SOURCE CODE

Lab program no 1 :

Develop a Java program that prints all real solutions to the quadratic equation ax2 +bx+c = 0. Read in a, b, c and use the quadratic formula. If the discriminate b2 -4ac is negative, display a message stating that there are no real solutions

```
import java.util.Scanner;


public class QuadraticEquationSolver {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);


        // Input coefficients a, b, and c
        System.out.print("Enter coefficient a: ");
        double a = scanner.nextDouble();


        System.out.print("Enter coefficient b: ");
        double b = scanner.nextDouble();


        System.out.print("Enter coefficient c: ");
        double c = scanner.nextDouble();


        // Calculate the discriminant
        double discriminant = b * b - 4 * a * c;


        // Check if the discriminant is negative
        if (discriminant < 0) {
            System.out.println("No real solutions exist.");
        } else {
            // Calculate real solutions
            double root1 = (-b + Math.sqrt(discriminant)) / (2 * a);
```

```
            double root2 = (-b - Math.sqrt(discriminant)) / (2 * a);

            System.out.println("Real solutions:");
            System.out.println("Root 1: " + root1);
            System.out.println("Root 2: " + root2);
        }

        scanner.close();
    }
}
```

Lab program no 2:

Develop a Java program to create a class Student with members usn, name, an array credits and an array marks. Include methods to accept and display details and a method to calculate SGPA of a student.

```
import java.util.Scanner;

public class Student {
    private String usn;
    private String name;
    private int[] credits;
    private int[] marks;

    // Constructor
    public Student(String usn, String name, int numSubjects) {
        this.usn = usn;
        this.name = name;
        this.credits = new int[numSubjects];
        this.marks = new int[numSubjects];
    }

    // Method to accept details of the student
    public void acceptDetails(Scanner scanner) {
        System.out.println("Enter details for student " + name + " (" + usn + "):");
        for (int i = 0; i < credits.length; i++) {
            System.out.print("Enter credits for subject " + (i + 1) + ": ");
            credits[i] = scanner.nextInt();
            System.out.print("Enter marks for subject " + (i + 1) + ": ");
            marks[i] = scanner.nextInt();
        }
    }
```

```
// Method to display details of the student
public void displayDetails() {
    System.out.println("Details for student " + name + " (" + usn + "):");
    for (int i = 0; i < credits.length; i++) {
        System.out.println("Subject " + (i + 1) + ": Credits - " + credits[i] + ", Marks - " + marks[i]);
    }
}

// Method to calculate SGPA of the student
public double calculateSGPA() {
    double totalCredits = 0;
    double totalGradePoints = 0;

    for (int i = 0; i < credits.length; i++) {
        totalCredits += credits[i];
        totalGradePoints += calculateGradePoints(marks[i]) * credits[i];
    }

    return totalGradePoints / totalCredits;
}

// Helper method to calculate grade points based on marks
private double calculateGradePoints(int marks) {
    if (marks >= 90) {
        return 10.0;
    } else if (marks >= 80) {
        return 9.0;
    } else if (marks >= 70) {
        return 8.0;
    } else if (marks >= 60) {
```

```
            return 7.0;
        } else if (marks >= 50) {
            return 6.0;
        } else if (marks >= 40) {
            return 5.0;
        } else {
            return 0.0;
        }
    }

    // Main method for testing
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Create a student object
        Student student = new Student("1MS17CS001", "John Doe", 3);

        // Accept details
        student.acceptDetails(scanner);

        // Display details
        student.displayDetails();

        // Calculate and display SGPA
        double sgpa = student.calculateSGPA();
        System.out.println("SGPA: " + sgpa);

        scanner.close();
    }
}
```

Lab program no 3 :

Create a class Book which contains four members: name, author, price, num_pages. Include a constructor to set the values for the members. Include methods to set and get the details of the objects. Include a toString( ) method that could display the complete details of the book. Develop a Java program to create n book objects.

```
import java.util.Scanner;

public class Book {
    private String name;
    private String author;
    private double price;
    private int numPages;

    // Constructor to initialize the members
    public Book(String name, String author, double price, int numPages) {
        this.name = name;
        this.author = author;
        this.price = price;
        this.numPages = numPages;
    }

    // Getters and setters for the members
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getAuthor() {
```

```
    return author;
}

public void setAuthor(String author) {
    this.author = author;
}

public double getPrice() {
    return price;
}

public void setPrice(double price) {
    this.price = price;
}

public int getNumPages() {
    return numPages;
}

public void setNumPages(int numPages) {
    this.numPages = numPages;
}

// toString() method to display complete details of the book
@Override
public String toString() {
    return "Book{" +
            "name='" + name + '\'' +
            ", author='" + author + '\'' +
            ", price=" + price +
            ", numPages=" + numPages +
```

```
            '}';
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of books: ");
        int n = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        Book[] books = new Book[n];

        // Input details for each book
        for (int i = 0; i < n; i++) {
            System.out.println("Enter details for book " + (i + 1) + ":");
            System.out.print("Name: ");
            String name = scanner.nextLine();
            System.out.print("Author: ");
            String author = scanner.nextLine();
            System.out.print("Price: ");
            double price = scanner.nextDouble();
            System.out.print("Number of pages: ");
            int numPages = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            // Create a book object and store it in the array
            books[i] = new Book(name, author, price, numPages);
        }

        // Display details of all books
        System.out.println("Details of all books:");
```

```
        for (int i = 0; i < n; i++) {
            System.out.println("Book " + (i + 1) + ": " + books[i]);
        }

        scanner.close();
    }
}
```

Lab program no 4 :

Develop a Java program to create an abstract class named Shape that contains two integers and an empty method named printArea( ).Provide three classes named Rectangle, Triangle and Circle such that each one of the classes extends the class Shape. Each one of the classes contain only the method printArea( ) that prints the area of the given shape.

```
abstract class Shape {
    protected int dimension1;
    protected int dimension2;

    public Shape(int dimension1, int dimension2) {
        this.dimension1 = dimension1;
        this.dimension2 = dimension2;
    }

    // Abstract method to print area
    public abstract void printArea();
}

class Rectangle extends Shape {
    public Rectangle(int length, int breadth) {
        super(length, breadth);
    }

    // Override printArea method for Rectangle
    @Override
    public void printArea() {
        int area = dimension1 * dimension2;
        System.out.println("Area of Rectangle: " + area);
    }
}
```

```
class Triangle extends Shape {
    public Triangle(int base, int height) {
        super(base, height);
    }

    // Override printArea method for Triangle
    @Override
    public void printArea() {
        double area = 0.5 * dimension1 * dimension2;
        System.out.println("Area of Triangle: " + area);
    }
}

class Circle extends Shape {
    public Circle(int radius) {
        super(radius, 0); // For circle, we only need one dimension
    }

    // Override printArea method for Circle
    @Override
    public void printArea() {
        double area = Math.PI * dimension1 * dimension1;
        System.out.println("Area of Circle: " + area);
    }
}

public class Main {
    public static void main(String[] args) {
        // Create objects of Rectangle, Triangle, and Circle
        Rectangle rectangle = new Rectangle(5, 10);
        Triangle triangle = new Triangle(4, 6);
```

```
        Circle circle = new Circle(7);

        // Call printArea method for each object
        rectangle.printArea();
        triangle.printArea();
        circle.printArea();
    }
}
```

Lab program no 5 :

Develop a Java program to create a class Bank that maintains two kinds of account for its customers, one called savings account and the other current account. The savings account provides compound interest and withdrawal facilities but no cheque book facility. The current account provides cheque book facility but no interest. Current account holders should also maintain a minimum balance and if the balance falls below this level, a service charge is imposed.

• Create a class Account that stores customer name, account number and type of account. From this derive the classes Cur-acct and Sav-acct to make them more specific to their requirements. Include the necessary methods in order to achieve the following tasks:

• a)Accept deposit from customer and update the balance.

• b)Display the balance.

• c)Compute and deposit interest

• d)Permit withdrawal and update the balance

• Check for the minimum balance, impose penalty if necessary and update the balance.

```
import java.util.Scanner;

// Account class to store customer name, account number, and type of account
class Account {
    protected String customerName;
    protected int accountNumber;
    protected String accountType;
    protected double balance;

    // Constructor to initialize Account
    public Account(String customerName, int accountNumber, String accountType, double balance) {
        this.customerName = customerName;
        this.accountNumber = accountNumber;
        this.accountType = accountType;
        this.balance = balance;
    }

    // Method to accept deposit from customer and update the balance
```

```
    public void deposit(double amount) {
        balance += amount;
        System.out.println("Deposit of " + amount + " successful.");
    }

    // Method to display the balance
    public void displayBalance() {
        System.out.println("Balance: " + balance);
    }
}

// Savings Account class extending Account
class SavAcct extends Account {
    private double interestRate;

    // Constructor to initialize Savings Account
    public SavAcct(String customerName, int accountNumber, String accountType, double balance,
double interestRate) {
        super(customerName, accountNumber, accountType, balance);
        this.interestRate = interestRate;
    }

    // Method to compute and deposit interest
    public void computeAndDepositInterest() {
        double interest = balance * (interestRate / 100);
        deposit(interest);
    }

    // Method to permit withdrawal and update the balance
    public void withdraw(double amount) {
        if (balance >= amount) {
```

```
            balance -= amount;
            System.out.println("Withdrawal of " + amount + " successful.");
        } else {
            System.out.println("Insufficient balance.");
        }
    }
}

// Current Account class extending Account
class CurAcct extends Account {
    private double minimumBalance;
    private double serviceCharge;

    // Constructor to initialize Current Account
    public CurAcct(String customerName, int accountNumber, String accountType, double balance,
double minimumBalance, double serviceCharge) {
        super(customerName, accountNumber, accountType, balance);
        this.minimumBalance = minimumBalance;
        this.serviceCharge = serviceCharge;
    }

    // Method to permit withdrawal and update the balance
    public void withdraw(double amount) {
        if (balance - amount >= minimumBalance) {
            balance -= amount;
            System.out.println("Withdrawal of " + amount + " successful.");
        } else {
            System.out.println("Insufficient balance. Service charge applied.");
            balance -= serviceCharge;
        }
    }
```

```
}

public class Bank {
    public static void main(String[] args) {
        // Creating Savings Account object
        SavAcct savingsAccount = new SavAcct("John Doe", 123456789, "Savings", 5000, 5);

        // Creating Current Account object
        CurAcct currentAccount = new CurAcct("Jane Smith", 987654321, "Current", 10000, 2000, 50);

        // Deposit and display balance for Savings Account
        savingsAccount.deposit(1000);
        savingsAccount.displayBalance();
        savingsAccount.computeAndDepositInterest();
        savingsAccount.displayBalance();
        savingsAccount.withdraw(2000);
        savingsAccount.displayBalance();

        // Deposit and display balance for Current Account
        currentAccount.deposit(2000);
        currentAccount.displayBalance();
        currentAccount.withdraw(5000);
        currentAccount.displayBalance();
    }
}
```

Lab program no 6 :

Create a package CIE which has two classes- Student and Internals. The class Student has members like usn, name, sem. The class Internals derived from Student has an array that stores the internal marks scored in five courses of the current semester of the student. Create another package SEE which has the class External which is a derived class of Student. This class has an array that stores the SEE marks scored in five courses of the current semester of the student. Import the two packages in a file that declares the final marks of n students in all five courses.

Student.java:

```
package CIE;

public class Student {
    protected String usn;
    protected String name;
    protected int sem;
}
```

Internals.java:

```
package CIE;

public class Internals extends Student {
    protected int[] internalMarks = new int[5];
}
```

External.java:

```
package SEE;

import CIE.Student;

public class External extends Student {
    protected int[] externalMarks = new int[5];
}
```

FinalMarks.java:

```
import CIE.Internals;

import SEE.External;


public class FinalMarks {
```

```
public static void main(String[] args) {
    // Assuming you have the necessary data structures to store n students' final marks
    // and these data structures are named internalMarksArray and externalMarksArray
    // Let's assume we have 3 students for demonstration purposes

    // Creating objects for Internals and External classes
    Internals[] internalMarksArray = new Internals[3];
    External[] externalMarksArray = new External[3];

    // Initialize internal and external marks for each student
    for (int i = 0; i < 3; i++) {
        internalMarksArray[i] = new Internals();
        internalMarksArray[i].usn = "InternalUSN" + (i + 1);
        internalMarksArray[i].name = "InternalName" + (i + 1);
        internalMarksArray[i].sem = 5;

        externalMarksArray[i] = new External();
        externalMarksArray[i].usn = "ExternalUSN" + (i + 1);
        externalMarksArray[i].name = "ExternalName" + (i + 1);
        externalMarksArray[i].sem = 5;
    }

    // Assigning marks (Just for demonstration, you would assign actual marks here)
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 5; j++) {
            internalMarksArray[i].internalMarks[j] = 50 + i; // Just for demonstration
            externalMarksArray[i].externalMarks[j] = 60 + i; // Just for demonstration
        }
    }

    // Displaying final marks
```

```
        for (int i = 0; i < 3; i++) {
            System.out.println("Student: " + internalMarksArray[i].name + ", USN: " +
internalMarksArray[i].usn);
            System.out.println("Final Marks:");
            for (int j = 0; j < 5; j++) {
                int finalMarks = internalMarksArray[i].internalMarks[j] +
externalMarksArray[i].externalMarks[j];
                System.out.println("Subject " + (j + 1) + ": " + finalMarks);
            }
            System.out.println();
        }
    }
}
```

Lab program 7 :

Write a program that demonstrates handling of exceptions in inheritance tree. Create a base class called “Father” and derived class called “Son” which extends the base class. In Father class, implement a constructor which takes the age and throws the exception WrongAge( ) when the input age<0. In Son class, implement a constructor that cases both father and son’s age and throws an exception if son’s age is >=father’s age.

```
// Custom exception WrongAge
class WrongAge extends Exception {
    public WrongAge(String message) {
        super(message);
    }
}

// Father class
class Father {
    private int age;

    // Constructor for Father class
    public Father(int age) throws WrongAge {
        if (age < 0) {
            throw new WrongAge("Age cannot be negative");
        }
        this.age = age;
    }

    public int getAge() {
        return age;
    }
}

// Son class extending Father
```

```
class Son extends Father {
    private int sonAge;

    // Constructor for Son class
    public Son(int fatherAge, int sonAge) throws WrongAge {
        super(fatherAge);
        if (sonAge >= fatherAge) {
            throw new WrongAge("Son's age should be less than father's age");
        }
        this.sonAge = sonAge;
    }

    public int getSonAge() {
        return sonAge;
    }
}

// Main class
public class ExceptionInheritanceDemo {
    public static void main(String[] args) {
        try {
            // Creating a Father object with age 50
            Father father = new Father(50);
            System.out.println("Father's age: " + father.getAge());

            // Creating a Son object with father's age as 50 and son's age as 20
            Son son = new Son(father.getAge(), 20);
            System.out.println("Son's age: " + son.getSonAge());
        } catch (WrongAge e) {
            System.out.println("Exception caught: " + e.getMessage());
        }
```

```
    }
}
```

Lab program no 8 :

Write a program which creates two threads, one thread displaying “BMS College of Engineering” once every ten seconds and another displaying “CSE” once

Lab program no 9 :

Write a program that creates a user interface to perform integer divisions.The user enters two numbers in the text fields, Num1 and Num2. The division of Num1 and Num2 is displayed in the Result field when the Divide button is clicked. If Num1 or Num2 were not an integer, the program would throw a NumberFormatException. If Num2 were Zero, the program would throw an Arithmetic Exception Display the exception in a message dialog box.

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

public class DivisionMain1 extends Frame implements ActionListener
{
	TextField num1,num2;
	Button dResult;
	Label outResult;
	String out="";
	double resultNum;
	int flag=0;

	public DivisionMain1()
	{
		setLayout(new FlowLayout());

		dResult = new Button("RESULT");
		Label number1 = new Label("Number 1:",Label.RIGHT);
		Label number2 = new Label("Number 2:",Label.RIGHT);
		num1=new TextField(5);
		num2=new TextField(5);
		outResult = new Label("Result:",Label.RIGHT);

		add(number1);
		add(num1);
```

```
        add(number2);
        add(num2);
        add(dResult);
        add(outResult);

        num1.addActionListener(this);
        num2.addActionListener(this);
        dResult.addActionListener(this);
        addWindowListener(new WindowAdapter()
        {
                public void windowClosing(WindowEvent we)
                {
                        System.exit(0);
                }
        });
}

public void actionPerformed(ActionEvent ae)
{
        int n1,n2;
        try
        {
                if (ae.getSource() == dResult)
                {
                        n1=Integer.parseInt(num1.getText());
                        n2=Integer.parseInt(num2.getText());

                        /*if(n2==0)
                                throw new ArithmeticException();*/
                        out=n1+" "+n2;
                        resultNum=n1/n2;
```

```
                        out+=String.valueOf(resultNum);
                        repaint();

                    }
                }
                catch(NumberFormatException e1)
                {
                    flag=1;
                    out="Number Format Exception!  "+e1;
                    repaint();
                }
                catch(ArithmeticException e2)
                {
                    flag=1;
                    out="Divide by 0 Exception!  "+e2;
                    repaint();
                }

        }

        public void paint(Graphics g)
        {
            if(flag==0)

        g.drawString(out,outResult.getX()+outResult.getWidth(),outResult.getY()+outResult.getHeig
ht()-8);
            else
            g.drawString(out,100,200);
            flag=0;
        }

        public static void main(String[] args)
```

```
        {
                DivisionMain1 dm=new DivisionMain1();
                dm.setSize(new Dimension(800,400));
                dm.setTitle("DivionOfIntegers");
                dm.setVisible(true);
        }

}
```

Lab program 10 :

1.) correct implementation of a producer an consumer.

```
class Q {

int n;

boolean valueSet = false;

synchronized int get() {

while(!valueSet)

try {

System.out.println("\nConsumer waiting\n");

wait();

} catch(InterruptedException e) {

System.out.println("InterruptedException
caught");

}

System.out.println("Got: " + n);

valueSet = false;

System.out.println("\nIntimate Producer\n");

notify();

return n;

}

synchronized void put(int n) {

while(valueSet)

try {

System.out.println("\nProducer waiting\n");

wait();
```

```
} catch(InterruptedException e) {
System.out.println("InterruptedException caught");
}
this.n = n;
valueSet = true;
System.out.println("Put: " + n);
System.out.println("\nIntimate Consumer\n");
notify();
}
}
class Producer implements Runnable {
Q q;
Producer(Q q) {
this.q = q;
new Thread(this, "Producer").start();
}
public void run() {
int i = 0;
while(i<15) {
q.put(i++);
}
}
}
class Consumer implements Runnable {
```

```
Q q;

Consumer(Q q) {

this.q = q;

new Thread(this, "Consumer").start();

}

public void run() {

int i=0;

while(i<15) {

int r=q.get();

System.out.println("consumed:"+r);

i++;

}

}

}

class PCFixed {

public static void main(String args[]) {

Q q = new Q();

new Producer(q);

new Consumer(q);

System.out.println("Press Control-C to stop.");

}

}
```

every two seconds.

```
class DisplayMessage extends Thread {
    private String message;
    private int interval;

    // Constructor to initialize message and interval
    public DisplayMessage(String message, int interval) {
        this.message = message;
        this.interval = interval;
    }

    // Run method to display message at given intervals
    public void run() {
        try {
            while (true) {
                System.out.println(message);
                Thread.sleep(interval * 1000); // Convert seconds to milliseconds
            }
        } catch (InterruptedException e) {
            System.out.println("Thread interrupted.");
        }
    }
}

public class DisplayMessages {
    public static void main(String[] args) {
        // Creating and starting threads
        DisplayMessage thread1 = new DisplayMessage("BMS College of Engineering", 10);
        DisplayMessage thread2 = new DisplayMessage("CSE", 2);

        thread1.start();
```

```
        thread2.start();
    }
}
```

2.) Deadlock

```
class A {

synchronized void foo(B b) {

String name =
Thread.currentThread().getName();

System.out.println(name + " entered
A.foo");

try {

Thread.sleep(1000);

} catch(Exception e) {

System.out.println("A Interrupted");

}

System.out.println(name + " trying to
call B.last()");

b.last();

}

void last() {

System.out.println("Inside A.last");

}

}

class B {

synchronized void bar(A a) {

String name =
Thread.currentThread().getName();
```

```
System.out.println(name + " entered
B.bar");

try {

Thread.sleep(1000);

} catch(Exception e) {

System.out.println("B Interrupted");

}

System.out.println(name + " trying to
call A.last()");

a.last();

}

void last() {

System.out.println("Inside A.last");

}

}

class Deadlock implements Runnable
{

A a = new A();

B b = new B();

Deadlock() {

Thread.currentThread().setName("M
ainThread");

Thread t = new Thread(this,
"RacingThread");

t.start();

a.foo(b); // get lock on a in this
thread.
```

```
System.out.println("Back in main
thread");

}

public void run() {

b.bar(a); // get lock on b in other
thread.

System.out.println("Back in other
thread");

}

public static void main(String args[]) {

new Deadlock();

}

}
```