BREAKPOINT CODING PRACTICE

Working with Classes

User Information

Create the Class

Person Class (Two Properties, Two Methods)

- Create a class that will store the following information about the user
  - Name
  - Birthday
- Create two methods that will do the following
  - Calculate the user's age
  - Determine the number of days until the user's birthday
  - The class CAN NOT have any Console method

Instantiate the Objects

- Create an object to get information about your user.
  - Use the default constructor to create the object
- Create two more objects to store information about two of the user's friend
- Determine who is oldest out of the two friends and display that information in the Console window

Exercise Information

Create the Classes

Athlete Class (Four Properties, Three Methods)

- Create a class that will store the following information about an athlete
  - Name
  - Daily Calorie Goal
  - Current Heart Rate – Read Only
    - DO NOT use an auto property
  - Current Calories Burned – Read Only
    - Starting value should be zero
    - DO NOT use an auto property
- Create the methods that will do the following
  - Update the Current Heart Rate (one integer parameter)
    - Thrown an exception if the parameter is less than 1
  - Update the Current Calories Burned (one integer parameter)
    - Thrown an exception if the parameter is less than 1
  - Determine if the Calorie Goal was met (boolean return type)
  - The class CAN NOT have any Console method
- Create a custom constructor that requires the following input:
  - Name, Daily Calorie Goal, Current Heart Rate

Exercise Class (3 Properties, 1 Method)

- Create a class that will store the following information about an exercise
  - Name
  - Calories Per Minute
  - Heartrate Adjustment Value
    - How much does this exercise increase or decrease the heart rate each minute
  - A flag to determine if the exercise is currently being performed.
    - This default value for this flag is false
- Create a method that will switch the flag to the opposite of its current value.
- Create a custom constructor for the class

Instantiate the Objects

- Create an instance of the Athlete Class
  - Set the required information
- Display the following information in the Console window

    - Athlete Name and Daily Calorie Goal
    - Current Heart Rate and Current Calories Burned
    - A message stating whether or not the Daily Calorie Goal has been reached
- Create several instances of the exercise class, storing them in a collection
    - Make sure to populate the required information.

Write the application

In your main method, write the following logic.

Create a variable to store the max heart rate of 175
Create a variable to store the minimum heart rate of 120

Ask the athlete what exercise they want to do first and for how long they want to do it. Iterate through each minute, increase the calories burned, and adjusting the heart rate. If the heart rate reaches the max heart rate or minimum heart rate, end the exercise early.

At the end of the exercise session, Display the following information in the Console window

- Athlete Name and Daily Calorie Goal
- Current Heart Rate and Current Calories Burned
- A message stating whether or not the Daily Calorie Goal has been reached

If the session ended early, display a message explaining why and recommend the next session be the one that will increase or decrease the heartrate the quickest.

Continue work out sessions until the Daily Calorie Goal is reached.

Board Game Collection

Create the Enum

Game Mechanism Enum
Create an enum for the main game mechanisms of board games (e.g., Deck Building, Set Building, Worker Placement)

Create the Classes

Board Game Class (Five Properties, No Methods)

- Create a class that will store the following information about a board game
    - Name
    - Max number of players (default to 4)
    - Recommended Age (default to 8)
    - Approximate Playing Time (default to 90)
    - Main Game Mechanism
- Create a custom constructor that requires the Name, Max number of players, and Main Game Mechanism but allows the other value to be set if desired.

Board Game Collection Class (Properties, Methods)

- Create a class that will store the following information about a board game collection
    - Name of the collection
    - Collection of Board Game objects – the collection object is private
        - Make the collection a dictionary with the name of the board game as the key and a board game object the value.
- Create a custom constructor that requires the Name of the collection and instantiates an empty board game collection
- Create a method that accepts a Board Game object as a parameter and adds the object to the collection
- Create a method that accepts a Board Game object as a parameter as well as a string parameter that contains the original name of the board game. The method will update an existing board game in the collection
    - Throw an exception if the string is invalid
    - Throw an exception if the provided name doesn't exist in the collection
- Create a method that will accept the name of a board game and remove it from the collection
    - Throw an exception if the string is invalid
    - Throw an exception if the provided name doesn't exist in the collection
- Create a method that will return the collection as an array that can not be updated outside of the class

Instantiate the Objects

- Create an instance of the Board Game Collection Class
    - Set the required information
- Create several instances of the Board Game class, adding them to Board Game Collection

Write the application

Write an application that allows the following

- Display a list of the board games
- Allow the application to filter on the following
    - Game Mechanism
    - Number of Players
    - Approximate Playing Time
- Add a board game to the collection
- Update an existing board game
- Delete a board game from the collection