READ this ENTIRE specification sheet before writing any code or asking for clarifications. These specifications will be discussed during the onsite class sessions on Nov 15, and during the Section ZZZ online session on Nov 16.

Overview

Using Java, write a Lifestyle Tracker application. This system comes in handy with the upcoming Christmas Season (with lots of eating), or even in planning some healthy resolutions for the New Year.

The system should be able to record several kinds of food and the calories gained from eating one serving of each. It should also be able to record several kinds of activities and the calories burned from performing each activity for an hour.

After creating a list of available foods and activities, the system proceeds to track the user's diet and activities for the day.

The system should then be able to print a report displaying the different foods eaten and activities performed for the day, the net calories gained or lost for the day, and based on that, the amount of weight that the user is expected to gain or lose in specific time frames.

Input

The system accepts five (5) commands, described below. Refer to the sample input.

- **`Food <name> <calories>`**
  Record that the food named `<name>` contains `<calories>` calories per serving.
- **`Activity <name> <calories>`**
  Record that performing the activity named `<name>` burns `<calories>` calories per hour.
- **`Eat <name> <servings>`**
  Record that the user ate `<servings>` servings of the food named `<name>`.
- **`Perform <name> <hours>`**
  Record that the user performed the activity named `<name>` for `<hours>` hours.
- **`Report`**
  Print the report (refer to the sample output).

Specifications

1. Create a folder named **Final-Section-Surname-GivenName-IDNumber**. This is your submission folder.
   - We encourage students to make use of plain text editors (like Notepad++) over IDEs for now. As beginners, it is important that you are encouraged to type commands from scratch and to analyze your code without relying on tools other than your own ability to pay attention to details.

2. In that folder, create the following classes to implement your Lifestyle Tracker application. A complete description of each class can be found in subsequent specification items.
   - **Food.java** – represents a single food item with a name and the corresponding calories of one serving
   - **Activity.java** – represents a single activity with a name and the corresponding calories burned in one hour
   - **LifestyleTracker.java** – responsible for recording food eaten and activities performed based on the aggregated data on food and activities, provides feedback messages and reports based on the recorded food and activities
   - **TrackerConsole.java** – provides the user with a console (terminal) interface through a main method that accepts input from the user through a Scanner object and calls methods on an associated LifestyleTracker object

3. **Food.java** represents a single type of Food. An instance of Food will have its own name and calorie value for one serving of the Food.
   - **`public Food( String f, double c )`**
     - initializes the instance fields with the values provided in the parameters
     - **`f`** - the name of the Food
     - **`c`** - the calorie value of the Food

- **`public String getFoodName()`**
  - returns the name of the Food
- **`public double getFoodCalories()`**
  - returns the calorie value of one serving of the Food
- **`public void updateCalories( double c )`**
  - enables the Food to update the calorie value
  - **`c`** - the new calorie value of one serving of the Food

4. **Activity.java** represents a single type of Activity. An instance of Activity will have its own name and calorie value for one hour of performing the activity.
   - **`public Activity( String n, double c )`**
     - initializes the instance fields with the values provided in the parameters
     - **`n`** - the name of the Activity
     - **`c`** - the calorie value of the Activity
   - **`public String getActivityName()`**
     - returns the name of the Activity
   - **`public double getActivityCalories()`**
     - returns the calorie value burned when performing the Activity for one hour
   - **`public void updateCalories( double c )`**
     - enables the Activity to update the calorie value
     - **`c`** - the new calorie value burned when performing the Activity for one hour

5. **LifestyleTracker.java** is responsible for recording food eaten and activities performed based on the aggregated data on food and activities. It provides feedback messages upon the creation of each record. It provides a report based on the recorded food and activities. The LifestyleTracker may use several ArrayLists to store collections of Food, Activity, and String objects as deemed necessary. The LifestyleTracker may use several variables to keep track of total calories consumed and burned. The LifestyleTracker may implement private methods to assist in searching through the collections of Food and Activity objects for items with matching names.
   NOTE: A week denotes 7 days. A month denotes 30 days. 1 kcal = 0.00012959782 kg
   - **`public LifestyleTracker()`**
     - initializes the instance fields
   - **`public String addFood( String n, double c )`**
     - creates a Food object, given a name and a calorie value
     - The new Food object is added to the collection of Food objects.
     - If a Food object with the same name already exists in the collection of Food objects, the calorie value of the existing Food object is updated with the provided calorie value.
     - A feedback message is returned for printing to the console or display on a GUI. Refer to the sample output.
     - **`n`** - the name of the Food
     - **`c`** - the calorie value of one serving of the Food
   - **`public String addActivity( String n, double c )`**
     - creates an Activity object, given a name and a calorie value
     - The new Activity object is added to the collection of Activity objects.
     - If an Activity object with the same name already exists in the collection of Activity objects, the calorie value of the existing Activity object is updated with the provided calorie value.
     - A feedback message is returned for printing to the console or display on a GUI.
     - **`n`** - the name of the Activity
     - **`c`** - the calorie value of performing the Activity for one hour
   - **`public String eat( String foodName, double servings )`**
     - records the Food eaten by the user, given the name of the Food and the number of servings consumed, provided that the identified Food object exists in the LifestyleTracker's collection of Food objects
     - A record of the consumed Food is added to the records of the LifestyleTracker.
     - A feedback message is returned for printing to the console or display on a GUI. Refer to the sample output.
     - If the Food object does not exist, the corresponding feedback message is returned.

    - A negative value of the number of servings is not allowed. The corresponding feedback message is returned.
    - **`foodName`** - the name of the Food
    - **`servings`** - the number of servings of Food eaten
- **`public String perform( String actName, double hours )`**
    - records the Activity done by the user, given the name of the Activity and the number of hours that the activity was performed, provided that the identified Activity object exists in the LifestyleTracker's collection of Activity objects
    - A record of the performed Activity is added to the records of the LifestyleTracker.
    - A feedback message is returned for printing to the console or display on a GUI. Refer to the sample output.
    - If the Activity object does not exist, the corresponding feedback message is returned.
    - A negative value of the number of hours is not allowed. The corresponding feedback message is returned.
    - **`actName`** - the name of the Activity
    - **`hours`** - the number of hours the Activity was performed
- **`public String report()`**
    - displays the recorded items (Refer to the sample output.)
    - It displays all the records of the Food eaten, as well as the total calories consumed.
    - It displays all the records of the Activities done, as well as the total calories burned.
    - It provides a prediction of weight loss or weight gain based on the net calories.

6. **TrackerConsole.java** contains the **`main()`** method, which creates a **LifestyleTracker** instance and repeatedly accepts input from the user, through a **Scanner** object.
   It accepts one command line argument to indicate the name of the user:
   `java TrackerConsole <name>`
   The **`name`** is used in the first line of output.
   The program accepts five commands:
   - Food - calls **`addFood()`** on the **LifestyleTracker** object, given the values that follow
   - Activity - calls **`addActivity()`** on the **LifestyleTracker** object, given the values that follow
   - Eat - calls **`eat()`** on the **LifestyleTracker** object, given the values that follow
   - Perform - calls **`perform()`** on the **LifestyleTracker** object, given the values that follow
   - Report - calls **`report()`** on the **LifestyleTracker** object

7. Each of your Java files should contain class header information enclosed in javadoc-formatted comments, containing a description of your program (in your own words), your full name, your ID number, and the date you created your program. This is followed by a comment block containing the certification of authorship.

8. In your submission folder, include a ***properly accomplished*** Individual Certificate of Authorship with the following details:
   - Title of Submission: CSCI 21 Final Project - Lifestyle Tracker
   - Type of Submission: Program
   - Cite your sources in the source citation text input area. This area automatically adjusts to accommodate the length of the text provided.
   - Fill in the Student and Course Information.
   - File name: **LifestyleTracker-IDNumber.pdf**

9. Archive your folder. Ensure that it is named according to the convention:
   Final-Section-Surname-GivenName-IDNumber.zip
   The archived folder **only** contains ***at least four*** (4) and ***at most six*** (6) **.java** files and one (1) **.pdf** file.

10. Submit it through Canvas by **Monday, 05 December 2022, 23:59.**

Grading

The highest grade that a student can get by fulfilling **all** of the requirements stated above is **80 points** (high C+). To receive a higher grade, the student may implement any of the additional options described herein. Projects will be tested on the console / terminal.

Optional Add-Ons

Each add-on may be implemented independently. Add-ons implemented on the **TrackerConsole** must **also** be implemented in the GUI, **if** the GUI add-on is implemented. ***Students must cite all external sources that are used as references for implementing any add-on.***

Add-On 1: Edit a Record Food Eaten and a Record of Activity Performed (+5 max)

Add the function to allow the user to periodically view all of the recorded food eaten or activities performed (so far), with corresponding record numbers (index), then select a record number and update the data of that record.

Add-On 2: Delete a Record of Food Eaten and a Record of Activity Performed (+5 max)

Add the function to allow the user to periodically view all of the recorded food eaten or activities performed (so far), with corresponding record numbers (index), then select a record number to delete.

Add-On 3: Create Non-Existent Food Item / Activity, when Recorded (+5 max)

Instead of providing the feedback message "`The specified food does not exist.`" when creating a record of eating a Food item that does not exist in the collection, or "`The specified activity does not exist.`" when creating a record of performing an Activity that does not exist in the collection, provide the user with the option to create the corresponding Food or Activity item and then proceed with the record.

Add-On 4: GUI (+20 max)

Implement a graphical user interface (GUI) using a JFrame object that performs the same functions as the TrackerConsole. This is **not** a replacement for the TrackerConsole, but an additional way to use the program.

- **TrackerGUI.java** – provides the user with a graphical user interface and calls methods on an associated LifestyleTracker object. Additional methods may be created to aid in creating the GUI components and handling events. The TrackerGUI **must** enable the user to do the following:
  - Add a Food object with a given name and corresponding calorie value
  - Add an Activity object with a given name and corresponding calorie value
  - Update the calorie value of a specific Food item
  - Update the calorie value of a specific Activity item
  - Record food eaten
  - Record activities performed
  - Display the report
  - all other add-on functions implemented
- **TrackerApp.java** – creates an instance of a TrackerGUI in the main method.

Add-On ?: Mystery Mode (+?)

A student may also receive higher marks for adding unique or rare features, **without deviating** from the required specifications.

Project Defense (Code Review)

Project defenses start on **05 December 2022**, which is the day of the deadline. Sign-ups will be announced on Canvas as the deadline approaches. Schedules for online students (Section ZZZ) will be made available as well.

Students who submit and defend **before** the deadline will receive a bonus of 5 points. There will be limited slots for this.

Grades for projects that are submitted **beyond** the deadline (Dec 05) ***and*** defended (by Dec 09) will be **capped at 75**.

Grades for projects that are submitted beyond 5:00 PM on the last day of defenses, **Friday, 09 December 2022**, will be **capped at 65**, with the defense being forfeited. **Projects will not be accepted after Saturday, 10 December 2022.**

A student who does not appear within the ***first five minutes*** of their selected time slot forfeits their project defense. There is no make-up defense, except in extenuating circumstances.

Important Notes

- Carefully and attentively read all specifications.
- Spell all words correctly, according to the sample output.
- Be precise with spaces that appear, according to the sample output.
- Do not add unnecessary prompts that are not described in the specifications.
- Do not add extra spaces to the end of lines. Each line must end with a non-space character.
- Do not add extra lines to the end of the output. Follow the format of the sample output exactly.
- Follow file naming conventions.
- Follow submission procedures.
- Do not submit any excess files.