CS 211, ALL SECTIONS

PROJECT 3

DUE SUNDAY, OCTOBER 25 AT MIDNIGHT

The objective of this project is to implement a mini personal fitness app. This project uses deeper class hierarchies than previously, and introduces the use of exceptions, enum datatypes as well as ArrayList. It also demonestrates the use of ploymorphism in problem sloving

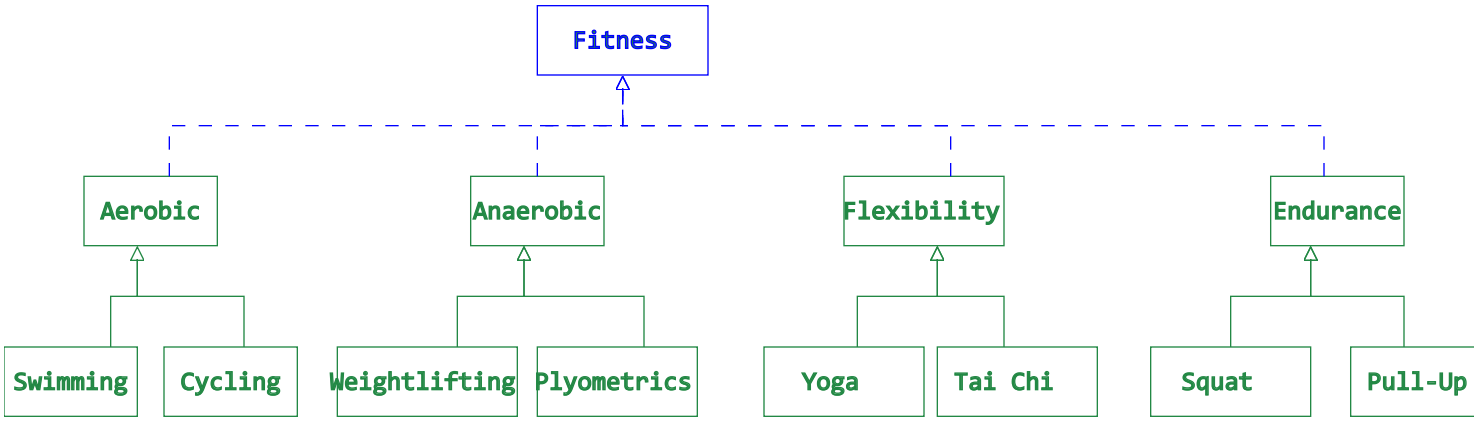
OVERVIEW:

1. Implement the basics of `Fitness` and types of Fitnesses: `Aerobic`, `Anaerobic`,`Flexibility`, and `Endurance` which are derived from `Fitness`.
2. Implement specific Fitness types such as `Swimming`, `Cycling`, `Yoga`, `PullUp`, etc.
3. Define the class `Profile`, which stores information about users.
4. Implement the class `DailyExercise`, which gives the daily exercise routines to the user.
5. Implement the class `WeeklyExercise`, which gives a weekly workout plan to the user based on personal weight loss target.
6. Implement a custom exception class `TargetWeightException`
7. Define different `enum` data types to be used throughout the project.
8. Download and use the tester module to ensure that your program is correct.
9. Prepare the assignment for submission and submit it.

According to the United States Department of Health and Human Services, physical fitness is defined as "a set of attributes that people have or achieve that relates to the ability to perform physical activity" . There are four different types of fitnesses that can be incorporated into your exercise routine: aerobic, anaerobic, flexibility, and endurance fitnesses. Each of these fitness types target different body parts. There are different fitness exercises/activities that can be categorized in one or more of the fitness types. Aerobic activities condition your heart and lungs. Anaerobic exercise is defined as short duration, high intensity exercise lasting anywhere from merely seconds up to around two minutes. Flexibility is a result of physical activity. Flexibility comes from stretching. Endurance exercises exert maximum force in short intervals of time, with the goal of increasing power (speed-strength).

For this project, we will develop a simple personal fitness app. The fitness app will provide users the functionality to create and modify daily and weekly fitness plans based on personal goals.

This project will use slightly deeper class hierarchies than we have in the past. The following diagram depicts the class hierarchies we will implement in this project.



We will introduce the use of exceptions. Exceptions are mechanisms used by Java (and other languages) to report errors and illegal conditions. By using exceptions, we can streamline the task of error checking and avoid the need to use the return type of a method to pass back error codes. In this project, enum data types are also introduced.

RULES

1. This project is an individual effort; the Honor Code applies
2. You may import Java built-in libraries (java.util.*).
3. All data fields should be declared `private` or `protected`
4. You may add your own helper methods or data fields, but they should be `private` or `protected`. You may add additional constructors which are public.
5. Use of `@Override` tags on overridden methods is not required but is strongly recommended.
6. When commenting code, use JavaDoc-style comments. Include `@param` and/or `@return` tags to explain what is passed in or returned wherever it isn't obvious. Any other JavaDoc tags are optional.
7. There is a separate question for Honors section students.

FITNESS TASK:

`public interface Fitness`

(*5 pts*) This will be used as a starting point for deriving any specific Fitness type. Every fitness exercise type has one or more muscle group it affects. Therefore, a `Fitness` has the following abstract method (Note that all methods in an interface are `abstract` by default):

- `public Muscle [ ] muscleTargeted()` - A method that returns the muscle that is going to be affected by the fitness. Note that the return type of the method is Muscle. A human body has a finite number of muscle parts that can be affected by fitness exercises. Define an `enum` datatype called `Muscle` with the following member values `Abs, Back, Biceps, Chest, Arms, Glutes, Shoulders, Triceps, Legs, Cardio`
- `public double calorieLoss(Intensity intensity, double weight, int duration)` - returns the total amount of calorie burnt by the exercise for the duration number of minutes for a person with the given weight. Intensity can be `HIGH, MEDIUM, LOW`. Note that Intensity is a good candidate to be defined as `enum`.
- `public String description()` - a method that returns a short decription of the fitness type.

AEROBIC TASK:

(*10pts*) Aerobic means "with oxygen." The purpose of aerobic conditioning is to increase the amount of oxygen that is delivered to your muscles, which allows them to work longer. `Aerobic` is a `Fitness`. However, we cannot give the actual implementation for the methods `muscleTargeted() and calorieLoss()` as we don't know the actual aerobic exercise. The `descripton()` method returns the string `Aerobic means "with oxygen."`. Note that `Aerobic` is a good candidate to be `abstract` class.

`public class Swimming`, which is an `Aerobic`

This class represents an actual `Aerobic` exercise, i.e., it extends `Aerobic` class, that a user can do to burn calories. The calculation of how much calorie will be burnt relies on a key value known as a MET, which stands for metabolic equivalent. One "MET" is "roughly equivalent to the energy cost of sitting quietly," and can be considered 1 kcal/kg/hour. Since sitting quietly is one MET, a 70 kg person would burn 70 calories (kcal) if they sat quietly for an hour. If an activity's MET value was two, that same person would burn 140 calories in an hour. The MET values of some of the exercises that we consider of this project are displayed in the following table. The MET value multiplied by weight in kilograms tells you calories burned per hour (MET*weight in kg=calories/hour). Note that the values shown in the table are per hour, but the inputs will typically be per minute, so be sure to take that into account. There are different types of swimming: Butterflystroke, Freestyle, and Breaststroke. These different types of swimming activities affect different muscles.

- Butterflystroke: `Abs, Back, shoulders, Biceps, Triceps`
- Breaststsroke: `Glutes, Cardio`
- Freestyle: `Arms, Legs, Cardio`

Define a class `Swimming`. The class must include the following:

- `public Swimming (SwimmingType type)` - defines the constructor for the class. You need to define an `enum` datatype called `SwimmingType` with members Butterflystroke,Breaststroke,Freestyle
- `public Swimming ()` - a default constructor for the class that initilizes SwimmingType to Freestyle.
- `public void setSwimmingType(SwimmingType type)` - A setter for the swimmingType.
- `public SwimmingType getSwimmingType()` - A getter for the swimmingType.
- `@Override public String description()` - returns the name of the class.

`public class Cycling`, which is an `Aerobic`

(*10pts*) This class represents an actual `Aerobic` exercise, i.e., it extends `Aerobic` class, that a user can do to burn calories. Cycling affects muscles: `Glutes, Cardio, Legs`. Define a class `Cycling`. The class also has:

- `@Override public String description()` - returns the name of the class.

Exercise	HIGH	MEDIUM	LOW
Swimming	10.0	8.3	6.0
Cycling	14.0	8.5	4.0
Yoga	4.0	3.0	2.0
Weightlifting	6.0	5.0	3.5
Plyometrics	7.4	4.8	2.5
Tai Chi	5.0	3.0	1.5
Squat	7.0	5.0	2.5
Pull-Up	7.5	6.0	4.8

ANAEROBIC TASK:

(*10pts*) Examples of anaerobic exercise include heavy weight training, sprinting (running or cycling) and jumping. Basically, any exercise that consists of short exertion, high-intensity movement is an anaerobic exercise. Heavy weight training is an excellent way to build strength and muscle mass. Like `Aerobic` class, `Anaerobic` is a `Fitness` and we cannot give the actual implementation for the methods `muscleTargeted() and calorieLoss()` as we don't know the actual Anaerobic exercise. The `descripton()` method returns the string `Anaerobic means "without oxygen."`. Note that `Anaerobic` is a good candidate to be `abstract` class.

Define classes `WeightLifting` and `Plyometrics`, which are `Anaerobic` fitness exercises. Use the following table to define the classes. The classes also have:

- `@Override public String description()` - returns the name of the respective classes.

Exercise Type	Muscle affected
Yoga	Triceps, Biceps
Weightlifting	Shoulders, Legs, Arms, Triceps
Plyometrics	Abs, Legs, Glutes
Tai Chi	Arms, Legs
Squat	Glutes, Abs, Back
Pull-Up	Biceps, Arms

FLEXIBILITY TASK:

(*10pts*) Flexibility exercises stretch your muscles and can help your body stay flexible. These exercises may not improve your endurance or strength, but being flexible gives you more freedom of movement for other exercise as well as for your everyday activities. `Flexibility` is a `Fitness` and we cannot give the actual implementation for the methods `muscleTargeted()` and `calorieLoss()` as we don't know the actual Anaerobic exercise. The `descripton()` method returns the string `Flexibility is uncomfortable and it takes time, so people don't like to do it`. Note that `Flexibility` is a good candidate to be `abstract` class.

Define classes `Yoga` and `TaiChi`, which are `Flexibility` fitness exercises. Use the table above to define the classes. The classes also have:

- `@Override public String description()` - returns the name of the respective classes.

Endurance Task:

(*10pts*) You can improve your endurance by doing an activity for increasing periods of time. `Endurance` is a `Fitness` and we cannot give the actual implementation for the methods `muscleTargeted()` and `calorieLoss()` as we don't know the actual Anaerobic exercise. The `descripton()` method returns the string `Endurance is all about sweat and patience`. Note that `Endurance` is a good candidate to be `abstract` class.

Define classes `Squat` and `PullUp`, which are `Endurance` fitness exercises. Use the table above to define the classes. The classes also have:

- `@Override public String description()` - returns the name of the respective classes.

Profile Task:
`public class Profile`

(*10pts*) This class represents the profile of the user. It has the age, weight, height, and gender (MALE or FEMALE) of the user. Assume the weight is in kgs, height in meters and gender is `enum` data type called `Gender`. In addition, this class must contain:

- `public Profile(int age, double height, double weight, Gender gender)` - A constructor, which accepts the age, weight, height, and gender.
- `public void setHeight(double height)` - the setter that changes the height of the user.
- `public void setWeight(double weight)` - the setter that changes the weight of the user.
- `public void setAge(int age)` - the setter that changes the age of the user.
- `public void setGender(Gender gender)` - the setter that changes the gender of the user.
- `public double getHeight()` - the getter that returns the height of the user.
- `public double getWeight()` - the getter that returns the weight of the user.
- `public int getAge()` - the getter that returns the age of the user.
- `public Gender getGender()` - the getter that returns the gender of the user.
- `@Override public String toString()` - returns the string that represents the profile of the user as `Age 26, Weight 78.0kg, Height 1.7m, Gender MALE`. Use one digit after the decimal point.
- `public double calcBMI()` - this method calculates and returns the Body Mass Index (BMI) of the user. BMI is calculated by dividing the weight by the square of the height (in kg and in meters respectively).
- `public double dailyCalorieIntake()` - this method calculates and returns the rough daily calorie intake necessary to maintain the current weight based on Body Mass Ratio (BMR) of the user. Assume the BMR value is the daily calorie need. BMR is calculated as follows:

  ```
  BMR for men = 66.47 + (13.75 * weight in kg) + (5.003 * height in cm) - (6.755 * age)

  BMR for women = 655.1 + ( 9.563 * weight in kg ) + ( 1.85 * height in cm ) - ( 4.676 * age )
  ```

Planner Tasks:
`public class DailyExercise`

(*10pts*) This class represents a daily exercise plan. It has a list of `Fitness` exercises a user plans to do, the duration s/he willing to workout, and the targeted calorie loss for a day. The implementation of this class will most likely involve the use of some kind of `ArrayList` to store all of the `Fitness`es for the daily workout. In addition, this class must contain:

- `public DailyExercise(ArrayList<Fitness> exerciseList, int duration, double calorieTarget, Profile profile)` - A constructor, which accepts the list of exercises, number of minutes to workout, and the amount of calories to be burnt.
- `public DailyExercise(ArrayList<Fitness> exerciseList, Profile profile)` - A constructor, which sets `duration` to 1 hour and `calorieTarget` to 500.
- `public void addExercise(Fitness ex)` - add a new Fitness in the exerciseList.
- `public void removeExercise(Fitness ex)` - removes an Exercise from the exerciseList. If the exercise does not exist, it will leave the exerciseList unchanged.
- `public void setExerciseList(ArrayList<Fitness> list)` - A setter method, which sets the exerciseList of the DailyExercise.
- `public void setDuration(int duration)` - A setter method, which sets the duration of the DailyExercise.
- `public void setCalorieTarget(double target)` - A setter method, which sets the amount of calorie to be burnt of the DailyExercise.
- `public void setProfile(Profile profile)` - A setter method, which sets the profile of the user.
- `public ArrayList<Fitness> getExerciseList( )` - A getter method, which returns the exerciseList of the DailyExercise.
- `public int getDuration()` - A getter method, which returns the duration of the DailyExercise.
- `public double getCalorieTarget()` - A getter method, which returns the amount of calorie to be burnt of the DailyExercise.
- `public Profile getProfile()` - A getter method, which returns the profile of the user.
- `public Fitness[] getExercises(Muscle[] targetMuscle)` - returns an array of `Fitness` exercises from the exerciseList that fullfills all the target muscle groups (`targetMuscle`) the user wants to work on for that specific day. The method will return `null` if there is no exercise that targets all the muscle groups.

`public class WeeklyExercise`

(*25pts*) This class represents a weekly exercise plan. It has a list of `Fitness` exercises a user plans to do, the number of times s/he is willing to workout per week, and the targeted calorie loss for a week. The implementation of this class will most likely involve the use of some kind of `ArrayList` to store all of the exercises for the week. In addition, this class must contain:

- `public WeeklyExercise(ArrayList<Fitness> exerciseList, int days, double weeklyCalorieTarget, Profile profile)` - A constructor, which accepts the list of daily exercises, number of days to workout per week , the amount of calories to be burnt, and a profile of a user.
- `public WeeklyExercise(ArrayList<Fitness> exerciseList, Profile profile)` - A constructor which sets number of days to 7 and weeklyCaloryTarget to 3500.
- `public void addExercise(Fitness ex)` add a Fitness in the exerciseList.
- `public void removeExercise(Fitness ex)` - removes a Fitness from the exerciseList. If the fitness does not exist, it will leave the exerciseList unchanged.
- `public void setExerciseList(ArrayList<Fitness> list)` - A setter method, which sets the exerciseList of the WeeklyExercise.
- `public void setDays(int days)` - A setter method, which sets the number of days the user plans to workout for the week.
- `public void setWeeklyCalorieTarget(double target)` - A setter method, which sets the amount of calorie to be burnt per week.
- `public void setProfile(Profile profile)` - A setter method, which sets the user's profile.
- `public ArrayList<Fitness> getExerciseList( )` - A getter method, which returns the exerciseList of the WeeklyExercise.
- `public int getDays()` A getter method, which returns the number of days the user plans to workout per week.
- `public Profile getProfile()` - A getter method which returns the user's profile.
- `public double getWeeklyCalorieTarget()` - A getter method, which returns the amount of calorie to be burnt for the week.

- `public ArrayList<DailyExercise>getWeeklyExercises(Intensity intensity)` - A method that returns a list of `DailyExercise`s that the user should do in order to meet the targeted calorie loss. The method evenly distributes the calorie loss over the number of `days` the user plans to workout. This method accepts the intensity of the exercises. One exercise will be used per day and we have exactly `days` number of exercises in the `exerciseList`. Note that the `DailyExercise` suggessions include the duration (minutes) the user should workout towards the targeted calorie loss.
  We may assume that the input number of days matches the number of `Fitness` exercises in the exercise list, and that each one gets its own day in the output and an equal fraction of the total calorie loss responsibility. See the example below. We must somehow figure out how to use a class's `calorieLoss` to help us deduce the number of minutes needed to achieve the calorie loss goal. Note that the duration we calculate may be a fractional value. When converting the floating point value of duration to an integer, please use an ordinary typecast without a rounding operation, to be consistent with the tester. See the example below.
- `public ArrayList<DailyExercise>getWeeklyExercises()` - A method that returns a list of `DailyExercise`s that the user should do in order to meet the targeted calorie loss. The method evenly distributes the calorie loss over the number of `days` the user plans to work out. This method is similar to the previous version except that it assumes LOW intensity exercises. One exercise will be used per day and we have exactly `days` number of exercises in the `exerciseList`. Note that the `DailyExercise` suggessions include the duration (minutes) the user should work out towards the targeted calorie loss.
- `public String targetedCalorieLoss(double targetWeight, int withInDays)` - this method returns the string that contains a suggestion on how to loss the targeted weight within the specified number of days. Assume that you need to burn (or decrease your intake by) 7000 calories to lose 1Kg. The format of the output is "`You need to lose 755.00 calories per day or decrease your intake from 1700.00 to 945.00 in order to lose 10.00 kg of weight`". (Note that you can use "%.2f" to have two decimal places after the dot). If the target weight is greater than the actual weight of the user, the method throws a TargetWeighException. You need to define a TargetWeightException that extends Java's `RuntimeException` class and displays the string "Only works to lose weight".

Example:

```
> import java.util.*;
> Profile p = new Profile(25, 1.85, 88, Gender.MALE);
> p
Age 25, Weight 88.0kg, Height 1.9m, Gender MALE
> p.calcBMI()
25.712198685171657
> p.dailyCalorieIntake()
2033.15
> ArrayList<Fitness> exercises = new ArrayList();
> Collections.addAll(exercises, new Swimming(), new Yoga(), new WeightLifting());
> exercises
[Swimming@758b306a, Yoga@330e4651, WeightLifting@3cb2992a]
> WeeklyExercise w = new WeeklyExercise(exercises, 3, 1000, p);
> w.targetedCalorieLoss(85.0, 30)
"You need to lose 700.00 calories per day or decrease your intake from 2033.15 to 1333.15 in order to lose 3.00 kg of weight"
> w.targetedCalorieLoss(85.0, 90)
"You need to lose 233.33 calories per day or decrease your intake from 2033.15 to 1799.82 in order to lose 3.00 kg of weight"
> ArrayList<DailyExercise> exArray = w.getWeeklyExercises(Intensity.MEDIUM); // medium intensity exercise
> exArray.get(0).getExerciseList() // the first day's exercise is Swimming
[Swimming@758b306a]
> exArray.get(0).getCalorieTarget() // splitting a 1000.0 kcal loss across 3 days yields this
333.3333333333333
> exArray.get(0).getDuration() // it takes 27 minutes of medium-intensity swimming to achieve the 333.3 kcal loss
27
> exArray.get(1).getExerciseList() // the second day is Yoga
[Yoga@330e4651]
> exArray.get(1).getCalorieTarget()
333.3333333333333
> exArray.get(1).getDuration()
75
> exArray.get(2).getExerciseList() // the third day is WeightLifting
[WeightLifting@3cb2992a]
> exArray.get(2).getCalorieTarget()
333.3333333333333
> exArray.get(2).getDuration()
45
```

HONORS SECTION:
If you are in the honors section, you must complete this part and it is worth 10 points of the project grade. If you are not in the honors section, you are welcome to attempt this but you do not need to complete it and it is not worth any points if you do.

`public class DailyExercise`:

- *(10 pts.)*Add a new method in the `DailyExercise` class that returns an array of `Fitness` exercises from the exerciseList that fullfills all or some of the target muscle groups (`targetMuscle`) the user wants to work on for that specific day. The method will return `null` if there is no exercise or combination of exercises that targets all/some of the muscle groups. For example, if the targeted muscles are `Arms`, `Cardio` and the exerciseList has `WeightLifting`, `Yoga`, the method returns `WeightLifting` as it works on `Arms`. Note that none of the fitnesses fulfill both of the target muscles (only `WeightLifting` fulfills it partially). Use the method signature:
  - `public Fitness[] getAllExercises(Muscle[] targetMuscle)`

TESTING:

- https://cs.gmu.edu/~tmengis/courses/FA20/junit-cs211.jar
- https://cs.gmu.edu/~tmengis/courses/FA20/P3Tester.java

The additional files are for some of the test cases, and should be copied into your working directory.

SUBMISSION:

Submission instructions are as follows.

1. Let *xxx* be your lab section number, and let *yyyyyyyy* be your GMU userid. Create the directory `xxx_yyyyyyyy_P3/`
2. Place your files in the directory you've just created.
3. Create the file `ID.txt` in the format shown below, containing your name, userid, G#, lecture section and lab section, and add it to the directory.

   Full Name: Donald Knuth
   userID: dknuth
   G#: 00123456
   Lecture section: 004
   Lab section: 213

4. compress the folder and its contents into a .zip file, and upload the file to Blackboard.