UNK CSIT 150
Lab 3: Fundamentals of Classes Part Two

Objectives

- Practice defining a class
- Practice the use of object data members in a class
- Practice using deep copy of objects
- Practice dealing with exceptions (bonus)
-

General Requirements

In this lab, you must write your code following the proper programming style. The bottom line is:

- Use indentation to show the logical structure of your code
- Use blank line to separate code blocks and give each code block a comment
- Give documentation comment to the class and each method

Programming Practice

Create a package for this lab 3. Copy the Car class you created in Lab 2 into this package and add the lab code provided. If you did not get the Car class completed, you can use the Car class provided with the lab code.

1. Finish the class named **Race**. The Race class has the following fields (i.e., data members):
    - Array or ArrayList of Cars, for cars to participate in the race.
    - double distance, for the distance of the race.
    - String raceType, for the type of race (such as stock car, sprint car, etc.).
    - int carCount, to keep track of the number of cars that have "entered" the race.
    - Car winner, for the winner of the race (initially set to null when a race is created)

    The Race class has the following public methods (with JavaDoc documentation for each method):
    - Accessor methods for distance and raceType.
    - Accessor method for the array/ArrayList of cars. (Return a copy of the array/ArrayList of Cars for the getRaceCars accessor. Each car in the copied array is a copy of the car in the race's array of cars.)
    - Accessor method for the winning car. Returns a string. Either returns the winning car's information or the phrase **Race winner is unknown at this time**
    - Mutator methods for the distance and raceType.
    - addCar – a method that takes in a car and creates a copy of the input car parameter in the race car array, and adds it in the next open spot in the array. (Hinit: see the CD Collection example or the BankAccountArrayList example.)
    - **Constructors**: create a default constructor and constructor that takes in the starting number of cars in the race.
    - **runRace**: a void method to run the race and set the winner.
      To simulate the race, this method rolls a Dice object to select the current action, until one of the cars reaches the finish line.

      ```
      Dice raceDice = new Dice(carCount*3);
      ```

      Select which car and what action number based on the dice roll

```
int diceRoll = (raceDice.roll()
int carToMove = (diceRoll -1)/3
int action = (diceRoll -1)%3
```

If the action is 0, then move the car for one minute. If action is 1, then brake the car. If action is 2, then accelerate the car. Remember, the car cannot exceed its max speed, nor go below 0 mph.

Continue selecting cars and actions in this manner until one of the car reaches (or passes the finish line) as the action is executed.

Set the winner of the race appropriately.

(For bonus, how would this method need to change if the number of cars is not set at three cars?)

- toString method. Output the race information in an appropriate manner.
- Create a private mutator method (with JavaDoc) to set the winner of the race. This method is called from the "runRace" method.
- Add any additional methods that you need to complete this lab. Note, I used a private helper method called "increaseSize" when adding cars to the race. You do not need to use this method. See the CD Collection for an example implementation of this method.

2. Modify the class named **RaceTest**. This class just has a main method that creates the races. Uncomment the parts of the code to test your Race class.

3. BONUS:

- Add the following to the heading of each constructor in the Race class:

```
throws NegativeStartingCount
```

- Then inside the Race constructors:
  if you are using arrays:
  If the input starting number is zero, set the array to null
  If the starting number is more than zero, create an array of that size
  If you are using an ArrayList: simply create an empty ArrayList, no matter what size requested by the input.
  In either case, add a check to see if the input starting number of cars is <0, then:

```
throw new NegativeStartingCount(inCarCount);
```

- In the RaceTest, add the try/catch blocks needed to implement the exception handling code, and test the third race.

Make sure to include JavaDoc for all of your classes.

Report to the Instructor

Before you leave, show the instructor

- The source code
- How the code runs
- The generated technical documents