Q1. Design a stack class. Provide your own stack exceptions namely Push Exception and

Pop Exception, which throw exceptions when the stack is full and when the stack is

empty respectively. Show the usage of these exceptions in handling a stack object in

the main.

Code :

// Custom exception for stack overflow

class PushException extends Exception

{ public PushException(String message) { super(message); } } // Custom exception for stack underflow class PopException extends Exception

{ public PopException(String message)

{ super(message); } }

// Stack class with custom push and pop operations class Stack

{ private int[] stackArray; private int top; private int maxSize; // Constructor to initialize the stack public Stack(int size)

{ this.maxSize = size; stackArray = new int[maxSize]; top = -1; } // Push method to add an element to the stack public void push(int value) throws PushException {

if (top == maxSize – 1

) { throw new PushException("Stack is full. Cannot push element: " + value); }

stackArray[++top] = value; }

// Pop method to remove the top element from the stack public int pop() throws PopException

{ if (top == -1)

{ throw new PopException("Stack is empty. Cannot pop element."); }

return stackArray[top--]; } // Method to display the elements in the stack public void display() { if (top == -1)

{ System.out.println("Stack is empty."); return; }

System.out.print("Stack elements: ");

for (int i = 0; i <= top; i++) { System.out.print(stackArray[i] + " "); }

System.out.println(); } }

public class StackDemo

{ public static void main(String[] args)

{ Stack stack = new Stack(3); // Creating a stack with a max size of 3 try

{ // Pushing elements into the stack stack.push(10); stack.push(20); stack.push(30); stack.display(); // Displaying the stack // Attempting to push another element (should throw PushException) stack.push(40); }

catch (PushException e)

{ System.out.println(e.getMessage()); }

try { // Popping elements from the stack System.out.println("Popped: " + stack.pop()); System.out.println("Popped: " + stack.pop());

System.out.println("Popped: " + stack.pop()); // Attempting to pop from an empty stack (should throw PopException) stack.pop(); }

catch (PopException e)

{ System.out.println(e.getMessage()); } } }

Output

Stack elements: 10 20 30 Stack is full.

Cannot push element: 40 Popped: 30 Popped: 20 Popped: 10 Stack is empty.

Cannot pop element

Q2 Define a class CurrentDate with data members day, month and year. Define a method

createDate() to create date object by reading values from keyboard. Throw a user

defined exception by name InvalidDayException if the day is invalid and

InvalidMonthException if month is found invalid and display current date if the date

is valid. Write a test program to illustrate the functionality.

Code:

import java.util.Scanner;

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

class InvalidMonthException extends Exception

{ public InvalidMonthException(String message)

{ super(message); } }

class CurrentDate

{ private int day; private int month; private int year; public void createDate() throws InvalidDayException, InvalidMonthException

{ Scanner scanner = new Scanner(System.in);

System.out.print("Enter day: "); day = scanner.nextInt(); System.out.print("Enter month: ");

month = scanner.nextInt(); System.out.print("Enter year: ");

year = scanner.nextInt();

if (month < 1 || month > 12)

{ throw new InvalidMonthException("Invalid month! Must be between 1 and 12."); }

if (day < 1 || day > getDaysInMonth(month, year))

{ throw new InvalidDayException("Invalid day! Must be between 1 and " + getDaysInMonth(month, year)); }

System.out.println("Current date: " + day + "/" + month + "/" + year); }

private int getDaysInMonth(int month, int year)

{ switch (month) { case 4, 6, 9, 11: return 30; case 2: return (isLeapYear(year)) ? 29 : 28; default: return 31; } } private boolean isLeapYear(int year) { return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0); } }

public class TestCurrentDate

{ public static void main(String[] args)

CurrentDate date = new CurrentDate();

try { date.createDate(); }

catch (InvalidDayException | InvalidMonthException e) { System.out.println(e.getMessage()); } } }

output

Enter day: 28 Enter month: 2 Enter year: 2024 Current date: 28/2/2024

03 Design a Student class with appropriate data members as in Lab 5. Provide your own

exceptions namely Seats Filled Exception, which is thrown when Student registration

number is >XX25 (where XX is last two digits of the year of joining) Show the usage

of this exception handling using Student objects in the main. (Note: Registration

number must be a unique number)

code :

import java.util.HashSet;

import java.util.Scanner;

class SeatsFilledException extends Exception

{ public SeatsFilledException(String message)

{ super(message); } }

class Student

{ private static HashSet<String> registeredNumbers = new HashSet<>();

private String name;

private String regNumber;

private int yearOfJoining;

public Student(String name, String regNumber, int yearOfJoining) throws SeatsFilledException

{ if (isRegistrationInvalid(regNumber, yearOfJoining))

{ throw new SeatsFilledException("Seats filled! Registration number exceeds limit."); }

if (!registeredNumbers.add(regNumber))

{ throw new IllegalArgumentException("Registration number must be unique."); }

this.name = name; this.regNumber = regNumber; this.yearOfJoining = yearOfJoining; }

private boolean isRegistrationInvalid(String regNumber, int yearOfJoining)

{ String limit = String.format("%d25", yearOfJoining % 100);

return Integer.parseInt(regNumber.substring(2)) > Integer.parseInt(limit); }

@Override

public String toString()

{ return "Name: " + name + ", Registration Number: " + regNumber + ", Year: " + yearOfJoining; } }

public class TestStudent

{ public static void main(String[] args)

{ Scanner scanner = new Scanner(System.in); try

{ System.out.print("Enter name: ");

String name = scanner.nextLine();

System.out.print("Enter registration number: ");

String regNumber = scanner.nextLine();

System.out.print("Enter year of joining: ");

int yearOfJoining = scanner.nextInt(); Student student = new Student(name, regNumber, yearOfJoining); System.out.println(student); }

catch (SeatsFilledException e)

{ System.out.println(e.getMessage()); }

catch (IllegalArgumentException e)

{ System.out.println(e.getMessage()); } } }

Output

Enter name: Alice Enter registration number: 2301 Enter year of joining: 2023 Name: Alice, Registration Number: 2301, Year: 2023

03.

Q1 Create a class by extending Thread Class to print a multiplication table of a number

supplied as parameter. Create another class Tables which will instantiate two objects

of the above class to print multiplication table of 5 and 7.

class MultiplicationTable extends Thread

{ private int number; public MultiplicationTable(int number)

{ this.number = number; } @Override public void run()

{ printTable(); }

private void printTable()

{ System.out.println("Multiplication Table of " + number + ":");

for (int i = 1; i <= 10; i++)

{ System.out.println(number + " x " + i + " = " + (number \* i)); } } }

public class Tables

{ public static void main(String[] args)

{ MultiplicationTable tableOf5 = new MultiplicationTable(5);

MultiplicationTable tableOf7 = new MultiplicationTable(7);

tableOf5.start(); tableOf7.start(); } }

output

Multiplication Table of 5: 5 x 1 = 5 5 x 2 = 10 5 x 3 = 15 ... 5 x 10 = 50 Multiplication Table of 7: 7 x 1 = 7 7 x 2 = 14 7 x 3 = 21 ... 7 x 10 = 70

Q2. Write and execute a java program to create and initialize a matrix of integers. Create n

threads( by implementing Runnable interface) where n is equal to the number of rows

in the matrix. Each of these threads should compute a distinct row sum. The main

thread computes the complete sum by looking into the partial sums given by the

threads.

import java.util.Random;

import java.util.Scanner;

class RowSumCalculator implements Runnable {

private int[] row;

private int rowIndex;

private int[] rowSums;

public RowSumCalculator(int[] row, int rowIndex, int[] rowSums) {

this.row = row;

this.rowIndex = rowIndex;

this.rowSums = rowSums;

}

@Override

public void run() {

int sum = 0;

for (int value : row) {

sum += value;

}

rowSums[rowIndex] = sum;

System.out.println("Row " + rowIndex + " sum: " + sum);

}

}

public class MatrixSum {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter number of rows: ");

int rows = scanner.nextInt();

System.out.print("Enter number of columns: ");

int cols = scanner.nextInt();

int[][] matrix = new int[rows][cols];

int[] rowSums = new int[rows];

initializeMatrix(matrix);

displayMatrix(matrix);

Thread[] threads = new Thread[rows];

for (int i = 0; i < rows; i++) {

threads[i] = new Thread(new RowSumCalculator(matrix[i], i, rowSums));

threads[i].start();

}

for (int i = 0; i < rows; i++) {

try {

threads[i].join();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

int totalSum = 0;

for (int sum : rowSums) {

totalSum += sum;

}

System.out.println("Total matrix sum: " + totalSum);

}

private static void initializeMatrix(int[][] matrix) {

Random random = new Random();

for (int i = 0; i < matrix.length; i++) {

for (int j = 0; j < matrix[i].length; j++) {

matrix[i][j] = random.nextInt(10); // Random values between 0 and 9

}

}

}

private static void displayMatrix(int[][] matrix) {

System.out.println("Matrix:");

for (int[] row : matrix) {

for (int value : row) {

System.out.print(value + " ");

}

System.out.println();

}

}

}

Output

Enter number of rows: 3

Enter number of columns: 4

Matrix:

2 5 3 1

7 1 4 0

3 6 2 8

Row 0 sum: 11

Row 1 sum: 12

Row 2 sum: 19

Total matrix sum: 42

Q3 Write and execute a java program to implement a producer and consumer problem

using Inter-thread communication.

class Buffer {

private int data;

private boolean isEmpty = true;

public synchronized void produce(int value) {

while (!isEmpty) { // Wait if buffer is full

try {

wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

data = value;

isEmpty = false;

System.out.println("Produced: " + value);

notify(); // Notify consumer that new data is available

}

public synchronized int consume() {

while (isEmpty) { // Wait if buffer is empty

try {

wait();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

isEmpty = true;

System.out.println("Consumed: " + data);

notify(); // Notify producer that buffer is empty

return data;

}

}

class Producer implements Runnable {

private Buffer buffer;

public Producer(Buffer buffer) {

this.buffer = buffer;

}

@Override

public void run() {

for (int i = 1; i <= 5; i++) {

buffer.produce(i);

try {

Thread.sleep(500); // Simulate production delay

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

class Consumer implements Runnable {

private Buffer buffer;

public Consumer(Buffer buffer) {

this.buffer = buffer;

}

@Override

public void run() {

for (int i = 1; i <= 5; i++) {

buffer.consume();

try {

Thread.sleep(1000); // Simulate consumption delay

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

public class ProducerConsumerTest {

public static void main(String[] args) {

Buffer buffer = new Buffer();

Thread producerThread = new Thread(new Producer(buffer));

Thread consumerThread = new Thread(new Consumer(buffer));

producerThread.start();

consumerThread.start();

}

}

Output

Produced: 1

Consumed: 1

Produced: 2

Consumed: 2

Produced: 3

Consumed: 3

Produced: 4

Consumed: 4

Produced: 5

Consumed: 5

Q1. Write a JavaFX application program to do the following:

a. Display the message “Welcome to JavaFX programming” using Label in

the Scene.

b. Set the text color of the Label to Magenta.

c. Set the title of the Stage to “This is the first JavaFX Application”.

d. Set the width and height of the Scene to 500 and 200 respectively.

e. Use FlowPane layout and set the hgap and vgap of the FlowPane to desired

values.

import javafx.application.Application;

import javafx.scene.Scene;

import javafx.scene.control.Label;

import javafx.scene.layout.FlowPane;

import javafx.stage.Stage;

import javafx.scene.paint.Color;

public class JavaFXDemo extends Application {

@Override

public void start(Stage primaryStage) {

Label messageLabel = new Label("Welcome to JavaFX programming");

messageLabel.setTextFill(Color.MAGENTA); // Set label text color to Magenta

FlowPane flowPane = new FlowPane();

flowPane.setHgap(20); // Set horizontal gap

flowPane.setVgap(10); // Set vertical gap

flowPane.getChildren().add(messageLabel); // Add label to the pane

Scene scene = new Scene(flowPane, 500, 200); // Set Scene dimensions

primaryStage.setTitle("This is the first JavaFX Application"); // Set Stage title

primaryStage.setScene(scene); // Attach Scene to the Stage

primaryStage.show(); // Display the Stage

}

public static void main(String[] args) {

launch(args); // Launch the JavaFX application

}

}

Q2. Write a JavaFX program to accept an integer from the user in a text field and

display the multiplication table (up to number \*10) for that number. Use FlowPane

layout for the application.

Code

import javafx.application.Application;

import javafx.geometry.Pos;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.scene.control.TextField;

import javafx.scene.layout.FlowPane;

import javafx.stage.Stage;

public class MultiplicationTableApp extends Application {

@Override

public void start(Stage primaryStage) {

Label promptLabel = new Label("Enter a number:");

TextField inputField = new TextField();

Button generateButton = new Button("Generate Table");

Label resultLabel = new Label();

inputField.setPrefWidth(100);

generateButton.setOnAction(e -> {

try {

int number = Integer.parseInt(inputField.getText());

StringBuilder table = new StringBuilder();

for (int i = 1; i <= 10; i++) {

table.append(number).append(" x ").append(i)

.append(" = ").append(number \* i).append("\n");

}

resultLabel.setText(table.toString());

} catch (NumberFormatException ex) {

resultLabel.setText("Please enter a valid integer.");

}

});

FlowPane flowPane = new FlowPane(10, 10); // Set horizontal and vertical gaps

flowPane.setAlignment(Pos.CENTER);

flowPane.getChildren().addAll(promptLabel, inputField, generateButton, resultLabel);

Scene scene = new Scene(flowPane, 400, 300);

primaryStage.setTitle("Multiplication Table Generator");

primaryStage.setScene(scene);

primaryStage.show();

}

public static void main(String[] args) {

launch(args);

}

}

Q3. Write a JavaFX program to display a window as shown below. Use TextField for

import javafx.application.Application;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.scene.control.PasswordField;

import javafx.scene.control.TextArea;

import javafx.scene.control.TextField;

import javafx.scene.layout.GridPane;

import javafx.stage.Stage;

public class LoginApp extends Application {

@Override

public void start(Stage primaryStage) {

// Create labels and input fields

Label usernameLabel = new Label("Username:");

TextField usernameField = new TextField();

Label passwordLabel = new Label("Password:");

PasswordField passwordField = new PasswordField();

Button signInButton = new Button("Sign in");

TextArea messageArea = new TextArea();

messageArea.setEditable(false); // Make the text area read-only

// Set up the GridPane layout

GridPane gridPane = new GridPane();

gridPane.setHgap(10);

gridPane.setVgap(10);

gridPane.add(usernameLabel, 0, 0); // Column 0, Row 0

gridPane.add(usernameField, 1, 0); // Column 1, Row 0

gridPane.add(passwordLabel, 0, 1); // Column 0, Row 1

gridPane.add(passwordField, 1, 1); // Column 1, Row 1

gridPane.add(signInButton, 1, 2); // Column 1, Row 2

gridPane.add(messageArea, 0, 3, 2, 1); // Column 0, Row 3 spanning 2 columns

// Handle button click event

signInButton.setOnAction(e -> {

String username = usernameField.getText();

messageArea.setText("Welcome " + username);

});

// Set up the scene and stage

Scene scene = new Scene(gridPane, 300, 200);

primaryStage.setTitle("Login Application");

primaryStage.setScene(scene);

primaryStage.show();

}

public static void main(String[] args) {

launch(args);

}

}

Q4. Define a class called Employee with the attributes name, empID, designation,

basicPay, DA, HRA, PF, LIC, netSalary. DA is 40% of basicPay, HRA is 15% of

basicPay, PF is 12% of basicPay. Display all the employee information in a

JavaFX application.

import javafx.application.Application;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.scene.control.TextField;

import javafx.scene.layout.GridPane;

import javafx.stage.Stage;

class Employee {

private String name;

private String empID;

private String designation;

private double basicPay;

private double DA;

private double HRA;

private double PF;

private double LIC;

private double netSalary;

public Employee(String name, String empID, String designation, double basicPay) {

this.name = name;

this.empID = empID;

this.designation = designation;

this.basicPay = basicPay;

calculateSalaries();

}

private void calculateSalaries() {

this.DA = 0.4 \* basicPay;

this.