

INDEX

Exp No	Date	Experiment Name	Page No	Marks	Staff Signature
1.1		Sum of Individual Digits of a Positive Integer			
1.2		Generate the First N Terms of the Sequence			
1.3		Generate All the Prime Numbers Between 1 and N			
1.4		Largest and Smallest Number in a List of Integers			
1.5		Program Using Packages			
2.1		Employee payroll calculation using Hierarchical Inheritance			
2.2		Calculating Area of different shapes using Abstract classes			
2.3		String Operations Using ArrayList			
3.1		Exception Handling and User-Defined Exceptions			
3.2		Java Interface for ADT Stack			
3.3		File Handling in Java			
4.1		Multi-threaded Application			
4.2		Importing Classes from User-Defined Package			
4.3		Find the Maximum Value from the Given Type of Elements Using a Generic Function			
5.1		Adding elements using ArrayList			
5.2		Shuffle Elements in Array List			
5.3		Iterate Through All Elements in a Linked List			
5.4		ArrayList of Student (ID, Name, Dept, Age) Objects and Search Operations			
5.5		ArrayList – Store only Char and String			
5.6		Queue Collection for Cinema Ticket Sale			
6.1		Finding sum and difference using Applet			
6.2		Keyboard events using Applet			
6.3		Student Registration Form Using Swing Controls			
6.4		Digital Watch in Swing			

MODULE 1

Ex.No.1.1 SUM OF INDIVIDUAL DIGITS OF A POSITIVE INTEGER

Date :

AIM : To write a java program to find the sum of individual digits of a positive integer.

ALGORITHM :

1. Read or initialize an integer N.
2. Declare a variable (sum) to store the sum of numbers and initialize it to 0.
3. Find the remainder by using the modulo (%) operator. It gives the last digit of the number (N).
4. Add the last digit to the variable sum.
5. Divide the number (N) by 10. It removes the last digit of the number.
6. Repeat the above steps (3 to 5) until the number (N) becomes 0.

PROGRAM :

```
import java.util.Scanner;
public class SumOfDigits
{
    public static void main(String args[])
    {
        int number, digit, sum = 0;
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number: ");
        number = sc.nextInt();
        while(number > 0)
        {
            digit = number % 10;
            sum = sum + digit;
            number = number / 10;
        }
    }
}
```

```
    }  
    System.out.println("Sum of Digits: "+sum);  
}  
}
```

OUTPUT :

Enter the number: 7896

Sum of Digits: 30

RESULT :

Thus the java program to find the sum of individual digits of a positive integer is executed and verified successfully.

Ex.No.1.2**GENERATE THE FIRST N TERMS OF THE SEQUENCE**

Date :

AIM : To write a Java program to generate the first n terms of the sequence.

ALGORITHM :

1. Start the program
2. Read a number n using Scanner class
3. Iterate a for loop using variable i from 1 to n
4. Print i

PROGRAM :

```
import java.util.*;

public class Sequence {
    public static void main(String[] args)
    {
        System.out.println("Enter Number: ");
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        for(int i = 1; i <= n; i++)
            System.out.print(i+" ");
    }
}
```

OUTPUT :

Enter Number:

10

1 2 3 4 5 6 7 8 9 10

RESULT :

Thus the Java program to generate the first n terms of the sequence is executed and verified successfully.

Ex.No.1.3 GENERATE ALL THE PRIME NUMBERS BETWEEN 1 AND N

Date :

AIM : To write a Java program to generate all the prime numbers between 1 and n, where n is a value supplied by the user.

ALGORITHM :

Step 1 – Start

Step 2 – Declare an integer: n

Step 3 – Prompt the user to enter an integer value

Step 4 – Read the value using Scanner class

Step 5 – Using a for loop from 1 to n, check if the 'i' value is divisible by any number from 2 to i

Step 6 – If yes, check the next number

Step 7 – If no, store the number as a prime number

Step 8 – Stop

PROGRAM :

```
import java.util.Scanner;
public class PrimeNumbers {
    public static void main(String arg[]) {
        int i, n, counter, j;
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the n value : ");
        n = scanner.nextInt();
        System.out.print("Prime numbers between 1 to n are ");
        for(j = 2; j <= n; j++) {
            counter = 0;
            for(i = 1; i <= j; i++) {
                if(j % i == 0) {
                    counter++;
                }
            }
            if(counter == 2)
                System.out.print(j + " ");
        }
    }
}
```

OUTPUT :

Enter the n value : 32

Prime numbers between 1 to n are 2 3 5 7 11 13 17 19 23 29 31

RESULT :

Thus the Java program to generate all the prime numbers between 1 and n is executed and verified successfully.

Ex. No.1.4**Largest and Smallest number in a list of integers.****Date :****AIM :** To write a Java program to find both the largest and smallest number in a list of integers.**ALGORITHM:**

1. Start
2. Read the number of values “n” in the array
3. Declare an array and read values using Scanner class
4. Initialize two variables min and max with arr[0]
5. Iterate over the array using a for loop
6. If current element is greater than max, then assign current element to max
7. If current element is smaller than min, then assign current element to min
8. Print smallest and largest element

PROGRAM :

```
import java.util.*;
public class MinMax
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter array size: ");
        int n = sc.nextInt();
        int arr[] = new int[n];
        System.out.print("Enter array elements: ");
        for(int i = 0; i < n; i++)
            arr[i] = sc.nextInt();
        System.out.println("Entered Array: ");
        for(int i = 0; i < n; i++)
            System.out.print(arr[i] + " ");
        int min = arr[0], max = arr[0];
        for(int i = 0; i < n; i++)
        {
```



```
        if(min > arr[i])
            min = arr[i];
        if(max < arr[i])
            max = arr[i];
    }
    System.out.println("\nMaximum is : " + max);
    System.out.println("Minimum is : " + min);
}
}
```

OUTPUT :

Enter array size: 5

Enter array elements:

67

45

89

80

3

Entered Array:

67 45 89 80 3

Maximum is : 89

Minimum is : 3

RESULT :

Thus the Java program to find both the largest and smallest number in a list of integers is executed and verified successfully.

Ex.No.1.5**PROGRAM USING PACKAGES****Date :**

AIM : To write a Java program to implement a package for currency, distance and time converter.

ALGORITHM :

1. Create a package named converter
2. Inside that package create 3 classes named Currency, Distance and Time
3. Define the methods for conversion in the created class
4. Create objects for the classes in Converter class
5. Call the methods from the Converter class using switch case statement
6. Print the result

PROGRAM :**Currency.java**

```
package converter;
import java.util.Scanner;
import java.io.*;

public class Currency {
    double inr, usd;
    Scanner s = new Scanner(System.in);

    public void dollartorupee() {
        System.out.println("Enter dollars to convert into Rupees");
        usd = s.nextDouble();
        inr = usd * 67;
        System.out.println("Dollar=" + usd + " equal to INR=" + inr);
    }

    public void rupeetodollar() {
        System.out.println("Enter Rupee to convert into Dollars:");
        inr = s.nextDouble();
        usd = inr / 67;
        System.out.println("Rupee=" + inr + " equal to Dollars=" + usd);
    }
}
```

Distance.java

```
package converter;
import java.util.;
import java.io.;

public class Distance {
    Scanner sc = new Scanner(System.in);
    double m, km;

    public void mtokm() {
        System.out.print("Enter in meter");
        m = sc.nextDouble();
        km = (m / 1000);
        System.out.println(m + "m equal to " + km + " kilometres");
    }

    public void kmtom() {
        System.out.print("Enter in km");
        km = sc.nextDouble();
        m = (km * 1000);
        System.out.println(km + "km equal to " + m + " metres");
    }
}
```

Time.java

```
package converter;
import java.util.;
import java.io.;

public class Time {
    int hours, seconds, minutes;
    int input;
    Scanner sc = new Scanner(System.in);

    public void secondtohours() {
        System.out.print("Enter the number of seconds");
        input = sc.nextInt();
        hours = input / 3600;
        minutes = (input % 3600) / 60;
        seconds = (input % 3600) % 60;
        System.out.println("Hours: " + hours);
        System.out.println("Minutes: " + minutes);
    }
}
```

```

        System.out.println("Seconds: " + seconds);
    }
    public void minutestohours() {
        System.out.print("Enter the number of minutes.");
        minutes = sc.nextInt();
        hours = minutes / 60;
        minutes = minutes % 60;
        System.out.println("Hours: " + hours);
        System.out.println("Minutes: " + minutes);
    }
}

```

Converter.java

```

package converter;
import java.util.*;
import java.io.*;

public class Converter {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        int choice = 1, ch;
        Currency c = new Currency();
        Distance d = new Distance();
        Time t = new Time();
        do {
            System.out.println("1. dollar to rupee");
            System.out.println("2. rupee to dollar");
            System.out.println("3. meter to kilometer");
            System.out.println("4. kilometer to meter");
            System.out.println("5. seconds to hours");
            System.out.println("6. minutes to hours");
            choice = s.nextInt();
            switch(choice) {
                case 1: c.dollartorupee(); break;
                case 2: c.rupeetodollar(); break;
                case 3: d.mtokm(); break;
            }
        } while (choice != 0);
    }
}

```

```

        case 4: d.kmtom(); break;
        case 5: t.secondtohours(); break;
        case 6: t.minutestohours(); break;
    }
    System.out.println("Enter 0 to quit and 1 to continue");
    ch = s.nextInt();
    } while(ch == 1);
}
}

```

OUTPUT :

```

1. dollar to rupee
2. rupee to dollar
3. meter to kilometer
4. kilometer to meter
5. seconds to hours
6. minutes to hours
Enter your choice: 1
Enter dollars to convert into Rupees
20
Dollar = 20.0 equal to INR = 1340.0
Enter 0 to quit and 1 to continue
1
1. dollar to rupee
2. rupee to dollar
3. meter to kilometer
4. kilometer to meter
5. seconds to hours
6. minutes to hours
Enter your choice: 4
Enter in km
6
6.0 km equal to 60000.0 metres
Enter 0 to quit and 1 to continue

```

RESULT :

Thus the Java program to implement a package for currency, distance and time converter has been executed and verified successfully.

MODULE 2

Ex.No.2.1 Employee payroll calculation using Hierarchical Inheritance

Date :

AIM : To demonstrate the concepts of inheritance and class hierarchy in Java by creating an Employee class and its subclasses, and to implement salary calculation and pay slip generation.

ALGORITHM :

1. Create a class Employee with member variables like Emp_name, Emp_id, Address, Mail_id, and Mobile_no.
2. Create subclasses Programmer, AssistantProfessor, AssociateProfessor, and Professor that inherit the Employee class.
3. Add a member variable BasicPay to each subclass and calculate the allowances:
DA = 97% of Basic Pay
HRA = 10% of Basic Pay
PF = 12% of Basic Pay
Staff Club Fund = 0.1% of Basic Pay
4. Calculate the gross and net salary.
5. Create a method to generate the pay slip with the employee's details and salary details.

Java Code:

```
class Employee {
    String Emp_name;
    String Emp_id;
    String Address;
    String Mail_id;
    String Mobile_no;

    Employee(String name, String id, String address, String mail, String mobile) {
        Emp_name = name;
        Emp_id = id;
        Address = address;
        Mail_id = mail;
        Mobile_no = mobile;
    }
}

class Programmer extends Employee {
    double BasicPay;

    Programmer(String name, String id, String address, String mail, String mobile, double pay) {
        super(name, id, address, mail, mobile);
        BasicPay = pay;
    }
}
```

```

    }

    double calculateSalary() {
        double DA = 0.97 * BasicPay;
        double HRA = 0.10 * BasicPay;
        double PF = 0.12 * BasicPay;
        double StaffClubFund = 0.001 * BasicPay;
        double grossSalary = BasicPay + DA + HRA;
        double netSalary = grossSalary - PF - StaffClubFund;
        return netSalary;
    }

    void generatePaySlip() {
        double netSalary = calculateSalary();
        System.out.println("Pay Slip for " + Emp_name);
        System.out.println("Employee ID: " + Emp_id);
        System.out.println("Basic Pay: " + BasicPay);
        System.out.println("Net Salary: " + netSalary);
    }
}

public class EmployeeSalary {
    public static void main(String[] args) {
        Programmer p = new Programmer("John", "P123", "123 Street", "john@example.com",
        "1234567890", 50000);
        p.generatePaySlip();
    }
}

```

OUTPUT :

```

Pay Slip for John
Employee ID: P123
Basic Pay: 50000.0
Net Salary: 57300.0

```

RESULT :

Thus, the Java program to calculate Employee Salary using Hierarchical inheritance has been executed and the output is verified successfully.

Ex.No.2.2 Calculating Area of different shapes using Abstract classes

Date :

AIM : To illustrate the use of abstract classes and methods in Java by creating an abstract Shape class and implementing area calculation for different shapes.

ALGORITHM :

1. Create an abstract class Shape with an integer array and an abstract method printArea().
2. Create three classes (Rectangle, Triangle, Circle) that extend Shape.
3. Implement the printArea() method in each class to compute the area of the respective shape.

Java Code:

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

    Shape(int d1, int d2) {
        dimension1 = d1;
        dimension2 = d2;
    }

    abstract void printArea();
}

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

    void printArea() {
        int area = dimension1 * dimension2;
        System.out.println("Area of Rectangle: " + area);
    }
}

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

    void printArea() {
        double area = 0.5 * dimension1 * dimension2;
        System.out.println("Area of Triangle: " + area);
    }
}
```



```

class Circle extends Shape {
    Circle(int radius) {
        super(radius, 0);
    }

    void printArea() {
        double area = Math.PI * Math.pow(dimension1, 2);
        System.out.println("Area of Circle: " + area);
    }
}

public class ShapeTest {
    public static void main(String[] args) {
        Shape s1 = new Rectangle(10, 20);
        s1.printArea();

        Shape s2 = new Triangle(10, 20);
        s2.printArea();

        Shape s3 = new Circle(7);
        s3.printArea();
    }
}

```

OUTPUT :

```

Area of Rectangle: 200
Area of Triangle: 100.0
Area of Circle: 153.93804002589985

```

RESULT :

Thus the java program for abstract class is executed successfully and output is verified.

Ex.No.2.3**String Operations Using ArrayList****Date :**

AIM : To demonstrate the use of ArrayList in Java for performing various string operations such as appending, inserting, searching, and listing strings based on a given criterion.

ALGORITHM :

1. Create an ArrayList to store the strings.
2. Implement functions to:
 - o Append: Add a string at the end.
 - o Insert: Add a string at a specific index.
 - o Search: Check if a string exists.
 - o List strings that start with a given letter.

Java Code:

```
import java.util.ArrayList;
public class StringOperations {
    ArrayList<String> strings = new ArrayList<>();

    void appendString(String str) {
        strings.add(str);
    }

    void insertString(int index, String str) {
        strings.add(index, str);
    }

    boolean searchString(String str) {
        return strings.contains(str);
    }

    void listStringsStartingWith(char letter) {
        for (String str : strings) {
            if (str.charAt(0) == letter) {
                System.out.println(str);
            }
        }
    }

    public static void main(String[] args) {
        StringOperations so = new StringOperations();

        so.appendString("Apple");
        so.appendString("Banana");
    }
}
```

```
so.appendString("Avocado");
so.insertString(1, "Apricot");

System.out.println("Search for 'Banana': " + so.searchString("Banana"));
System.out.println("Strings starting with 'A':");
so.listStringsStartingWith('A');
}
}
```

OUTPUT :

```
Search for 'Banana': true
Strings starting with 'A':
Apple
Apricot
Avocado
```

RESULT :

Thus the java program for String Operations Using ArrayList is executed successfully and output is verified.

MODULE 3

Exp.No.3.1 Exception Handling and User-Defined Exceptions

Date :

AIM : To demonstrate exception handling in Java, specifically the creation and use of user-defined exceptions.

ALGORITHM :

1. Create a user-defined exception class InvalidAgeException.
2. Write a method that checks for valid age input (e.g., age > 0).
3. If the age is invalid, throw the custom exception.

Java Code:

```
class InvalidAgeException extends Exception {
    public InvalidAgeException(String message) {
        super(message);
    }
}

public class ExceptionHandling {
    public static void validateAge(int age) throws InvalidAgeException {
        if (age < 1) {
            throw new InvalidAgeException("Age cannot be less than 1");
        } else {
            System.out.println("Valid Age: " + age);
        }
    }

    public static void main(String[] args) {
        try {
            validateAge(0);
        } catch (InvalidAgeException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

OUTPUT :

Age cannot be less than 1

RESULT :

Thus the java program for Exception Handling is executed successfully and output is verified

Exp.No.3.2**Java Interface for ADT Stack****Date :**

AIM : To illustrate the implementation of an Abstract Data Type (ADT) Stack using a Java interface, and to demonstrate exception handling for stack overflow and underflow.

ALGORITHM :

1. Define an interface StackADT with methods like push(), pop(), peek(), and isEmpty().
2. Implement this interface using an array-based class Stack.
3. Provide necessary exception handling for underflow and overflow.

Java Code:

```
package stacktest;

interface StackADT {
    void push(int item);
    int pop();
    int peek();
    boolean isEmpty();
    void display();
}

class Stack implements StackADT {
    private int maxSize = 5;
    private int[] stackArray = new int[maxSize];
    private int top = -1;

    public void push(int item) {
        if (top >= maxSize - 1) {
            System.out.println("Stack Overflow");
        } else {
            stackArray[++top] = item;
        }
    }

    public int pop(){
        if(top==-1)
        {
            System.out.println("Stack Underflow");
            return -1;
        }
        else
```

```

        return stackArray[top--];

    }

    public int peek() {
        if (top == -1) {
            System.out.println("Stack is Empty");

            return -1;
        }
        else {
            return stackArray[top];
        }
    }

    public boolean isEmpty()
    {
        return top == -1;
    }

    public void display()
    {
        for(int i=0;i<=top;i++)
        {
            System.out.println(stackArray[i]);
        }

    }

}

public class StackTest {
    public static void main(String[] args) {
        Stack s = new Stack();
        s.push(10);
        s.push(30);
        s.push(24);
        s.push(45);
        s.push(20);
        System.out.println(" Stack elements after push");
        s.display();
        System.out.println("top element:"+s.peek());
        System.out.println("popped element"+s.pop());
        s.display();
    }
}

```

```

        System.out.println("popped element"+s.pop());
        s.display();
        System.out.println("popped element"+s.pop());
        s.display();
        System.out.println("popped element"+s.pop());
        s.display();
        System.out.println("popped element"+s.pop());
        s.display();
        System.out.println("Is stack empty? " + s.isEmpty());
        s.display();
    }
}

```

OUTPUT :

Stack elements after push

```

10
30
24
45
20
top element:20
popped element20
10
30
24
45
popped element45
10
30
24
popped element24
10
30
popped element30
10
popped element10
Is stack empty? true

```

RESULT:

Thus the java program for Java Interface for ADT Stack is executed successfully and output is verified

Exp.No.3.3**File Handling in Java****Date :**

AIM : To demonstrate basic file handling operations in Java, including checking file existence, readability, writability, and displaying file details.

ALGORITHM :

1. Read a file name from the user.
2. Check if the file exists and if it is readable/writable.
3. Display the file details, such as file type and size.

Java Code:

```
java
Copy
import java.io.File;

public class FileInfo {
    public static void main(String[] args) {
        File file = new File("test.txt");

        if (file.exists()) {
            System.out.println("File exists");
            System.out.println("Readable: " + file.canRead());
            System.out.println("Writable: " + file.canWrite());
            System.out.println("File Type: " + (file.isDirectory() ? "Directory" : "File"));
            System.out.println("File Size: " + file.length() + " bytes");
        } else {
            System.out.println("File does not exist");
        }
    }
}
```

OUTPUT :

```
File exists
Readable: true
Writable: true
File Type: File
File Size: 1024 bytes
```

RESULT :

Thus the java program for File Handling in Java is executed successfully and output is verified.

MODULE 4

Ex No: 4.1

Multi-threaded application

Date :

AIM : To write a Java program that implements a multi-threaded application with three threads.

ALGORITHM :

1. Create three threads: Number, Square, and Cube.
2. Generate a random number in the Number thread using the Random class.
3. If the generated number is even, start the Square thread and display the square of the number.
4. If the generated number is odd, start the Cube thread and display the cube of the number.

PROGRAM :

```
package multithreading;
import java.util.Random;

class Square extends Thread {
    int x;
    Square(int n) {
        x = n;
    }
    public void run() {
        int sqr = x * x;
        System.out.println("Square of " + x + " = " + sqr);
    }
}

class Cube extends Thread {
    int x;
    Cube(int n) {
        x = n;
    }
    public void run() {
        int cub = x * x * x;
        System.out.println("Cube of " + x + " = " + cub);
    }
}

class Number extends Thread {
    public void run() {
        Random random = new Random();
```

```

for (int i = 0; i < 10; i++) {
    int randomInteger = random.nextInt(100);
    System.out.println("Random Integer generated : " + randomInteger);

    if ((randomInteger % 2) == 0) {
        Square sThread = new Square(randomInteger);
        sThread.start();
    } else {
        Cube cThread = new Cube(randomInteger);
        cThread.start();
    }

    try {
        Thread.sleep(1000);
    } catch (InterruptedException ex) {
        System.out.println(ex);
    }
}

}

}

public class Multithreading {
    public static void main(String args[]) {
        Number n = new Number();
        n.start();
    }
}

```

OUTPUT :

Random Integer generated : 80

Square of 80 = 6400

Random Integer generated : 39

Cube of 39 = 59319

Random Integer generated : 76

Square of 76 = 5776

Random Integer generated : 5

Cube of 5 = 125

Random Integer generated : 18

Square of 18 = 324

Random Integer generated : 83

Cube of 83 = 571787

Random Integer generated : 1

Cube of 1 = 1

Random Integer generated : 62

Square of 62 = 3844

Random Integer generated : 98

Square of 98 = 9604

Random Integer generated : 44

Square of 44 = 1936

RESULT :

Thus the Java program that implements a multi-threaded application has been completed successfully and the result is verified.

Ex No: 4.2**Importing classes from user-defined package****Date :**

AIM : To write a program to implement the concept of importing classes from user-defined package and creating packages.

ALGORITHM :

1. Create classes inside the package: Define one or more classes in the package. Implement required functionality in these classes.
2. Create a package: Define a package using the package keyword. Save the file inside a folder with the package name.
3. Compile the package.
4. Import the package in another Java file using `import package_name.ClassName;`
5. Use the imported class in the main program: Create an instance of the imported class, call its methods, and execute the program.

PROGRAM :

Java program that demonstrates:

1. Creating a user-defined package (mypackage).
2. Defining a class (Calculator) inside the package with arithmetic operations.
3. Importing and using the package in another class (MainApp).

Steps to Run the Program

1. Create a folder where you want to store your Java files.
2. Inside that folder, create a subfolder named mypackage (this will be the package).
3. Save the first file (Calculator.java) inside the mypackage folder.
4. Save the second file (MainApp.java) outside the mypackage folder.
5. Compile and run MainApp.java.

Create the Package (mypackage) and Define a Class

Save the following code as Calculator.java inside the mypackage folder.

```
package mypackage;

public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }

    public int subtract(int a, int b) {
        return a - b;
    }
}
```

```

    }

    public int multiply(int a, int b) {
        return a * b;
    }

    public double divide(int a, int b) throws ArithmeticException {
        if (b == 0) {
            throw new ArithmeticException("Cannot divide by zero");
        }
        return (double) a / b;
    }
}

```

Import and Use the Package

Save the following code as MainApp.java outside the mypackage folder.

```

import mypackage.Calculator;
import java.util.Scanner;

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

        System.out.print("Enter first number: ");
        int num1 = scanner.nextInt();

        System.out.print("Enter second number: ");
        int num2 = scanner.nextInt();

        System.out.println("Addition: " + calc.add(num1, num2));
        System.out.println("Subtraction: " + calc.subtract(num1, num2));
        System.out.println("Multiplication: " + calc.multiply(num1, num2));

        try {
            System.out.println("Division: " + calc.divide(num1, num2));
        } catch (ArithmeticException e) {
            System.out.println("Error: " + e.getMessage());
        }

        scanner.close();
    }
}

```

OUTPUT :

Compile and Run the Program

Compiling the Package (mypackage)

Navigate to the folder where mypackage is located and compile the Calculator.java file:

```
javac -d . mypackage/Calculator.java
```

This -d . option ensures that mypackage is created as a package.

Compiling and Running MainApp.java

Compile MainApp.java : `javac MainApp.java`

Run the program : `java MainApp`

Enter first number: 10

Enter second number: 5

Addition: 15

Subtraction: 5

Multiplication: 50

Division: 2.0

RESULT :

Thus the program to implement the concept of importing classes from user-defined package and creating packages has been completed successfully, and the output is verified.

Ex No: 4.3 Find the maximum value from the given type of elements using a generic function.

Date :

AIM : To write a java program to find the maximum value from the given type of elements using a generic function.

ALGORITHM :

1. Define a generic method: Create a method that takes an array of any comparable type (e.g., Integer, Double, String). Use a type parameter <T extends Comparable<T>> to ensure elements can be compared.
2. Initialize maximum value: Assume the first element is the maximum.
3. Iterate through the array: Compare each element with the current maximum using compareTo(). If an element is greater, update the maximum.
4. Return the maximum value: Once all elements are checked, return the maximum.
5. Test the generic method in the main method: Call the method with different data types (Integer, Double, String).

PROGRAM :

```
class MaxFinder {
    public static <T extends Comparable> T findMax(T[] array) {
        if (array == null || array.length == 0) {
            throw new IllegalArgumentException("Array is empty or null");
        }
        T max = array[0];
        for (T element : array) {
            if (element.compareTo(max) > 0) {
                max = element;
            }
        }
        return max;
    }
}

public class GenericMaxFinder {
    public static void main(String[] args) {
        Integer[] intArray = {10, 25, 5, 88, 42};
        System.out.println("Max Integer: " + MaxFinder.findMax(intArray));
        Double[] doubleArray = {12.5, 7.8, 20.9, 2.3};
        System.out.println("Max Double: " + MaxFinder.findMax(doubleArray));
        String[] stringArray = {"Apple", "Orange", "Banana", "Peach"};
        System.out.println("Max String: " + MaxFinder.findMax(stringArray));
    }
}
```

OUTPUT :

Max Integer: 88
Max Double: 20.9
Max String: Peach

RESULT :

Thus the java program to find the maximum value from the given type of elements using a generic function has been completed successfully and the output is verified.

MODULE 5

Ex No 5.1

Adding elements using ArrayList

Date :

AIM : To write a Java program to create a new array list, add some colors and print the collection.

ALGORITHM :

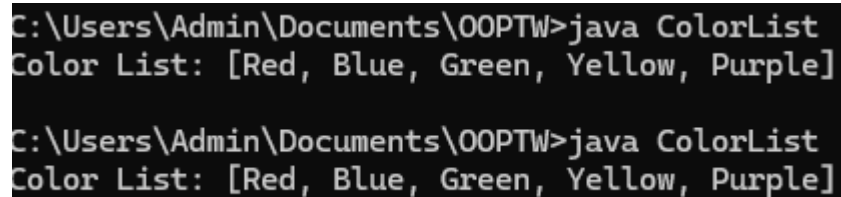
1. Import Required - Package Import java.util.ArrayList.
2. Create an ArrayList - Declare and initialize an ArrayList<String>.
3. Add Colors to the List - Use add() method to insert color names (e.g., "Red", "Blue", "Green").
4. Print the Collection - Use a loop or System.out.println() to display the colors.
5. End the Program - Ensure the program executes successfully.

PROGRAM :

```
import java.util.ArrayList;

public class ColorList {
    public static void main(String[] args) {
        ArrayList<String> colors = new ArrayList<>();
        colors.add("Red");
        colors.add("Blue");
        colors.add("Green");
        colors.add("Yellow");
        colors.add("Purple");
        System.out.println("Color List: " + colors);
    }
}
```

OUTPUT :



```
C:\Users\Admin\Documents\OOPTW>java ColorList
Color List: [Red, Blue, Green, Yellow, Purple]

C:\Users\Admin\Documents\OOPTW>java ColorList
Color List: [Red, Blue, Green, Yellow, Purple]
```

RESULT :

Thus a java program to create a new array list, add some colors and print the collection has been completed successfully and output is verified.

Ex No 5.2**Shuffle elements in array list****Date :****AIM :** To write a Java program to shuffle elements in array list**ALGORITHM :**

1. Import Required Packages - Import java.util.ArrayList and java.util.Collections.
2. Create an ArrayList - Declare and initialize an ArrayList<String>.
3. Add Elements to the List - Use add() method to insert elements (e.g., colors, numbers, etc.).
4. Shuffle the Elements - Use Collections.shuffle() to randomize the order of elements.
5. Print the Shuffled List - Use System.out.println() to display the shuffled elements.
6. End the Program - Ensure the program runs successfully.

PROGRAM :

```
import java.util.ArrayList;
import java.util.Collections;

public class ShuffleArrayList {
    public static void main(String[] args) {

        ArrayList<String> colors = new ArrayList<>();
        colors.add("Red");
        colors.add("Blue");
        colors.add("Green");
        colors.add("Yellow");
        colors.add("Purple");

        System.out.println("Original List: " + colors);

        Collections.shuffle(colors);

        System.out.println("Shuffled List: " + colors);
    }
}
```

OUTPUT :

```
C:\Users\Admin\Documents\OOPTW>javac ShuffleArrayList.java

C:\Users\Admin\Documents\OOPTW>java ShuffleArrayList
Original List: [Red, Blue, Green, Yellow, Purple]
Shuffled List: [Green, Blue, Purple, Yellow, Red]

C:\Users\Admin\Documents\OOPTW>java ShuffleArrayList
Original List: [Red, Blue, Green, Yellow, Purple]
Shuffled List: [Purple, Yellow, Red, Green, Blue]

C:\Users\Admin\Documents\OOPTW>java ShuffleArrayList
Original List: [Red, Blue, Green, Yellow, Purple]
Shuffled List: [Green, Red, Purple, Blue, Yellow]

C:\Users\Admin\Documents\OOPTW>java ShuffleArrayList
Original List: [Red, Blue, Green, Yellow, Purple]
Shuffled List: [Green, Blue, Red, Purple, Yellow]
```

RESULT :

Thus, writing a Java program to shuffle elements in an array list has been completed successfully and output is verified.

Ex No 5.3**Iterate through all elements in a linked list****Date :****AIM :** To write a Java program to iterate through all elements in a linked list**ALGORITHM :**

1. Import Required Packages - Import java.util.LinkedList and java.util.Iterator.
2. Create a LinkedList - Declare and initialize a LinkedList<String>.
3. Add Elements to the List - Use add() method to insert elements.
4. Iterate Using a Loop - Use a for-each loop or for loop to access each element.
5. Iterate Using an Iterator - Create an Iterator and use a while loop with hasNext().
6. Print Each Element - Use System.out.println() to display elements.
7. End the Program - Ensure smooth execution.

PROGRAM :

```
import java.util.LinkedList;
import java.util.Iterator;

public class LinkedListIteration {
    public static void main(String[] args) {
        LinkedList<String> colors = new LinkedList<>();
        colors.add("Red");
        colors.add("Blue");
        colors.add("Green");
        colors.add("Yellow");
        colors.add("Purple");

        System.out.println("Iterating using for-each loop:");
        for (String color : colors) {
            System.out.println(color);
        }

        System.out.println("\nIterating using Iterator:");
        Iterator<String> iterator = colors.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }

        System.out.println("\nIterating using for loop with index:");
        for (int i = 0; i < colors.size(); i++) {
            System.out.println(colors.get(i));
        }
    }
}
```

OUTPUT :

```
C:\Users\Admin\Documents\OOPTW>javac LinkedListIteration.java

C:\Users\Admin\Documents\OOPTW>java LinkedListIteration
Iterating using for-each loop:
Red
Blue
Green
Yellow
Purple

Iterating using Iterator:
Red
Blue
Green
Yellow
Purple

Iterating using for loop with index:
Red
Blue
Green
Yellow
Purple
```

RESULT :

Thus, writing a Java program to iterate through all elements in a linked list has been completed successfully and output is verified.

Ex No 5.4 Adding and searching Students details using Arraylist.

Date :

AIM : To write a Java program to create an ArrayList of Student (id,name,dept,age) objects and search for particular Student objects based on id number.

ALGORITHM :

1. Import Required Packages - Import java.util.ArrayList.
2. Define a Student Class - Create a class with attributes: id, name, dept, and age. Define a constructor to initialize the attributes.
3. Create an ArrayList of Student Objects - Declare and initialize ArrayList<Student>.
4. Add Student Objects to the List - Use add() method to insert multiple student records.
5. Search for a Student by ID - Use a loop (for or for-each) to iterate through the list.
6. Compare each student's id with the given ID.
7. Print the Student Details if Found If a match is found, display student details using System.out.println().
8. Handle Case When Student is Not Found - Print a message if no matching student is found.
9. End the Program - Ensure smooth execution.

PROGRAM :

```
import java.util.ArrayList;
import java.util.Scanner;
class Student {
    int id;
    String name;
    String department;
    int age;

    public Student(int id, String name, String department, int age) {
        this.id = id;
        this.name = name;
        this.department = department;
        this.age = age;
    }

    public void display() {
        System.out.println("ID: " + id + ", Name: " + name + ", Dept: " + department + ", Age: " +
age);
    }
}

public class StudentSearch {
    public static void main(String[] args) {
        ArrayList<Student> students = new ArrayList<>();
```

```

students.add(new Student(101, "Alice", "Computer Science", 20));
students.add(new Student(102, "Bob", "Mechanical", 22));
students.add(new Student(103, "Charlie", "Electrical", 21));
students.add(new Student(104, "David", "Civil", 23));

Scanner scanner = new Scanner(System.in);
System.out.print("Enter Student ID to search: ");
int searchId = scanner.nextInt();
scanner.close();

boolean found = false;
for (Student student : students) {
    if (student.id == searchId) {
        System.out.println("Student Found:");
        student.display();
        found = true;
        break;
    }
}

if (!found) {
    System.out.println("Student with ID " + searchId + " not found.");
}
}

```

OUTPUT :

```
C:\Users\Admin\Documents\OOPTW>javac StudentSearch.java

C:\Users\Admin\Documents\OOPTW>java StudentSearch
Enter Student ID to search: 101
Student Found:
ID: 101, Name: Alice, Dept: Computer Science, Age: 20

C:\Users\Admin\Documents\OOPTW>java StudentSearch
Enter Student ID to search: 203
Student with ID 203 not found.

C:\Users\Admin\Documents\OOPTW>java StudentSearch
Enter Student ID to search: 102
Student Found:
ID: 102, Name: Bob, Dept: Mechanical, Age: 22

C:\Users\Admin\Documents\OOPTW>java StudentSearch
Enter Student ID to search: 105
Student with ID 105 not found.

C:\Users\Admin\Documents\OOPTW>java StudentSearch
Enter Student ID to search: 102
Student Found:
ID: 102, Name: Bob, Dept: Mechanical, Age: 22
```

RESULT :

Thus, writing a Java program to create an ArrayList of Student (id,name,dept,age) objects and search for particular Student objects based on id number. has been completed successfully and output is verified.

Ex No 5.5**ArrayList - Store only char and String.****Date :**

AIM : To write a Java program to create an ArrayList which will be able to store only char and String but not any other data type.

ALGORITHM :

1. Create an ArrayList of Type Character and String - Use ArrayList<Object> to store both Character and String values.
2. Add Only char and String Values - Use instanceof to ensure only Character and String types are added.
3. Iterate and Display Elements
4. Loop through the list and print the stored values.

PROGRAM :

```
import java.util.ArrayList;

public class CharStringArrayList {
    public static void main(String[] args) {
        ArrayList<Object> list = new ArrayList<>();

        addElement(list, 'A'); // Character
        addElement(list, "Hello"); // String
        addElement(list, 'B');
        addElement(list, "Java");

        addElement(list, 100); // This should be rejected

        System.out.println("Valid Char & String List: " + list);
    }

    public static void addElement(ArrayList<Object> list, Object element) {
        if (element instanceof Character || element instanceof String) {
            list.add(element);
        } else {
            System.out.println("Error: Only Character and String are allowed! Attempted to add: " +
            element);
        }
    }
}
```

OUTPUT :

```
C:\Users\Admin\Documents\OOPW>javac CharStringArrayList.java  
C:\Users\Admin\Documents\OOPW>java CharStringArrayList  
Error: Only Character and String are allowed! Attempted to add: 100  
Valid Char & String List: [A, Hello, B, Java]
```

RESULT :

Thus, writing a Java program to create an ArrayList which will be able to store only char and String but not any other data type. has been completed successfully and output is verified.

Ex No 5.6**Queue Collection for Cinema Ticket Sale.****Date :****AIM :** To write a Java program using Queue Collection for Cinema Ticket Sale.**ALGORITHM :**

1. Create a Queue for Customers - Use Queue<String> with LinkedList to store customer names in the order they arrive.
2. Process Ticket Sales - Use poll() to remove and serve customers one by one.
3. Display the Queue Status - Print the queue before and after ticket sales to show remaining customers.

PROGRAM :

```
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;

class CinemaTicketQueue {
    public static void main(String[] args) {
        Queue<String> ticketQueue = new LinkedList<>();
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter customer names (type 'done' to stop): ");
        while (true) {
            String name = scanner.nextLine();
            if (name.equalsIgnoreCase("done")) break;
            ticketQueue.add(name);
        }

        System.out.println("\nProcessing ticket sales...");
        while (!ticketQueue.isEmpty()) {
            String customer = ticketQueue.poll(); // Serve the first customer
            System.out.println("Ticket sold to: " + customer);
        }

        System.out.println("\nAll tickets sold. Queue is empty!");
        scanner.close();
    }
}
```

OUTPUT :

How It Works:

- Customers enter their names → They are added to the Queue.
- First customer in line gets served first (poll() method).
- Loop continues until all customers are served.

```
C:\Users\Admin\Documents\OOPTW>javac CinemaTicketQueue.java

C:\Users\Admin\Documents\OOPTW>java CinemaTicketQueue
Enter customer names (type 'done' to stop):
Priya
Varsha
Sanchana
Josephine
Vijay
done

Processing ticket sales...
Ticket sold to: Priya
Ticket sold to: Varsha
Ticket sold to: Sanchana
Ticket sold to: Josephine
Ticket sold to: Vijay

All tickets sold. Queue is empty!
```

RESULT :

Thus, writing a Java program using Queue Collection for Cinema Ticket Sale has been completed successfully and output is verified.

MODULE 6

Ex No 6.1

Finding sum and difference using Applet

Date :

AIM : To develop an Applet program to accept two numbers from the user and output the sum and difference in the respective text boxes.

ALGORITHM :

- **Initialize Applet Components:**
 1. Create two TextField components for the user to enter the two numbers.
 2. Create two TextField components to display the sum and difference.
 3. Create a Button to trigger the calculation.
- **Accept User Input:** Use TextField components to get the two numbers from the user.
- **Perform Calculation:**
 1. On pressing the button, retrieve the numbers entered by the user.
 2. Convert these input values (strings) to integers.
 3. Calculate the sum and the difference of the two numbers.
- **Display the Results:** Set the sum and difference values to the respective result text boxes
- **Handle Events:**

Implement an event listener for the button click to trigger the calculation.

PROGRAM :

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

public class SumDifferenceApplet extends Applet implements ActionListener {
    TextField num1Field, num2Field, sumField, diffField;
    Button calculateButton;

    public void init() {
        Label num1Label = new Label("Enter Number 1:");
        Label num2Label = new Label("Enter Number 2:");
        Label sumLabel = new Label("Sum:");
        Label diffLabel = new Label("Difference:");

        num1Field = new TextField(10);
        num2Field = new TextField(10);
        sumField = new TextField(10);
```

```

diffField = new TextField(10);

sumField.setEditable(false);
diffField.setEditable(false);

calculateButton = new Button("Calculate");
calculateButton.addActionListener(this);

setLayout(new GridLayout(5, 2)); // 5 Rows, 2 Columns

add(num1Label);
add(num1Field);
add(num2Label);
add(num2Field);
add(sumLabel);
add(sumField);
add(diffLabel);
add(diffField);
add(calculateButton);
}

public void actionPerformed(ActionEvent e) {
    try {
        int num1 = Integer.parseInt(num1Field.getText());
        int num2 = Integer.parseInt(num2Field.getText());

        int sum = num1 + num2;
        int diff = num1 - num2;

        sumField.setText(String.valueOf(sum));
        diffField.setText(String.valueOf(diff));
    } catch (NumberFormatException ex) {
        sumField.setText("Invalid Input");
        diffField.setText("Invalid Input");
    }
}
}

```

Steps to Run the Applet

Save the file as SumDifferenceApplet.java

Compile the Java file

```
javac SumDifferenceApplet.java
```

Create an HTML file (applet.html)

```
<html>
```

```
<body>
```

```
    <applet code="SumDifferenceApplet.class" width="300" height="200"></applet>
```

```
</body>
```

```
</html>
```

Run using Applet Viewer

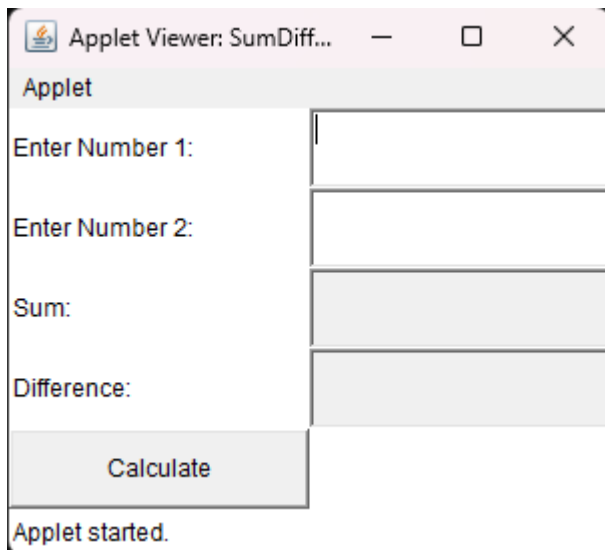
```
appletviewer applet.html
```

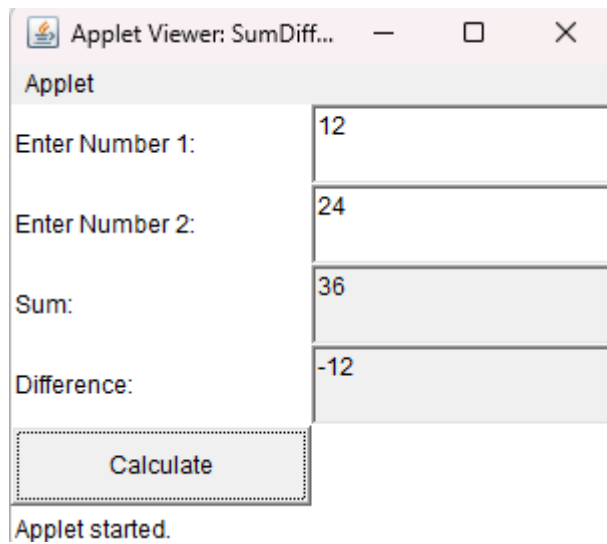
OUTPUT :

```
C:\Users\Admin\Documents\OOPTW>javac SumDifferenceApplet.java

C:\Users\Admin\Documents\OOPTW>appletviewer applet.html
Warning: Can't read AppletViewer properties file: C:\Users\Admin\.hotjava\properties Using defaults.

C:\Users\Admin\Documents\OOPTW>|
```



**RESULT :**

Thus the program to develop an Applet program to accept two numbers from the user and output the sum and difference in the respective text boxes is completed successfully and the output is verified.

Ex No 6.2**Keyboard events using Applet****Date :****AIM :** To write a program that identifies key-up key-down event user entering text in a Applet**ALGORITHM :**

1. Initialize Applet Components:
 - Create a TextField or TextArea for user text input.
 - Create Label or TextField components to display the messages for keyPressed (key-down) and keyReleased (key-up) events.
2. Add KeyListener:
 - Attach a KeyListener to the text input component (e.g., TextField).
 - Implement the methods keyPressed, keyReleased, and keyTyped of the KeyListener interface.
3. Handle Key Events:
 - keyPressed(KeyEvent e): This method will be triggered when a key is pressed down (key-down event).
 - keyReleased(KeyEvent e): This method will be triggered when a key is released (key-up event).
 - keyTyped(KeyEvent e): This method can be used to identify the key typed (if necessary).
4. Display Event Information:
 - Use Label or TextField components to display the key information (such as the key code or the key character) for both keyPressed and keyReleased.

PROGRAM :

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

public class KeyEventApplet extends Applet implements KeyListener {
    String message = "";

    public void init() {
        addKeyListener(this);
        setFocusable(true);
    }

    public void keyPressed(KeyEvent e) {
        message = "Key Down: " + e.getKeyChar();
```

```
        repaint();
    }

    public void keyReleased(KeyEvent e) {
        message = "Key Up: " + e.getKeyChar();
        repaint();
    }

    public void keyTyped(KeyEvent e) {
        message = "Key Typed: " + e.getKeyChar();
        repaint();
    }

    public void paint(Graphics g) {
        g.drawString(message, 20, 50);
    }
}
```

OUTPUT :

Save the file as KeyEventApplet.java

Compile the Java file

```
javac KeyEventApplet.java
```

Create an HTML file (applet.html)

```
<html>
```

```
<body>
```

```
<applet code="KeyEventApplet.class" width="300" height="200"></applet>
```

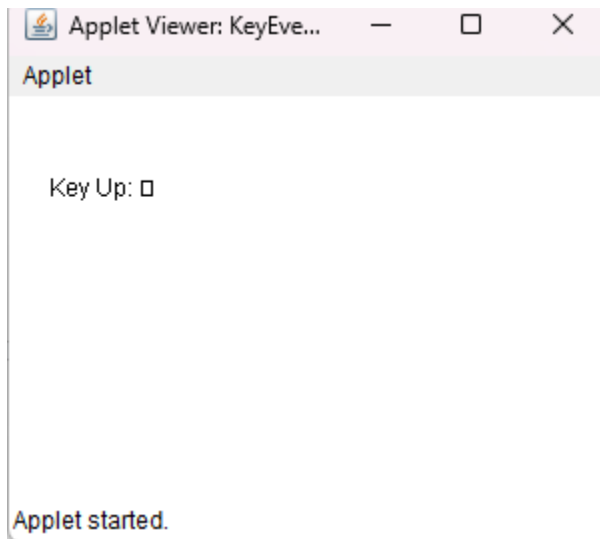
```
</body>
```

```
</html>
```

Run using Applet Viewer

```
appletviewer applet.html
```

```
C:\Users\Admin\Documents\OOPTW>javac KeyEventApplet.java
C:\Users\Admin\Documents\OOPTW>appletviewer appletkeyevent.html
```



RESULT :

Thus to write a program that identifies key-up key-down event user entering text in a Applet is completed successfully and output is verified.

Ex No 6.3**Student registration form using Swing controls.**

Date :

AIM : To write a Java program to design student registration forms using Swing controls.

ALGORITHM :**Initialize the Swing Component**

1. Create a JFrame for the main window.
2. Add JLabels for form fields (Name, Age, Gender, Course, etc.).
3. Add JTextFields for text input.
4. Add JRadioButtons for gender selection (Group them using ButtonGroup).
5. Add JComboBox for selecting a course.
6. Add JCheckBox for agreement or terms selection.
7. Add JButton for form submission and reset.

Set Layout and Position Components

Use GridLayout or GridBagLayout for proper alignment. Place labels and corresponding input fields properly.

Add Action Listeners

Implement an ActionListener for the submit button to fetch input values. Implement an ActionListener for the reset button to clear fields.

Perform Data Validation

Check if all required fields are filled. Validate the age field to accept only numbers. Ensure gender is selected.

Display Confirmation Message

Show a message dialog on successful registration.

Run the Application

Create the frame object in main and set it to be visible.

PROGRAM :

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

public class StudentRegistrationForm extends JFrame implements ActionListener {
    JTextField nameField, ageField;
    JRadioButton male, female;
    JComboBox<String> departmentBox;
    JCheckBox javaCheck, pythonCheck, cppCheck;
```

```

JButton submitButton, resetButton;

public StudentRegistrationForm() {
    setTitle("Student Registration Form");
    setSize(400, 400);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLayout(new GridLayout(7, 2, 10, 10)); // 7 rows, 2 columns

    add(new JLabel("Name:"));
    nameField = new JTextField();
    add(nameField);

    add(new JLabel("Age:"));
    ageField = new JTextField();
    add(ageField);

    add(new JLabel("Gender:"));
    JPanel genderPanel = new JPanel();
    male = new JRadioButton("Male");
    female = new JRadioButton("Female");
    ButtonGroup genderGroup = new ButtonGroup();
    genderGroup.add(male);
    genderGroup.add(female);
    genderPanel.add(male);
    genderPanel.add(female);
    add(genderPanel);

    add(new JLabel("Department:"));
    String[] departments = { "Computer Science", "Electronics", "Mechanical", "Civil" };
    departmentBox = new JComboBox<>(departments);
    add(departmentBox);

    add(new JLabel("Courses:"));
    JPanel coursePanel = new JPanel();
    javaCheck = new JCheckBox("Java");
    pythonCheck = new JCheckBox("Python");
    cppCheck = new JCheckBox("C++");
    coursePanel.add(javaCheck);
    coursePanel.add(pythonCheck);
    coursePanel.add(cppCheck);
    add(coursePanel);

    submitButton = new JButton("Submit");
    submitButton.addActionListener(this);
    add(submitButton);

```

```

resetButton = new JButton("Reset");
resetButton.addActionListener(e -> {
    nameField.setText("");
    ageField.setText("");
    genderGroup.clearSelection();
    departmentBox.setSelectedIndex(0);
    javaCheck.setSelected(false);
    pythonCheck.setSelected(false);
    cppCheck.setSelected(false);
});
add(resetButton);

setVisible(true);
}

public void actionPerformed(ActionEvent e) {
    String name = nameField.getText();
    String age = ageField.getText();
    String gender = male.isSelected() ? "Male" : female.isSelected() ? "Female" : "Not
Selected";
    String department = (String) departmentBox.getSelectedItem();
    String courses = "";
    if (javaCheck.isSelected()) courses += "Java ";
    if (pythonCheck.isSelected()) courses += "Python ";
    if (cppCheck.isSelected()) courses += "C++ ";

    JOptionPane.showMessageDialog(this,
        "Name: " + name + "\nAge: " + age + "\nGender: " + gender + "\nDepartment: " +
department + "\nCourses: " + courses,
        "Registration Details",
        JOptionPane.INFORMATION_MESSAGE);
}

public static void main(String[] args) {
    new StudentRegistrationForm();
}
}

```

OUTPUT :

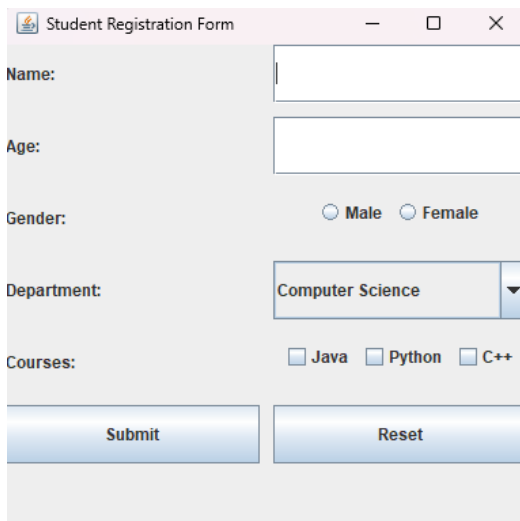
Save the file as: StudentRegistrationForm.java

Compile the program:

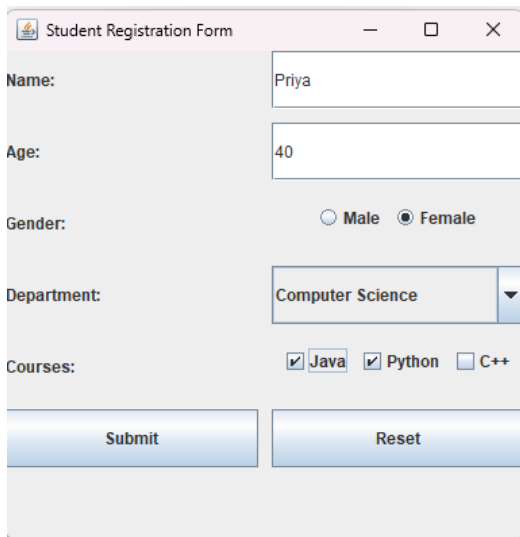
```
javac StudentRegistrationForm.java
```

Run the program:

```
java StudentRegistrationForm
```



A screenshot of a Java Swing window titled "Student Registration Form". The window contains several input fields and controls: a text field for "Name:", a text field for "Age:", radio buttons for "Gender:" (Male and Female), a dropdown menu for "Department:" (currently showing "Computer Science"), and checkboxes for "Courses:" (Java, Python, and C++). At the bottom, there are two buttons: "Submit" and "Reset".



A second screenshot of the same "Student Registration Form" window, but with data entered into the fields. The "Name:" field contains "Priya", the "Age:" field contains "40", the "Gender:" radio buttons have "Female" selected, the "Department:" dropdown still shows "Computer Science", and the "Courses:" checkboxes have "Java" and "Python" selected, while "C++" remains unselected. The "Submit" and "Reset" buttons are still present at the bottom.

RESULT :

Thus to write a Java program to design student registration forms using Swing controls has been completed successfully and output is verified.

Ex No 6.4

Digital watch in swing

Date :

AIM : To write a program to display the digital watch in swing

ALGORITHM :

Initialize Swing Components

Create a JFrame for the main window. Add a JLabel to display the time.

Set Layout and Styling

Use a suitable layout (FlowLayout, BorderLayout, etc.). Set font size and style for better visibility.

Create a Timer Using javax.swing.Timer

Set a timer to trigger every second (1000ms).

Get the Current Time

Use LocalDateTime or SimpleDateFormat to fetch the system time.

Format the Time Properly

Convert time to a readable format (HH:mm:ss).

Update the JLabel with Current Time

Set the formatted time as text in the label.

Start the Timer

Use timer.start() to continuously update the time.

Handle Window Closing

Set setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE).

Make the Frame Visible

Use setVisible(true) to display the watch.

Run the Application in the Main Method

Instantiate the watch class and execute it.

PROGRAM :

```
import javax.swing.*;
import java.awt.*;
import java.text.SimpleDateFormat;
import java.util.Date;

public class DigitalClock extends JFrame {
```



```

private JLabel timeLabel;
private SimpleDateFormat timeFormat;

public DigitalClock() {
    setTitle("Digital Clock");
    setSize(300, 150);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLayout(new BorderLayout());

    timeFormat = new SimpleDateFormat("hh:mm:ss a");

    timeLabel = new JLabel();
    timeLabel.setHorizontalAlignment(SwingConstants.CENTER);
    timeLabel.setFont(new Font("Arial", Font.BOLD, 40));
    timeLabel.setForeground(Color.BLUE);

    add(timeLabel, BorderLayout.CENTER);

    Timer timer = new Timer(1000, e -> updateTime());
    timer.start();

    updateTime();
    setVisible(true);
}

private void updateTime() {
    String time = timeFormat.format(new Date());
    timeLabel.setText(time);
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(DigitalClock::new);
}
}

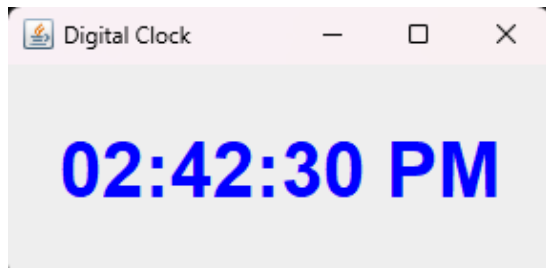
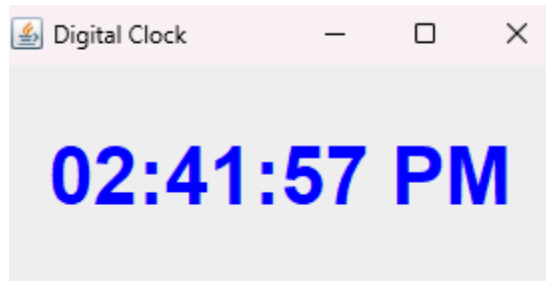
```

OUTPUT :

Save the file as: DigitalClock.java

Compile the program: javac DigitalClock.java

Run the program: java DigitalClock



RESULT :

Thus to write a java program to display the digital watch in swing has been completed successfully and output is verified.

