Core Java: Declarations and Access Control, Object Orientation, Operators, Flow Control, Exceptions, Gradle Fundamentals, TDD with Junit 5, Strings, /O, Formating, and Parsing, Generics and Collections, Threads.

1. Declarations and Access Control

Q: You are designing a library system. Create a class Book with appropriate access control such that only the library system can modify the book's availability status.

2. Object Orientation

Q: Design a Shape class hierarchy with classes Circle, Rectangle, and Triangle. Implement a method calculateArea() in each subclass.

3. Operators

Q: Write a method to determine if a given year is a leap year. Use the appropriate operators to perform the checks.

4. Flow Control

Q: Implement a simple text-based menu system that allows users to select options from a list until they choose to exit.

5. Exceptions

Q: Create a custom exception InvalidAgeException that is thrown if a user tries to register with an age less than 18. Handle this exception in your registration method.

6. Gradle Fundamentals

Q: Create a basic Gradle project structure for a Java application with dependencies on JUnit for testing.

7. TDD with JUnit 5

Q: Write test cases using JUnit 5 for a method int add(int a, int b) that returns the sum of two integers. Implement the method using TDD principles.

8. Strings

Q: Write a method that takes a comma-separated string of numbers and returns the sum of those numbers.

9. I/O (Input/Output)

Q: Write a program that reads a text file containing names and prints out the names in alphabetical order.

10. Formatting and Parsing

Q: Write a method that takes a date string in the format "dd/MM/yyyy" and returns a formatted date string in the format "MMMM dd, yyyy".

11. Generics and Collections

Q: Implement a generic class Box<T> that can hold an item of any type. Provide methods to set and get the item.

12. Threads

Q: Create a program that starts two threads: one thread prints even numbers and the other prints odd numbers up to 100. Ensure the threads coordinate to print numbers in order.

Core Java 8: Concurrent Paterns in Java, Concurrent Collections, Lambda Expressions, Stream API, Introduction to Design Pattern, GitHub, Introduction to JDBC: Connection, Statement, PreparedS tatement, ResultSet.

1. **Concurrent Patterns in Java:**
    - **Question:** Imagine you are developing a web server that handles multiple client requests concurrently. Describe how you would implement this using Java's concurrency utilities. What pattern would you use, and why? Provide a code snippet to illustrate your approach.
2. **Concurrent Collections:**
    - **Question:** You have a multi-threaded application where threads frequently read from and update a shared map. Which concurrent collection would you choose to ensure thread safety and high performance? Justify your choice with an example of how you would implement this.
3. **Lambda Expressions:**
    - **Question:** Given a list of **Employee** objects, each having properties **id**, **name**, and **salary**, write a lambda expression to filter out employees with a salary greater than $50,000 and sort the remaining employees by name. Discuss how lambda expressions improve the readability and maintainability of this code.
4. **Stream API:**
    - **Question:** Using the Stream API, write a code snippet to process a list of transactions where each transaction has an **id**, **type**, and **amount**. Filter out transactions of type "debit" with an amount greater than $1000, map them to their IDs, and collect the result into a list. Explain the benefits of using the Stream API for this task.
5. **Introduction to Design Pattern:**
    - **Question:** You need to ensure that only one instance of a **DatabaseConnection** class is created throughout your application. Explain which design pattern you would use and provide a thread-safe implementation in Java. Additionally, discuss potential issues you might encounter with this pattern and how you would address them.
6. **GitHub:**
    - **Question:** Describe a scenario where you are working on a project with multiple team members, and you need to implement a new feature without affecting the main codebase. Explain how you would use GitHub's branching and pull request features to manage this workflow. Include the specific Git commands you would use.
7. **Introduction to JDBC:**
    - **Question:** Consider a scenario where you need to connect to a PostgreSQL database, execute a query to retrieve a list of users, and handle potential SQL exceptions gracefully. Write a code snippet to demonstrate this and discuss the importance of proper exception handling in JDBC.
8. **Statement vs. PreparedStatement:**
    - **Question:** You have identified a SQL injection vulnerability in your application caused by using a **Statement** object. Refactor the following code to use **PreparedStatement** to prevent SQL injection and explain how this change improves security.

```
String query = "SELECT * FROM users WHERE username = '" + username + "' AND password = '" + password + "'";

 Statement stmt = connection.createStatement();
```

ResultSet rs = stmt.executeQuery(query);

9. **ResultSet:**

    - **Question:** Given a **ResultSet** object containing user data, write a method that iterates through the **ResultSet** and prints each user's details. Additionally, explain how you would handle any SQL exceptions that might occur during this process to ensure robust error handling.

10. **Combining Concepts:**

    - **Question:** Design a small Java application that uses JDBC to connect to a database and retrieve product data, processes the data using the Stream API to filter and sort products by price, and then prints the results. Explain how you would structure your code to ensure it is modular, maintainable, and easy to test. Provide relevant code snippets.