scroll Bar**

A Scrollbar control represents a scroll bar component in order to enable user to select from range of values.

- **Dialog**

A Dialog control represents a top-level window with a title and a border used to take some form of input from the user.

- **File Dialog**

A FileDialog control represents a dialog window from which the user can select a file.

Event handling

Event handling is fundamental to Java programming because it is integral to the creation of applets and other types of GUI-based programs.

Events

In the delegation model, an event is an object that describes a state change in a source. It can be generated as a consequence of a person interacting with the elements in a graphical user interface.

Event Sources

A source is an object that generates an event. This occurs when the internal state of that object changes in some way. Sources may generate more than one type of event. A source must register listeners in order for the listeners to receive notifications about a specific type of event.

Event Listeners
A listener is an object that is notified when an event occurs.

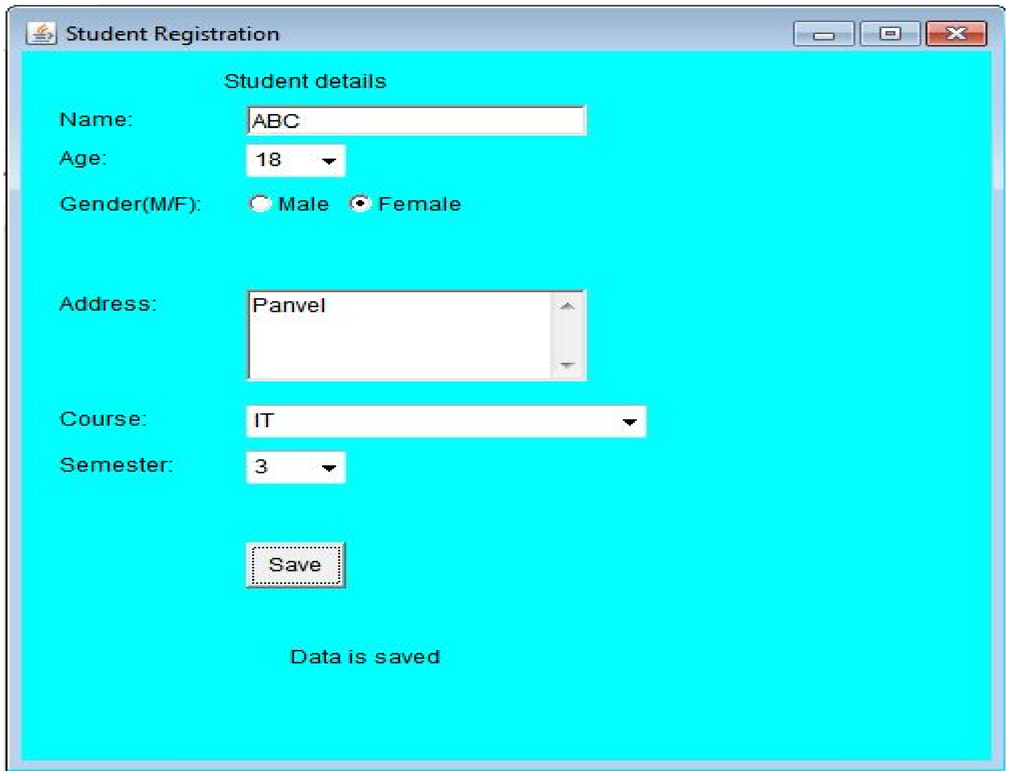
Event Classes
The classes that represent events are at the core of Java's event handling mechanism. For example, ActionEvent, KeyEvent and MouseEvent etc.

Conclusion: Thus we have implemented a java program to display the GUI Form to accept Student's information using AWT components (TextField, TextArea Button, Label, etc) and creating frame to add all the components and event handling.

Program: WAP to display the following GUI Form to accept Student's information using AWT components (TextField, TextArea Button, Label, etc) and creating frame to add all the components and event handling.

```
import java.awt.*;
import java.awt.event.*;

public class AWTGUI extends Frame implements ActionListener
{
  Button b1=new Button("Save");
  Label l1=new Label("Name:",Label.LEFT);
  Label l2=new Label("Age:",Label.LEFT);
  Label l3=new Label("Gender(M/F):",Label.LEFT);
  Label l4=new Label("Address:",Label.LEFT);
  Label l5=new Label("Course:",Label.LEFT);
  Label l6=new Label("Semester:",Label.LEFT);
  Label l7=new Label("Student details",Label.CENTER);
  Label label= new Label();

  TextField t1=new TextField();

  Choice c1=new Choice();

  CheckboxGroup cbg=new CheckboxGroup();

  Checkbox ck1=new Checkbox("Male",false,cbg);
  Checkbox ck2=new Checkbox("Female",false,cbg);

  TextArea t2 = new TextArea("",180,90,TextArea.SCROLLBARS_VERTICAL_ONLY);

  Choice course=new Choice();
  Choice sem=new Choice();
  Choice age=new Choice();
  public AWTGUI()
   {
          setBackground(Color.cyan);
          setForeground(Color.black);
          setLayout(null);

    add(l1);add(l2);add(l3);add(l4);add(l5);add(l6);add(l7);
    add(t1);add(t2);add(ck1);add(ck2);
          add(course);add(sem);add(age);add(b1);
          course.add("Computer Engg");course.add("IT");course.add("Mech");course.add("AUTO");
          sem.add("1");sem.add("2");sem.add("3");sem.add("4");sem.add("5");sem.add("6");
          sem.add("7");sem.add("8");
          age.add("17");age.add("18");age.add("19");age.add("20");age.add("21");
          add(label);

          l1.setBounds(25,65,90,20);l2.setBounds(25,90,90,20);l3.setBounds(25,120,90,20);
          l4.setBounds(25,185,90,20);l5.setBounds(25,260,90,20);l6.setBounds(25,290,90,20);
          l7.setBounds(10,40,280,20);

          t1.setBounds(120,65,170,20);t2.setBounds(120,185,170,60);

          ck1.setBounds(120,120,50,20);ck2.setBounds(170,120,60,20);


course.setBounds(120,260,200,20);sem.setBounds(120,290,50,20);age.setBounds(120,90,50,20);
          b1.setBounds(120,350,50,30);
          label.setBounds(140,400,90,50);

          b1.addActionListener(this);

}

public void actionPerformed(ActionEvent ae)
{
if(ae.getActionCommand().equals("Save"))
{
label.setText("Data is saved");
}
}
public static void main(String args[])
{
          AWTGUI stu=new AWTGUI();
          stu.setSize(500,500);
          stu.setTitle("Student Registration");
          stu.setVisible(true);
          }
}
```

Output



Student Registration

Student details

Name:	ABC
Age:	18
Gender(M/F):	Male [✓ Female]
Address:	Panvel
Course:	IT
Semester:	3

Save

Data is saved

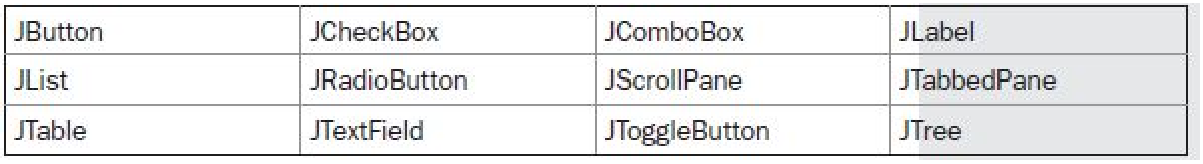
Experiment 15

Title: Write a program to create a window with four text fields for the name, street, city and zipcode with suitable labels. Also windows contains a button MyInfo. When the user types the name, his street, city and pincode and then clicks the button,the typed details must appear in Arial Font with Size 32, Italics.

Theory:

Java Swing is a part of Java Foundation Classes (JFC) that is used to create window-based applications. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java. Unlike AWT, Java Swing provides platform-independent and lightweight components. Several Swing components, such as buttons, check boxes, trees, and tables. The Swing components provide rich functionality and allow a high level of customization. These components are all lightweight, which means that they are all derived from JComponent.

The Swing component classes are:



JButton	JCheckBox	JComboBox	JLabel
JList	JRadioButton	JScrollPane	JTabbedPane
JTable	JTextField	JToggleButton	JTree

Difference between AWT and Swing

There are many differences between java awt and swing that are given below.

No.	Java AWT	Java Swing
1)	AWT components are **platform-dependent**.	Java swing components are **platform-independent**.
2)	AWT components are **heavyweight**.	Swing components are **lightweight**.
3)	AWT **doesn't support pluggable look and feel**.	Swing **supports pluggable look and feel**.
4)	AWT provides **less components** than Swing.	Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.

5)	AWT **doesn't follows MVC**(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing **follows MVC**.

Commonly used Methods of Component class

The methods of Component class are widely used in java swing that are given below.

Method	**Description**
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.
public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

Conclusion: Thus we have implemented a java program to create a window with four text fields for the name, street, city and zipcode with suitable labels using Swings

Program:

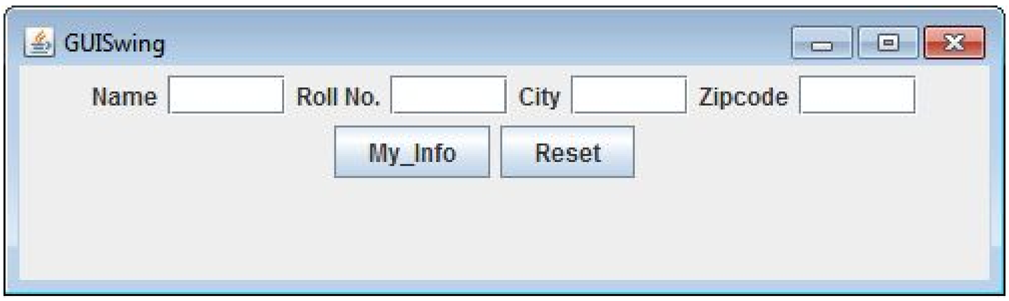
```
import javax.swing.*;

import java.awt.*;

import java.awt.event.*;

class GUISwing1 implements ActionListener

{

        JLabel l1,l2,l3,l4;
```

```
JTextField tf1,tf2,tf3,tf4;
GUISwing1()
{ //Creating the Frame
JFrame frame = new JFrame("GUISwing");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //JFrame.HIDE_ON_CLOSE
                                                       //JFrame.DO_NOTHING_ON_CLOSE

frame.setSize(500,500);
//Creating the panel at bottom and adding components
JPanel panel1 = new JPanel(); // the panel is not visible in output

JLabel label1 = new JLabel("Name");
tf1 = new JTextField(5); // initial space for 5 characters
JLabel label2 = new JLabel("Roll No.");
tf2 = new JTextField(5);
JLabel label3 = new JLabel("City");
tf3 = new JTextField(5);
JLabel label4 = new JLabel("Zipcode");
tf4 = new JTextField(5);

JButton b = new JButton("My_Info");
JButton reset = new JButton("Reset");
   l1= new JLabel("");
   l2= new JLabel("");
   l3= new JLabel("");
   l4= new JLabel("");
panel1.add(label1); // Components Added using Flow Layout
panel1.add(tf1);
panel1.add(label2); // Components Added using Flow Layout
```

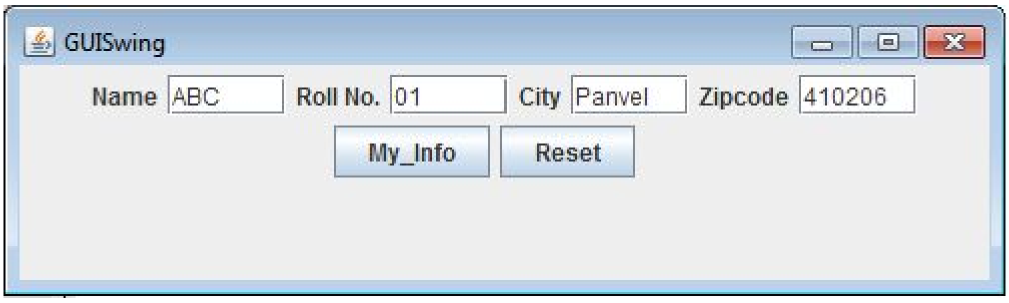
```
panel1.add(tf2);
panel1.add(label3);
panel1.add(tf3);
panel1.add(label4);
panel1.add(tf4);
panel1.add(b);
panel1.add(reset);
 panel1.add(l1);
panel1.add(l2);
panel1.add(l3);
panel1.add(l4);

//Adding Components to the frame.
frame.getContentPane().add(BorderLayout.CENTER, panel1);
frame.setVisible(true);
b.addActionListener(this);
reset.addActionListener(this);
 }

public void actionPerformed(ActionEvent ae)
   {
            if(ae.getActionCommand().equals("My_Info"))
            {
                     String s1,s2,s3,s4;
                     s1 = tf1.getText();
                     tf1.setFont(new Font("Arial", Font.ITALIC, 16));
                     tf1.setText(s1);

                     s1 = tf2.getText();
```

```
                    tf2.setFont(new Font("Arial", Font.ITALIC, 16));
                    tf2.setText(s1);

                    s1 = tf3.getText();
                    tf3.setFont(new Font("Arial", Font.ITALIC, 16));
                    tf3.setText(s1);

                    s1 = tf4.getText();
                    tf4.setFont(new Font("Arial", Font.ITALIC, 16));
                    tf4.setText(s1);
               }
               else if(ae.getActionCommand().equals("Reset"))
               {
                    tf1.setText(" ");
                    tf1.setFont(new Font("Dialog",Font.PLAIN ,12));
                    tf2.setText(" ");
                    tf2.setFont(new Font("Dialog",Font.PLAIN ,12));
                    tf3.setText(" ");
                    tf3.setFont(new Font("Dialog",Font.PLAIN ,12));
                    tf4.setText(" ");
                    tf4.setFont(new Font("Dialog",Font.PLAIN ,12));
               }
          }
          public static void main(String args[])
         {
          GUISwing1 stu=new GUISwing1();
         }
    }
```

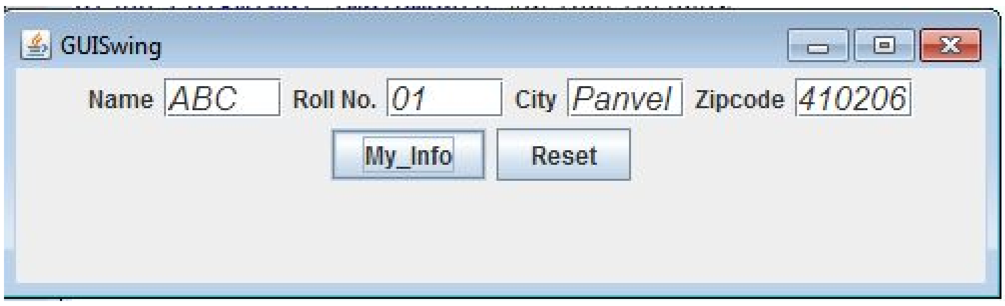
Output:


GUISwing

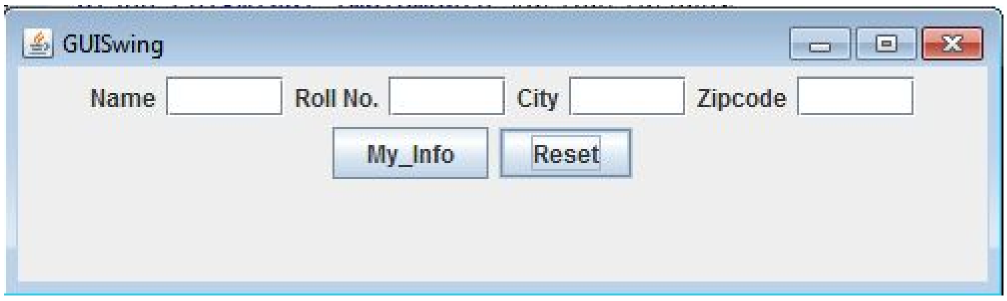
Name Roll No. City Zipcode


GUISwing

Name ABC Roll No. 01 City Panvel Zipcode 410206


GUISwing

Name *ABC* Roll No. *01* City *Panvel* Zipcode *410206*


GUISwing

Name Roll No. City Zipcode