Java Lab Assignments for MCA-II(Sem) students of Group-2

Week-6

1. **Java Assignment Question: File Handling and Exception Handling in Java**

Objective:

This question focuses on file handling, exception handling, and proper design practices in Java. It assesses the candidate's ability to use concepts such as try-catch blocks, `FileInputStream`, `BufferedReader`, and `FileOutputStream` effectively to implement a complete solution.

Instructions:

You are tasked with implementing a Java program that reads numeric values from a text file, calculates the average of these values, and outputs the result to another text file. The program should handle file-related issues and invalid data using exception handling.

Requirements:

1. File Input:

- Design a Java class named `DataProcessor` with a method `readValuesFromFile(String filePath)` that takes a file path as a parameter and reads numeric values from the specified text file.

- Implement exception handling to catch and handle `FileNotFoundException` and `IOException` appropriately.

2. Data Validation:

- Within the `DataProcessor` class, implement a method `validateData(List<Double> values)` to validate the read numeric values. Ensure that only valid numeric data is considered for calculating the average.

- Handle invalid data by throwing a custom exception (`InvalidDataException`) for non-numeric values.

3. Average Calculation:

- Create a method `calculateAverage(List<Double> validValues)` within the `DataProcessor` class to calculate the average of the valid numeric values.

4. File Output:

- Implement a method `writeResultToFile(double average, String outputPath)` within the `DataProcessor` class to write the calculated average to another text file specified by the `outputPath`.

- Handle `IOException` during the file writing process.

5. Main Program:

- In the main program, instantiate the `DataProcessor` class, call the methods in sequence to read values, validate data, calculate the average, and write the result to another file.

6. Exception Handling:

- Define a custom exception class `InvalidDataException` that extends `Exception`. Use this exception for handling invalid data.

2. **Java Assignment Question: Counting Exceptions in a Java Program**

Objective:
The objective of this assignment is to assess your understanding of exception handling in Java and your ability to develop a program that counts and reports the total number of exceptions in a given Java program.

Instructions:

1. Create a Java program named “ExceptionCounter.java”.

2. Implement a method named “countExceptions” that takes the path to a Java program file as input and returns the total number of exceptions present in that program.

3. The program should read the specified Java file, analyze its content, and count occurrences of exception-related keywords or patterns.

4. Use appropriate exception handling mechanisms to handle scenarios such as file not found or syntax errors in the Java program.

5. Print the total number of exceptions found on the screen.

3. **Java Assignment Question: Handling Null Pointer Exceptions**

Objective:
The objective of this assignment is to assess your understanding of handling Null Pointer Exceptions in Java through various approaches, including the use of Try-Catch blocks, Default Exception Handlers, and preventive measures.

a. Write down the same program using Try-Catch block?
b. Write down the same program using Default Exceptional Handler?
c. Write the program by Preventing unchecked exceptions?

Instructions:
1. Create a Java program named “NullPointerExceptionHandler.java”.
2. Implement three different versions of the program using the provided blocks of code (a, b, and c) to identify, handle, or prevent Null Pointer Exceptions.
3. Each version should include appropriate comments explaining the chosen approach and the logic behind it.

4. Demonstrate the functionality of each version by specifying scenarios in which Null Pointer Exceptions may occur.
5. Print the details of the Null Pointer Exception on the screen in each version.

Tasks:
1. Implement the three versions of the program using the provided code blocks (a, b, c).
2. Ensure each version handles or prevents Null Pointer Exceptions appropriately.
3. Demonstrate the functionality by specifying scenarios that trigger Null Pointer Exceptions and printing the details on the screen.
4. Provide comments explaining the logic and approach used in each version.

<u>Week-7</u>

4. **Java Assignment Question: Preventing Unchecked Exceptions**

Objective:
The objective of this assignment is to assess your ability to write robust and defensive code to prevent unchecked exceptions in Java. Focus on handling potential issues before they lead to exceptions.

Instructions:
1. Implement a Java program that includes a method susceptible to unchecked exceptions. For example, you can create a method that performs a mathematical operation that might throw an "ArithmeticException" (e.g., division by zero).

2. Rewrite the method to make it more robust and prevent the occurrence of unchecked exceptions. Use defensive coding techniques such as checking for null references, division by zero, or out-of-bounds array indices before performing operations.

3. Provide a set of test cases to demonstrate the effectiveness of your modifications. Include scenarios that would have caused unchecked exceptions in the original implementation.

5. **Java Assignment Question: Logging and Reporting**

Objective:

The objective of this assignment is to assess your understanding of logging and reporting in Java, specifically dealing with unchecked exceptions using logging frameworks.

Instructions:
1. Create a Java program that simulates a scenario where an unchecked exception might occur. This could involve operations such as reading from a file, parsing data, or any other operation that could potentially throw an unchecked exception.

2. Implement exception handling using a try-catch block for the specific type of exception that might be thrown.

3. Integrate a logging framework, either Log4j or SLF4J, into your program to log the details of the exception.

4. Ensure that your program handles the exception gracefully by logging the exception details and providing a meaningful message.

6. **Java Assignment Question: Graceful Termination**

Objective:
The objective of this assignment is to assess your understanding of handling severe runtime exceptions in Java and implementing graceful termination to prevent further damage or data corruption.

Instructions:
1. Create a Java program that simulates a scenario where a severe runtime exception might occur. This could involve critical operations or conditions that, if not handled properly, could lead to potential issues.

2. Implement exception handling using a try-catch block for the specific type of exception that might be considered severe.

3. Integrate a logging framework (e.g., Log4j or SLF4J) into your program to log the details of the severe runtime exception.

4. Use “System.exit()” to gracefully terminate the application with an error code after logging the exception.

Week-8

7. **Java Assignment Question: Opening a Database Connection (Checked Exception)**

Objective:
The objective of this assignment is to assess your mastery of handling checked exceptions, specifically when dealing with database connections in Java. Focus on utilizing advanced techniques for exception handling and resource management.

Instructions:
1. Extend the provided Java program to include advanced techniques for handling checked exceptions related to opening a database connection.

2. Implement the following enhancements:
  - Use the “try-with-resources” statement to manage the “Connection” object. Ensure proper closure of the connection, even if an exception occurs.

- Add a “finally” block to log a message indicating the successful closure of the database connection, regardless of whether an exception occurred or not.
- Introduce a custom exception class for representing database connection-related issues. Throw this custom exception instead of the generic “SQLException” when encountering problems.

3. Demonstrate the enhanced program by intentionally providing incorrect database connection details. Include a scenario where the custom exception is thrown and caught, and ensure that the connection is properly closed.

8. **Java Assignment Question: Parsing a Date (Unchecked Exception)**
Objective:
The objective of this assignment is to assess your advanced understanding of handling unchecked exceptions, specifically when parsing dates in Java. Focus on employing advanced techniques for exception handling and providing meaningful feedback.

Instructions:
1. Enhance the provided Java program to include advanced techniques for handling unchecked exceptions related to parsing dates.

2. Implement the following enhancements:
- Use the “try-with-resources” statement to manage the “DateFormat” object. Ensure proper closure of the formatter, even if an exception occurs.
- Introduce a custom exception class for representing date parsing-related issues. Throw this custom exception instead of the generic “ParseException” when encountering problems.
- Add a “finally” block to log a message indicating the successful closure of the “DateFormat” object, regardless of whether an exception occurred or not.

3. Demonstrate the enhanced program by intentionally providing an incorrectly formatted date string. Include a scenario where the custom exception is thrown and caught, and ensure that the “DateFormat” object is properly closed.

9. **Advanced Java Assignment Question: Network Connection (Checked Exception)**
Objective:
The objective of this assignment is to evaluate your advanced understanding of handling checked exceptions, specifically when dealing with network connections in Java. Focus on utilizing advanced techniques for exception handling, providing meaningful feedback, and managing resources effectively.

Instructions:
1. Extend the provided Java program to include advanced techniques for handling checked exceptions related to establishing a network connection.

2. Implement the following enhancements:

- Use the “try-with-resources” statement to manage the “HttpURLConnection” object. Ensure proper closure of the connection, even if an exception occurs.
- Introduce a custom exception class for representing network connection-related issues. Throw this custom exception instead of the generic “IOException” when encountering problems.
- Add a “finally” block to log a message indicating the successful closure of the “HttpURLConnection” object, regardless of whether an exception occurred or not.

3. Demonstrate the enhanced program by intentionally providing an incorrect URL. Include a scenario where the custom exception is thrown and caught, and ensure that the “HttpURLConnection” object is properly closed.

Week-9

10. Advanced Java Assignment Question: File Input (Checked Exception)

Objective:
The objective of this assignment is to assess your advanced understanding of handling checked exceptions, specifically when dealing with file input in Java. Focus on utilizing advanced techniques for exception handling, providing meaningful feedback, and managing resources effectively.

Instructions:
1. Extend the provided Java program to include advanced techniques for handling checked exceptions related to file input.

2. Implement the following enhancements:
- Use the “try-with-resources” statement to manage the “FileReader” object. Ensure proper closure of the file reader, even if an exception occurs.
- Introduce a custom exception class for representing file input-related issues. Throw this custom exception instead of the generic “FileNotFoundException” when encountering problems.
- Add a “finally” block to log a message indicating the successful closure of the “FileReader” object, regardless of whether an exception occurred or not.

3. Demonstrate the enhanced program by intentionally providing the name of a nonexistent file. Include a scenario where the custom exception is thrown and caught, and ensure that the “FileReader” object is properly closed.

11. Advanced Java Assignment Question: Handling OutOfMemoryError
Objective:
The objective of this assignment is to assess your advanced understanding of handling severe runtime errors, specifically dealing with “OutOfMemoryError” in Java. Focus on implementing strategies to gracefully handle the error and prevent unintended consequences.
Instructions:

1. Extend the provided Java program to implement advanced techniques for detecting and handling the “OutOfMemoryError”.

2. Implement the following enhancements:
- Use a monitoring mechanism to detect the potential occurrence of “OutOfMemoryError” before it causes the program to crash. This can involve checking available memory, setting memory limits, or using specific JVM options.
- Gracefully handle the situation by terminating the infinite loop and providing a meaningful message when the system is about to run out of memory.

3. Demonstrate the enhanced program by simulating an environment where memory is limited. Include a scenario where your implemented strategy detects the potential “OutOfMemoryError” and takes appropriate action.

12. Java Assignment Question: String to Integer Conversion (NumberFormatException)

Objective:

The objective of this assignment is to assess your understanding of handling “NumberFormatException” in Java when converting a “String” to an “int”. Focus on utilizing proper exception handling techniques and providing meaningful feedback.

Instructions:

1. Extend the provided Java program to include additional scenarios for string-to-integer conversion, not limited to a single input.

2. Implement the following tasks:
- Create an array or a list of strings containing both valid and invalid representations of integers.
- Use a loop to attempt the conversion of each string to an integer inside a “try-catch” block.
- Handle the “NumberFormatException” for each conversion attempt and print an appropriate error message.
- Keep track of the successfully converted integers and display them along with a total count at the end.

3. Demonstrate the enhanced program by providing a variety of strings for conversion, including valid and invalid representations.

<u>Week-10</u>

13. Java Assignment Question: Type Casting Exception (ClassCastException)

Objective:

The objective of this assignment is to assess your understanding of handling “ClassCastException” in Java when performing type casting. Focus on utilizing proper exception handling techniques and providing meaningful feedback.

Instructions:
1. Extend the provided Java program to include additional scenarios for type casting, involving different data types.

2. Implement the following tasks:
- Use an array or a collection to store objects of different data types (e.g., String, Integer, Double, etc.).
- Attempt to cast each object to a specific type inside a loop and catch the “ClassCastException” if the casting is not possible.
- Print appropriate error messages for each casting attempt, specifying the source and target data types.
- Keep track of the successfully casted objects and display them along with a total count at the end.

3. Demonstrate the enhanced program by providing a variety of objects for type casting, including both valid and invalid scenarios.

14. Java Genetic Algorithm Assignment Question: Finding Prime Numbers

Objective:
The objective of this assignment is to assess your understanding of Genetic Algorithms (GA) and your ability to implement a Java program to find a prime number using a genetic algorithm.

Instructions:
1. Implement a Java program named “GeneticPrimeFinder.java” that uses a Genetic Algorithm to find a prime number.
2. The program should have the following components:
- **Chromosome Representation:** Encode potential prime numbers as chromosomes. You can use binary representation for simplicity.
- **Initialization:** Generate an initial population of chromosomes representing potential prime numbers.
- **Fitness Function:** Define a fitness function that evaluates how close a chromosome is to being a prime number. Consider using factors, divisibility, or other relevant criteria.
- **Selection:** Implement a selection mechanism (e.g., roulette wheel, tournament) to choose chromosomes for reproduction based on their fitness.
- **Crossover:** Apply crossover (e.g., one-point or two-point crossover) to combine genetic material from selected parents.
- **Mutation:** Introduce mutation in the population to maintain diversity.
- **Termination:** Decide on a termination condition (e.g., a certain number of generations or reaching a fitness threshold).
- **Result Output:** Print the prime number found and other relevant information.

Tasks:

1. Implement the components of the Genetic Algorithm within the "geneticAlgorithm" method.
2. Apply suitable chromosome representations, genetic operators, and termination conditions.
3. Demonstrate the functionality by setting a target prime number (e.g., 97) and running the "GeneticPrimeFinder" program.
4. Ensure that the program outputs the found prime number and other relevant information.

15. **Java Assignment Question: Particle Swarm Optimization (PSO) Algorithm Implementation**

Objective:
The objective of this assignment is to assess your understanding of the Particle Swarm Optimization (PSO) algorithm and your ability to implement it in Java for solving a simple example problem.

Instructions:
1. Create a Java program named "PSOImplementation.java" that implements the Particle Swarm Optimization algorithm.
2. Choose a simple example problem for optimization (e.g., finding the minimum of a mathematical function).
3. Implement the PSO algorithm components, including particle initialization, fitness evaluation, velocity and position updates, and termination conditions.
4. Allow customization of key parameters such as the number of particles, iterations, and any algorithm-specific parameters.
5. Print the final optimized solution and the corresponding fitness value.

Tasks:
1. Choose a simple example problem (e.g., a mathematical function) for optimization.
2. Implement the PSO algorithm components within the "psoAlgorithm" method.
3. Define the fitness function specific to the chosen example problem within the "fitnessFunction" method.
4. Allow customization of key parameters in the main method.
5. Demonstrate the functionality by running the "PSOImplementation" program and printing the final optimized solution and its fitness value.

<u>Week-11</u>

16. **Java Assignment Question: Hybrid PSO-GA for Triangle Classification Test Case Generation**

Objective:

The objective of this assignment is to assess your understanding of hybrid algorithms and your ability to implement a hybrid Particle Swarm Optimization (PSO) - Genetic Algorithm (GA) for test case generation in the context of a triangle classification problem.

Instructions:

1. Create a Java program named “HybridPSOGATriangleTestCases.java” that implements the hybrid PSO-GA algorithm for test case generation.

2. Define a simple triangle classification problem with specific rules (e.g., based on side lengths, angles) for which test cases need to be generated.

3. Implement the PSO and GA components separately, and then integrate them into a hybrid algorithm.

4. Design the chromosome representation, fitness evaluation, and genetic operators for the GA.

5. Allow customization of key parameters such as the number of particles, iterations, population size, crossover rate, mutation rate, etc.

6. Print the final set of generated test cases for triangle classification.

Tasks:

1. Define the rules for triangle classification based on specific conditions (e.g., side lengths, angles).

2. Implement the PSO and GA components within the “hybridPSOGAAlgorithm” method.

3. Design the chromosome representation, fitness function, and genetic operators for the GA.

4. Allow customization of key parameters in the main method.

5. Demonstrate the functionality by running the “HybridPSOGATriangleTestCases” program and printing the final set of generated test cases.

17. Java Assignment Question: Generics and Collections in Java

Problem Statement:

Your task is to create a Java program that showcases the use of generics and various collection classes. The program should involve the manipulation of data using generic classes and demonstrate the functionality of different collection classes provided by the Java Collections Framework.

Requirements:

1. Generic Class:

- Design a generic class named `DataProcessor<T>` that can process and manipulate a collection of elements of type `T`.

- Implement methods to add elements, remove elements, and display the elements in the collection.

2. Collection Manipulation:

- Instantiate multiple instances of `DataProcessor` with different types (e.g., `Integer`, `String`, custom class).

- Demonstrate the addition, removal, and display of elements for each `DataProcessor` instance.

3. Use of ArrayList:

- Create a method within the `DataProcessor` class to utilize `ArrayList` for storing and manipulating elements.

- Implement operations such as adding, removing, and searching for elements in the `ArrayList`.

4. Use of HashMap:

- Extend the `DataProcessor` class to work with `HashMap`. The key-value pairs should represent unique identifiers and corresponding elements.

- Implement methods to add, remove, and retrieve elements using keys.

5. Generics with Wildcards:

- Modify the `DataProcessor` class to include a method that accepts a collection of elements (using wildcard generics) and combines it with the existing collection.

6. Main Program:

- In the main program, create instances of `DataProcessor` with different types, perform operations on them, and demonstrate the functionality of generic methods and various collection classes.

18. Java Assignment Question: Multithreading and Generics in Java

Problem Statement:

You are assigned the task of developing a Java program that leverages both multithreading and generics. The program should involve the creation of a multithreaded application with generic classes to perform parallel processing on different data types.

Requirements:

1. Generic Threaded Processor:

- Design a generic class named `ThreadedProcessor<T>` that processes a collection of elements of type `T` in a multithreaded environment.

- Implement methods for processing the elements concurrently using threads. This could involve tasks like computation, transformation, or any other operation suitable for parallel processing.

2. Multithreading:

- Utilize the `Thread` class or the `ExecutorService` framework to implement multithreading within the `ThreadedProcessor` class.

- Demonstrate the concurrent processing of elements using multiple threads.

3. Data Types:

- Instantiate multiple instances of `ThreadedProcessor` with different types (e.g., `Integer`, `String`, custom class).

- Implement specific operations in the `ThreadedProcessor` class that are relevant to the type being processed (e.g., finding the sum of integers, concatenating strings).

4. Thread Synchronization:

- Implement proper synchronization mechanisms to ensure thread safety during the concurrent processing of elements.

- Use synchronization primitives or techniques to avoid race conditions and ensure consistency.

5. Main Program:

- In the main program, create instances of `ThreadedProcessor` with different types, define tasks for each processor, and demonstrate the parallel processing of elements.

6. Generics with Bounds:

- Modify the `ThreadedProcessor` class to incorporate generics with bounds (e.g., `<T extends Number>`).

- Implement operations that are specific to the bounds of the generic type.

Week-12

19. Java Assignment Question: Designing compiler

Object:

Design your own java pre compiler for the following:-

1. Write a comment using **"$"** symbol instead of using **slash("/ /").**
2. Write an end of statement symbol using any special character instead semi colon (**" ; ").**
3. Write a block using **square brackets ("[ ]")** instead of using **curly brackets ("{ }").**

20. Java Assignment Question: Designing pre-compiler

Object:

Design your own java pre compiler for the following:-

1. Count the Number of integer, float variables and String variables.

2. Count the number of for loop(s), while loop(s), do-while loop(s).
3. Count the number of conditional statements.
4. Count the number of user defined methods.

21. Java Assignment Question: Online Library Catalog System in Java

Problem Statement:

You are tasked with designing and implementing a Java program that simulates an online library catalog system. The system should have classes representing books, users, and the catalog itself. Users should be able to borrow and return books, and the catalog should maintain information about available books and their status. Ensure that proper encapsulation, inheritance, and polymorphism principles are applied.

Requirements:

1. Book Class:

- Design a class named `Book` to represent individual books in the library.

- Include attributes such as book ID, title, author, genre, and availability status.

- Implement appropriate methods, including getters and setters.

2. User Class:

- Design a class named `User` to represent library users.

- Include attributes such as user ID, name, and a list to store borrowed books.

- Implement methods to borrow and return books.

3. Catalog Class:

- Design a class named `Catalog` to represent the library catalog.

- Use collections (e.g., ArrayList) to store a collection of books.

- Implement methods to add books to the catalog, display available books, and update book availability.

4. Encapsulation:

- Ensure that proper encapsulation is applied to protect the internal state of the classes. Use access modifiers appropriately.

5. Inheritance and Polymorphism:

- Utilize inheritance to create a more specific type of book (e.g., FictionBook, NonFictionBook) that extends the base `Book` class.

- Demonstrate polymorphism through a method like `displayDetails()` that can provide different information based on the book type.

6. Main Program:

- In the main program, instantiate the classes, add books to the catalog, create users, and simulate the borrowing and returning of books.

- Display the catalog's status after each transaction.

Note:

- Pay attention to proper error handling and validation, such as checking if a book is available before borrowing.

- Consider using meaningful names for classes, variables, and methods.

22. **Java Assignment Question: Concurrent Collection Handling in Java**

Consider a scenario where you are designing a system that needs to handle concurrent access to a shared collection. The collection stores information about bank accounts, and multiple threads may simultaneously deposit and withdraw funds from different accounts. Additionally, a mechanism is required to ensure that the collection is accessed safely in a multithreaded environment.

Class Definitions:

Define a class BankAccount that represents a bank account with attributes like accountNumber (int) and balance (double).

Implement a class AccountManager that manages a collection of BankAccount objects. The manager should support methods to deposit and withdraw funds from an account.

Concurrency Handling:

Modify the AccountManager class to handle concurrent access using appropriate synchronization mechanisms. Ensure that deposit and withdrawal operations are thread-safe.

Multithreaded Simulation:

In the main method of your program, create a simulation with multiple threads concurrently depositing and withdrawing funds from different bank accounts.

Use the ExecutorService and related concurrency classes to manage the threads.

Output Verification:

Print the balance of each bank account after the multithreaded simulation. Ensure that the final balances are correct, considering the concurrent deposit and withdrawal operations.