LAB 8: Exception handling

Exercise 1:

Create a class called Calculator with a method called divide(int num1, int num2) that performs division of two numbers. Implement exception handling to handle the scenario when the second number is zero and throw a custom exception called DivideByZeroException. Test the Calculator class by calling the divide method with different inputs and handle the exceptions gracefully.

Exercise 2:

Create a class called BankAccount with attributes like accountNumber and balance. Implement a method called withdraw(double amount) that allows users to withdraw money from the bank account. Implement exception handling to handle scenarios when the withdrawal amount exceeds the available balance and throw a custom exception called InsufficientFundsException. Test the BankAccount class by creating an account, depositing money, and making withdrawals while handling the exceptions.

Exercise 3:

Create a class called FileParser with a method called parseFile(String filename) that reads and processes a file. Implement exception handling to handle scenarios when the file is not found or cannot be read and throw a custom exception called FileParsingException. Test the FileParser class by calling the parseFile method with different file names and handle the exceptions gracefully.

Exercise 4:

Create a class called DataValidator with a method called validateData(String data) that validates a data string based on certain rules. Implement exception handling to handle scenarios when the data string does not meet the validation criteria and throw a custom exception called InvalidDataException. Test the DataValidator class by calling the validateData method with different data strings and handle the exceptions gracefully.

Exercise 5:

Create a class called DatabaseConnection with a method called connect() that establishes a connection with a database. Implement exception handling to handle scenarios when the database connection fails and throw a custom exception called DatabaseConnectionException. Test the DatabaseConnection class by calling the connect method and handle the exceptions gracefully.

Exercise 6:

Create a class called DataProcessor with a method called processData(List<String> data) that processes a list of data elements. Implement exception handling to handle scenarios when the data list is empty or null and throw a custom exception called DataProcessingException. Test the DataProcessor class by calling the processData method with different data lists and handle the exceptions gracefully.

Exercise 7:

Create a class called InputReader with a method called readInput() that reads user input from the console. Implement exception handling to handle scenarios when the user input cannot be read or does not meet certain criteria and throw a custom exception called InputValidationException. Test the InputReader class by calling the readInput method and handle the exceptions gracefully.

Exercise 8:

Create a class called EmailSender with a method called sendEmail(String recipient, String message) that sends an email to a recipient. Implement exception handling to handle scenarios when the email cannot be sent and throw a custom exception called EmailSendingException. Test the EmailSender class by calling the sendEmail method with different inputs and handle the exceptions gracefully.

Exercise 9:

Create a class called DataReader with a method called readDataFromFile(String filename) that reads data from a file and processes it. Implement exception handling to handle scenarios when the file cannot be read or the data cannot be processed and throw a custom exception called DataProcessingException. Test the DataReader class by calling the readDataFromFile method with different file names and handle the exceptions gracefully.

Exercise 10:

Create a class called NetworkConnector with a method called connectToServer(String ipAddress) that establishes a connection with a server. Implement exception handling to handle scenarios when the server connection fails and throw a custom exception called ConnectionException. Test the NetworkConnector class by calling the connectToServer method and handle the exceptions gracefully.

Exercise 11:

Create a class called TemperatureConverter with a method called convertToFahrenheit(double celsius) that converts a temperature from Celsius to Fahrenheit. Implement exception handling to handle scenarios when the input temperature is below absolute zero and throw a custom exception called InvalidTemperatureException. Test the TemperatureConverter class by calling the convertToFahrenheit method with different temperature values and handle the exceptions gracefully.

Exercise 12:

Create a class called DataAnalyzer with a method called analyzeData(List<Integer> data) that performs analysis on a list of integer values. Implement exception handling to handle scenarios when the list is empty or contains null elements and throw a custom exception called DataAnalysisException. Test the DataAnalyzer class by calling the analyzeData method with different data lists and handle the exceptions gracefully.

Exercise 13:

Create a class called UserValidator with a method called validateUser(String username, String password) that validates a user's credentials. Implement exception handling to handle scenarios when the username or password does not meet certain criteria and throw a custom exception called InvalidCredentialsException. Test the UserValidator class by calling the validateUser method with different username and password inputs and handle the exceptions gracefully.

Exercise 14:

Create a class called DataReader with a method called readData(String filename) that reads data from a file. Implement exception handling to handle scenarios when the file is empty or contains invalid data and throw a custom exception called InvalidDataException. Test the DataReader class by calling the readData method with different file names and handle the exceptions gracefully.

Exercise 15:

Create a class called ArrayCalculator with a method called calculateSum(int[] numbers) that calculates the sum of an array of numbers. Implement exception handling to handle scenarios when the array is null or empty and throw a custom exception called InvalidArrayException. Test the ArrayCalculator class by calling the calculateSum method with different arrays and handle the exceptions gracefully.

Exercise 16:

Create a class called PaymentProcessor with a method called processPayment(double amount) that processes a payment. Implement exception handling to handle scenarios when the payment amount is negative or exceeds a certain limit and throw a custom exception called InvalidPaymentException. Test the PaymentProcessor class by calling the processPayment method with different amounts and handle the exceptions gracefully.

Exercise 17:

Create a class called DataTransformer with a method called transformData(String data) that transforms a data string. Implement exception handling to handle scenarios when the data is null or cannot be transformed and throw a custom exception called DataTransformationException. Test the DataTransformer class by calling the transformData method with different data strings and handle the exceptions gracefully.

Exercise 18:

Create a class called FileValidator with a method called validateFile(String filename) that validates a file. Implement exception handling to handle scenarios when the file is not found or cannot be accessed and throw a custom exception called FileValidationException. Test the FileValidator class by calling the validateFile method with different file names and handle the exceptions gracefully.

Exercise 19:

Create a class called EmailValidator with a method called validateEmail(String email) that validates an email address. Implement exception handling to handle scenarios when the email format is invalid and throw a custom exception called InvalidEmailException. Test the EmailValidator class by calling the validateEmail method with different email addresses and handle the exceptions gracefully.

Exercise 20:

Create a class called InputParser with a method called parseInput(String input) that parses user input. Implement exception handling to handle scenarios when the input is null or cannot be parsed and throw a custom exception called InputParsingException. Test the InputParser class by calling the parseInput method with different inputs and handle the exceptions gracefully.

Exercise 21:

Create a class called DatabaseConnection with a method called connect() that establishes a connection with a database. Implement exception handling to handle scenarios when the database connection fails due to invalid credentials and throw a custom exception called DatabaseConnectionException. Test the DatabaseConnection class by calling the connect method and handle the exceptions gracefully.

Exercise 22:

Create a class called FileReader with a method called readFile(String filename) that reads data from a file. Implement exception handling to handle scenarios when the file is not found or cannot be read and throw a custom exception called FileReadingException. Test the FileReader class by calling the readFile method with different file names and handle the exceptions gracefully.

Exercise 23:

Create a class called ImageProcessor with a method called processImage(String imagePath) that processes an image file. Implement exception handling to handle scenarios when the image file is corrupted or of an unsupported format and throw a custom exception called ImageProcessingException. Test the ImageProcessor class by calling the processImage method with different image file paths and handle the exceptions gracefully.

Exercise 24:

Create a class called NetworkClient with a method called sendData(String data) that sends data over a network. Implement exception handling to handle scenarios when the network connection fails or the data transmission is interrupted and throw a custom exception called NetworkException. Test the NetworkClient class by calling the sendData method with different data inputs and handle the exceptions gracefully.

Exercise 25:

Create a class called ConfigurationReader with a method called readConfiguration(String configPath) that reads configuration settings from a file. Implement exception handling to handle scenarios when the configuration file is missing or contains invalid data and throw a custom exception called ConfigurationException. Test the ConfigurationReader class by calling the readConfiguration method with different configuration file paths and handle the exceptions gracefully.

Exercise 26:

Create a class called DataValidator with a method called validateData(String data) that validates a data string. Implement exception handling to handle scenarios when the data is not in the expected format or does not meet certain criteria and throw a custom exception called DataValidationException. Test the DataValidator class by calling the validateData method with different data inputs and handle the exceptions gracefully.

Exercise 27:

Create a class called TransactionManager with a method called performTransaction(double amount) that performs a financial transaction. Implement exception handling to handle scenarios when the transaction amount is negative or exceeds a certain limit and throw a custom exception called

TransactionException. Test the TransactionManager class by calling the performTransaction method with different amounts and handle the exceptions gracefully.

Exercise 28:

Create a class called UserRegistration with a method called registerUser(String username, String password) that registers a new user. Implement exception handling to handle scenarios when the username or password is not valid and throw a custom exception called UserRegistrationException. Test the UserRegistration class by calling the registerUser method with different username and password inputs and handle the exceptions gracefully.

Exercise 29:

Create a class called DataExporter with a method called exportData(List<String> data, String filePath) that exports data to a file. Implement exception handling to handle scenarios when the file path is invalid or the data cannot be written to the file and throw a custom exception called DataExportException. Test the DataExporter class by calling the exportData method with different data lists and file paths and handle the exceptions gracefully.

Exercise 30:

Create a class called LogManager with a method called logEvent(String event) that logs events to a log file. Implement exception handling to handle scenarios when the log file is not accessible or the event cannot be logged and throw a custom exception called LogEventException. Test the LogManager class by calling the logEvent method with different event inputs and handle the exceptions gracefully.