PRACTICE QUESTIONS FROM DAY 10 TO DAY 14

DAY 10 EXCEPTION HANDLING PRACTICE QUESTIONS

1. Write a Java program that throws an exception and catch it using a try-catch block.
2. Write a Java program to create a method that takes an integer as a parameter and throws an exception if the number is odd.
3. Write a Java program to create a method that reads a file and throws an exception if the file is not found.
4. Write a Java program that reads a list of numbers from a file and throws an exception if any of the numbers are positive.
5. Write a Java program that reads a file and throws an exception if the file is empty.
6. Write a Java program that reads a list of integers from the user and throws an exception if any numbers are duplicates.
7. Write a Java program to create a method that takes a string as input and throws an exception if the string does not contain vowels.

DAY 12 WRAPPER CLASS

1. Implement a method that accepts a List of String and returns a List of Integer by parsing the strings. Handle potential NumberFormatException.
2. Write a program that takes user input as a String and validates whether it can be converted to a double.
3. Write a program to compare two Integer objects using == and .equals(). Explain the difference in behavior.
4. Demonstrate the use of Integer.parseInt(), Double.parseDouble(), and Boolean.parseBoolean() in a Java program.
5. Write a Java program to convert a primitive int to an Integer and back to int.
6. Convert a String to a Double using the appropriate wrapper class method.

DAY 13 STRING HANDLING

1. Write a Java program to get the character (Unicode code point) at the given index within the string.
2. Write a Java program to count Unicode code points in the specified text range of a string.
3. Write a Java program to test if a given string contains the specified sequence of char values.
4. Write a Java program to compare a given string to a specified string buffer.
5. Write a Java program to create a String object with a character array.
6. Write a Java program to check whether a given string ends with another string.
7. Write a Java program to get the index of all the characters of the alphabet.
   Sample Output:
   a b c d e f g h i j
   ==========
   36 10 7 40 2 16 42 1 6 20
   k l m n o p q r s t
   ==========
   8 35 22 14 12 23 4 11 24 31
   u v w x y z
   ==========
   5 27 13 18 38 37
   Sample Input:
   "The quick brown fox jumps over the lazy dog."
8. Write a program to append, insert, replace, delete, and reverse a string using both StringBuffer and StringBuilder.
9. Write a method that checks if a given string is a palindrome using StringBuffer or StringBuilder.
10. Write a program to extract a substring and a subsequence from a StringBuffer/StringBuilder object.

FILE HANDLING

1. Write a Java program to check if a file or directory specified by pathname exists or not.
2. Write a Java program to check if a file or directory has read and write permissions.
3. Write a Java program to read the contents of a text file line by line using FileReader.
4. Write a program that writes some text to a file using FileWriter.
5. Write a program to copy the contents of one file to another using FileInputStream and FileOutputStream.
6. Create a program to read a file and count the number of words, characters, and lines in the file.

DAY 14 COLLECTION FRAMEWORKS

1. Write a program to create an ArrayList of strings, add elements, remove an element, and iterate through the list.
2. Implement a program to find the largest and smallest elements and sum of elements in a List of integers.
3. Write a program to remove duplicates from a List of integers using a Set.
4. Write a program to read a text file and count the frequency of each word using a Map.
5. Write a program to sort a List of strings in alphabetical order using Collections.sort().
6. Write a program to simulate a simple phone book using a HashMap where keys are names and values are phone numbers. Include functionalities to add, remove, search, and display contacts.
7. Write a program to store countries and their capitals in a HashMap. Implement functionalities to:
   Add a new country-capital pair.
   Retrieve the capital by country.
   Display all countries and capitals.
8. Write a program to store student names and their grades in a HashMap. Provide functionalities to add a new student, update grades, and print all student grades.
9. Create a HashMap where the keys are custom Employee objects (with attributes id, name) and the values are their salaries. Ensure proper equals() and hashCode() methods for Employee.
10. Write a program that takes a Set of integers and removes all even numbers from it.

GENERICS

1. Write a generic class Box that can store any type of data. Implement methods to set and get the value stored in the Box.
2. Write a generic method printArray that can print the elements of any type of array.
3. Define a generic interface Pair<K, V> with methods getKey() and getValue(). Implement this interface in a class OrderedPair.

LAMBDA EXPRESSION

1. Write a program to implement a simple Runnable using a lambda expression.
2. Create a functional interface Operation with a method int operate(int a, int b). Use a lambda expression to implement addition.
3. Define a functional interface MathOperation with a method double operate(double a, double b). Implement this interface using a lambda expression to perform multiplication.
4. Define a functional interface MaxFinder with a method int findMax(int a, int b). Use a lambda expression to implement a method that returns the maximum of two integers.
5. Define a functional interface StringOperation with a method String concatenate(String a, String b). Implement this interface using a lambda expression to concatenate two strings.
6. Create a functional interface NumberChecker with a method boolean isEven(int a). Use a lambda expression to implement the method to check if a number is even.
7. Define a functional interface StringLengthCalculator with a method int length(String str). Use a lambda expression to implement the method that returns the length of a given string.

DAY 11 THREAD QUESTIONS

1. Basic Thread Creation

Problem Statement: Write a Java program that creates two threads. Each thread prints numbers from 1 to 5 with a delay of 500 milliseconds between each number. Ensure that both threads run concurrently.

Test Cases:

1. **Test Case 1: Verify Thread Concurrency**
   - **Description:** Ensure both threads are printing numbers concurrently.
   - **Expected Output:** Both threads should print numbers from 1 to 5 in an interleaved manner, i.e., not in strict sequence like 1, 2, 3, 4, 5 for both threads.
2. **Test Case 2: Verify Thread Delay**
   - **Description:** Verify that each thread has a 500ms delay between prints.
   - **Expected Output:** The time difference between each number printed by the same thread should be approximately 500ms.
3. **Test Case 3: Verify Completeness**
   - **Description:** Check if both threads complete their execution and print all numbers from 1 to 5.
   - **Expected Output:** Both threads should complete and print: `1 2 3 4 5` at least once in the output.

2. Thread Synchronization

Problem Statement: Create a class `Counter` that has a synchronized method `increment()` which increments a shared counter. Create two threads that both increment the counter 100 times. Ensure that the final value of the counter is 200.

Test Cases:

1. **Test Case 1: Verify Final Counter Value**
   - **Description:** Ensure the counter value is exactly 200 after both threads have finished executing.
   - **Expected Output:** Final counter value should be `200`.
2. **Test Case 2: Verify Concurrent Increment**
   - **Description:** Check if the counter increment is happening concurrently and not sequentially.
   - **Expected Output:** Observing the counter increments, there should be no race conditions.
3. **Test Case 3: Verify Method Synchronization**
   - **Description:** Ensure that the `increment()` method is synchronized and avoids race conditions.
   - **Expected Output:** No incorrect counter values during increments.

3. Thread Synchronization on Shared Resource

Problem Statement: Write a program where two threads are incrementing and decrementing the balance of a shared bank account. Use synchronized methods to prevent inconsistent states where the balance could become negative due to concurrent access.

Test Cases:

1. **Test Case 1: Verify Initial Balance**

    - **Description:** Ensure that the initial balance is set correctly.
    - **Expected Output:** Initial balance should match the predefined value.
2. **Test Case 2: Verify Balance Consistency**
    - **Description:** Verify that at no point the balance goes negative during concurrent operations.
    - **Expected Output:** Balance should never be negative.
3. **Test Case 3: Verify Final Balance**
    - **Description:** After all operations, check if the final balance is consistent with the expected result after all increments and decrements.
    - **Expected Output:** Final balance should be correct and consistent.

4. Producer-Consumer Problem (Inter-thread Communication)

Problem Statement: Implement the classic producer-consumer problem using `wait()` and `notify()`. One thread should produce integers and add them to a buffer, while another thread consumes the integers from the buffer. Ensure that the buffer has a limited size, and the producer should wait if the buffer is full, while the consumer should wait if the buffer is empty.

Test Cases:

1. **Test Case 1: Buffer Overflow Check**
    - **Description:** Verify that the producer does not add elements to the buffer when it is full.
    - **Expected Output:** Producer waits when the buffer reaches its maximum size.
2. **Test Case 2: Buffer Underflow Check**
    - **Description:** Verify that the consumer does not consume elements when the buffer is empty.
    - **Expected Output:** Consumer waits when the buffer is empty.
3. **Test Case 3: Proper Synchronization**
    - **Description:** Verify that the producer and consumer are synchronized using `wait()` and `notify()`.
    - **Expected Output:** No race conditions or deadlocks, and both threads function correctly.

5. Odd-Even Thread Alternation

Problem Statement: Create two threads: one prints odd numbers and the other prints even numbers. Ensure that the two threads alternate their execution, i.e., the odd-number thread prints first, then the even-number thread, and so on. Use inter-thread communication techniques to ensure proper alternation.

Test Cases:

1. **Test Case 1: Verify Odd-Even Alternation**
    - **Description:** Ensure that the odd and even numbers are printed alternately.
    - **Expected Output:** Numbers should be printed in the sequence: 1, 2, 3, 4, 5, 6, etc.
2. **Test Case 2: Verify Correct Printing Range**
    - **Description:** Verify that the odd thread only prints odd numbers and the even thread only prints even numbers.
    - **Expected Output:** Odd thread prints `1, 3, 5, ...`, and even thread prints `2, 4, 6, ...`.
3. **Test Case 3: Verify Thread Synchronization**
    - **Description:** Ensure that both threads are synchronized using inter-thread communication techniques.
    - **Expected Output:** No race conditions, and threads do not print out of sequence.