EN.500.132 Bootcamp: Java, Intersession 2020

Project D: Workout Tracker

Dates

- Warm-up due by 11pm on Thursday 1/23
- Project implementation due by 11pm on Friday 1/24

Topics covered

Object-oriented programming, algorithms, polymorphism.

Purpose & background

The purpose of this project is to demonstrate skills with object-oriented programming, including writing classes, constructing arrays of objects, and utilizing polymorphism through inheritance.

General description

This project requires you to design and implement a Client class representing information about members of a gym, and a Workout class that stores information about specific exercise sessions. You will then create subclasses for specific types of Workouts, including Bicycle and Yoga. Finally, you'll create a main method which allows a gym owner to track gym member workouts (instances of your classes) over time.

You must break down the program into classes and subclasses and utilize objects and polymorphism. Your project is required to have the following instantiable classes:

- Workout
- Yoga, a subclass of Workout
- Bicycle, a subclass of Workout
- Client

Specifications for required Classes

Class Workout:

- Stores client, a user identification value representing the particular person completing the workout (String)
- Stores isCardio, a representation of the workout type, either cardio or strength (boolean)
    - isCardio should default to true
- Stores the duration of the workout in minutes (int)
- Has a method named getCaloriesBurned which calculates and returns the estimated calories burned (double)
    - For a cardio workout, the estimated calories burned are 8 per minute; for a strength workout, the estimated calories burned are 5 per minute
- Has an overridden toString method that shows Workout details in the format *[USER completed TYPE workout: DUR mins, CALS cals]*, where USER is replaced by the client's identification String, TYPE is replaced by either cardio or strength, DUR is

replaced by the workout duration, and CALS is replaced by the number of calories burned. For example:

```
[achen09 completed cardio workout: 25 mins, 200 cals]
```

- Has a compareTo method which compares two workouts and returns an int. Workout A is considered to "come before" Workout B if the number of calories burned during Workout A is smaller than the number of calories burned during Workout B. Return values are as follows:
    - When A comes before B, A.compareTo(B) returns a negative integer.
    - When B comes before A, A.compareTo(B) returns a positive integer.
    - When neither A nor B comes before the other, A.compareTo(B) returns 0
- A class user (such as a main program) should be permitted to check the duration and/or type (i.e. cardio or strength) of an existing Workout object.
- No class user (such as a main program) should be permitted to set the calories directly, but a Workout object's duration and workout type may be adjusted by the class user after the object is created.
- Stores a class-wide (shared) defaultDuration, initially set to 60 minutes, and a method by which the user can modify this value, so that default Workout objects created after the modification have the new default value.

Warm-up for Project D - submit via Blackboard by 11pm on Thursday 1/23.
Complete the Workout class as described above. Your class should compile and run with the provided file ProjectDWarmupDriver.java, providing expected outputs as indicated in a comment in the provided file. Do not modify ProjectDWarmupDriver.java; instead, make sure that your Workout class has methods named appropriately so that it compiles with the driver and executes correctly. Your submitted code must be checkstyle-compliant. Zip your completed Workout.java into a file named PDWarmup-jhed.zip (using your jhed) and submit.

Class Bicycle:

- Inherits from class Workout
- Stores a resistance level (int) that can range from 1 to 10, with a default value of 1.
- Has a method which reports the travelled distance in miles during the workout, calculated as: *distance = duration / (8 * resistance)*
- Has an overridden getCaloriesBurned method which calculates and reports the estimated calories burned, calculated as: *calories = duration * (resistance / 3) * 2*
- Has an overridden toString method that shows bike workout details in the format: *[USER completed bicycle workout: DUR mins, CALS cals, resistance RES, DIST mi]*, where USER is replaced by the client's identification String, DUR is replaced by the workout duration, CALS is replaced by the number of calories burned, RES is replaced by the resistance setting, and DIST is replaced by the calculated distance traveled. For example:

```
[achen09 completed bicycle workout: 15 mins, 49.99 cals,
resistance 5, 0.375 mi]
```

Class Yoga:

- Inherits from class Workout
- Stores the workout level (int), where 1 signifies "beginner" (the default), 2 signifies "intermediate", and 3 signifies "advanced".
- Has an overridden getCaloriesBurned method which calculates and reports the estimated calories burned, calculated as: *calories = duration * 2.1 * workout level*
  This method does NOT round fractional parts of the result up to the next higher integer.
- Has an overridden toString method that shows yoga workout details in the format: *[USER completed yoga workout: DUR mins, LEV level, CALS cals]*, where USER is replaced by the client's identification String, DUR is replaced by the workout duration, LEV is replaced by the *word* representing the level of the yoga workout, and CALS is replaced by the number of calories burned, RES is replaced by the resistance setting, and DIST is replaced by the calculated distance traveled. For example:
  `[achen09 completed yoga workout: 40 mins, intermediate level, 168 cals]`

Class Client:

- Stores an identification number for each client (auto-generated by the Client class to be unique and sequential starting at 100).
- Stores the client's name as a single String in the format *Last, First*
- Stores a list of workouts the client has completed. You may not use Java's ArrayList class in your solution. You must assume a maximum number of workouts per member of 100.
- Contains an overridden toString(method that shows a member's identification number and name in the format *[ID_NUM, NAME]*

You can assume a gym capacity of 500 clients.

It is likely that you will need to add other methods to the classes above in order to support the main program operations described below.

Main Program Interface

When the main program (ProjectD.java) launches, the following menu is displayed on the screen for the user:

0) Quit the program
1) Add clients from a plain text file
Enter file name: *[Get user input for the name of the input file]*
2) Add workouts for clients from a plain text file
Enter file name: *[Get user input for the name of the input file]*
3) Display all client identification strings and their names, along with total number of workouts completed, and total numbers of bicycle workouts and yoga workouts
4) Display details of all workouts completed by a given client
Enter client identification string: *[Get user input for the name of the client]*
5) Display total calories burned by a given client
Enter client identification string: *[Get user input for the name of the client]*
6) Display average calories burned per workout for a given client
Enter client identification string: *[Get user input for the name of the client]*
7) Display all workouts in the system, sorted by calories burned, with the lowest-calorie-burning workout listed first
8) Display a list of client identification strings containing only those clients who have completed at least one bicycle workout and at least one yoga workout

Enter the number of your choice ->

The program keeps prompting for choice numbers and executing them until choice 0 “Quit the program” is selected. A sample program execution is posted here for your review: sample.txt
Note that this file relies on members.txt, workouts.txt and workouts2.txt, which are linked below.

Note: You may assume that the format of the files containing members and workout details are always valid (i.e., no invalid values, no malformed files etc). Assume there will be no empty or malformed lines at the beginning, middle, or end of any of the text files.

For choice “1) Add clients from a plain text file”, a member text file has one member per line in the format:

Last, First

See the posted sample file members.txt for examples.

Also, for “2) Add workouts for clients from a plain text file”, a workout text file has one workout per line in the format:

memberIDNumber, workoutType, duration, other details as needed

See the posted sample files workouts.txt and workouts2.txt for examples. Notably, you only need to handle workout types of **yoga**, **bike**, and **other**, as shown in the example files.

Reminders/Suggestions

- Developing a program incrementally and testing as you go generally requires less time that trying to write everything in one shot and testing at the end.
- The jGRASP debugger can be helpful when tracking down bugs in your code.
- Testing with a diverse set of inputs is the best way to ensure that your code is correct. Carefully hand-trace with small inputs so you know what output to expect.

Implementation Requirements

- **You will receive a 0 for any code that does not compile.**
- Your solution files must be named as indicated above.
- Each program file must have a class comment that explains briefly the purpose of the program, the author's name, JHED, and the date.
- Each program must be fully checkstyle-compliant using our course required check112.xml configuration file. You must also use good style with respect to variable names, method names, etc.
- Submit via Blackboard a zipped file called PD-**jhed**.zip (using your own jhed login name in place of **jhed**) that contains the source files mentioned above. Do not include .class files or textual data files.
- Your solution should contain no "global" variables. Use method parameters and return values instead!
- You must not use Java ArrayLists or Vectors (or any more advanced Collections) in any form to solve this problem. If you're not sure if something is allowed, please ask before using it!

Grading (remember that code that doesn't compile gets a 0 grade)

- 2 pts warm-up
- 18 pts implementation
    - 16 pts functionality
    - 2 pts style & submission