Q.1.

import java.awt.\*;

import java.awt.event.\*;

public class button extends Frame {

private TextField textField;

private Label resultLabel;

public button() {

setLayout(new BorderLayout());

Label enterLabel = new Label("Enter No:");

textField = new TextField();

resultLabel = new Label("Result:");

Panel panel = new Panel();

panel.setLayout(new GridLayout(4, 1));

Button primeButton = new Button("Check Prime");

Button factorialButton = new Button("Factorial");

Button evenOddButton = new Button("Even or Odd");

Button divisibleBy5Button = new Button("Divisible by 5");

primeButton.addActionListener(e -> prime());

factorialButton.addActionListener(e -> factorial());

evenOddButton.addActionListener(e -> evenOdd());

divisibleBy5Button.addActionListener(e -> divisibleBy5());

panel.add(primeButton);

panel.add(factorialButton);

panel.add(evenOddButton);

panel.add(divisibleBy5Button);

add(enterLabel, BorderLayout.WEST);

add(textField, BorderLayout.CENTER);

add(panel, BorderLayout.EAST);

add(resultLabel, BorderLayout.SOUTH);

setTitle("Button");

setSize(400, 300);

setVisible(true);

addWindowListener(new WindowAdapter() {

public void windowClosing(WindowEvent e) {

System.exit(0);

}

});

}

private void prime() {

setResultLabel("is Prime: " + isPrime(getInput()));

}

private void factorial() {

setResultLabel("Factorial: " + calculateFactorial(getInput()));

}

private void evenOdd() {

setResultLabel("Even or Odd: " + (isEven(getInput()) ? "Even" : "Odd"));

}

private void divisibleBy5() {

setResultLabel("Divisible by 5: " + (isDivisibleBy5(getInput()) ? "Yes" : "No"));

}

private int getInput() {

try {

return Integer.parseInt(textField.getText());

} catch (NumberFormatException ex) {

setResultLabel("Invalid input. Please enter a valid number.");

return 0;

}

}

private void setResultLabel(String result) {

resultLabel.setText("Result: " + result);

}

private boolean isPrime(int num) {

if (num <= 1) {

return false;

}

for (int i = 2; i <= Math.sqrt(num); i++) {

if (num % i == 0) {

return false;

}

}

return true;

}

private int calculateFactorial(int num) {

int result = 1;

for (int i = 1; i <= num; i++) {

result \*= i;

}

return result;

}

private boolean isEven(int num) {

return num % 2 == 0;

}

private boolean isDivisibleBy5(int num) {

return num % 5 == 0;

}

public static void main(String[] args) {

new button();

}

}

Q.2.

import java.awt.\*;

import java.awt.event.\*;

public class StudentIDCardFormAWT extends Frame {

private TextField nameField, idField, courseField;

private Label resultLabel;

public StudentIDCardFormAWT() {

setTitle("Student ID Card Form");

setSize(400, 300);

setLayout(new GridLayout(5, 2));

Label nameLabel = new Label("Name:");

nameField = new TextField();

Label idLabel = new Label("Student ID:");

idField = new TextField();

Label courseLabel = new Label("Course:");

courseField = new TextField();

Button submitButton = new Button("Submit");

resultLabel = new Label("");

add(nameLabel);

add(nameField);

add(idLabel);

add(idField);

add(courseLabel);

add(courseField);

add(new Label()); // Empty cell for spacing

add(submitButton);

add(new Label()); // Empty cell for spacing

add(resultLabel);

submitButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

displayStudentDetails();

}

});

addWindowListener(new WindowAdapter() {

public void windowClosing(WindowEvent e) {

System.exit(0);

}

});

}

private void displayStudentDetails() {

String name = nameField.getText();

String studentID = idField.getText();

String course = courseField.getText();

if (name.isEmpty() || studentID.isEmpty() || course.isEmpty()) {

resultLabel.setText("Please fill in all fields.");

} else {

String details = "Name: " + name + ",\n Student ID: " + studentID + ",\n Course: " + course;

resultLabel.setText(details);

}

}

public static void main(String[] args) {

EventQueue.invokeLater(new Runnable() {

public void run() {

new StudentIDCardFormAWT().setVisible(true);

}

});

}

}

Q.3

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class extends Frame implements ActionListener {

private TextField display;

private String input = "";

public CalculatorWithMenu() {

setTitle("Calculator");

setSize(300, 400);

setLayout(new BorderLayout());

display = new TextField();

display.setEditable(false);

add(display, BorderLayout.NORTH);

Panel buttonPanel = new Panel();

buttonPanel.setLayout(new GridLayout(4, 4));

String[] buttonLabels = {

"7", "8", "9", "/",

"4", "5", "6", "\*",

"1", "2", "3", "-",

"0", ".", "=", "+"

};

for (String label : buttonLabels) {

Button button = new Button(label);

button.addActionListener(this);

buttonPanel.add(button);

}

add(buttonPanel, BorderLayout.CENTER);

MenuBar menuBar = new MenuBar();

setMenuBar(menuBar);

Menu viewMenu = new Menu("View");

MenuItem standardItem = new MenuItem("Standard");

MenuItem scientificItem = new MenuItem("Scientific");

MenuItem programmerItem = new MenuItem("Programmer");

standardItem.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

// Add code for switching to the standard calculator view

setTitle("Standard Calculator");

}

});

scientificItem.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

// Add code for switching to the scientific calculator view

setTitle("Scientific Calculator");

}

});

programmerItem.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

// Add code for switching to the programmer calculator view

setTitle("Programmer Calculator");

}

});

viewMenu.add(standardItem);

viewMenu.add(scientificItem);

viewMenu.add(programmerItem);

menuBar.add(viewMenu);

addWindowListener(new java.awt.event.WindowAdapter() {

public void windowClosing(java.awt.event.WindowEvent windowEvent) {

System.exit(0);

}

});

}

public void actionPerformed(ActionEvent e) {

String actionCommand = e.getActionCommand();

if (actionCommand.equals("=")) {

evaluateExpression();

} else {

input += actionCommand;

display.setText(input);

}

}

private void evaluateExpression() {

try {

String result = String.valueOf(eval(input));

display.setText(result);

input = result;

} catch (Exception ex) {

display.setText("Error");

input = "";

}

}

private double eval(String expression) {

return new Object() {

int pos = -1, ch;

void nextChar() {

ch = (++pos < expression.length()) ? expression.charAt(pos) : -1;

}

boolean isDigit(char c) {

return Character.isDigit(c);

}

double parse() {

nextChar();

double x = parseExpression();

if (pos < expression.length()) throw new RuntimeException("Unexpected: " + (char) ch);

return x;

}

double parseExpression() {

double x = parseTerm();

for (; ; ) {

if (eat('+')) x += parseTerm(); // addition

else if (eat('-')) x -= parseTerm(); // subtraction

else return x;

}

}

double parseTerm() {

double x = parseFactor();

for (; ; ) {

if (eat('\*')) x \*= parseFactor(); // multiplication

else if (eat('/')) x /= parseFactor(); // division

else return x;

}

}

double parseFactor() {

if (eat('+')) return parseFactor(); // unary plus

if (eat('-')) return -parseFactor(); // unary minus

double x;

int startPos = this.pos;

if (eat('(')) { // parentheses

x = parseExpression();

eat(')');

} else if (isDigit((char) ch) || ch == '.') { // numbers

while (isDigit((char) ch) || ch == '.') nextChar();

x = Double.parseDouble(expression.substring(startPos, this.pos));

} else {

throw new RuntimeException("Unexpected: " + (char) ch);

}

return x;

}

boolean eat(int charToEat) {

while (Character.isWhitespace(ch)) nextChar();

if (ch == charToEat) {

nextChar();

return true;

}

return false;

}

}.parse();

}

public static void main(String[] args) {

EventQueue.invokeLater(new Runnable() {

public void run() {

new CalculatorWithMenu().setVisible(true);

}

});

}

}

Q.4 & Q.5 From Notebook

Q6.

public class ExceptionHandlingExample {

// User-defined exception class

static class CustomException extends Exception {

public CustomException(String message) {

super(message);

}

}

public static void main(String[] args) {

try {

// System-defined exception (ArithmeticException)

int result = 10 / 0; // ArithmeticException: division by zero

// User-defined exception

if (result == 0) {

throw new CustomException("Result is zero");

}

} catch (ArithmeticException e) {

// Handling system-defined exception

System.out.println("Caught system-defined exception: " + e.getMessage());

} catch (CustomException e) {

// Handling user-defined exception

System.out.println("Caught user-defined exception: " + e.getMessage());

} finally {

// Code in finally block always executes, whether exception occurred or not

System.out.println("Finally block executed");

}

}

}

Q7.

First Program:

package myPackage;

// Class in the user-defined package

public class Cal {

    public int add(int a, int b) {

        return a + b;

    }

}

import myPackage.\*;

// Class in the user-defined sub-package

public class SubClass {

public static void main(String[] args) {

Cal cal = new Cal();

int result = cal.add(5, 7);

System.out.println("Addition result: " + result);

}

}

Second Program :

package myPackage;

// Class in the user-defined package

public class MyClass {

public void display() {

System.out.println("This is from MyClass in the user-defined package.");

}

}

;

import myPackage.\*;

// Class in the user-defined sub-package

public class SubClass {

public void display() {

System.out.println("This is from SubClass in the user-defined sub-package.");

}

public static void main(String[] args) {

MyClass myObj = new MyClass();

myObj.display();

}

}

Q8 .

First Program :

// Define an interface with three methods

interface Calculation {

void add(int a, int b);

void subtract(int a, int b);

void multiply(int a, int b);

}

// Implement the interface in a class

class Calculator implements Calculation {

// Implementing add method

public void add(int a, int b) {

System.out.println("Addition: " + (a + b));

}

// Implementing subtract method

public void subtract(int a, int b) {

System.out.println("Subtraction: " + (a - b));

}

// Implementing multiply method

public void multiply(int a, int b) {

System.out.println("Multiplication: " + (a \* b));

}

// Main method to demonstrate the usage

public static void main(String[] args) {

Calculator calculator = new Calculator();

calculator.add(10, 5);

calculator.subtract(10, 5);

calculator.multiply(10, 5);

}

}

Second program :

interface MyInterface {

void method1();

void method2();

void method3();

}

class MyClass implements MyInterface {

public void method1() {

System.out.println("Method 1 implemented.");

}

public void method2() {

System.out.println("Method 2 implemented.");

}

public void method3() {

System.out.println("Method 3 implemented.");

}

// Main method to demonstrate the usage

public static void main(String[] args) {

MyClass obj = new MyClass();

obj.method1();

obj.method2();

obj.method3();

}

}

Q9.

// Abstract class representing a Vehicle

abstract class Vehicle {

// Encapsulated field

private String model;

// Constructor

public Vehicle(String model) {

this.model = model;

}

// Abstract method to start the vehicle

abstract void start();

// Getter method for model

public String getModel() {

return model;

}

}

// Concrete subclass Car inheriting from Vehicle

class Car extends Vehicle {

// Constructor

public Car(String model) {

super(model);

}

// Implementation of abstract method to start the car

@Override

void start() {

System.out.println("Starting the car " + getModel());

}

}

public class Main {

public static void main(String[] args) {

// Create a Car object

Car car = new Car("Toyota");

// Start the car

car.start();

}

}

Explanation:

* **Vehicle** is an abstract class representing a vehicle. It contains an encapsulated field **model** and an abstract method **start()** to start the vehicle.
* **Car** is a concrete subclass of **Vehicle**, representing a specific type of vehicle. It inherits from **Vehicle** and provides an implementation for the **start()** method.
* Abstraction is achieved through the use of an abstract class (**Vehicle**) and an abstract method (**start()**).
* Inheritance is demonstrated by **Car** inheriting from **Vehicle**.
* Encapsulation is demonstrated by encapsulating the **model** field in the **Vehicle** class and providing a getter method to access it.

In the **Main** class, we create an object of **Car**, set its model, and start the car.

Q10. New Stage : public class NewStateDemo {

public static void main(String[] args) {

Thread thread = new Thread(() -> {

System.out.println("Thread is created but not yet started.");

});

System.out.println("Thread created.");

}

}

Run Stage : public class A extends Thread {

public void run() {

System.out.println("Thread is Running");

}

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

A t1 = new A();

t1.start();

}

}

Sleep Stage : Sleep ka Program Terminate Stage : public class A extends Thread {

public void run() {

System.out.println("Thread is running");

}

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

A t1 = new A();

t1.start();

t1.join();

System.out.println("Thread terminated.");

}

}

Q11.

Sleep aur Join ka program notebook se

The setPriority() method is used to set the priority of a thread. Threads with higher priority are given preference in CPU scheduling over threads with lower priority.

The priority range is from 1 to 10, where 1 is the lowest priority and 10 is the highest.

The syntax for setPriority() is thread.setPriority(priority), where thread is the thread whose priority you want to set, and priority is the priority value. public class P extends Thread

{

public void run()

{

System.out.println("Running");

}

public static void main(String args[])

{

P t1=new P();

P t2=new P();

t1.setPriority(4);

t2.setPriority(7);

System.out.println("Priority of thread t1 is: " + t1.getPriority());

System.out.println("Priority of thread t2 is: " + t2.getPriority());

t1.start();

}

}

Q12.

Program using **InputStream**:

import java.io.FileInputStream;

import java.io.IOException;

public class InputStreamExample {

public static void main(String[] args) {

try {

FileInputStream inputStream = new FileInputStream("input.txt");

int data;

while ((data = inputStream.read()) != -1) {

System.out.print((char) data);

}

inputStream.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

Program using **OutputStream**:

import java.io.FileOutputStream;

import java.io.IOException;

public class OutputStreamExample {

public static void main(String[] args) {

try {

FileOutputStream outputStream = new FileOutputStream("output.txt");

String data = "Hello, World!";

byte[] byteArray = data.getBytes();

outputStream.write(byteArray);

outputStream.close();

System.out.println("Data has been written to output.txt.");

} catch (IOException e) {

e.printStackTrace();

}

}

}

Program using both **InputStream** and **OutputStream**:

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;

public class InputStreamOutputStreamExample {

public static void main(String[] args) {

try {

FileInputStream inputStream = new FileInputStream("input.txt");

FileOutputStream outputStream = new FileOutputStream("output.txt");

int data;

while ((data = inputStream.read()) != -1) {

outputStream.write(data);

}

inputStream.close();

outputStream.close();

System.out.println("Data has been copied from input.txt to output.txt.");

} catch (IOException e) {

e.printStackTrace();

}

}

}