Student name: Matheus Almeida

SN: 33149385

Program 1

Test plan to American Flag

This test plan describes the testing activities for the AmericanFlagTest program, which generates a text-based representation of the American flag.

This plan covers functional testing of the program, verifying correct output of the stars and stripes patterns. It does not cover performance testing, security testing, or testing on different operating systems.

Items to be tested:

Star line generation: Correct output of alternating star lines.

Stripes line generation: Correct output of equal lines.

Overall flag structure: Correct combination and order of stars and stripes patterns.

Entry and Exit Criteria:

* Entry Criteria:

Java code compiled successfully.

Test environment set up.

* Exit Criteria:

All test cases executed.

All identified defects resolved and retested.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Description | Input | Output | Comments |
| 01 | Here we will test if the flag has the expected output without errors. |  |  | I didn't notice that the system.out.println was used incorrectly and the pattern flag was printed with a blank line next to it. |
| 02 | Here I fixed the error that I got previous. |  |  | After fix the system.out.println to system.out.print the error was fixed and now the flag is printed correctly. |

Program 2

Test plan to BMICalculator

This test plan describes the testing activities for the BMICalculator program, which calculates a user's Body Mass Index based on their weight, height, and gender.

This plan covers functional testing, input validation, and basic limits testing. It does not cover performance testing, security testing, or compatibility testing (different operating systems or Java versions).

Items to be tested:

Weight input validation: Handling valid and invalid weight inputs.

Height input validation: Handling valid and invalid height inputs.

Gender input validation: Handling valid and invalid gender inputs.

BMI calculation: Correct calculation of BMI based on weight and height.

BMI Interpretation: Correct classification of BMI into categories.

Multiple calculations: Ability to perform multiple calculations in a single program run.

Program termination: Correct program termination.

Types of tests: Functional testing, Black box testing focusing on inputs and outputs.

Entry and Exit Criteria:

Entry Criteria:

Java code compiled successfully.

Test environment setup.

Exit Criteria:

All test cases executed.

All identified defects resolved and retested.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Description | Input | Output | Comments |
| 01 | In this first test we will use an input that is expected to be the output of an underweight male person and test the handling validation. | 01:  aqs  02:  one meter  03:  Weight:50  Height: 1.80  Gender: M |  | As expected, after I type a letter instead of a number, the program automatically shows me an error message and asks me to repeat the input correctly. After the input is correctly entered, I get the expected output. |
| 02 | In this test we will use an input that is expected to be the output of an underweight female person. | Weight:42.1  Height: 1.75  Gender: F |  | As expected, after I type the entry's correctly it’s give me the result without errors. |
| 03 | In this test we will use an input that is expected to be the output of a normal weight female person. | Weight:53.9  Height: 1.61  Gender: F |  | As expected, after I type the entry's correctly it’s give me the result without errors. |
| 04 | In this test we will use an input that is expected to be the output of a normal weight male person. | Weight:73.9  Height: 1.78  Gender: M |  | As expected, after I type the entry's correctly it’s give me the result without errors. |
| 05 | In this test we will use an input that is expected to be the output of an overweight male person. | Weight:89  Height: 1.73  Gender: M |  | As expected, after I type the entry's correctly it’s give me the result without errors. |
| 06 | In this test we will use an input that is expected to be the output of an overweight female person. | Weight:66.9  Height: 1.58  Gender: F |  | As expected, after I type the entry's correctly it’s give me the result without errors. |
| 07 | In this test we will use an input that is expected to be the output of an obese weight male person. | Weight:53.9  Height: 1.61  Gender: M |  | As expected, after I type the entry's correctly it’s give me the result without errors. |
| 08 | In this test we will use an input that is expected to be the output of an obese weight female person. | Weight:53.9  Height: 1.61  Gender: F |  | As expected, after I type the entry's correctly it’s give me the result without errors. |

Program 3

Test plan to OnlineStore

This test plan outlines the testing activities for the OnlineStore program, which simulates a simple online store where users can browse products, select quantities, and view their total cost.

This plan covers functional testing, input validation, and basic boundary testing.

**Items to be Tested:**

**Product Display:** Verification that all four products are displayed with their correct prices.

**Product Selection:** Testing user input for product selection using valid product numbers and invalid numbers.

**Quantity Input:** Testing user input for quantity using positive values, zero, and negative values.

**Cost Calculation:** Verification that the cost of each product is calculated correctly based on quantity and price.

**Total Cost:** Verification that the total cost is correctly accumulated as items are added to the cart.

**Program Termination:** Testing that the program terminates gracefully when the user enters 'q' to quit.

**Entry and Exit Criteria:**

**Entry Criteria:**

Java code compiled successfully.

Test environment set up.

**Exit Criteria:**

All test cases executed.

All identified defects resolved and retested.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Description | Input | Output | Comments |
| 01 | Verification for all four products if they do not output any error. | Product number:1  Quantity:5  Expected output: 1245.0 |  | As I expected the program was executed without any errors, and displaying all messages and providing a right total cost. |
| 02 | Verification the keep buying, adding more products. | Product number:1,3  Quantity:5,2  Expected output:5145.0 |  | As I expected the program was executed without any errors, adding the right quantity and displaying all messages, providing the right total cost. |
| 03 | Verification the keep buying, adding more products. | Product number:1,3,2  Quantity:5,2,6  Expected output:8739.0 |  | As I expected the program was executed without any errors, adding the right quantity and displaying all messages, providing the right total cost. |
| 04 | Verification of the end buy and output message with total amount. | Q to quit  Expected output:  Display the thank you message with the total amount. |  | As I expected the program ended without any errors, displaying the quit thank you message, providing the right total cost. |

Program 4

Test plan to Election program

This test plan outlines the testing activities for the Election program, which simulates an election voting system. Users can enter candidate names, and the program tallies the votes and announces the winner.

This plan covers functional testing, input validation, and basic boundary testing. It does *not* cover error handling beyond what's implemented in the code.

**Items to be Tested:**

**Candidate Input:** Testing user input for candidate names, including valid names, empty strings, and special characters.

*Vote Counting:* Verification that votes are correctly added to the corresponding candidate's count.

*New Candidate Handling:* Testing how the program handles new candidate entries that are not already on the list.

*Program Termination:* Testing that the program terminates gracefully when the user enters '-1' or leaves the input empty.

*Empty Election:* Testing how the program handles an election with no votes cast.

*Tie Handling:* Testing how the program handles a scenario where multiple candidates have the same highest number of votes.

*Input Validation:* Testing that the program handles invalid inputs.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Description | Input | Output | Comments |
| 01 | Verification for main head and to the candidate's name input and counts votes. | Candidate name: Matheus  Votes:2  Output expected: “Matheus: 2 votes  And the winner with most votes is: Matheus with 2 votes!” |  | As I expected the program was executed correctly without any errors. |
| 02 | Verification for the candidates handling, how the programs work with new inputs. | Candidate name: Matheus  Tomas  Mason  Votes: 2,4,3  Output expected:  “And the results for the Election is:  Matheus: 2 votes  Tomas: 4 votes  Mason: 3 votes  And the winner with most votes is: Tomas with 4 votes!” |  | As I expected the program was executed correctly without any errors. |
| 03 | Verification for the program executed with no candidates and votes. | Empty space |  | As I expected the program was executed correctly displaying the “No votes.” message. |
| 04 | Verification for the program executed with candidates with same counts of votes. | Candidate name: Matheus  Tomas  Mason  Votes: 3,3,1 |  | I didn’t expect this output, because I have not programmed a logic to case if was a draw, so the program assume that the name what was input first is the winner. |
| 05 | Verification for the different types of inputs | Candidate name: 321312  212  3@d |  | As I expected the program have not a handling validation for name, just for numbers, the program assume that the name have no restrictions. |

Program 5

Test plan to Account program

This test plan outlines the testing activities for the Account class and Main program, which simulates a simple bank account with basic functionalities like deposit, withdrawal, and balance inquiry.

This plan covers functional testing, basic input validation, and error handling. It does *not* cover security testing or database interactions.

Items to be Tested:

Account Creation: Testing the creation of accounts with valid and invalid initial values (e.g., negative balance).

*Deposit Functionality:* Testing deposits with positive amounts and handling insufficient funds scenarios during deposits.

*Withdrawal Functionality:* Testing withdrawals with positive amounts, insufficient balance scenarios, and negative withdrawal amounts.

*Balance Inquiry:* Verification that the getBalance method returns the correct current balance.

*Monthly Interest Calculation:* Verification that the getMonthlyInterestRate method correctly calculates the monthly interest rate based on the annual rate.

*Date Handling:* Verification that the getDateCreated method returns the correct date the account was created.

*Account ID:* Testing the getId and setId methods.

Entry and Exit Criteria:

Entry Criteria:

Java code compiled successfully.

Test environment set up.

Exit Criteria:

All test cases executed.

All identified defects resolved and retested.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Description | Input | Output | Comments |
| 01 | Verification for the account date creation |  |  | As I expected the account was generated in the current date, with the account id,balance and monthly interest rate. |