Task 1 — Project Domain:

Personal Health app with data analysis

- User input their personal health data (ie. height, weight, age, gender)
- User indicates which functionality they would like to use:
    - Generating Workout moves, Generating Meal plan, DiseaseRiskFactor
- Depending on the functionality they chose, the program outputs prompts the user need to provide inputs to
- Use the inputs provided to provide suggestions on exercise/meals or comments on health
- Access a local database on nutritional facts
    - Process the data and categorizes the food into different types according to their nutrient level

Task 2 — Writing the specification:

Entity Classes

- **User**: a class containing all data relevant to the user
    - Instance attributes:
        - username
        - Information
        - IsMale: boolean (true if male, false otherwise)
- **Food**:
    - A Class that represents a type of food
    - Stores Calories, nutritional breakdown, and the category of the food
    - Instant attributes:
        - Calories
        - Carbohydrate (percentage of DV)
        - Protein (percentage of DV)
        - Fat (percentage of DV)
        - Sugar (percentage of DV)
        - FoodType
        - VegetarianFriendly
        - NutrientScore
        - Id
- **Exercise:**
    - A Class that represents a workout move
    - Stores the difficulty of the move, what muscle it exercises, and what equipment it needs
    - Instance Attributes:
        - type

        - muscleExercised
        - equipmentNeeded
- **Disease**
    - A Class that represent a disease
    - Stores the name of the disease and the symptoms of it
    - Instance Attributes:
        - disease
        - symptoms
- **UserInformation**:
    - A class that stores user's information
    - Instance Attributes:
        - foodPreference
        - exercisePreference
        - riskFactor
        - personalData

Use Cases

- **UserAnalyzer**: a parent class that manipulates different information of the user based on the functionality selected by the user.
    - Children Class:
    - BMIAnalyzer, EERAnalyzer, DiseaseAnalyzer
- **MealPlanGenerator**: generates a specific meal plan for the user according to the user's food preference and the user's energy requirement per day (measured in Calories)

Interface Adapters:

- **Controller:** The controller class
    - Outputs the prompt to the terminal
    - Takes input from user and direct the input to the correct use cases

Framework and/or Drivers:

- Basic Command Line interface:
    - **Console**
    - (open to the option of creating a more user-friendly interface if there is enough time)

- **API**: read and process data from our databases
    - FoodAPI
    - ExerciseAPI
    - DiseasesAPI

Task 4 - Scenario walkthrough

For the scenario walkthrough, we will only be focusing on one functionality to show that the code can run. For the scenario we will be focused on some calculations and print out the BMI as the output.

A user will first open up the app, and the app will then prompt the user for several inputs. First, we will ask for some basic information, such as their name, age, gender, height, and weight. Then the program will prompt the user to select a functionality/feature that they would like to use. In this case, they will select the BMI function, and the program will output the BMI of the user.

How it works is that the user will first interact with the Console class. The Console will take in all the data entered by the user and pass them to the Controller, which then passes the data into UserAnalyzer. After that, UserAnalyzer will create an User object along with an Information Object to store the data. After the User and Information Object have been created for the user and the data saved, the Console will then prompt the user to choose a functionality of the application. In this case, the user will select the "Calculate BMI" option. After receiving the user input, the Console will pass this message to the Controller, which will then contact the BMIAnalyzer (Subclass of UserAnalyzer). The BMIAnalyzer will calculate the BMI, store the value into the corresponding Information Object of the user, and return it to the Controller. The Controller will then pass the BMI value into the Console which outputs the BMI value to the user.

Questions:

Should we make instance attributes private or public?
Do we still need getters if all instance attributes are public?

!!! Do we still need UserInformation (the Parent class)? Since its subclasses have nothing in common. If we get rid of it, how can we organize?

Does Console need to implement an interface since it’s an UI?

Does API directly connect to entity classes?