# Fitness tracker documentation

Anhelina Romanchuk, SD 11

## Clean Code Practices (3 Examples)

1. Descriptive Naming

**- Example:**

In the `FitnessTracker` class, methods and variables are named meaningfully to describe their purpose. For example, `addWorkout()` clearly indicates it adds a workout, and `calculateTotalProgress()` describes its purpose of calculating progress.

**Code snippet:**

A computer screen shot of text

AI-generated content may be incorrect.

**Why it's clean:** Clear, descriptive method names make it easy to understand the function without needing to read through the entire implementation.

2. Proper Use of Access Modifiers:

**- Example:**

In the `Goal` class, the `targetCalories` and `targetMinutes` are defined as `static` because they are shared across instances, but the class allows controlled access through public methods for getters and setters.

**Code snippet:**

A computer screen with text

AI-generated content may be incorrect.

**Why it's clean:** The use of static variables and encapsulation ensures controlled access to critical data.

3. Consistent Formatting:

**- Example:**

Throughout the classes, the formatting of code follows consistent indentation, spacing, and line breaks for readability. This can be observed in the `Goal` class constructor and methods.

**Code snippet:**

A black background with white text

AI-generated content may be incorrect.

**Why it's clean:** Consistent indentation helps make the code easier to read, understand, and debug.

## Project Overview:

**- Purpose:**

The project is a **Fitness Tracker** application that helps users track their workouts and progress towards fitness goals. It calculates calories burned, workout duration, and displays progress toward the set goals.

**- How It Works:**

1. The user sets a fitness goal with target calories burned and workout duration (in minutes).

2. The user can add workouts (e.g., running, cycling) and the app tracks the calories burned and workout duration.

3. The app will track progress and notify the user when they reach their goals. The user can then update their goals.

**- Main Components:**

- Workout Class: Represents individual workouts (type, duration, calories burned).

- Goal Class: Represents the user's fitness goals (target calories and minutes).

- FitnessTracker Class: Handles adding workouts, calculating progress, and checking goal completion.

- Main Class: Contains the user interface (CLI), where the user interacts with the program.

## Test Cases Explanation:

**- Test 1: `testAddWorkout()`**

- Purpose: To test if the `addWorkout()` method correctly adds a new workout to the list and updates the total progress.

- Input: Adds a running workout with 800 calories and 80 minutes.

- Expected Output: The workout list should have 1 workout, with progress showing 800 calories and 80 minutes.

**Code snippet:**

A computer screen shot of a program

AI-generated content may be incorrect.

**- Test 2: `testCalculateTotalProgress()`**

- Purpose: To ensure that `calculateTotalProgress()` returns correct totals for calories burned and workout duration after multiple workouts are added.

- Input: Adds a running workout (800 calories) and cycling workout (400 calories).

- Expected Output: The total calories should be 1200, and the total time should be 160 minutes.

**Code snippet:**

A computer screen with text and numbers

AI-generated content may be incorrect.

**- Test 3: `testGoalCompletion()`**

- Purpose: To verify that goal completion logic correctly congratulates the user once they reach their target calories and minutes.

- Input: Simulates workouts and checks if goals are met for both calories and minutes.

- Expected Output: The congratulatory flags for calories and minutes are set to `false` after goals are completed, indicating the user can set new goals.

**Code snippet:**

A computer screen shot of a program

AI-generated content may be incorrect.

## Dependencies:

**- JUnit:**

- Purpose: Used for unit testing.

- How it's used: Provides annotations (`@Test`, `@BeforeEach`) to structure and run tests.

**- Scanner:**

- Purpose: Used for reading user input from the command line.

## **Challenges Faced:**

**1. GitHub Actions Configuration:**

Configuring GitHub Actions to automatically run tests on every Pull Request was a challenging task. While I was able to set up the workflow and include the necessary configurations for testing, I encountered some difficulties in ensuring everything worked as expected. Specifically, I was unsure whether the tests were being triggered correctly and whether the configuration was set up to run the tests seamlessly with each pull request.

Although I followed the necessary steps to set up the GitHub Actions workflow, I am not entirely confident that it functions as intended. There were some uncertainties about the environment configuration and potential issues with the test execution, and while the workflow runs, it’s unclear whether all tests pass successfully without configuration-related problems.