Phase 3: Testing Strategies

The testing strategies we used were condition/decision coverage. Each test case involved all entry and exit points and also took every possible outcome. We made sure that each condition in a decision all affected the outcome in some way independently. For example, to check that we inputted the name correctly, we can try with both a valid and invalid input and confirm we get the correct output. The testing method we will follow is a white-box testing method. Each test will be conducted with a predefined expected outcome, calculated by the programmer.

1. User Data System
   1.1. Put in the right password for the user
      - Type: Security
      - Purpose: To make sure the user can access their account with a saved password
      - Method: Junit testing
      - Input: password of the user
      - Output: N/A

   1.2. Put in the wrong password for the user
      - Type: Security
      - Purpose: So that it does not allow any password to work
      - Method: Junit testing
      - Input: invalid password of the user
      - Output: N/A

   1.3. Check if the name is inputted
      - Type: Usability
      - Purpose: so that it will display a name for reference on the app
      - Method: Junit testing
      - Input: name of the user
      - Output: N/A

   1.4. Check if the name has invalid characters
      - Type: functionality
      - Purpose: makes sure there are valid characters so there are no errors
      - Method: Junit testing
      - Input: symbols/ numbers in name
      - Output: error

   1.5. Check if age is inputted
      - Type: Usability

- Purpose: so that it will display an age for reference on the app
- Method: Junit testing
- Input: no input
- Output: return an error

1.6. Check if age has invalid characters
- Type: functionality
- Purpose: makes sure there are valid characters so there are no errors
- Method: Junit testing
- Input: symbols/ numbers in name
- Output: return an error

1.7. Check if height is inputted
- Type: functionality
- Purpose: so that it will display a height for reference on the app
- Method: Junit testing
- Input: no input
- Output: return an error

1.8. Check if height has valid characters
- Type: : functionality
- Purpose: makes sure there are valid characters so there are no errors
- Method: Junit testing
- Input: symbols/ numbers in name
- Output: return an error

1.9. Check if weight is inputted
- Type: Usability
- Purpose: so there is a weight to reference for the user
- Method: Junit testing
- Input: no input
- Output: return an error

1.10. Check if the weight has valid characters
- Type: Usability
- Purpose: so that it will display an age for reference on the app
- Method: Junit testing
- Input: no input
- Output: return an error

1.11. Check if it allows selecting the same user
- Type: functionality/accessibility
- Purpose: so it doesn't allow refreshing of the page for no reason
- Method: Junit testing
- Input: Select the same name
- Output: none

1.12. Check if the user has the same name
- Type: functionality/accessibility
- Purpose: so there are not multiple users with the same name
- Method: Junit testing
- Input: 2 of the same names
- Output: none

1.13. Make sure the name is saved properly
- Type: functionality
- Purpose: so the name is saved after the program is closed
- Method: Junit testing
- Input: user name

1.14. Check if it saves user record
- Type: functionality
- Purpose: so it can save workouts for each user
- Method: Junit testing
- Input: user exercises

1.15. Check if load the last user loads user properly
- Type: functionality
- Purpose: so the user is saved for the next time the program is launched
- Method: Junit testing
- Output: user name

2. Exercise System

2.1. Load CSV file with all exercises
- Type: Functionality
- Purpose: To make sure all exercises are loaded correctly under normal conditions
- Method: Junit testing
- Input: CSV file with all exercises to load
- Output: all exercises loaded into the program

2.2. Load CSV file with all exercises with missing images
- Type: Functionality
- Purpose: All exercises are inputted expected if its missing and image
- Method: Junit testing
- Input: CSV file with all exercises to load
- Output: all exercises loaded into the program expected if its missing and image

2.3. Load CSV file with all exercises with missing Information
- Type: Functionality
- Purpose: Exercise import should not need all fields full
- Method: Junit testing
- Input: CSV file with all exercises to load, some missing fields
- Output: all exercises loaded into the program, with data they should have

2.4. Store CSV file with all exercises
- Type: Functionality
- Purpose: All exercises are inputted expected if its missing and image
- Method: Junit testing
- Input: CSV file with all exercises to load
- Output: all exercises loaded into the program expected if its missing and image

2.5. Get all exercises by type
- Type: Functionality
- Purpose: Exercises should be able to be filtered by type
- Method: Junit testing
- Input: Set of all exercises, type keyword
- Output: Filtered list of exercises of specific type

2.6. Iterate through exercise library
- Type: Functionality
- Purpose: User should be able to browse exercises in library
- Method: Junit testing
- Input: Set of all exercises
- Output: a formatted list of exercises to choose from

2.7. Check for existing exercise by name
- Type: Functionality
- Purpose: User should be able to search for an exercise by name
- Method: Junit testing
- Input: Set of all exercises, search keyword
- Output: A set of possible results for search keyword

2.8. Check for non existing exercise by name
- Type: Functionality
- Purpose: User should be able to search for an exercise by name; sometimes that exercise does not exist
- Method: Junit testing
- Input: Set of all exercises, search keyword
- Output: A message specifying no results found.

2.9. Check for existing exercise by ID
- Type: Functionality
- Purpose: User should be able to search for an exercise by ID.
- Method: Junit testing
- Input: Set of all exercises, ID keyword.
- Output: Exercise data for matching ID exercise object.

2.10. Check for non-existing exercise by ID
- Type: Functionality
- Purpose: User should be able to search for an exercise by ID. System should be able to accommodate a non existing ID search.
- Method: Junit testing
- Input: Set of all exercises, ID keyword.
- Output: No exercise found for this ID.

2.11. Create a new exercise
- Type: Functionality
- Purpose: User should be able to create a custom exercise.
- Method: Junit testing
- Input: User interacts with interface with all possible fields.
- Output: Created exercise data, added to the set of all exercises and tagged as a custom exercise with a user ID.

2.12. Create exercise with an existing name
- Type: Functionality
- Purpose: System should be able to distinguish between exercises with the same name, provided they use different equipment.
- Method: Junit testing
- Input: User creates a new exercise with the same name as an existing exercise; new exercise has different equipment than its counterpart.
- Output: New exercise should be added to the exercise set; searching for ‘name’ exercise should yield both results.

2.13. Create a new exercise with missing information
- Type: Functionality
- Purpose: Exercise creation should not need all fields full
- Method: Junit testing
- Input: User/test creates a new exercise with missing fields
- Output: Displaying exercise should not show the blank fields.

2.14. Check calorie counter calculate
- Type: Functionality
- Purpose: App should use user data and workout history to calculate calorie count from certain exercises.
- Method: Junit testing
- Input: User data, some exercises that have been completed.
- Output: An estimated calorie count.

2.15. Save history of workouts
- Type: Functionality
- Purpose: Each user should have a history of completed exercises and timestamps on each entry.
- Method: Junit testing
- Input: User marks exercises as completed
- Output: User should be able to view a formatted list of exercises

3. Schedule system

3.1. Save weekly schedule
- Type: Functionality
- Purpose: saves weekly schedule to serialized file
- Method: Junit testing
- Input: user inputs workouts in schedule
- Output: they can view the week

3.2. Load weekly schedule
- Type: Functionality
- Purpose: Loads weekly schedule from serialized file
- Method: Junit testing
- Input: serialized file of the weekly schedule
- Output: Loads schedule into program for user to select

3.3. Create new Schedule
- Type: Functionality
- Purpose: Creates a new weekly schedule that lets a user store workouts
- Method: Junit testing
- Output: A new schedule for the week

3.4. Add schedule day name
- Type: Accessibility
- Purpose: Adds the name of the day to the schedule.
- Method: Junit testing
- Input: Schedule day name

3.5. Remove schedule day name
- Type: Accessibility
- Purpose: Removes schedule day name from the schedule
- Method: Junit testing
- Input: the day removed

3.6. Add workout to each day
- Type: Accessibility
- Purpose: User can add workouts to a date on the weekly schedule
- Method: Junit testing
- Input: user inputs workouts in schedule
- Output: user can view the week

3.7. Remove workout from each day
- Type: Accessibility
- Purpose: removes workout from a day
- Method: Junit testing
- Input: user inputs day to remove

3.8. Add rest day
- Type: Accessibility
- Purpose: Adds a rest day to a date on the weekly schedule
- Method: Junit testing
- Input: user inputs rest day in schedule
- Output: user can view the week

3.9. Get Schedule history
- Type: Functionality
- Purpose: Adds a rest day to a date on the weekly schedule
- Method: Junit testing
- Input: user inputs rest day in schedule
- Output: user can view the week

3.10. Reset schedule
- Type: Accessibility
- Purpose: Adds a rest day to a date on the weekly schedule
- Method: Junit testing
- Input: user inputs rest day in schedule
- Output: user can view the week

3.11. Save workout progress for date
- Type: Functionality
- Purpose: User data is saved when a workout is completed by user
- Method: Junit testing
- Input: workout details to save
- Output: file of saved data

3.12. Load workout progress on date
- Type: Functionality
- Purpose: User data is loaded for user to view previous information
- Method: Junit testing
- Input: file of saved data
- Output: previous workout information

3.13. Show the previous record for workout
- Type: Accessibility
- Purpose: shows the record data including reps from prior workouts
- Method: Junit testing
- Input: Workout object
- Output: Display workout history object corresponding to record.

3.14. Get time elapsed for current workout
- Type: Accessibility
- Purpose: Show time taken to complete workout session
- Method: Junit testing
- Output: Time in minutes for how long session has taken

3.15. Show previous times for the current workout
- Type: Accessibility
- Purpose: Shows the previous time taken to complete the same workout on prior weeks
- Method: Junit testing
- Input: workout that was completed
- Output: Previous time taken to complete workout

The main tool we will be using for testing is JUnit 5 for testing our program. This will allow us to use decision/ condition coverage when testing and test all inputs and outputs to be correct. This way we can test every aspect of our program continuously as changes are being made and verify the output is correct.