Aim: Write a Java program to print all the real roots of a quadratic equation. If there are no real roots, display an appropriate message.

Description: A quadratic equation is of the form . The nature of the roots of a quadratic equation can be found out using the discriminant . If d>0, then the roots are real and distinct. If b=0, then the roots are real and equal. If b<0, then the roots are imaginary. The roots can be found out using the formula .

Program:

```
import java.util.*;
class Quadratic{
	public static void main(String args[]){
		int a,b,c;
		float d;
		Scanner s=new Scanner(System.in);
		System.out.println("Enter the values of a,b,c:");
		a=s.nextInt();
		b=s.nextInt();
		c=s.nextInt();
		d=b*b-4*a*c;
		if(d<0)
			System.out.println("There are no real solutions for this Quadratic
equation");
		else{
			double r1,r2;
			r1=(-b+Math.sqrt(d))/(2*a);
			r2=(-b-Math.sqrt(d))/(2*a);
			System.out.println("The roots of the Quadratic equation are: "+r1+",
"+r2);
		}
	}
}
```

Output:

```
Enter the values of a,b,c:
1 -5 6
The roots of the Quadratic equation are: 3.0, 2.0
```

Aim: Write a Java program to find the number in the Fibonacci series using non-recursive functions.

Description: The Fibonacci series is the following sequence of numbers:

Mathematically, the recurrence relation of the sequence can be represented by
where
This program generates the Fibonacci series using iterative function.

Program:

```
import java.util.*;
class FibNR{
        public static void main(String args[]){
                int f1=1,f2=1,f3=0,n;
                Scanner s=new Scanner(System.in);
                System.out.print("Enter n value: ");
                n=s.nextInt();
                for(int i=3;i<=n;i++){
                        f3=f1+f2;
                        f1=f2;
                        f2=f3;
                }
        System.out.println("The Fibonacci number at position "+n+" is "+f3);
        }
}
```

Output:

```
Enter n value: 25
The Fibonacci number at position 25 is 75025
```

Aim: Write a Java program to find the number in the Fibonacci series using recursive function.

Description: The Fibonacci series is the following sequence of numbers:

Mathematically, the recurrence relation of the sequence can be represented by
where
This program generates the Fibonacci series using recursion.

Program:

```
import java.util.*;
class FibR{
	public static void main(String args[]){
		int f1=1,f2=1,f3=0,n;
		Scanner s=new Scanner(System.in);
		System.out.print("Enter n value: ");
		n=s.nextInt();
		if(n==1||n==2)
			System.out.println("Result=1");
		else{
			f3=fibonacci(f1,f2,f3,n,3);
			System.out.println("Result="+f3);
		}
	}
	public static int fibonacci(int a,int b,int c,int p,int i){
		if(p<=i)
			return a+b;
		else{
			i++;
			c=a+b;
			return fibonacci(b,c,c,p,i);
		}
	}
}
```

Output:

```
Enter n value: 15
Result=610
```

Aim: Write a Java program that prompts the user for an integer and prints all the prime numbers up to that integer.

Description: A prime number is a natural number that has no other factors other than one and itself. Mathematically, *p* is a prime number if for all *i* other than 1 and *p*.

This program prints all the prime number up to a limit by checking each number for its divisibility from 2 to half of the number itself.

Program:

```
import java.util.*;
class Prime{
	public static void main(String args[]){
		Scanner s=new Scanner(System.in);
		System.out.print("Enter the limit:");
		int num=s.nextInt();
		System.out.println("Prime numbers from 1 to "+num);
		for(int i=2;i<=num;i++)
			if(isPrime(i))
				System.out.print(i+" ");
	}
	static boolean isPrime(int n){
		for(int i=2;i<=(n/2);i++)
			if(n%i==0)
				return false;
		return true;
	}
}
```

Output:

```
Enter the limit:25
Prime numbers from 1 to 25
2 3 5 7 11 13 17 19 23
```

Aim: Write a Java program to multiply two given matrices.

Description: Matrices can be represented using 2 dimensional arrays. In order to multiply two matrices, the number of columns in the first matrix must be equal to the number of rows in the second matrix. Matrix multiplication is carried out in iterative fashion. The matrices are fed into two source arrays and an additional array is used to store the resultant matrix.

Program:

```
import java.util.*;
class Matrix{
	public static void main(String args[]){
		int A[][],B[][],C[][],m,n,p,q,i=0,j=0,k=0;
		Scanner s=new Scanner(System.in);
		System.out.println("Enter order of matrix A:");
		m=s.nextInt();
		n=s.nextInt();
		System.out.println("Enter order of matrix B:");
		p=s.nextInt();
		q=s.nextInt();
		if(n!=p){
			System.out.println("Matrix multiplication not possible!");
			System.exit(0);
		}
		A=new int[m][n];
		B=new int[p][q];
		C=new int[m][q];
		System.out.println("Enter elements of matrix A:");
		for(i=0;i<m;i++)
			for(j=0;j<n;j++)
				A[i][j]=s.nextInt();
		System.out.println("Enter elements of matrix B:");
		for(i=0;i<p;i++)
			for(j=0;j<q;j++)
				B[i][j]=s.nextInt();
		for(i=0;i<m;i++)
			for(j=0;j<q;j++){
				C[i][j]=0;
				for(k=0;k<n;k++)
					C[i][j]+=A[i][k]*B[k][j];
			}
		System.out.println("The resultant matrix is: ");
		for(i=0;i<m;i++){
			for(j=0;j<q;j++)
				System.out.print(C[i][j]+"\t");
			System.out.println();
		}
	}
}
```

Output:

```
Enter order of matrix A:
2 3
Enter order of matrix B:
3 2
Enter elements of matrix A:
1 2 3
4 5 6
Enter elements of matrix B:
1 2
3 4
5 6
The resultant matrix is:
22      28
49      64
```

Aim: Write a Java program that reads a line of integers and then displays each integer and the sum of each integer.

Description: This program reads a line of integers as input and displays each integer separately along with the sum of all the integers. In order to achieve this functionality, the StringTokenizer class is used. The StringTokenizer splits the string into tokens based on a delimiter provided as an argument. The nextToken() function returns the next token in the string. The token is initially a String and must be converted into integer to get its value. This is done by utilising the parseInt() function of Integer class.

Program:

```
import java.util.*;
class Token
{
	public static void main(String[] args)
	{
		Scanner s=new Scanner(System.in);
		System.out.println("Enter the string: ");
		String str=s.nextLine();
		StringTokenizer st=new StringTokenizer(str," ");
		int num,sum=0;
		while(st.hasMoreTokens()){
			num=Integer.parseInt(st.nextToken());
			System.out.println(num);
			sum+=num;
		}
		System.out.println("The sum of the integers is "+sum);
	}
}
```

Output:

```
Enter the string:
12 14 17 11 15 5
12
14
17
11
15
5
The sum of the integers is 74.
```

Aim: Write a Java program that checks whether a given string is a palindrome or not without actually reversing the string.

Description: A palindrome is a word which read the same both left-to-right and right-to-left. For example, the word MADAM is a palindrome. In order to check a string for this property, we use the charAt() function of the String class to check the first and last character for equality and then the subsequent inner characters and so on. If each pair of characters is equal, then the string is a palindrome, else it isn't. _

Program:

```
import java.util.*;
class Palindrome{
        public static void main(String[] args){
                Scanner s1=new Scanner(System.in);
                System.out.println("Enter the string: ");
                String s=s1.nextLine();
                boolean isPal=true;
                int len=s.length();
                for(int i=0;i<(len/2);i++)
                        if(s.charAt(i)!=s.charAt(len-i-1)){
                                isPal=false;
                                break;
                        }
                if(isPal)
                        System.out.println(s+" is a palindrome.");
                else
                        System.out.println(s+" is not a palindrome!");
        }
}
```

Output:

```
Enter the string:
CIVIC
CIVIC is a palindrome.
```

Aim: Write a Java program for sorting a given list of names in ascending order.

Description: This program accepts an array of strings, sorts the array in ascending order and displays the sorted array. This program uses the compareTo() method of the String class. The compareTo() method returns a negative number if the invoking string is less than the passed string, a positive number if the it is greater than the passed string and 0 if the two strings are equal.

Program:

```
import java.io.*;
class StringSort
{
        public static void main(String[] args)throws IOException
        {
                int i=0,j=0;
                String t=null;
                BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
                System.out.print("Enter the number of strings: ");
                int n=Integer.parseInt(br.readLine());
                String s[]=new String[n];
                System.out.println("Enter the list of strings:");
                for(i=0;i<n;i++)
                        s[i]=br.readLine();
                for(i=0;i<n;i++)
                        for(j=i+1;j<n;j++)
                                if(s[i].compareTo(s[j])>0){
                                        t=s[i];
                                        s[i]=s[j];
                                        s[j]=t;
                                }
                System.out.println("\nThe list after sorting:");
                for(i=0;i<n;i++)
                        System.out.println(s[i]);
        }
}
```

Output:

```
Enter the number of strings: 5
Enter the list of strings:
kin
bat
dog
den
urn
```

```
The list after sorting:
bat
den
dog
kin
urn
```

Aim: Write a Java program to count the frequency of each word appearing in a given text.

Description: This program takes a string as input, splits it into words and counts the frequency of each word and displays it. In order to achieve this, the StringTokenizer class is used. The functions hasMoreTokens() and nextToken() are used to extract the tokens. The delimiters for separating the words are space, new line character and the tab character. A second StringTokenizer object is used to match the tokens and a counter is maintained to count the frequency of the words.

Program:

```
import java.util.*;
import java.io.*;
class WordCounter{
	public static void main(String[] args)throws Exception{
		BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
		String text=new String();
		String temp=new String();
		System.out.println("Enter the text(# on a new line to stop):");
		do{
			temp=br.readLine();
			text+=temp+" \n";
		}while(!temp.equals("#"));

		StringTokenizer st1=new StringTokenizer(text," \n\t#");
		int wordcount=0,tokencount=0;

		while(st1.hasMoreTokens()){
			wordcount=0;
			StringTokenizer st2=new StringTokenizer(text," \n\t#");
			String word1=st1.nextToken();
			if(isWordRepeated(text,word1,tokencount)){
				tokencount++;
				continue;
			}
			tokencount++;
			while(st2.hasMoreTokens()){
				String word2=st2.nextToken();
				if(word2.equals(word1))
				wordcount++;
			}
			System.out.println("Count of "+word1+" is "+wordcount);
		}
	}
	static boolean isWordRepeated(String text,String str, int count){
		StringTokenizer st=new StringTokenizer(text," \n\t#");
		for(int i=0;i<count;i++)
			if(st.nextToken().equals(str))
				return true;
		return false;
```

```
        }
}
```

Output:

```
Enter the text(# on a new line to stop):
bat cat rat mat
dog cat rat ant
bat bug mat dog
#

Count of bat is 2
Count of cat is 2
Count of rat is 2
Count of mat is 2
Count of dog is 2
Count of ant is 1
Count of bug is 1
```

Aim: Write a Java program that reads a filename from the user and then displays information about whether the file exists, whether the file is readable, whether the file is writable, the type of file and the length of the file in bytes.

Description: The File class in Java provides many methods to set and get various properties from a file. Properties like name, size, and read-only attribute can be get from the file. In Java, even directories can be handled using the File class.

Program:

```
import java.io.*;
class FileDetails{
	static void p(String str){
		System.out.println(str);
	}
	public static void main(String args[])throws IOException{
		BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
		p("Enter the file path: ");
		String s=br.readLine();
		File f=new File(s);
		if(f.exists()){
			if(f.isFile()){
				p(f.getName()+" is a file.\n");
				p(f.canRead()?"is readable.\n":"is not readable.\n");
				p(f.canWrite()?"is writeable.\n":"is not writeable.\n");
				int pos=(f.getName()).lastIndexOf('.');
				p("is a "+f.getName().substring(pos+1)+" file.\n");
				p("File size: "+f.length()+" bytes");
			}
			else if(f.isDirectory()){
				p(f.getName()+" is a directory.\n");
				p("List of files:\n");
				String dir[]=f.list();
				for(int i=0;i<dir.length;i++)
					p(dir[i]);
			}
		}
		else
			p("File/Directory not found!");
	}
}
```

Output:

```
Enter the file path: FileDetails.java
FileDetails.java is a file.
is readable.
is writeable.
is a java file.
File size: 915 bytes
```

Aim: Write a Java program that reads a file and displays the file on the screen, with a line number before each line.

Description: This program utilises the FileReader class and the LineNumberReader class of the java.io package. The LineNumberReader class provides a method getLineNumber() which returns the number of the current line. The line number along with the line is prined as output.

Program:

```
import java.io.*;
class LineNumbers{
	public static void main(String args[])throws IOException{
		BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
		System.out.println("Enter the file path: ");
		String name=br.readLine();
		try{
			FileReader fr=new FileReader(name);
			LineNumberReader ln=new LineNumberReader(fr);
			String s;
			while((s=ln.readLine())!=null)
				System.out.println(ln.getLineNumber()+". "+s);
			fr.close();
		}
		catch(FileNotFoundException e){
			System.out.println("Invalid file/File not found!");
		}
	}
}
```

Sample file: sample.txt

```
This is a
sample txt file.

This file is written
to check the output
of a Java program.
```

Output:

```
Enter the file path:
sample.txt

1. This is a
2. sample text file.
3.
4. This file is written
5. to check the output
6. of a Java program.
```

Aim: Write a Java program that displays the number of characters, lines and words in a text file.

Description: This program reads a file, counts the number of character, words and lines in the file and displays the count as output. Words are distinguished by the space or tab character and lines are distinguished by the new line character.

Program:

```
class FileCount{
        static int chars=0,lines=0,words=0;
        public static void main(String[] args)throws IOException{
                BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
                System.out.print("Enter the name of the file: ");
                String name=br.readLine();
                try{
                        FileReader fr=new FileReader(name);
                        count(fr);
                        fr.close();
                }
                catch(FileNotFoundException e){
                        System.out.println("Invalid file/File not found!");
                }
                System.out.println("Number of characters: "+chars);
                System.out.println("Number of words: "+words);
                System.out.println("Number of lines: "+lines);
        }
        static void count(InputStreamReader isr)throws IOException{
                int c=0;
                while((c=isr.read())!=-1){
                        chars++;
                if((c==' ')||(c=='\t')||(c=='\n'))
                        words++;
                if(c=='\n')
                        lines++;
                }
                lines+=1;//To count the last line without a \n character
        }
}
```

Output:

```
Enter the name of the file: sample.txt
Number of characters: 92
Number of words: 18
Number of lines: 6
```

Aim: Write a Java program that (i) Implements the stack ADT, (ii) Convert an infix expression to a postfix expression and (iii) Evaluates the postfix expression.

Description:

A stack is a linear data structure that follows the LIFO(Last In First Out) principle. The stack has 2 main methods PUSH and POP. The PUSH method inserts data into the stack. The POP method removes data from the stack. A stack can be implemented using arrays or linked-lists.

An infix expression follows the notation of placing the operators in between the operands they act on.

e.g. 2+3*8/6

In an infix expression, parentheses surrounding groups of operands and operators are necessary to indicate the intended order in which operations are to be performed. In the absence of parentheses, certain precedence rules determine the order of operations.

Precedence rules:
Highest
( )
/ * %
+ -
Lowest

A postfix expression follows the notation of placing the operators after the operands they act on. Postfix expressions are easy for the computers to parse and evaluate due to lack of parentheses and precedence rules.

e.g: 238*6/+

To convert an infix expression to a postfix expression, the shunting yard algorithm is commonly used.

The shunting yard algorithm requires a stack data structure for processing.

In this algorithm, the infix expression is scanned from left to right.

1. When a left parenthesis '(' is encountered it is pushed onto the stack.
2. When an operand is encountered, it is appended to the postfix expression.
3. When an operator is encountered,
   i. If the precedence of the operator on the top of the stack is greater than or equal to the precedence of the operator encountered, the stack is popped until the precedence of the operator on the top of the stack is less than the precedence of the operator encountered, and is appended to the postfix expression.
   ii. The encountered operator is pushed onto the stack.
4. When a right paranthesis ')' is encountered,
   i. the stack is popped until a left paranthesis '(' is encountered and is appended to the postfix expression.
   ii. the left paranthesis '(' is ignored.

A postfix expression is easily parsed and evaluated by a computer. To evaluate the postfix expression, a stack data structure is required.

The postfix expression is scanned from left to right.

1. When an operand is encountered, push it onto the stack.

2.When an operator is encountered, pop the two top most elements of the stack, perform the operation and push the result onto the stack.

Care must be taken while working with non-associative operations like -,/,%,etc.

Program:

```
import java.util.*;
//Define a stack class
class Stack{
	final int SIZE=20;
	//Stack is implemented using a String array. Cannot use char array because operands
//by contain more than one digit.
	String stk[]=new String[SIZE];
	int top=-1;
	void push(String s){
		if(top==SIZE)
			System.out.println("Stack Overflow!");
		else
			stk[++top]=s;
	}
	String pop(){
		if(top==-1){
			System.out.println("Stack Underflow!");
			return null;
		}
		return stk[top--];
	}
	//This method is used to return the top most element of the stack. But the element will
not be removed from the stack.
	String peek(){
		if(top==-1){
			System.out.println("Stack Underflow!");
			return null;
		}
		return stk[top];
	}
}
class Expression{
	//Create an object of Stack class.
	//All objects and methods in this class must be static because static method main()
//should access them.
	static Stack sk=new Stack();
	public static void main(String args[]){
		//Create a Scanner class object and read the infix expression.
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the infix expression: ");
		String infix=sc.next();
		String postfix=toPostfix(infix);
		System.out.println("Equivalent postfix expression: "+postfix);
		String result=evaluate(postfix);
```

```
                System.out.println("Result of postfix expression evaluation: "+result);
        }
        //This method takes an infix string as input and returns equivalent postfix expression.

        static String toPostfix(String infix){
                //The infix expression needs to be enclosed within a pair of parentheses for
//proper evaluation.
                infix+=")";
                sk.push("(");
                String postfix="";
                //This overloaded constructor of the StringTokenizer class takes an additional
parameter which when set to true returns the delimiters along with the tokens.
                //The possible operators and the two parentheses are supplied as delimiters.
                StringTokenizer st=new StringTokenizer(infix,"+-*/%()",true);
                while(st.hasMoreTokens()){
                        String tok=st.nextToken();
                        if(tok.equals("("))
                                sk.push(tok);
                        else if(isOperator(tok)){
                                while(precedence(sk.peek())>=precedence(tok))
                                        postfix+=sk.pop()+" ";
                                sk.push(tok);
                        }
                        else if(tok.equals(")")){
                                while(sk.peek()!="(")
                                        postfix+=sk.pop()+" ";
                                String temp=sk.pop();
                        }
                        else
                                postfix+=tok+" ";
                }
                return postfix;
        }
        //This method takes a postfix expression as input, evaluates it and returns the result.
        //The operators and operands in the postfix string given as input must be delimited
with spaces " ".
        static String evaluate(String postfix){
                //The delimiter for this StringTokenizer constructor is space " " character.
                StringTokenizer st=new StringTokenizer(postfix," ");
                while(st.hasMoreTokens()){
                        String tok=st.nextToken();
                        double a=0,b=0,c=0;
                        if(isOperator(tok)){
                                //Popped String needs to be converted into numeric data type
for arithmetic operations.
                                a=Double.parseDouble(sk.pop());
                                b=Double.parseDouble(sk.pop());
                                //The operator token is a String object and cannot be used with
switch(), and therefore must be converted into char.
                                char op=tok.charAt(0);
```

```
                                        switch(op){
                                                case '+':
                                                        c=b+a;
                                                        break;
                                                case '-':
                                                        c=b-a;
                                                        break;
                                                case '*':
                                                        c=b*a;
                                                        break;
                                                case '/':
                                                        c=b/a;
                                                        break;
                                                case '%':
                                                        c=b%a;
                                                        break;
                                        }
                                        //The evaluation result must be converted into String type
before pushing onto the stack.
                                        sk.push(Double.toString(c));
                                }
                                else
                                        sk.push(tok);
                        }
                        return sk.pop();
                }
                //This method takes a token as input and return true if the token is an operator else
returns false.
                static boolean isOperator(String token){


if(token.equals("*")||(token.equals("/"))||(token.equals("%"))||(token.equals("+"))||(token.equa
ls("-")))
                                return true;
                        return false;
                }
                //This method takes an operator as input and returns the precedence of the operator.
                static int precedence(String token){
                        if((token.equals("*"))||(token.equals("/"))||(token.equals("%")))
                                return 2;
                        if((token.equals("+"))||(token.equals("-")))
                                return 1;
                        return 0;
                }
}
```

Output:

```
Enter the infix expression:
12+3-6/8*7
```

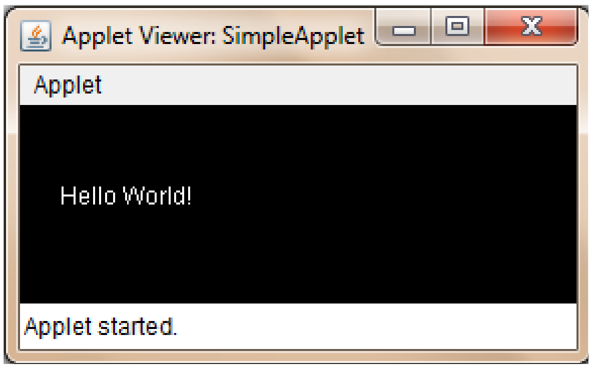
Equivalent postfix expression: 12 3 + 6 8 / 7 * -
Result of postfix expression evaluation: 9.75
Aim: Write a Java applet that displays a simple message

Description: An applet is a small Java program that runs inside a web browser. Applets are created by extending the Applet class or JApplet class in Java. A Java applet requires a GUI toolkit like AWT or Swing to display various GUI elements such as buttons, text fields, etc.

Program:

```
import java.awt.*;
import java.applet.*;
/*
<applet code="SimpleApplet" height="100" width="100">
</applet>
*/
public class SimpleApplet extends Applet{
        public void init(){
                setBackground(Color.black);
                setForeground(Color.white);
        }
        public void paint(Graphics g){
                g.drawString("Hello World!",20,50);
        }
}
```

Output:



```
Hello World!
```

Applet started.

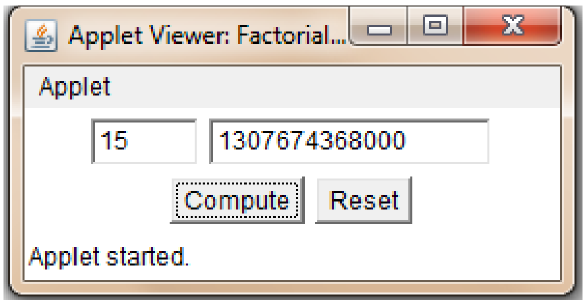
Aim: Write a Java applet that receives an integer in one text field, computes its factorial value and returns it in another text field when the button “Compute” is clicked.

Description: In this program, an applet is created with two text fields and a button. The first text field receives the input, parses it into an integer value, and displays the factorial of that value in the second text field. This program implements the ActionListener interface to define the method to be performed when the “Compute” button is clicked.

Program:

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="FactorialApplet" height="60" width="200">
</applet>
*/
public class FactorialApplet extends Applet implements ActionListener{
	TextField tx1,tx2;
	Button btn,rst;
	public void init(){
		tx1=new TextField(4);
		tx2=new TextField(16);
		btn=new Button("Compute");
		rst=new Button("Reset");
		add(tx1);
		add(tx2);
		add(btn);
		add(rst);
		btn.addActionListener(this);
		rst.addActionListener(this);
	}
	public void actionPerformed(ActionEvent ae){
		String str=ae.getActionCommand();
		if(str.equals("Compute")){
			int n=Integer.parseInt(tx1.getText().trim());
			long fact=1;
			for(int i=1;i<=n;i++)
				fact*=i;
			tx2.setText(Long.toString(fact));
		}
		if(str.equals("Reset")){
			tx1.setText("");
			tx2.setText("");
		}
	}
}
```

Output:



```
15    1307674368000
Applet started.
```

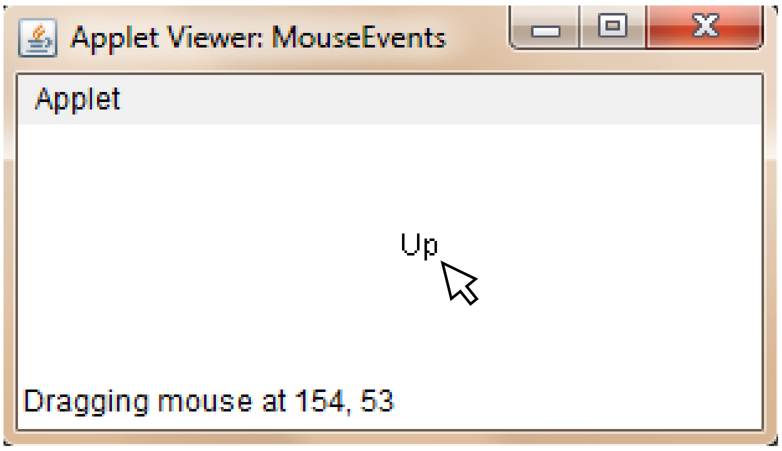
Aim: Write a Java program for handling mouse events.

Description: Mouse events can be handled by implementing the MouseListener and MouseMotionListener interfaces. Different mouse events like mouse entered, exited, moved, clicked, pressed, released and dragged can be handled by appropriately implementing the methods in the interfaces.

Program:

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="MouseEvents" width="300" height="100">
</applet>
*/
public class MouseEvents extends Applet implements MouseListener,MouseMotionListener{
        String msg="";
        int mouseX=0,mouseY=0;
        public void init(){
                addMouseListener(this);
                addMouseMotionListener(this);
        }
        public void mouseClicked(MouseEvent me){
                mouseX=0;
                mouseY=10;
                msg="Mouse Clicked!";
                repaint();
        }
        public void mouseEntered(MouseEvent me){
                mouseX=0;
                mouseY=10;
                msg="Mouse Entered!";
                repaint();
        }
        public void mouseExited(MouseEvent me){
                mouseX=0;
                mouseY=10;
                msg="Mouse Exited!";
                repaint();
        }
        public void mousePressed(MouseEvent me){
                mouseX=me.getX();
                mouseY=me.getY();;
                msg="Down";
                repaint();
        }
        public void mouseReleased(MouseEvent me){
                mouseX=me.getX();
                mouseY=me.getY();;
                msg="Up";
                repaint();
        }
        public void mouseDragged(MouseEvent me){
                mouseX=me.getX();
                mouseY=me.getY();;
                msg="*";
                showStatus("Dragging mouse at "+mouseX+", "+mouseY);
                repaint();
        }
        public void mouseMoved(MouseEvent me){
                showStatus("Moving mouse at "+me.getX()+", "+me.getY());
        }
        public void paint(Graphics g){
                g.drawString(msg,mouseX,mouseY);
        }
}
```

Output:



```
Dragging mouse at 154, 53
```

Aim: Write a Java program that creates three threads. First thread displays “Good Morning” every one second, second thread displays “Hello” every two seconds and the third thread displays “Welcome” every three seconds.

Description: A thread is a path for execution on the processor. A thread is light-weight and multiple threads can be run concurrently. Java provides built-in functionality for multi-threaded programming. In this program three threads are created each having a different sleep time. Also we have to ensure the proper order of execution of threads. This is achieved by thread priorities and pre-emption.

Program:

```
class Message extends Thread{
        String msg;
        long delay;
        Message(String msg,long delay){
                this.msg=msg;
                this.delay=delay;
        }
        public void run(){
                while(true){
                        try{
                                Thread.sleep(delay);
                        }catch(InterruptedException e){
                                System.out.println("Thread Execution Interruptred!");
                        }
                        System.out.println(msg);
                        Thread.yield();
                }
        }
}
class MessageMain{
        public static void main(String args[]){
                Message m1=new Message("Good Morning",1000);
                Message m2=new Message("Hello",2000);
                Message m3=new Message("Welcome",3000);
                m1.setPriority(Thread.MAX_PRIORITY);
                m2.setPriority(Thread.MAX_PRIORITY-1);
                m2.setPriority(Thread.MAX_PRIORITY-2);
                m1.start();
                m2.start();
                m3.start();
        }
}
```

Output:

```
Good Morning
```

Hello
Good Morning
Welcome
Good Morning
Hello
Good Morning
Good Morning
Hello
Welcome
Good Morning
Hello
Good Morning
Welcome
Good Morning
Hello
Good Morning
Good Morning

Aim: Write a Java program that correctly implements the producer consumer problem using the concept of inter-thread communication.

Description: The producer-consumer problem is a classical problem encountered in many real life situations. The producer and consumer share a common resource such as a buffer. The producer writes to the buffer and the consumer reads from it. The communication is constrained by two rules:

1. While the producer is accessing the buffer, the consumer should not access the buffer and vice versa.
2. The consumer must read the buffer only after the producer writes into buffer.

Program:

```
class Producer extends Thread{
        Buffer buf;
        Producer(Buffer b){
                buf=b;
        }
        public void run(){
                while(true)
                        buf.produce();
        }
}
class Consumer extends Thread{
        Buffer buf;
        Consumer(Buffer b){
                buf=b;
        }
        public void run(){
                while(true)
                        buf.consume();
        }
}
class Buffer{
        boolean value=false;
        int i=0;
        synchronized void produce(){
                while(value)
                        try{
                                wait();
                        }catch(InterruptedException e){}
                System.out.println("Producing Item "+(++i));
                value=true;
                notify();
        }
        synchronized void consume(){
                while(!value)
                        try{
```

```
                        wait();
                    }catch(InterruptedException e){}
                System.out.println("Consuming Item "+i);
                value=false;
                notify();
        }
}
class PC{
        public static void main(String args[]){
                Buffer b=new Buffer();
                new Producer(b).start();
                new Consumer(b).start();
        }
}
```

Output:

```
Producing Item 1
Consuming Item 1
Producing Item 2
Consuming Item 2
Producing Item 3
Consuming Item 3
Producing Item 4
Consuming Item 4
Producing Item 5
Consuming Item 5
Producing Item 6
Consuming Item 6
```

Aim: Write a Java program that creates a user interface to perform integer divisions. The user enters two numbers in two text fields, Num1 and Num2. The result of division is displayed in a third text field, Result when the "Divide" button is clicked. If Num1 or Num2 is not an integer, the program should throw a NumberFormatException. If Num2 is 0, the program should throw an ArithmeticException. Display the exceptions in a message dialog box.

Description: This program utilises the Dialog class of the AWT package to display the exceptions raised during division of two integers. The exceptions are caught using the try catch blocks. A Dialog is similar to a Frame. Any components can be added to a Dialog. A Dialog is primarily used to get input from the user and to display messages.

Program:

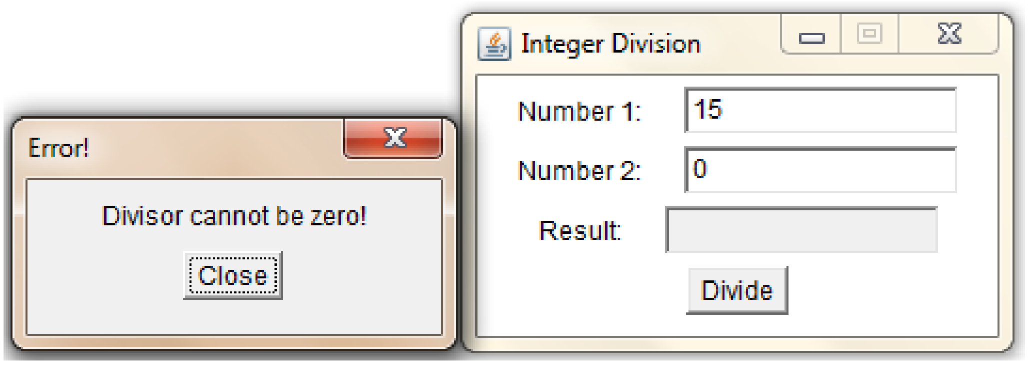
```
import java.awt.*;
import java.awt.event.*;
public class IntDiv extends Frame implements ActionListener{
        Message m;
        Label l1,l2,l3;
        TextField num1,num2,result;
        Button d;
        IntDiv(){
                super("Integer Division");
                setLayout(new FlowLayout());
                setLocation(250,150);
                setSize(250,150);
                setResizable(false);
                m=new Message(this,"Error!");
                l1=new Label("Number 1: ");
                num1=new TextField(15);
                l2=new Label("Number 2: ");
                num2=new TextField(15);
                l3=new Label("Result: ");
                result=new TextField(15);
                d=new Button("Divide");
                add(l1);
                add(num1);
                add(l2);
                add(num2);
                add(l3);
                add(result);
                result.setEditable(false);
                add(d);
                d.addActionListener(this);
                addWindowListener(new WindowAdapter(){
                        public void windowClosing(WindowEvent we){
                                System.exit(0);
                        }
                });
```

```
				setVisible(true);
		}
		public void actionPerformed(ActionEvent ae){
				if(ae.getSource()==d){
						try{


result.setText(Integer.toString(Integer.parseInt(num1.getText())/Integer.parseInt(num2.getTe
xt())));
						}catch(NumberFormatException e){
								m.display("Please enter integer values!");
								m.setVisible(true);
								result.setText("");
						}catch(ArithmeticException e){
								m.display("Divisor cannot be zero!");
								m.setVisible(true);
								result.setText("");
						}
				}
		}
		public static void main(String[] args){
				new IntDiv();
		}
}
class Message extends Dialog implements ActionListener{
		Label l=new Label("",Label.CENTER);
		Button b=new Button("Close");
		Message(Frame parent,String title){
				super(parent,title,true);
				setLocation(275,175);
				setSize(200,100);
				setLayout(new FlowLayout());
				add(l);
				add(b);
				b.addActionListener(this);
				addWindowListener(new WindowAdapter(){
						public void windowClosing(WindowEvent we){
								setVisible(false);
								dispose();
						}
				});
				setResizable(false);
				setVisible(false);
		}
		public void actionPerformed(ActionEvent ae){
				if(ae.getSource()==b){
						setVisible(false);
						dispose();
				}
		}
```

```
        void display(String str){
                l.setText(str);
        }
}
```

Output:



Aim: Write a Java program that implements a simple client-server application. The client sends data to a server. The server receives the data, uses it to produce a result and then sends the result back to the client. The client displays the result on the console. For example, the client sends the radius of a circle and the result returned by the server is the area of the circle.

Description: A client-server program can be developed using java.net package. Socket and ServerSocket classes are used to connect a client and a server and initiate protocol exchange. Socket is used on the client side and ServerSocket is used on the server side. Sockets are used to connect Java's I/O stream to the network and listen for clients.

Program:

```
//Server.java
import java.io.*;
import java.net.*;
class Server{
        public static void main(String[] args)throws Exception{
                ServerSocket ss=new ServerSocket(8080);
                System.out.println("Server Ready! Listening for Clients.....");
                Socket s=ss.accept();
                System.out.println("Client(s) Connected!");
                DataOutputStream dos=new DataOutputStream(s.getOutputStream());
                DataInputStream dis=new DataInputStream(s.getInputStream());
                double rad=dis.readDouble();
                double area=Math.PI*rad*rad;
                dos.writeDouble(area);
                s.close();
                ss.close();
        }
}

//Client.java
import java.io.*;
import java.net.*;
class Client{
        public static void main(String[] args)throws Exception{
                Socket s=new Socket("localhost",8080);
                System.out.println("Connected to Server!");
                DataOutputStream dos=new DataOutputStream(s.getOutputStream());
                DataInputStream dis=new DataInputStream(s.getInputStream());
                BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
                System.out.print("Enter radius: ");
                double rad=Double.parseDouble(br.readLine());
                dos.writeDouble(rad);
                double area=dis.readDouble();
                System.out.println("Area: "+area);
                s.close();
        }
}
```

Output:

Server:

```
Server Ready! Listening for Clients.....
Client(s) Connected!
```

Client:

```
Connection established with Server!
Enter radius: 22
Area: 1520.5308443374597
```

Aim: Write a Java program that simulates a traffic light. The program lets the user select one of the three lights: red, yellow or green. When a radio button is selected, the light is turned on and only one light can be on at a time. No light is on when the program starts.

Description: A traffic light structure can be composed using a rectangle and three circles. These can be drawn using the drawRect() and drawOval() methods of the Graphics class. The colours of the lights can be displayed using the Color class. A turned on light can be represented as a filled circle of that particular colour. This is achieved using the fillOval() method. The radio button controls can be generated using CheckboxGroup class. The Checkbox items should be registered with an ItemListener.

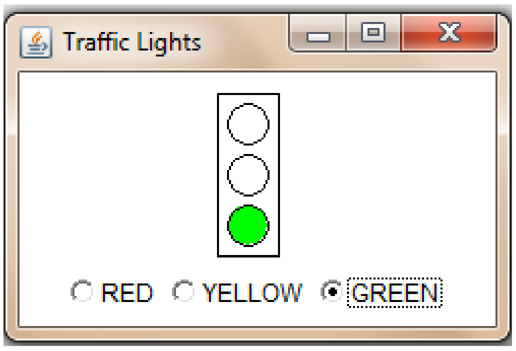
Program:

```
import java.awt.*;
import java.awt.event.*;
class TrafficLights extends Frame implements ItemListener{
	Lights l=new Lights();
	Panel p1,p2;
	CheckboxGroup control;
	Checkbox red,yellow,green;
	TrafficLights(){
		super("Traffic Lights");
		setLayout(new BorderLayout());
		setLocation(250,150);
		setSize(250,150);
		setResizable(false);
		p1=new Panel();
		l.setSize(50,100);
		p1.add(l);
		control=new CheckboxGroup();
		red=new Checkbox("RED",false,control);
		yellow=new Checkbox("YELLOW",false,control);
		green=new Checkbox("GREEN",false,control);
		p2=new Panel();
		p2.add(red);
		p2.add(yellow);
		p2.add(green);
		red.addItemListener(this);
		yellow.addItemListener(this);
		green.addItemListener(this);
		add(p1,BorderLayout.CENTER);
		add(p2,BorderLayout.SOUTH);
		addWindowListener(new WindowAdapter(){
			public void windowClosing(WindowEvent we){
				System.exit(0);
			}
		});
```

```
			setVisible(true);
		}
		public void itemStateChanged(ItemEvent ie){
			if(red.getState())
				l.red();
			if(yellow.getState())
				l.yellow();
			if(green.getState())
				l.green();
		}
		public static void main(String[] args){
			new TrafficLights();
		}
	}
	class Lights extends Canvas{
		boolean r,y,g;
		Lights(){
			r=y=g=false;
		}
		public void red(){
			r=true;
			y=g=false;
			repaint();
		}
		public void yellow(){
			y=true;
			r=g=false;
			repaint();
		}
		public void green(){
			g=true;
			r=y=false;
			repaint();
		}
		public void paint(Graphics gr){
			if(r){
				gr.setColor(Color.RED);
				gr.fillOval(10,10,20,20);
			}
			if(y){
				gr.setColor(Color.YELLOW);
				gr.fillOval(10,35,20,20);
			}
			if(g){
				gr.setColor(Color.GREEN);
				gr.fillOval(10,60,20,20);
			}
			gr.setColor(Color.BLACK);
			gr.drawRect(5,5,30,80);
			gr.drawOval(10,10,20,20);
```

```
                    gr.drawOval(10,35,20,20);
                    gr.drawOval(10,60,20,20);
          }
}
```

Output:



Traffic Lights

RED ( ) YELLOW ( ) [✓ GREEN]

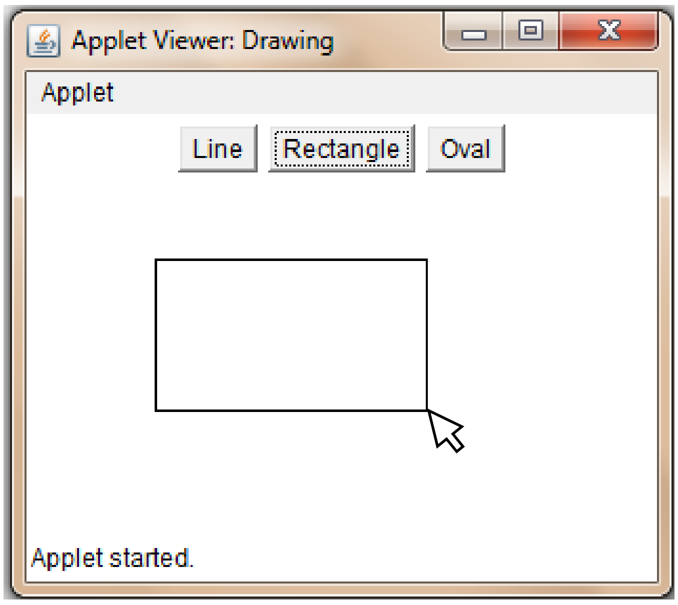
Aim: Write a Java program that allows the user to draw lines, rectangles and ovals.

Description: Java provides 2D graphics capabilities. The Canvas class is used to draw various shapes like lines, rectangles, ovals, polygons, etc. The methods to draw these shapes are provided by the Graphics class. To enable the user to draw these shapes, these methods are associated with the mouse movements. MouseAdapter and MouseMotionAdapter classes are used to listen to the mouse events like clicking, dragging, etc.

Program:

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
/*
<applet code="Drawing" width="300" height="200">
</applet>
*/
public class Drawing extends Applet implements ActionListener{
        int startX=0,startY=0,endX=0,endY=0;
        Button line,rect,oval;
        String shape="";
        public void init(){
                line=new Button("Line");
                rect=new Button("Rectangle");
                oval=new Button("Oval");
                line.addActionListener(this);
                rect.addActionListener(this);
                oval.addActionListener(this);
                add(line);
                add(rect);
                add(oval);
                addMouseListener(new MouseAdapter(){
                        public void mousePressed(MouseEvent me){
                                startX=me.getX();
                                startY=me.getY();
                        }
                        public void mouseReleased(MouseEvent me){
                                endX=me.getX();
                                endY=me.getY();
                                repaint();
                        }
                });
                addMouseMotionListener(new MouseMotionAdapter(){
                        public void mouseDragged(MouseEvent me){
                                endX=me.getX();
                                endY=me.getY();
                                repaint();
                        }
                });
        }
        public void actionPerformed(ActionEvent ae){
                if(ae.getSource()==line)
                        shape="line";
                if(ae.getSource()==rect)
                        shape="rect";
                if(ae.getSource()==oval)
                        shape="oval";
        }
        public void paint(Graphics g){
                if(shape.equals("line"))
                        g.drawLine(startX,startY,endX,endY);
                if(shape.equals("rect"))
                        g.drawRect(startX,startY,endX-startX,endY-startY);
                if(shape.equals("oval"))
                        g.drawOval(startX,startY,endX-startX,endY-startY);
        }
}
```

Output:



Applet started.

Aim: Write a Java program to create an abstract class named Shape that contains an abstract method named numberOfSides( ). Provide three classes named Trapezium, Triangle and Hexagon such that each one of these classes extend the abstract class Shape and override the abstract method numberOfSides( ) to display the number of sides in the given polygon.

Description: An abstract class is a class in which the methods are only declared but not defined. Such methods are called as abstract methods. An abstract class cannot be instantiated, i.e, no objects can be created for an abstract class. A class which extends an abstract class must either provide concrete implementation for all the abstract methods in the superclass or itself must be declared as abstract.

Program:

```
abstract class Shape{
        abstract void numberOfSides();
}
class Trapezium extends Shape{
        void numberOfSides(){
                System.out.println("Trapezium: 4 Sides");
        }
}
class Triangle extends Shape{
        void numberOfSides(){
                System.out.println("Triangle: 3 Sides");
        }
}
class Hexagon extends Shape{
        void numberOfSides(){
                System.out.println("Hexagon: 6 Sides");
        }
}
class ShapeDemo
{
        public static void main(String[] args)
        {
                Shape s=new Trapezium();
                s.numberOfSides();
                s=new Triangle();
                s.numberOfSides();
                s=new Hexagon();
                s.numberOfSides();
        }
}
```

Output:

```
Trapezium: 4 Sides
```

Triangle: 3 Sides
Hexagon: 6 Sides

Aim: Write a Java program to create a table using the JTable component. The data for the table is stored in a text file. The elements in the text file are separated using commas. Read the file and display the values in the table.

Description: A table is a component that displays data in rows and columns of data. Tables are implemented by the JTable class in Swing. One of its constructors is

JTable(object data[][],object colHeads[])

where data is 2-dimensional array of the information and colHeads is a 1-dimensional array with the columns headings. A table must be placed inside a scroll pane to facilitate scrolling through the data in rows and columns. JScrollPane class is used for this purpose.

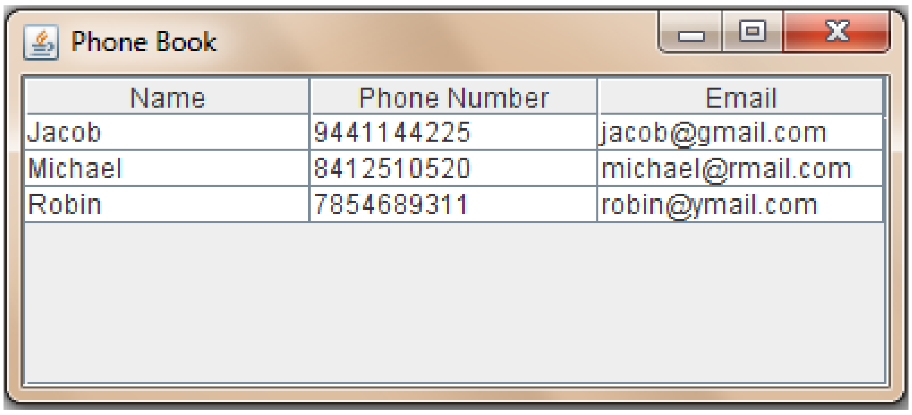
Program:

```
import java.io.*;
import java.util.StringTokenizer;
import javax.swing.*;
import java.awt.BorderLayout;
class PhoneBook extends JFrame{
	PhoneBook() throws Exception{
		super("Phone Book");
		setSize(300,300);
		File file=new File("phonebook.txt");
		BufferedReader br=new BufferedReader(new FileReader(file));
		String values[][]=new String[3][3];
		int row=0,col=0;
		String line=null;
		String head[]={"Name","Phone Number","Email"};
		while((line=br.readLine())!=null)
		{
			StringTokenizer st=new StringTokenizer(line,",");
			col=0;
			while(st.hasMoreTokens()){
				values[row][col]=st.nextToken();
				col++;
			}
			row++;
		}
		br.close();
		JTable table=new JTable(values,head);
		JScrollPane sp=new JScrollPane(table);
		add(sp,BorderLayout.CENTER);
		setVisible(true);
	}
	public static void main(String args[]) throws Exception{
		new PhoneBook();
	}
}
```

Sample file: phonebook.txt

```
Jacob,9441144225,jacob@gmail.com
Michael,8412510520,michael@rmail.com
Robin,7854689311,robin@ymail.com
```

Output:



Phone Book

Name	Phone Number	Email
Jacob	9441144225	jacob@gmail.com
Michael	8412510520	michael@rmail.com
Robin	7854689311	robin@ymail.com