**CS 1150 HW Objects and Files**

Due Date: **See Canvas**

Purpose: Demonstrate ability to problem solve, implement class relationships, arrays, objects and writing to a file.

Effort: **Individual**: Read [CS Academic Integrity .pdf](https://drive.google.com/file/d/1vHXGQPPRUXZQrUxfnM6hJQid9jio7xlt/view?usp=sharing)

Points: **100 (see rubric in canvas)**

Deliverables: **Upload your design document with your design/ reflection/learning question and your .java file as separate files. Do not upload as a zip file.**

**Assignment Description**

This assignment provides the opportunity to work with an array of objects and an object that contains another object. The array of objects will hold **race car** objects and each **race car** object has an **odometer** object. The code will simulate a race in which each race car must race at least 500 miles. The race is finished when all cars have raced at least 500 miles.

In the code, perform three tasks::

* Task 1: create race car objects as described below and place them into an array.
* Task 2: create a loop to simulate a race until **ALL** of the race cars reach at least 500 miles.
* Task 3: display all race cars that have finished during each hour of the race.

In this assignment, the completed program contains **three classes:**

* **HW08** – contains main (test program that creates and works with the other 2 classes)
* **RaceCar** – represents one race car
* **Odometer** – represents the odometer in a race car

Details for both **RaceCar** and **Odometer** classes are provided in the “Classes” section below.

**Specifications**

Read these specifications and output to get an overview of the assignment. When writing code, **first** create the classes.

1. Create a Java class **LastNameFirstNameHW08** within that project calledwithin the **CS1150HW** project. Place all classes in one Java file.
2. Follow [Problem Solving Using Software Development Life Cycle](https://docs.google.com/presentation/d/e/2PACX-1vSC4tM0BsOaYjRmd_emNeHtKrWaExKXfvvLxQC10rCt77CPT_WvF_s3W--o_yuj8XczYGo7qHxgjMIq/pub?start=false&loop=false&delayms=3000&slide=id.ge6997ea817_0_125)
3. **Design Task 2:** You only need to design the algorithm in main where you iterate until all race cars reach at least 500 miles. See highlighted information below and do the following to help improve your problem solving skills
   * Sketch out on paper what is happening in main for Donkey, Shrek and Dragon
   * Write pseudocode and think about
     1. How do you implement a loop until all race cars reach at least 500 miles?
     2. How do you increment to show each iteration represents 15 minutes?
     3. How do you represent minutes as hours?
     4. How do you determine if the iteration represents an hour of racing?
        1. 4 iterations = 1 hour of racing, so 8 iterations = 2 hours, etc.
        2. How do you determine if the current number of iteration is a factor of 4?
     5. At each hour how do you update each race cars mileage (i.e. on hour 1, then hour 2, etc.) and display any race cars that have finished?
4. Write a **test program** (i.e. main) that:
   * Creates an array that stores **6** race cars.
   * Calls ***createRaceCars*** method to fill array
     1. This method creates the 6 race cars using the information in the following table and places the race cars into the array.
     2. Note: creation of the race cars is done manually in this method, so no user input is required to create the race cars. Hardcoding is allowed only in this method.

| **Race Car #** | **Driver** | **Average Race Speed** |
| --- | --- | --- |
| 8 | Donkey | 80 mph |
| 11 | Shrek | 115 mph |
| 17 | Dragon | 70 mph |
| 18 | Fiona | 100 mph |
| 45 | Farquaad | 84 mph |
| 42 | Pinocchio | 68 mph |

* + Calls ***displayAllRaceCars*** method to display on the **console** the details for each race car.
    1. Display the race car details before the race starts
    2. Race car #, driver, average speed, miles raced (zero at this point)
  + Start the race!
  + Create a loop that iterates until **ALL** race cars reach at least 500 miles:
    1. Note: each loop iteration simulates 15 minutes of racing (4 iterations = 1 hour)
    2. Print the race time every 15 minutes (see output below)
    3. **At each hour**, update race cars mileage (i.e. on hour 1, then hour 2, etc.) and display any race cars that have finished
       1. For each race car in the array
          1. If the race car is still racing (finished is still false)

Update race car’s number of miles raced for 1 hour

Because the odometer is updated only each hour, use the average speed to update the odometer’s miles

Obtain the odometer for this race car

Call the ***updateMiles*** method in **Odometer** sending in the average speed

If car has raced 500 miles or more

Set race car’s ***finished*** state to true

Call ***displayRaceCar*** to display this race car’s driver name and miles raced.

* + - 1. After updating mileage, determine if all cars have finished the race
         1. Call ***isRaceFinished*** and set the flag for the while loop to either keep going or stop
  + After the while loop completes:
    1. Call ***displayAllRaceCars*** method to display on the **console** details for each car
       1. Display the race car details at race completion.
       2. Display race car#, driver, average speed, and miles raced.
       3. The miles raced should have changed for each car.
    2. Call ***writeCarDetailsToFile*** method to write the details for each race car to a file.
       1. The code for this method is provided for you, see step #6 below.

1. The test program HW08 (i.e. main) **must** have these methods
   * **Place these methods after main inside the hw class**

// Create the race cars for the race based on the assignment sheet table and

// fill the incoming array with the race cars

**public** **static** **void** createRaceCars(RaceCar[] raceCars)

// Display the details (driver’s name and miles raced) for a single race car

**public** **static** **void** displayRaceCar(RaceCar raceCar)

// Display the details for each race car in the array

// Display race car #, driver name, average speed, and miles raced

**public** **static** **void** displayAllRaceCars(RaceCar[] raceCars)

// Determines if the race is finished, that is, have all race cars finished

// If "finished" state for all race cars is true, return true otherwise false

// Print the details for each race car in the race cars array to a file

**public** **static** **void** writeRaceCarDetailsToFile(RaceCar[] raceCars) **throws** IOException

**Classes**

Design classes called **RaceCar** and **Odometer** containing the following fields and methods. Implement encapsulation.

**RaceCar Class**

* Description
  + Class that represents one race car
* Private Data Fields **(all instance variables must be defined as** **private**)
  + **driver –** string identifier for the driver’s name
  + **number –** int identifier for the race car’s racing number
  + **averageSpeed –** double identifier for the race car’s average speed
  + **finished –** boolean identifier indicating if the race car finished the race
  + **odometer –** odometer object that represents the race car’s odometer
    - **Note:** this is a “Has-A” relationship; the race car has an odometer
* Constructor and Methods
  + Constructor

***public RaceCar(String driver, int number, double averageSpeed)***

* + - Initialize instance variables **driver, number,** and **averageSpeed** to incoming driver, number and averageSpeed
    - Note that **finished** is initialized to false by default when the RaceCar is created. This information is not passed to the constructor.
    - Allocate memory for an odometer object (code provided)
      * The code is provided to show you how to allocate the memory.
      * Do you understand what this code is doing?

odometer = **new** Odometer();

* + Getters
    - Create getters for **driver, number, averageSpeed, finished,** and **odometer**
  + Setters
    - Only need a setter for the instance variable **finished**

**Odometer Class**

* Description
  + Class that represents the odometer in a race car
* Data Fields
  + **miles –** double identifier for the current miles raced
* Constructor and Methods
  + No constructor is needed in the Odometer class but include the default

***public Odometer ()***

* + - Note that **miles** are initialized to 0 by default when the Odometer is created.
    - Thus the odometer of the race car is zero at the beginning of the race.
  + Getters
    - Create getter for **miles**
  + **public void updateMiles(double milesTraveled)** 
    - Update the odometer miles by the number of miles the race car has traveled

**Output**

Your output must look like the following:

Race cars at start of race

---------------------------------------------------------

Race Car Driver Average Miles

Number Speed Raced

---------------------------------------------------------

8 Donkey 80.00 0.00

11 Shrek 115.00 0.00

17 Dragon 70.00 0.00

18 Fiona 100.00 0.00

45 Farquaad 84.00 0.00

42 Pinocchio 68.00 0.00

All cars are ready - let the race begin in 5..4..3..2..1

Racing time ... 0.25 hours

Racing time ... 0.5 hours

Racing time ... 0.75 hours

Racing time ... 1.0 hours

Racing time ... 1.25 hours

Racing time ... 1.5 hours Showing race time 

Racing time ... 1.75 hours every 15 minutes

Racing time ... 2.0 hours

Racing time ... 2.25 hours

Racing time ... 2.5 hours

Racing time ... 2.75 hours

Racing time ... 3.0 hours

Racing time ... 3.25 hours

Racing time ... 3.5 hours

Racing time ... 3.75 hours

Racing time ... 4.0 hours

Racing time ... 4.25 hours

Racing time ... 4.5 hours

Racing time ... 4.75 hours

Racing time ... 5.0 hours Show when a race

Finished: Shrek with 575.0 miles car finishes race

Finished: Fiona with 500.0 miles (reached 500 miles

Racing time ... 5.25 hours or more)

Racing time ... 5.5 hours

Racing time ... 5.75 hours

Racing time ... 6.0 hours

Finished: Farquaad with 504.0 miles

Racing time ... 6.25 hours

Racing time ... 6.5 hours

Racing time ... 6.75 hours

Racing time ... 7.0 hours

Finished: Donkey with 560.0 miles

Racing time ... 7.25 hours

Racing time ... 7.5 hours

Racing time ... 7.75 hours

Racing time ... 8.0 hours

Finished: Dragon with 560.0 miles

Finished: Pinocchio with 544.0 miles

Race cars at end of race

---------------------------------------------------------

Race Car Driver Average Miles

Number Speed Raced

---------------------------------------------------------

8 Donkey 80.00 560.00

11 Shrek 115.00 575.00

17 Dragon 70.00 560.00

18 Fiona 100.00 500.00

45 Farquaad 84.00 504.00

42 Pinocchio 68.00 544.00

Find the file here C:\JavaDev\CS1150HW\HW08.txt



This is where I created my eclipse workspace

Yours will be different

Check this location for Assignment11.txt file.

Reflection

1. What relationship is represented between the Car and Odometer class? Explain the difference between a “has-a” and “is-a” relationship between two classes.
2. Explain the code to write to a file and include explanations of the classes used
3. Look at where you started in this class and look at where you are now with analyzing, designing, implementing and testing solutions and understanding how information from a java program is stored in memory. What do you feel are your biggest improvements? What areas do you still want to improve?

1. The Car class Has an Odometer class. "Has-a" relationships are where one class contains an instance of another class as one of its members or fields. "Is-a" relationships are inherited,, where one class is a subtype of another class.

3. I feel my biggest improvement is in understanding scope, where as I used to have many issues with my code, now it is rare, and I understand how to fix scope issues. I have also made strides in writing more clean and easy to understand code and using comments to direct myself.