ISE Assignment (2023-S2-V1)

- **Assessment Name:** Introduction to Software Engineering
- **Your Name (as in Blackboard):** Mushaf Majeed
- **Curtin Student ID:** 21408410
- **Practical Class (Date/Time):** 20/10/2023 17/10/2023

Introduction

I created a GitHub repository via GitHub Desktop. This initial step involved setting up a version control system to manage the project efficiently. With the repository in place, I could easily track changes, collaborate with others, and maintain a comprehensive history of the project's development. Creating a repository from the outset is crucial for ensuring the project's organization and version control. Once the repository was established, I moved on to the core of the project. Thoroughly understanding the tasks that needed to be performed was pivotal. This involved a detailed analysis of the project requirements and objectives, breaking them down into manageable components. A clear understanding of the tasks was essential to ensure that the project would meet its goals effectively and efficiently. Upon a comprehensive grasp of the requirements, the next step was to create test cases. Writing test cases served as a critical quality assurance measure. These test cases provided a structured framework for evaluating the functionality and performance of the developed code. Each test case was designed to validate a specific aspect of the project, ensuring that it met the desired specifications. With the test cases in hand, I began the process of developing the code. Each line of code was written with the test cases in mind, ensuring that the project's functionality aligned with the predefined requirements. This approach not only guaranteed that the project met its goals but also made debugging and troubleshooting more systematic and efficient. In addition to diligent coding, I maintained a disciplined daily routine. At the end of each workday, I made it a practice to commit and push my work to the repository. This not only acted as a backup of the project but also allowed for a transparent and continuous record of project development. It provided insights into the progress made on a daily basis, facilitating collaboration and tracking the project's evolution.

Modularity design

Shape Identification Module

- **Description**: This module identifies the shape of a regular polygon based on the number of sides given.
- **Inputs**: The number of sides of the polygon.
- **Output**: Returns the name of the identified shape and, optionally, an image of the shape.
- **Assumptions**: Validates input for a valid range (e.g., between 3 and 7) and handles invalid inputs.

Area and Perimeter Calculation Module

- **Description**: This module calculates the area and perimeter of a regular polygon.

- **Inputs**: Length of a side and, optionally, the number of sides or the shape name.
- **Output**: Returns the area and perimeter. Can also display a message comparing the calculated area with two given situations.
- **Assumptions**: Validates input, handles situations when the number of sides is outside the expected range, and calculates area and perimeter correctly.

Image Display Module

- **Description**: This module displays images of regular polygons.
- **Inputs**: The name of the shape.
- **Output**: Displays the image on the screen.
- **Assumptions**: Has access to predefined images in the "ISEimages" folder.

Input Validation Module

- **Description**: This module validates user inputs for the number of sides and length of sides.
- **Inputs**: User inputs from the keyboard or parameters.
- **Output**: Provides validation feedback and ensures inputs are within the expected range.
- **Assumptions**: Handles invalid inputs and provides user-friendly error messages.

Message Display Module

- **Description**: This module handles the display of messages and results to the user.
- **Inputs**: Messages and results to be displayed.
- **Output**: Displays messages on the screen.
- **Assumptions**: Communicates results, validation messages, and comparisons between calculated and given values.

Main Module (Orchestrator)

- **Description**: The main module that coordinates the workflow, interacts with the user, and calls other modules.
- **Inputs**: User inputs and function calls to other modules.
- **Output**: Communicates with other modules and displays results to the user.
- **Assumptions**: Drives user interaction, handles the main program flow, and ensures the correct execution of various sub-modules.

Running the Production Code

Output

Enter the number of sides: 3 Enter the side length: 5 The shape with 3 sides is a Triangle Area: 10.825317547305483 Image not found for Triangle Area is too small.

1. **Setup**: Ensure you have the following in place:
    - Java Development Kit (JDK) installed.
    - The "ISEimages" folder with PNG images in the same directory as the code.

2. **Compile the Code**: Open your terminal or command prompt, navigate to the directory containing your Java file (e.g., `ChildrensMathSoftware.java`), and compile it with the following command: javac ChildrensMathSoftware.java
3. **Run the Code**: After successful compilation, run the code with the following command: java ChildrensMathSoftware
4. **Input**: You'll be prompted to enter the number of sides and the side length.
5. **Output**: The code will display information about the identified shape, area, and potentially an image.

Explanation of Modularity Concepts

1. **Functional Decomposition**: The code is structured into functions (methods) that encapsulate specific tasks like identifying shapes, calculating areas, displaying images, and user input/output.
2. **Abstraction**: Each method provides a high-level abstraction for a specific task, making the code easier to understand.
3. **Encapsulation**: The methods hide the internal logic of their respective tasks, promoting data encapsulation and making it easier to change or extend functionality.
4. **Reusability**: Functions like `identifyShape`, `calculateArea`, and `displayImage` can be reused in other parts of the program or other programs.
5. **Separation of Concerns**: The code separates concerns by having distinct modules for shape identification, image display, and area calculation.
6. **Low Coupling**: The modules are loosely coupled, which means changes to one module have minimal impact on others.
7. **High Cohesion**: Each module focuses on a single responsibility, promoting high cohesion.

Review Checklist

Here's a checklist to review the code:

- [ ] Is the code well-documented with comments?
- [ ] Are variable and method names descriptive?
- [ ] Is input validation handled effectively?
- [ ] Are exception-handling mechanisms in place?
- [ ] Are there any code smells or duplications?
- [ ] Is the code logically organized, easy to read, and maintain?
- [ ] Does the code produce the correct output for various input scenarios?
- [ ] Are there any edge cases or potential issues?

Refactoring Decisions

Based on the review checklist, if you find any issues or potential improvements, you can consider refactoring the code to address them. For example, you may want to enhance input validation, improve error handling, or provide more meaningful error messages.

Revised Module Descriptions

If you decide to refactor the code and make significant changes to the module structure, you may need to update the module descriptions accordingly. For example, you might create new modules or modify existing ones to improve modularity. The revised module descriptions should reflect these changes.

Blackbox Test Cases for ChildrensMathSoftware Modules

1. `identifyShape` Module (Using Equivalence Partitioning)

This module identifies the shape based on the number of sides provided. We'll test it with valid and invalid inputs.

Test Case	Input (Number of Sides)	Expected Output
Test 1	2	"Number is too small"
Test 2	3	"Triangle"
Test 3	4	"Square"
Test 4	6	"Invalid number of sides"
Test 5	8	"Number is too big"

2. `calculateArea` Module (Using Equivalence Partitioning and Boundary Value Analysis)

This module calculates the area of regular polygons based on the number of sides and side length. We'll consider both equivalence partitioning and boundary values.

Equivalence Partitioning:

Test Case	Inputs (Number of Sides, Side Length)	Expected Output
Test 1	(2, 5.0)	-1 (Invalid)
Test 2	(3, 5.0)	≈(Area of an Equilateral Triangle)
Test 3	(4, 5.0)	≈(Area of a Square)
Test 4	(6, 5.0)	-1 (Invalid)
Test 5	(3, 0.0)	-1 (Invalid)

Boundary Value Analysis:

Test Case	Inputs (Number of Sides, Side Length)	Expected Output

Test 1	(3, 1.0)	≈(Area of an Equilateral Triangle)
Test 2	(4, 1.0)	≈(Area of a Square)
Test 3	(3, 20.0)	≈(Area of an Equilateral Triangle)
Test 4	(4, 20.0)	≈(Area of a Square)
## White-Box Testing for `calculateArea` Module		
Test Case	Input (Number of Sides, Side Length)	Expected Output
----------	------------------------ ---------------	----------
Test 1	(3, 5.0)	≈ 10.83
Test 2	(4, 5.0)	25.0

White-Box Testing for `identifyShape` Module

Test Case	Input (Number of Sides)	Expected Output	Description
Test 1	3	"Triangle"	Statement coverage (Basic Block Testing)
Test 2	4	"Square"	Decision coverage (Branch Testing)
Test 3	6	"Invalid number of sides"	Path coverage (Invalid input)
## Summary			
Module Name	BB (EP)	BB (BVA)	WB
----------	----------	----------	----
calculateArea	done	done	done
identifyShape	done	not done	done
displayImage	not done	not done	not applicable
In the table:			

"BB" stands for Black-Box testing.			
"WB" stands for White-Box testing.			
"EP" stands for Equivalence Partitioning.			
"BVA" stands for Boundary Value Analysis.			
"Data Type/s" indicates the data types involved in testing.			
"Form of Input/Output" describes the form of input and output for each module.			
"done" indicates that the task has been completed for that module.			
"not done" indicates that the task has not been completed for that module.			
"not applicable" indicates that a specific type of testing is not relevant for the module.			
### Commit Log has been attached as an image in repository folder			
Achievements:			

1. **Module Design and Implementation**: I successfully designed and implemented the required modules for the `ChildrensMathSoftware`. These modules include `calculateArea`, `identifyShape`, and `displayImage`.
2. **Test Design**: I designed test cases for these modules using various testing techniques, including black-box testing with equivalence partitioning (EP) and boundary value analysis (BVA), as well as white-box testing with path coverage.
3. **Test Code Implementation**: I translated the test designs into actual Java code, using JUnit for white-box testing and plain Java for black-box testing.
4. **Execution and Results**: I executed the test cases and obtained results, identifying any test failures. I used the results to verify the correctness of the code and improve it where needed.

Challenges:

1. **Designing Equivalence Partitions**: One challenge was creating comprehensive equivalence partitions for all possible inputs. Determining meaningful partitions for boundary value analysis was also challenging.
2. **Implementing White-Box Tests**: Writing white-box tests, especially for path coverage, required a deep understanding of the code structure. Ensuring full path coverage can be complex and time-consuming.

Limitations and Ways to Improve:

1. **Limited Test Coverage**: The testing performed covers the basic functionalities of the program. However, for a production application, additional test scenarios, edge cases, and corner cases would need to be considered.
2. **Refinement of White-Box Testing**: White-box testing could benefit from more thorough path coverage testing, including edge cases and more complex control flow scenarios.
3. **Error Handling**: The code could be improved by implementing more robust error handling mechanisms. Currently, it assumes that inputs are within acceptable ranges.
4. **Automated Testing Framework**: For larger projects, it's advisable to set up an automated testing framework that runs tests automatically when code changes are made.
5. **User Interface**: The program could be enhanced with a user-friendly graphical user interface (GUI) to facilitate user interactions and improve the overall user experience.

In conclusion, the work done so far provides a foundation for testing the `ChildrensMathSoftware` program. However, there is room for improvement, particularly in terms of test coverage, error handling, and user interface. As with any software project, continuous testing and refinement are essential for ensuring its reliability and usability.