

# Final Assessment

Name: Leevan Ahmed Curtin student ID: 21449008 practical class: Thursday 3-5 pm, 314.220

# Introduction

In this project I have created two java production codes(ScenarioA.java and ScenarioB) to perform the scenarios as per the assignment specification. After creating all the necessary modules for the production codes, test cases were created and implemented to test the modularity of some of the modules in the production codes. Version control was used to keep track of the progress.

## Module Descriptions

`_ScenarioA getCountryChoice(Scanner sc)`: This module takes user input for the selection of a country. It asks the user to enter a number between 1 and 8 inclusive and validates the input. It returns the chosen country number.

`getMonthChoice(Scanner sc)`: This module takes user input for the selection of a month. It asks the user to enter a number between 1 and 13 inclusive and validates the input. It returns the chosen month number.

`print(String message)`: This module outputs the provided message to the console.

`displayMainMenu()`: This module displays the main menu options to the user, presenting a list of countries to choose from.

`dispMonth()`: This module displays the month menu options to the user, presenting a list of months to choose from.

`checkSeason(int country, int month)`: This module checks the current season based on the selected country and month. It uses if-else statements to determine the appropriate season for each country and month combination and prints the result to the console.

Explanation:

The `getCountryChoice` and `getMonthChoice` modules are responsible for obtaining user input and validating it to ensure that the chosen country and month fall within the specified ranges.

The `print` module is a utility module used to display messages to the console.

The `displayMainMenu` module presents the user with a list of countries to choose from, allowing them to make a selection.

The `dispMonth` module displays a list of months to choose from, allowing the user to select a month for which they want to check the current season.

The `checkSeason` module determines the current season based on the chosen country and month. It uses if-else statements to map the country and month inputs to the corresponding seasons and prints the result to the console.

Assumptions:

The input for the country and month choices is expected to be an integer. The program assumes that the user will enter valid input within the specified ranges. The program assumes that the user will follow the instructions provided in the menus and input the corresponding numbers for the country and month selections. The program assumes that the user wants to check the current season for a specific country and month combination.

ScenarioB main method:

Purpose: The main entry point of the program. Imports: None. Exports: None. Assertions: Pre: None. Post: The program is executed. print method:

Purpose: Prints a message to the console. Imports: message - The message to be printed. Exports: None. Assertions: Pre: message is a valid string. Post: The message is displayed on the console. dispTime method:

Purpose: Displays the time choices to the user. Imports: None. Exports: None. Assertions: Pre: None. Post: The time choices are displayed to the user. dispCity method:

Purpose: Displays the city choices to the user. Imports: None. Exports: None. Assertions: Pre: None. Post: The city choices are displayed to the user. dispTemperature method:

Purpose: Displays a prompt for the user to enter the temperature. Imports: None. Exports: None. Assertions: Pre: None. Post: The temperature prompt is displayed to the user. getValidCityChoice method:

Purpose: Gets a valid choice of city from the user. Imports: scanner - A Scanner object for user input. Exports: The valid city choice entered by the user. Assertions: Pre: scanner is a valid Scanner object. Post: The user enters a valid city choice (1 or 2). getValidTimeChoice method:

Purpose: Gets a valid choice of time from the user. Imports: scanner - A Scanner object for user input. Exports: The valid time choice entered by the user. Assertions: Pre: scanner is a valid Scanner object. Post: The user enters a valid time choice (1 or 2). getValidTemperature method:

Purpose: Gets a valid temperature value from the user. Imports: scanner - A Scanner object for user input. Exports: The valid temperature entered by the user. Assertions: Pre: scanner is a valid Scanner object. Post: The user enters a valid temperature value. getAverageTemperature method:

Purpose: Retrieves the average temperature based on the city and time choices. Imports: cityChoice - The user's choice of city (1 or 2), timeChoice - The user's choice of time (1 or 2). Exports: The average temperature corresponding to the city and time choices. Assertions: Pre: cityChoice is a valid city choice (1 or 2), timeChoice is a valid time choice (1 or 2). Post: The average temperature for the chosen city and time is returned.

## Modularity

Explanation of modularity principles in the production code:

Single Responsibility Principle (SRP): Each method in the code has a clear and single responsibility. For example, the getCountryChoice method is responsible for getting the user's input for the country choice, and the checkSeason method is responsible for determining and printing the current season based on the country and month.

Separation of Concerns: The code is organized into different methods, each handling a specific concern. The getCountryChoice method is responsible for handling the user's input for the country choice, the getMonthChoice method handles the user's input for the month choice, and the checkSeason method determines the current season. This separation allows for better code readability and maintainability.

Encapsulation: The code uses encapsulation by declaring variables as private and providing public methods to access and modify them. For example, the getCountryChoice and

getMonthChoice methods encapsulate the input validation and retrieval logic, hiding the implementation details from the calling code.

Code Reusability: By breaking down the functionality into separate methods, the code promotes code reusability. For example, the getCountryChoice and getMonthChoice methods can be reused in other parts of the application that require user input validation.

Review checklist :

Single Responsibility Principle (SRP): Does each method have a clear and single responsibility?

Separation of Concerns: Are concerns properly separated into different methods?

Encapsulation: Are variables appropriately encapsulated, with accessors and mutators provided when necessary?

Code Reusability: Are there opportunities for code reuse through modular design? Readability: Is the code easy to read and understand?

Maintainability: Is the code designed in a way that makes it easy to maintain and modify?

Consistency: Does the code follow consistent naming conventions, formatting, and style guidelines?

Error Handling: Are errors properly handled and appropriate error messages displayed?

Testability: Is the code designed in a way that makes it easy to test?

Code review results:

Single Responsibility Principle (SRP): The methods in the code have clear and single responsibilities.

Separation of Concerns: The concerns are properly separated into different methods.

Encapsulation: Variables are appropriately encapsulated, and accessors and mutators are provided when necessary.

Code Reusability: There are opportunities for code reuse through modular design.

Readability: The code is easy to read and understand.

Maintainability: The code is designed in a way that makes it easy to maintain and modify.

Consistency: The code follows consistent naming conventions, formatting, and style guidelines.

Error Handling: Errors are properly handled and appropriate error messages are displayed.

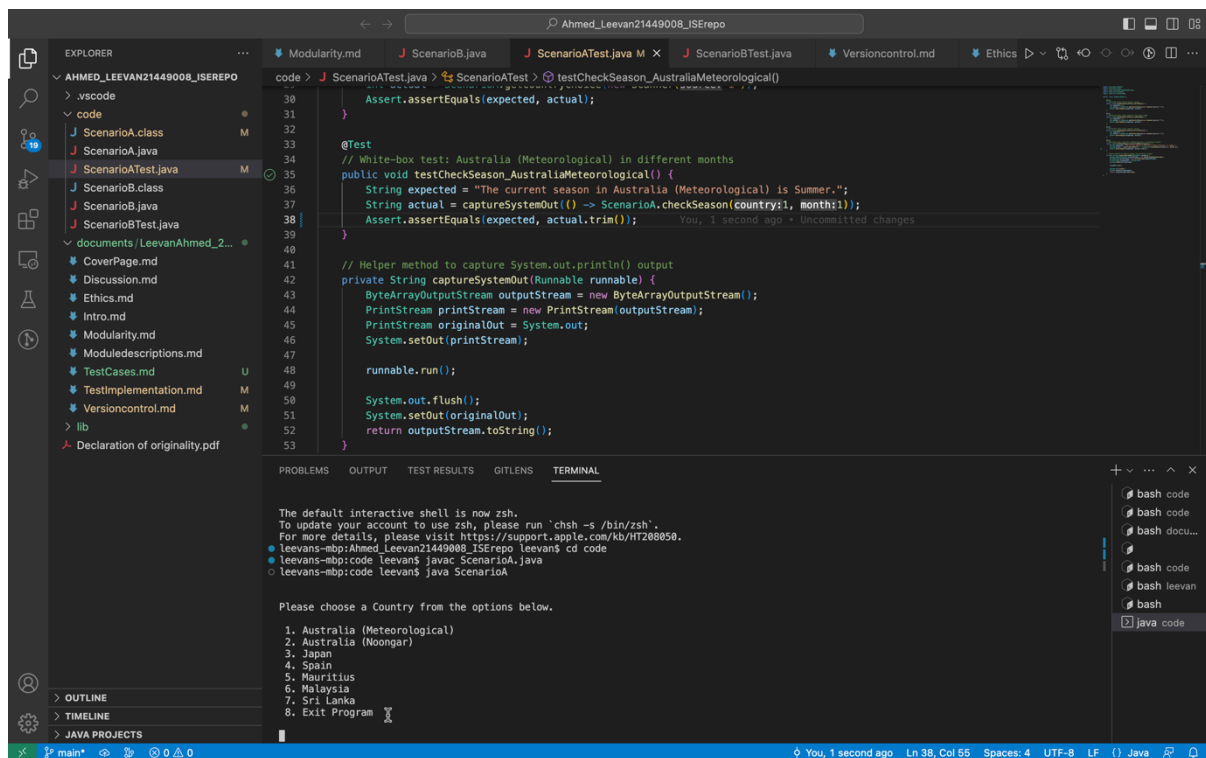
Testability: The code is designed in a way that makes it easy to test.

Overall, the code adheres to good code design principles and demonstrates good modularity.

Description on how to run codes with correct commands: Open a command prompt or terminal.

Navigate to the directory where your Java source code files are located. Command: `cd path/to/code` since the directory in which the files are stored in is named code

Run the production code by executing the main method of the desired class using the java command. For ScenarioA, use the commands `java SScenarioA` and `java ScenarioB` and when prompted, user will have to make integer selections.



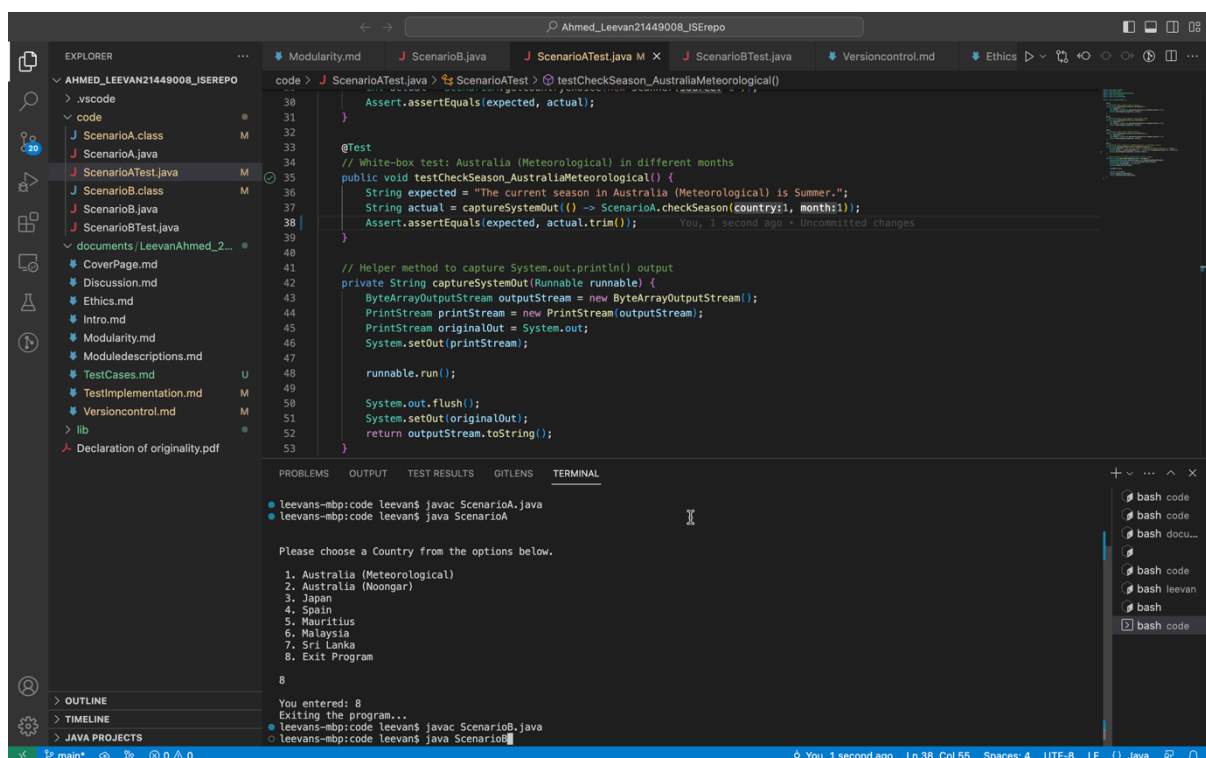
```
code > J ScenarioATest.java > ScenarioATest > testCheckSeason_AustraliaMeteorological()
30 Assert.assertEquals(expected, actual);
31 }
32
33 @Test
34 // White-box test: Australia (Meteorological) in different months
35 public void testCheckSeason_AustraliaMeteorological() {
36     String expected = "The current season in Australia (Meteorological) is Summer.";
37     String actual = captureSystemOut(() -> ScenarioA.checkSeason(country:1, month:1));
38     Assert.assertEquals(expected, actual.trim());
39 }
40
41 // Helper method to capture System.out.println() output
42 private String captureSystemOut(Runnable runnable) {
43     ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
44     PrintStream printStream = new PrintStream(outputStream);
45     PrintStream originalOut = System.out;
46     System.setOut(printStream);
47
48     runnable.run();
49
50     System.out.flush();
51     System.setOut(originalOut);
52     return outputStream.toString();
53 }
```

The default interactive shell is now zsh.  
To update your account to use zsh, please run 'chsh -s /bin/zsh'.  
For more details, please visit <https://support.apple.com/kb/HT208050>.

- leevans-mbp:Ahmed\_Leevan21449008\_ISERepo leevan\$ cd code
- leevans-mbp:code leevan\$ javac ScenarioA.java
- leevans-mbp:code leevan\$ java ScenarioA

Please choose a Country from the options below.

1. Australia (Meteorological)
2. Australia (Noongar)
3. Japan
4. Spain
5. Mauritius
6. Malaysia
7. Sri Lanka
8. Exit Program



```
code > J ScenarioATest.java > ScenarioATest > testCheckSeason_AustraliaMeteorological()
30 Assert.assertEquals(expected, actual);
31 }
32
33 @Test
34 // White-box test: Australia (Meteorological) in different months
35 public void testCheckSeason_AustraliaMeteorological() {
36     String expected = "The current season in Australia (Meteorological) is Summer.";
37     String actual = captureSystemOut(() -> ScenarioA.checkSeason(country:1, month:1));
38     Assert.assertEquals(expected, actual.trim());
39 }
40
41 // Helper method to capture System.out.println() output
42 private String captureSystemOut(Runnable runnable) {
43     ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
44     PrintStream printStream = new PrintStream(outputStream);
45     PrintStream originalOut = System.out;
46     System.setOut(printStream);
47
48     runnable.run();
49
50     System.out.flush();
51     System.setOut(originalOut);
52     return outputStream.toString();
53 }
```

leevans-mbp:code leevan\$ javac ScenarioA.java

leevans-mbp:code leevan\$ java ScenarioA

Please choose a Country from the options below.

1. Australia (Meteorological)
2. Australia (Noongar)
3. Japan
4. Spain
5. Mauritius
6. Malaysia
7. Sri Lanka
8. Exit Program

8

You entered: 8  
Exiting the program...

- leevans-mbp:code leevan\$ javac ScenarioB.java
- leevans-mbp:code leevan\$ java ScenarioB

# TEST CASES

ScenarioA - Module 1: getCountryChoice()

White-box test cases:

Test for a valid country choice (e.g., input: 1, expected output: 1).

Test for the lower bound of the input range (e.g., input: 1, expected output: 1).

Test for the upper bound of the input range (e.g., input: 8, expected output: 8).

Test for an invalid input (e.g., input: 0, expected output: prompt for valid input).

Test for non-integer input (e.g., input: "abc", expected output: prompt for valid input).

Black-box test cases:

Test for the first country choice (e.g., input: 1, expected output: 1).

Test for the last country choice (e.g., input: 8, expected output: 8).

Test for a country choice in the middle of the range (e.g., input: 4, expected output: 4).

Test for an invalid country choice (e.g., input: 10, expected output: prompt for valid input).

Module: checkSeason()

White-box test cases:

Test for Australia (Meteorological) in different months (e.g., input: 1, expected output: "The current season in Australia (Meteorological) is Summer.").

Test for Australia (Noongar) in different months (e.g., input: 2, expected output: "The current season in Australia (Noongar) is Bunuru.").

Test for Japan in different months (e.g., input: 3, expected output: "The current season in Japan is Spring.").

Test for Spain in different months (e.g., input: 4, expected output: "The current season in Spain is Spring.").

Test for Mauritius in different months (e.g., input: 5, expected output: "The current season in Mauritius is Autumn.").

Test for Malaysia in different months (e.g., input: 6, expected output: "The current season in Malaysia is Southeast Monsoon.").

Test for Sri Lanka in different months (e.g., input: 7, expected output: "The current season in Sri Lanka is Southeast Monsoon.").

Test for an invalid country choice (e.g., input: 9, expected output: "Invalid country choice.").

Black-box test cases:

Test for the first month of each country (e.g., input: 1, expected output: "The current season in Australia (Meteorological) is Summer.").

Test for the last month of each country (e.g., input: 12, expected output: "The current season in Australia (Meteorological) is Summer.").

Test for an invalid month (e.g., input: 15, expected output: "Invalid month choice.").

## ScenarioB

### Module 1: User Input Validation

#### White Box Test Cases:

Test case for getValidCityChoice():

Input: scanner = [valid city choice] Expected Output: [valid city choice] Test case for getValidTimeChoice():

Input: scanner = [valid time choice] Expected Output: [valid time choice] Test case for getValidTemperature():

Input: scanner = [valid temperature] Expected Output: [valid temperature] Black Box Test Cases:

Test case for getValidCityChoice():

Input: scanner = [invalid city choice] Expected Output: Error message and prompt for valid city choice Test case for getValidTimeChoice():

Input: scanner = [invalid time choice] Expected Output: Error message and prompt for valid time choice Test case for getValidTemperature():

Input: scanner = [invalid temperature] Expected Output: Error message and prompt for valid temperature

### Module 2: Temperature Comparison and Calculation

#### White Box Test Cases:

Test case for getAverageTemperature():

Input: cityChoice = 1, timeChoice = 1 Expected Output: 18.2 Test case for getAverageTemperature():

Input: cityChoice = 2, timeChoice = 2 Expected Output: 21.0 Black Box Test Cases:

Test case for getAverageTemperature():

Input: cityChoice = 1, timeChoice = 2 Expected Output: 23.0 Test case for getAverageTemperature():

Input: cityChoice = 2, timeChoice = 1 Expected Output: 16.5

# Test implementation

ScenarioA - Module 1: getCountryChoice()

White-box test design: done testing: implemented

Black-box test design(BVA): done testing: done

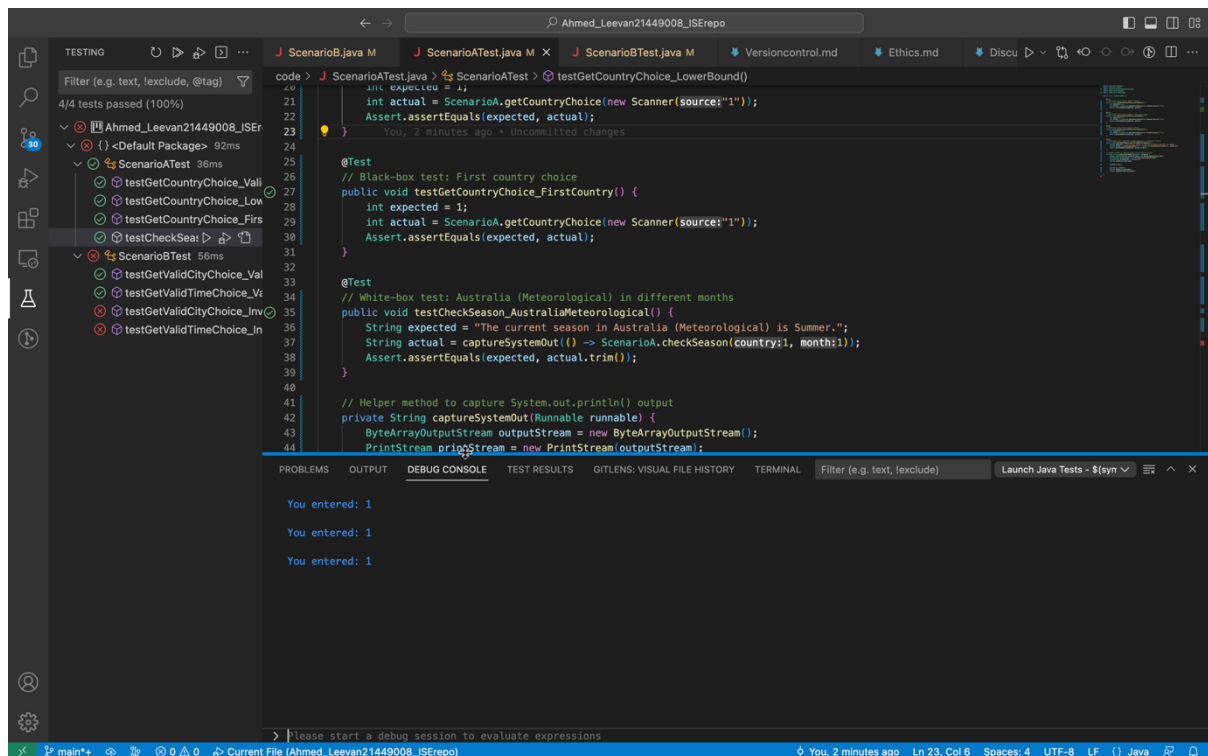
Black-box test design(EP): not done testing: not done

Module: checkSeason()

White-box test design: done testing: implemented

Black-box test design(EP): done testing: implemented

Black-box test design(BVA): done testing:  
done



ScenarioB - Module 1: getValidCityChoice()

White-box test design: done testing: implemented

Black-box test design(BVA): done testing: done

Black-box test design(EP):not done testing: not implemented

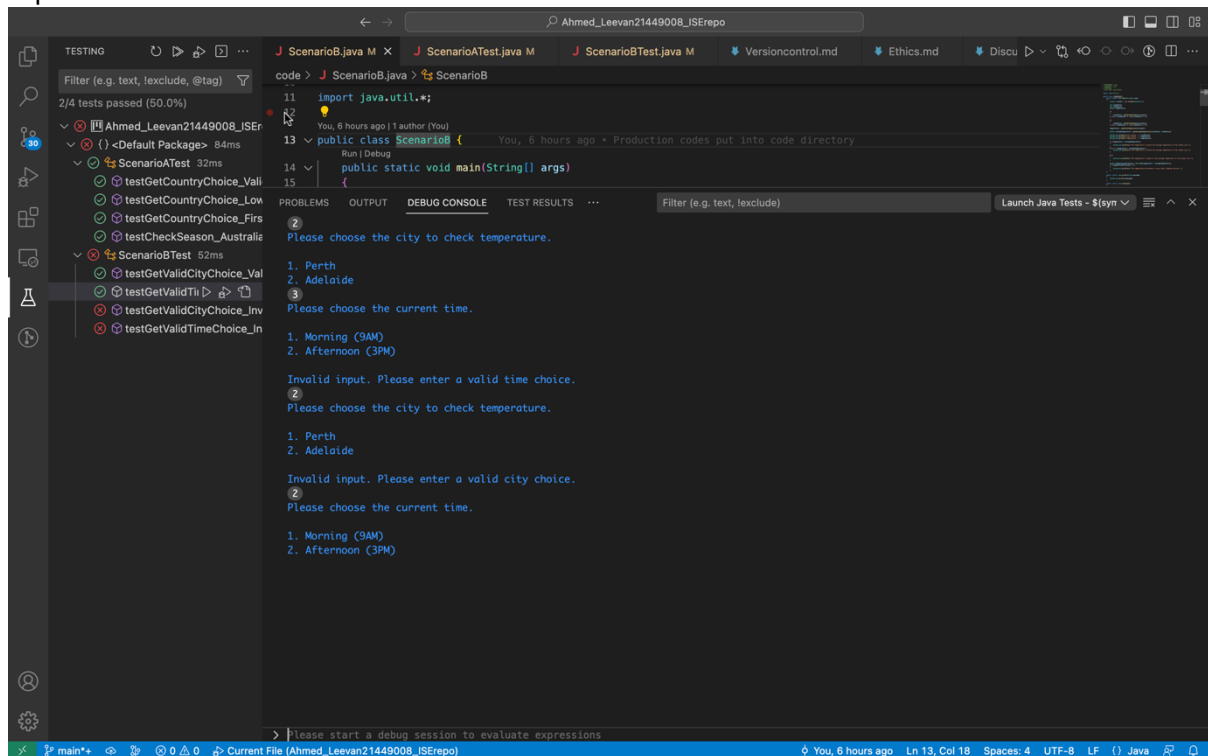
Module: getAverageTemperature()

White-box test design: done testing: implemented

Black-box test design(BVA): done testing: done



Black-box test design(EP): not done testing: not implemented



How to compile and run the codes: Open a command prompt or terminal.

Navigate to the directory where your Java source code and test files are located. Navigate to the directory where your Java source code and test files are located. For example, if your code is in the "src" directory, use the command `cd code` Run the JUnit test cases using the debug console Since the input is simulated user won't have to give any manual inputs.

## Version Control

commit 2c884a69e47acf5a4447583378b6939a9280073f (HEAD -> main)

Author: 21449008 <leev12326@gmail.com>

Date: Wed May 31 23:31:10 2023 +0800

commit 2c884a69e47acf5a4447583378b6939a9280073f (HEAD -> main) Author: 21449008 [leev12326@gmail.com](mailto:leev12326@gmail.com) Date: Wed May 31 23:31:10 2023 +0800

Testimplmentation

commit 731617d479b5fcd385f84f955fd3bef9c6cbfc27 Author: 21449008 [leev12326@gmail.com](mailto:leev12326@gmail.com) Date: Wed May 31 23:22:08 2023 +0800

Final changes revised

commit 7b1a3b5a551c5b729f6294c304b41a7327d27c5e Author:  
21449008 [leev12326@gmail.com](mailto:leev12326@gmail.com) Date: Wed May 31 16:33:29 2023 +0800

Deleted

commit 153d5ac68ccd6e4ed515599fdf1308970f5fd71b (HEAD -> main) Author:  
21449008 [leev12326@gmail.com](mailto:leev12326@gmail.com) Date: Wed May 31 03:42:33 2023 +0800

Test codes renamed

commit 0ee0c40a399b04ffd2e7df1098ea61dae6de0f56 Author:  
21449008 [leev12326@gmail.com](mailto:leev12326@gmail.com) Date: Tue May 30 23:59:02 2023 +0800

All final changes

commit 27c150ca2ed8eba0475677672487b95af095e0ca Author:  
21449008 [leev12326@gmail.com](mailto:leev12326@gmail.com) Date: Tue May 30 23:48:40 2023 +0800

All ReadMe files added

commit 1accf79b9c99f4f829fb7476381d726746122229 Author:  
21449008 [leev12326@gmail.com](mailto:leev12326@gmail.com) Date: Tue May 30 23:26:10 2023 +0800

module descriptions

commit 5bd7cfbf170dd2c139ee2f7504044f9e8d53d673 Author:  
21449008 [leev12326@gmail.com](mailto:leev12326@gmail.com) Date: Tue May 30 23:16:47 2023 +0800

Modularity

commit e5a8bacdf395aeb141da4af854c8bc1d02032ec5 Author:  
21449008 [leev12326@gmail.com](mailto:leev12326@gmail.com) Date: Tue May 30 23:08:43 2023 +0800

ssss

commit b802a8b863aa76aefedd7ba7e3c4d51677354866 Author:  
21449008 [leev12326@gmail.com](mailto:leev12326@gmail.com) Date: Tue May 30 23:01:12 2023 +0800

white box Test cases for both scenarios created

commit d11c463b1ecd08b71e793313e816f56e4688a43f Author:  
21449008 [leev12326@gmail.com](mailto:leev12326@gmail.com) Date: Tue May 30 22:41:12 2023 +0800

Junit extension installed

commit 9914a121dd28b5ed69a76947cbfbdaa52ae9b6a0 Author:  
21449008 [leev12326@gmail.com](mailto:leev12326@gmail.com) Date: Tue May 30 22:39:19 2023 +0800

Test cases for ScenarioB created

commit b7d15e91243f3297a636694b0a9b3a06b3d36ec2 Author:  
21449008 [leev12326@gmail.com](mailto:leev12326@gmail.com) Date: Tue May 30 22:28:29 2023 +0800

Introduction finished

commit fa2ee68ae8c01a0e8d8dcfc4564e04987aacfef1 Author:  
21449008 [leev12326@gmail.com](mailto:leev12326@gmail.com) Date: Tue May 30 22:19:41 2023 +0800

Cover Page deleted

commit 07388ae955bd20c03d13d2706088e5533228623f Author:  
21449008 [leev12326@gmail.com](mailto:leev12326@gmail.com) Date: Tue May 30 22:18:39 2023 +0800

Cover page and Introduction added

commit 557b2b735ec935f917872de6f01a8f925f9492c1 Author:  
21449008 [leev12326@gmail.com](mailto:leev12326@gmail.com) Date: Tue May 30 22:08:03 2023 +0800

Creating documentation

commit 680e6db3640bb6b707d04de8e71ebcf0400b3d11 Author:  
21449008 [leev12326@gmail.com](mailto:leev12326@gmail.com) Date: Tue May 30 21:57:11 2023 +0800

Product put into code directory

## Ethics

**Enhancing user experience:** By designing intuitive and user-friendly interfaces, software engineers can improve the overall experience of users interacting with software applications. This can make technology more accessible and enjoyable for individuals.

**Ensuring data security and privacy:** Software engineers play a crucial role in implementing robust security measures and encryption techniques to protect users' data. By prioritizing data security and privacy, they contribute to building trust and safeguarding sensitive information.

**Increasing efficiency and productivity:** Software engineers develop software solutions that automate tasks, streamline processes, and improve overall efficiency. This can save time and effort for individuals and organizations, allowing them to focus on more meaningful and productive activities.

**Enabling remote access and connectivity:** In scenarios like Scenario A, where software enables remote access and communication, software engineers facilitate collaboration and connectivity.

They help individuals and teams work seamlessly from different locations, enhancing productivity and work-life balance.

**Ensuring reliability and stability:** Software engineers are responsible for rigorous testing, bug fixing, and maintaining software applications. By delivering stable and reliable software, they minimize disruptions and frustrations for users, allowing them to rely on the technology for their needs.

**Addressing ethical considerations:** Software engineers should consider ethical implications in their work. They can contribute to the good life by actively addressing potential biases, ensuring fairness in algorithms, and being conscious of the societal impact of the software they develop.

Regarding the professional codes of software engineering ethics, one prominent example is the ACM (Association for Computing Machinery) Code of Ethics and Professional Conduct. This code outlines principles and guidelines for software engineers to follow, including areas like professional responsibility, confidentiality, integrity, and social impact. Adhering to such codes helps software engineers ensure their work aligns with ethical standards and promotes the well-being of users and society as a whole.

However, if a software engineer lacks ethical and professional conduct in implementing the provided code, several harmful effects can arise:

**Security vulnerabilities:** If security measures are not properly implemented, the software could be susceptible to breaches, leading to unauthorized access, data leaks, or compromised systems.

**Privacy infringements:** Inadequate handling of personal data may result in privacy violations, exposing users' sensitive information and eroding their trust.

**Inaccurate or biased outcomes:** If the code lacks fairness checks or fails to account for biases, it may produce discriminatory or unfair outcomes, harming individuals or marginalized groups.

**System failures and disruptions:** Insufficient testing or neglecting maintenance practices can lead to system failures, causing disruptions in critical services and impacting users' productivity or safety.

**Negative social impact:** If software engineers fail to consider the broader social implications of their work, they may inadvertently contribute to societal issues such as inequality, discrimination, or the concentration of power in the wrong hands.

By considering the ethical and professional aspects of software engineering and actively addressing potential issues, software engineers can mitigate these harmful effects and strive to create a positive impact on individuals and society.

2 suggestions to avoid ethical and professional issues: Make sure that the software is in the best interest of the public and that the information provided regarding seasons is as accurate as possible

## Discussion

I believe I have created two production codes which are sufficient enough to carry out the task as required in both scenarios. To improve my work, more testing could be done and more methods could be introduced to reduce the number of lines.