Practical No: 1 (A)

Aim: Write a program to create a class and implement a default, overloaded and copy Constructor.

I. Default Constructor

Code:

```
class Student
 {
int id;
String name;
 void display()
{
System.out.println(id+" "+name);
 }
public static void main(String args[])
{
Student s1=new Student();
Student s2=new Student();
s1.display();
s2.display();
}
 }
```

Output:

```
0 null
0 null
```

II. Parameterized Constructor

Code:

```
class Student
 {
 int id;
 String name;
 Student4 (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
```

III. Copy Constructor

Code:

```
public class Fruit
 {
    private double fprice;
    private String fname;
  Fruit(double fPrice, String fName)
{
        fprice = fPrice;
        fname = fName;
    }
 Fruit(Fruit fruit)
 {
        System.out.println("\nAfter invoking the Copy
Constructor:\n");
        fprice = fruit.fprice;
        fname = fruit.fname;
    }
double showPrice() {
        return fprice;
    }
 String showName() {
        return fname;
}
  public static void main(String args[])
{
        Fruit f1 = new Fruit(399, "Ruby Roman Grapes");
        System.out.println("Name of the first fruit: " + f1.showName());
        System.out.println("Price of the first fruit: " + f1.showPrice());
        Fruit f2 = new Fruit(f1);
        System.out.println("Name of the second fruit: " + f2.showName());
        System.out.println("Price of the second fruit: " + f2.showPrice());
    }
}
```

Output:

```
Name of the first fruit: Ruby Roman Grapes
Price of the first fruit: 399.0

After invoking the Copy Constructor:
Name of the second fruit: Ruby Roman Grapes
Price of the second fruit:399.0
```

Practical No: 1 (B)

Aim: Write a program to create a class and implement the concepts of Method Overloading

Code:

```
class Addition
{
    public void sum()
    {
     System.out.println("No addition is performed");
    }
    public void sum (int a, int b)
   {
    System.out.println("The addition is: " + (a+b));
   }
   public void sum (int a, int b, int c)
  {
   System.out.println("The addition is:" + (a+b+c));
  }
  public void sum (double a, double b)
 {
   System.out.println("The addition is: "+ (a+b));
  }
}
 public class Dome
{
  public static void main(String args[])
   {
       Addition obj = new Addition ();
```

```
        obj.sum ();
        obj.sum (2,6);
        obj.sum (4,5,6);
        obj.sum (2.3, 5.6);
  }
}
```

Output:

```
No addition is performed
The addition is: 8
The addition is:15
The addition is: 7.8999999999999995
```

Practical No: 1 (C)

Aim: **Write a program to create a class and implement the concepts of Static methods.**

Code:

```
class Cllg
{
    int rollno;
    String name;
    static String college="IT";
    static void change()
{
        college="LORDS";
    }
    Cllg(int r,String n)
{
        rollno=r;
        name=n;
    }
    void display()
{
        System.out.println(rollno+ " " +name+ " "+college);
    }
}
 public class Test
{
    public static void main(String args[])
{
        Cllg.change();
        Cllg s1=new Cllg(111,"karan");
        Cllg s2=new Cllg(222,"Neha");
        Cllg s3=new Cllg(333,"Rohit");
        s1.display();
        s2.display();
        s3.display();
            }
}
```

Output:

```
111 karan LORDS
222 Neha LORDS
333 Rohit LORDS
```

Practical No: 2 (A)

Aim: Write a program to implement the concepts of Inheritance and Method overriding.

I. Multilevel Inheritance

Code:

```
import java.util.*;
class Student
{
    Scanner sc = new Scanner(System.in);
    String name;
    int rollNo;
    void getDetails()
        {
        System.out.println("Enter your name:");
        name = sc.next();
        System.out.println("Enter your Roll no:");
        rollNo = sc.nextInt();
        }
}
class Marks extends Student
{
    int marks;
    void getMarks()
        {
        System.out.println("Enter the marks:");
        marks = sc.nextInt();
         }
}
```

```
class Result extends Marks
 {
    void display()
        {
        System.out.println("Roll No: " + rollNo);
        System.out.println("Name: " + name);
        System.out.println("Marks: " + marks);
    }
}
 public class Multilevel {
    public static void main(String[] args) {
        Result obj = new Result();
        obj.getDetails();
        obj.getMarks();
        obj.display();
    }
}
```

Output:

```
Enter your name: neha
Enter your Roll no: 49
Enter the marks: 60
Roll No: 49
Name: neha
Marks: 60
```

II. Hierarchical Inheritance

Code:

```
import java.util.*;
 class Account
{
      Scanner sc = new Scanner(System.in);
       int user_id;
       int password;
    void getDetails()
{
        System.out.println("Enter your user id:");
        user_id=sc.nextInt();
        System.out.println("Enter your password:");
        password = sc.nextInt();
    }
    void putDetails()
{
        System.out.println("User ID: " + user_id);
        System.out.println("Password: " + password);
    }
 }
 class saving extends Account
{
      Scanner sc = new Scanner(System.in);
       int s_account;
       int s_balance;
    void getSaving()
{
        System.out.println("Enter your s_account:");
        s_account=sc.nextInt();
        System.out.println("Enter your s_balance:");
        s_balance = sc.nextInt();
    }
     void putSaving()
{
        System.out.println("Saving account:" +  s_account);
        System.out.println("Saving balance: " + s_balance);
    }
}
class current extends Account
{
    Scanner sc = new Scanner(System.in);
      int c_account;
      int c_balance;
    void getCurrent()
{
        System.out.println("Enter your c_account:");
        c_account=sc.nextInt();
        System.out.println("Enter your c_balance:");
        c_balance = sc.nextInt();
    }
    void putCurrent()
{
        System.out.println("current account:" +  c_account);
        System.out.println("current balance: " + c_balance);
    } }
   public class Hierarchial
```

```
{
    public static void main(String[] args)
{
       Account obj1=new Account();
       saving obj2=new saving();
       current obj3 =new current();
          obj1.getDetails();
          obj1.putDetails();
          obj2.getSaving();
          obj2.putSaving();
          obj3.getCurrent();
          obj3.putCurrent();
   }
}
```

Output:

```
Enter your user id: 12
Enter your password:1234
User ID: 12
Password: 1234
Enter your s_account: 3
Enter your s_balance: 34000
Saving account:3
Saving balance: 34000
Enter your c_account:3
Enter your c_balance: 56789
current account:3
current balance: 56789
```

III. Method overriding

Code:

```
class Vehical {
        void run(){
        System.out.println("vehical run fast");
    }
}
class bike extends Vehical{
    @Override
    void run(){
        System.out.println("bike run fast");
    }
}
 public class road {
    public static void main(String args[]){
        bike b=new bike();
        b.run();
    }
}
```

Output:

```
bike run fast
```

Practical No: 2 (B)

Aim: Write a program to implement the concepts of Abstract classes and methods

Code:

```
abstract class Person
{
 public abstract void display();
}
class EmployeeA extends Person
{
 int empno ;
 String empname, addr;
 public EmployeeA(int empno , String empname, String addr)
{
 this.empno = empno;
 this.empname= empname;
 this.addr= addr;
 }
 public void display()
{
 System.out.println("Employee Details ");
 System.out.println("Employee Id is: "+empno);
 System.out.println("Employee Name is :"+empname);
 System.out.println("Employee Address is: "+addr);
 }
}
class Worker extends Person
{
 int hours ;
 String name;
 public Worker (int hours, String name)
{
 this.hours= hours;
 this.name= name;
 }
 public void display()
{
 System.out.println("Worker Details ");
 System.out.println("Worker working hours is "+hours);
 System.out.println("Worker name is "+name);
 }
}
public class AbstractDemo2
{
 public static void main(String args[])
{
 EmployeeA e = new EmployeeA(1,"Ajay","Mumbai");
 e.display();
 Worker w = new Worker (10,"Abhi");
 w.display();
 }
 }
```

Output:

```
Employee Details
Employee Id is: 1
Employee Name is: Ajay
Employee Address is: Mumbai
Worker Details:
Worker working hours is: 10
Worker name is: Abhi
```

Practical No: 2 (C)

Aim: Write a program to implement the concept of interfaces.

Code:

```
import java.util.*;
interface Student {
    void getDetails();
}
interface Sports {
    void getSportsMarks();
}
class Result implements Student, Sports {
    Scanner sc = new Scanner(System.in);
    int id;
    String name;
    int marks;
    int sMarks;

    public void getDetails() {
        System.out.println("Enter your id:");
        id = sc.nextInt();
        System.out.println("Enter your name:");
        name = sc.next();
        System.out.println("Enter your marks:");
        marks = sc.nextInt();
    }

    public void getSportsMarks() {
        System.out.println("Enter your sports marks:");
        sMarks = sc.nextInt();
    }

    void display() {
        System.out.println("Id: " + id);
        System.out.println("Name: " + name);
        System.out.println("Marks: " + marks);
        System.out.println("Sports marks: " + sMarks);
    }
}
public class MultipleInheritance {
    public static void main(String args[]) {
        Result obj = new Result();
        obj.getDetails();
        obj.getSportsMarks();
        obj.display();
    }
}
```

Output:

```
Enter your id: 30
Enter your name: Neha
Enter your marks: 80
Enter your sports marks: 90

Id: 30
Name: Neha
Marks: 80
Sports marks: 90
```

Practical No: 3 (A)

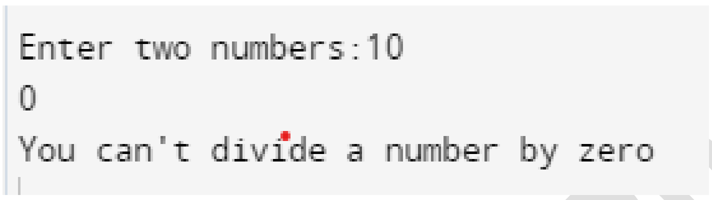
Aim: Write a program to raise built-in exceptions and raise them as per the requirements.

Code:

```
import java.util.Scanner;
class except
{
public static void main(String args[])
{
try
{
int a,b;
float c;
int x[]={10,20,30};
Scanner sc=new Scanner(System.in);
System.out.println("Enter two numbers:");
a=sc.nextInt();
b=sc.nextInt();
c=(int)a/b;
System.out.println("Division of a and b = " +c );
System.out.println("Element of array = " + x[6]);
}
catch(ArrayIndexOutOfBoundsException e1)
{
System.out.println("Array Index Out of range");
}
catch(ArithmeticException e2)
{
```

```
System.out.println("You can't divide a number by zero");
}
catch(Exception e)
{
System.out.println("Error");
}
}
}
```

Output:



```
Enter two numbers:10
0
You can't divide a number by zero
```

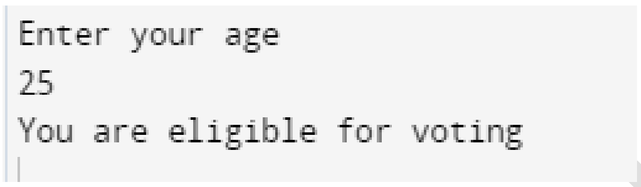
Practical No: 3 (B)

Aim: Write a program to define user defined exceptions and raise them as per the requirements

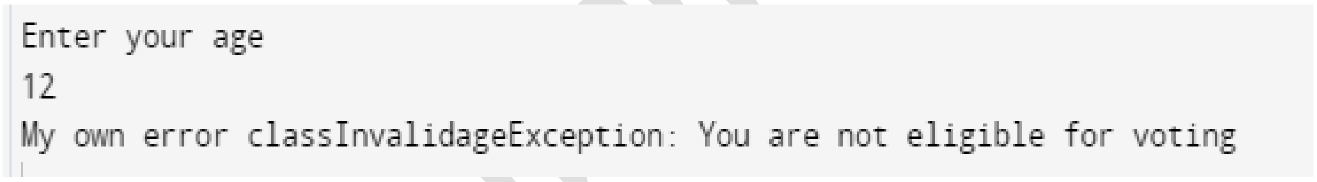
Code:

```
import java.util.*;
class InvalidageException extends Exception
{
public InvalidageException(String msg)
{
super (msg);
}
}
class UExcept
{
public static void main(String args[])
{
Try
{
int age;
Scanner sc=new Scanner(System.in);
System.out.println("Enter your age:");
age=sc.nextInt();
if(age>=18)
System.out.println("You are eligible for voting");
else
throw new InvalidageException("You are not eligible for voting");
}
catch(InvalidageException e)
{
System.out.println("My own error class"+e);
}
}
}
```

Output:



```
Enter your age
25
You are eligible for voting
```



```
Enter your age
12
My own error classInvalidageException: You are not eligible for voting
```

Practical No: 7 (A)

Aim: Write a program for the following layout: <u>FLOW LAYOUT</u>

Theory:

The Java FlowLayout class is used to arrange the components in a line, one after another (in a flow). It is the default layout of the applet or panel.

Fields of FlowLayout class

- public static final int LEFT
- public static final int RIGHT
- public static final int CENTER
- public static final int LEADING
- public static final int TRAILING

Constructors of FlowLayout class

- FlowLayout(): creates a flow layout with centred alignment and a default 5 unit horizontal and vertical gap.
- FlowLayout(int align): creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
- FlowLayout(int align, int hgap, int vgap): creates a flow layout with the given alignment and the given horizontal and vertical gap.

Code:

```
import java.awt.*;
import javax.swing.*;
   public class FlowLayoutExample
{
  JFrame frameObj;
 FlowLayoutExample()                    // constructor
{
   frameObj = new JFrame();             // creating a frame object
// creating the buttons
   JButton b1 = new JButton("1");
```

```
    JButton b2 = new JButton("2");
    JButton b3 = new JButton("3");
    JButton b4 = new JButton("4");
    JButton b5 = new JButton("5");
    JButton b6 = new JButton("6");
    JButton b7 = new JButton("7");
    JButton b8 = new JButton("8");
    JButton b9 = new JButton("9");
    JButton b10 = new JButton("10");

    // adding the buttons to frame
    frameObj.add(b1); frameObj.add(b2); frameObj.add(b3);
frameObj.add(b4);
    frameObj.add(b5); frameObj.add(b6);  frameObj.add(b7);
frameObj.add(b8);
    frameObj.add(b9);  frameObj.add(b10);

// parameter less constructor is used therefore, alignment is center &
horizontal as well as the vertical gap is 5 units.
    frameObj.setLayout(new FlowLayout());
    frameObj.setSize(300, 300);
    frameObj.setVisible(true);
}
 public static void main(String argvs[])
{
    new FlowLayoutExample();
}
}
```

Practical No: 7 (B)

Aim: Write program for the following layout: GRID LAYOUT

Theory:

The Java GridLayout class is used to arrange the components in a rectangular grid. One component is displayed in each rectangle.

Constructors of GridLayout class

- GridLayout(): creates a grid layout with one column per component in a row.
- GridLayout(int rows, int columns): creates a grid layout with the given rows and columns but no gaps between the components.
- GridLayout(int rows, int columns, int hgap, int vgap): creates a grid layout with the given rows and columns along with given horizontal and vertical gaps.

Code:

```
import java.awt.*;
import javax.swing.*;
public class MyGridLayout
{
MyGridLayout(){
   JFrame f =new JFrame();
   JButton b1=new JButton("1");
   JButton b2=new JButton("2");
   JButton b3=new JButton("3");
   JButton b4=new JButton("4");
   JButton b5=new JButton("5");
   JButton b6=new JButton("6");
   JButton b7=new JButton("7");
   JButton b8=new JButton("8");
   JButton b9=new JButton("9");
// adding buttons to the frame
    f.add(b1); f.add(b2); f.add(b3);
    f.add(b4); f.add(b5); f.add(b6);
    f.add(b7); f.add(b8); f.add(b9);

f.setLayout(new GridLayout(3,3));
// f.setLayout(new GridLayout(3, 3, 20, 25));
                                        //Different Constructor
    f.setSize(300,300);
    f.setVisible(true);
}
public static void main(String[] args)
{
    new MyGridLayout();
}
}
```

Practical No: 7 (C)

Aim: Write program for the following layout: Border LAYOUT

Code:

```
import java.awt.*;
import javax.swing.*;
public class Border
{
Border()
{
   JFrame f = new JFrame();
JButton b1 = new JButton("NORTH");;
JButton b2 = new JButton("SOUTH");;
JButton b3 = new JButton("EAST");;
JButton b4 = new JButton("WEST");;
JButton b5 = new JButton("CENTER");;

f.add(b1, BorderLayout.NORTH);
f.add(b2, BorderLayout.SOUTH);
f.add(b3, BorderLayout.EAST);
f.add(b4, BorderLayout.WEST);
f.add(b5, BorderLayout.CENTER);
        f.setSize(300, 300);
        f.setVisible(true);
}
public static void main(String[] args) {
   new Border();
}   }
```

Practical No: 8

Aim: Write a program to demonstrate the following events.

A. ActionEvent

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

public class ActionListenerExample implements ActionListener
{
public static void main(String[] args)
{
   Frame f=new Frame("ActionListener Example");
   final TextField tf=new TextField();
   tf.setBounds(50,50, 150,20);
   Button b=new Button("Click Here");
   b.setBounds(50,100,60,30);

   b.addActionListener(this);
   f.add(b);f.add(tf);
   f.setSize(400,400);
   f.setLayout(null);
   f.setVisible(true);
}
public void actionPerformed(ActionEvent e)
{
         tf.setText("Welcome to Javatpoint.");
}
}
```

B. MouseEvent

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

public class MouseListenerExample extends Frame implements
MouseListener
{
   Label l;
   MouseListenerExample()
{
      addMouseListener(this);
      l=new Label();
      l.setBounds(20,50,100,20);
      add(l);

      setSize(300,300);
      setLayout(null);
      setVisible(true);
   }
   public void mouseClicked(MouseEvent e) {
      l.setText("Mouse Clicked");
   }
   public void mouseEntered(MouseEvent e) {
      l.setText("Mouse Entered");
   }
   public void mouseExited(MouseEvent e) {
      l.setText("Mouse Exited");
   }
```

```
    public void mousePressed(MouseEvent e) {
        l.setText("Mouse Pressed");
    }
    public void mouseReleased(MouseEvent e) {
        l.setText("Mouse Released");
    }
public static void main(String[] args) {
    new MouseListenerExample();
}
}
```

C. KeyEvent

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

public class KeyListenerExample extends Frame implements
KeyListener
{
     Label l;
    TextArea area;

    KeyListenerExample()
{
        l = new Label();
        l.setBounds (20, 50, 100, 20);
        area = new TextArea();

        area.setBounds (20, 80, 300, 300);
        area.addKeyListener(this);
        add(l);
        add(area);
        setSize (400, 400);
        setLayout (null);
        setVisible (true);
    }
    public void keyPressed (KeyEvent e) {
        l.setText ("Key Pressed");
    }
```

```
  public void keyReleased (KeyEvent e) {
      l.setText ("Key Released");
   }
   public void keyTyped (KeyEvent e) {
      l.setText ("Key Typed");
   }
   public static void main(String[] args) {
      new KeyListenerExample();
   }
}
```