## Pre-assignment Tasks

---

1. What is exception handling? Explain with example.
2. List the benefits of writing Test first.
3. Discuss the difference between traditional methodologies where coding precedes testing and Test-Driven development(TDD) methodology where the test cases are written before the actual implementation code.

---

4. **(a).** Robust implementation in different programing language: Implement a robust method or function to calculate the area of a rectangle using two inputs: width and length. You are required to provide implementations in both **Java** and **Python**. After implementing, briefly discuss the key aspects that make your code robust. **(b).** Inside the main method of each implementation, write test cases to demonstrate the functionality of your method. Include scenarios that show normal operation, as well as cases that should trigger your error handling. This will help illustrate how your code behaves under various conditions.

5. Implement Test Cases for a Maximum Value Finder in an Array

**Objective:** Implement a Java class named `ArrayMaxFinder`` that includes a method to find the maximum value in an integer array.

```
public class ArrayMaxFinder {

    public int findMaxValue(int[] array) {
        if (array == null || array.length == 0) {
            throw new IllegalArgumentException("Array must not be null or
empty");
        }
        int max = array[0];
        for (int i = 1; i < array.length; i++) {
            if (array[i] > max) {
                max = array[i];
            }
        }
        return max;
    }
}
```

> **Test Cases to Implement:** Your test cases should verify the following scenarios:
>
> - **Normal Case:** Test with an array containing several elements to find the maximum value.
> - **Single Element:** Test with an array that contains only one element to ensure it returns that element as the max.
> - **Negative Values:** Test with an array containing all negative numbers to verify it finds the maximum correctly.

> - **Mixed Values:** Test with an array containing both positive and negative numbers, including zero, to ensure it identifies the maximum value correctly.
> - **Identical Elements:** Test with an array where all elements are the same to check if it returns that value as the max.
> - **Invalid Input:** Test with a null or empty array to verify the appropriate exception is thrown.

> **Guidelines:**

- Write clear and concise test cases ensuring they cover all the above scenarios.
- Use JUnit or a similar testing framework for implementing your test cases.

**maven**

```
<dependencies>
    [...]
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-engine</artifactId>
        <version>5.9.1</version>
        <scope>test</scope>
    </dependency>
    [...]
</dependencies>
```

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.2</version>
    <scope>test</scope>
</dependency>
```

- Provide comments to explain the purpose of each test case.
- Ensure your test cases are independent and do not affect each other.

---

Step 1.

> src | | main | -----|--- Java | com | example | -----|--- Java | com | example | --- ArrayMaxFinder.java | test | -----|--- Java | com | example | -----|--- Java | com | example --- ArrayMaxFinderTest.java

Step 2.

**ArrayMaxFinder.java**

```
public class ArrayMaxFinder {
    /*
     * 1. Check if the Array is empty or not
     * 2.      if true, the process leads to an exception throw
```

```
     * 3.      else, continue to the next activity
     * 4. Initializa 'max' with First Array Element: sets up the initial
condition for comparsion.
     * 5. Loop through Each Array Element Starting from Index 1:
     * 6.   Within the loop, a decision node checks if the current element
is greater thasn `max`
     * 7.   if true, update `max` within the current element value
     * 8.   else , proceed with the loop
     * 9. Returen `max`: Once all elements have been checked, the process
moves to return the `max` value
     * 10. End: The conclusion of the activity
     */
    public int findMaxValue(int[] array) {
        // default value
        return -1;
    }
}
```

**ArrayMaxFinderTest.java**

```
public class ArrayMaxFinderTest {

    // Normal Case: Test with an array containing several elements to find
the maximum value.

    // Single Element: Test with an array that contains only one element
to ensure it returns that element as the max.

    // Negative Values: Test with an array containing all negative numbers
to verify it finds the maximum correctly.

    // Mixed Values: Test with an array containing both positive and
negative numbers, including zero, to ensure it identifies the maximum
value correctly.

    // Identical Elements: Test with an array where all elements are the
same to check if it returns that value as the max.

    // Invalid Inputs: Test with a null or empty array to verify the
appropriate exception is thrown.
}
```

---

Step 3. Below is an example of how you might structure your test code. This example assumes you're using **JUnit 5**, but you can adjust it for other versions or testing frameworks as necessary.

```
import org.junit.Test;
```

```
public class ArrayMaxFinderTest {

    // Normal Case: Test with an array containing several elements to find
the maximum value.
    @Test
    public void testFindMaxValueNormalCase() {

    }

    // Single Element: Test with an array that contains only one element
to ensure it returns that element as the max.
    @Test
    public void testFindMaxValueWithSingleElement() {

    }

    // Negative Values: Test with an array containing all negative numbers
to verify it finds the maximum correctly.
    @Test
    public void testFindMaxValueWithNegativeValues() {

    }

    // Mixed Values: Test with an array containing both positive and
negative numbers, including zero, to ensure it identifies the maximum
value correctly.
    @Test
    public void testFindMaxValueWithMixedValues() {

    }

    // Identical Elements: Test with an array where all elements are the
same to check if it returns that value as the max.
    @Test
    public void testFindMaxValueWithIdenticalValues() {

    }

    // Invalid Inputs: Test with a null or empty array to verify the
appropriate exception is thrown.
    @Test
        public void testFindMaxValueThrowsExceptionForEmptyArray() {

    }

    @Test
        public void testFindMaxValueThrowsExceptionForNullArray() {
        //
    }
}
```

Step 4. Implement a Java class named `ArrayMaxFinder` that includes a method to find the maximum value in an integer array.

```
public class ArrayMaxFinder {

    public int findMaxValue(int[] array) {
        if (array == null || array.length == 0) {
            throw new IllegalArgumentException("Array must not be null or
empty");
        }
        int max = array[0];
        for (int i = 1; i < array.length; i++) {
            if (array[i] > max) {
                max = array[i];
            }
        }
        return max;
    }
}
```

---

6. Write comprehensive test case of basic Calculator class for the divide, methods of the Calculator class. The test cases should cover both normal and edge cases and for the divide method, include a test case for divison by zero.

7. Inventory Management System

---

**Answers**

1. **What is exception handling? Explain with example.**

> Exception handling in programming is a mechanism that deals with **runtime errors**, maintaining the normal flow of the application. An exception is an event that occurs during the execution of a program that **disrupts the normal flow of instructions**. Exception handling ensures that the code can handle these **errors gracefully without crashing**.

> In Java, exception handling is managed through five keywords: `try`, `catch`, `finally`, `throw`, and `throws`. (1) **try:** The try block contains a set of statements where an exception can occur. (2) **catch:** The catch block is used to handle the exception. It must be used after the try block only. You can have multiple catch blocks for different types of exceptions. (3) **finally:** The finally block is used to execute a set of statements, regardless of whether an exception is caught or not. It is often used to close resources like files or databases. (4) **throw:** This keyword is used to explicitly throw an ***exception*** from a ***method*** or any block of code. (5) **throws:** The throws keyword is used in ***method signatures*** to declare the exceptions that a method might throw.

**syntax**

```
try {
    // statement
} catch(args e) {
    // statement, throw
} finally {
    // statement
}
```

2. **Benefits of Writing Test First**

> 1. Early Bug Detection
> 2. Design Quality
> 3. Refactoring and Code Maintenance
> 4. Clear Requirementrs

3.

> Test-Driven Development (TDD) is a software development methodology where test cases are written before the actual implementation code. In TDD, developers first define the specifications and requirements through tests, then write code to pass these tests. This differs from traditional methodologies where coding precedes testing.

> ***Challenges and Considerations Challenges or Limitations:***
>
> - **Time-Consuming:** Initially, TDD can be more time-consuming, as writing tests before implementation requires additional upfront effort.
> - **Learning Curve:** There's a learning curve associated with adopting TDD, especially for teams used to traditional development methods.
> - **Complex Scenarios:** In complex scenarios, defining tests upfront can be challenging, potentially limiting the effectiveness of TDD.

4.

**Steps used to ensure the robustness of the codes are:**

> **Type Checking and Validation:** In Python, `isinstance`` is used to check the type of the inputs. This prevents type-related errors when the function is called with incorrect types. Java, being statically typed, inherently ensures type safety through its compilation checks. The parameters width and height are defined as doubles, ***so the method cannot be called with non-numeric types.***

> **Input Validation:** Both implementations check whether length and height are **non-negative**. This is crucial because negative values for these parameters don't make sense in the context of a rectangle's area and would lead to incorrect results. The use of conditional statements to **check** the value of the ***inputs*** and ***throw exceptions*** in case of **invalid data** is common in both languages.

> **Error Handling:** In Python, the function `raises` a `TypeError` for incorrect types and a `ValueError` for invalid values. This distinction helps the caller understand the nature of the error. Similarly, in Java, an `IllegalArgumentException` is *thrown* if either the width or length is negative, clearly informing the caller of the issue.

> **Language-Specific Considerations:** Python's dynamic typing requires explicit type checks at **runtime**, which is handled gracefully in the implementation. Java, on the other hand, relies on its **compile-time** type checking, making the code less prone to type-related errors at runtime. However, value checks are still crucial.

> **Clarity and Maintainability:** Both codes are written in a clear and concise manner, making them easy to understand and maintain. Proper naming conventions and error messages are used, which enhances readability and helps other developers understand the code quickly.

5.

> **The test cases should cover the following:**
>
> - **Normal Case:** Test with an array containing several elements, including the target element.
> - **No Occurrence:** Test with an array that does not contain the target element.
> - **Multiple Occurrences:** Test with an array where the target element occurs multiple times.
> - **Empty Array:** Test with an empty array to ensure no errors occur.
> - **Negative and Zero Values:** Test with an array containing negative numbers and zero, including these as target elements.