COMP10062: Assignment 3

The Assignment

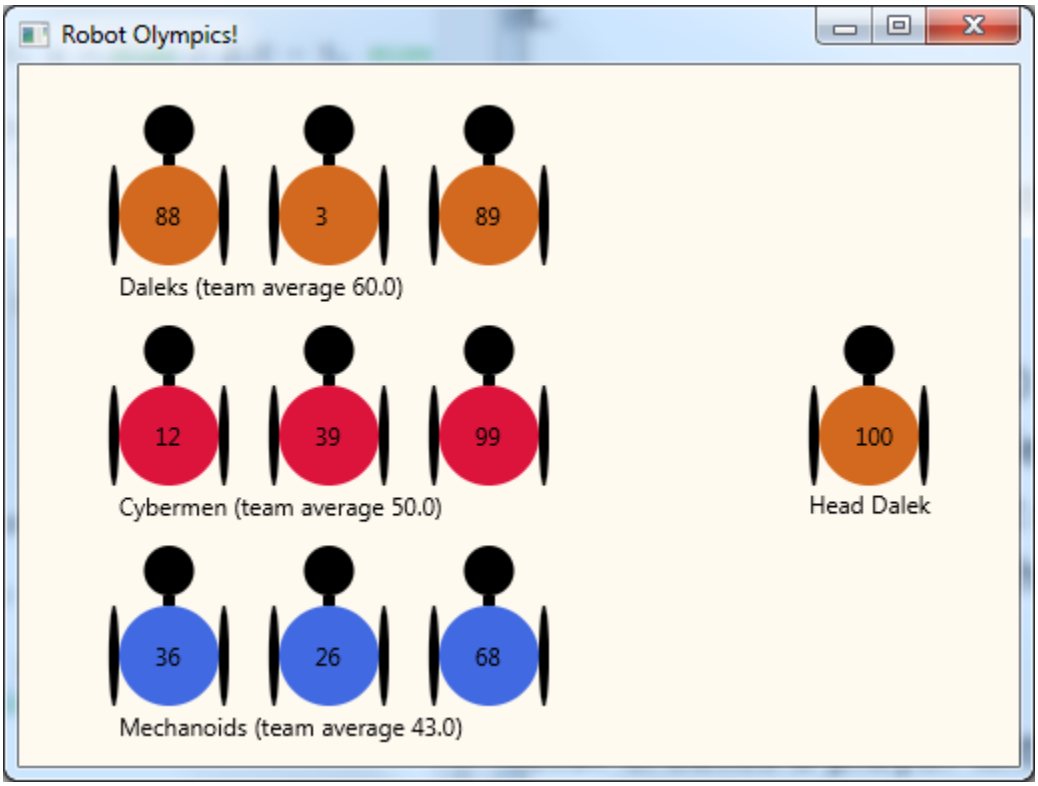
This assignment is mainly about association, but also covers constructors and overloading. You will define multiple classes for objects that can draw themselves on a `GraphicsContext`. All the objects will be related to one another through the relationship of association. Then you will create a simple graphical app to display the objects.

The Robot Olympics

The Robot Olympics consists of three teams and a referee – these four objects are the **model**. The **view** for this assignment is a class created using the FXGraphicsTemplate which creates and draws the model.

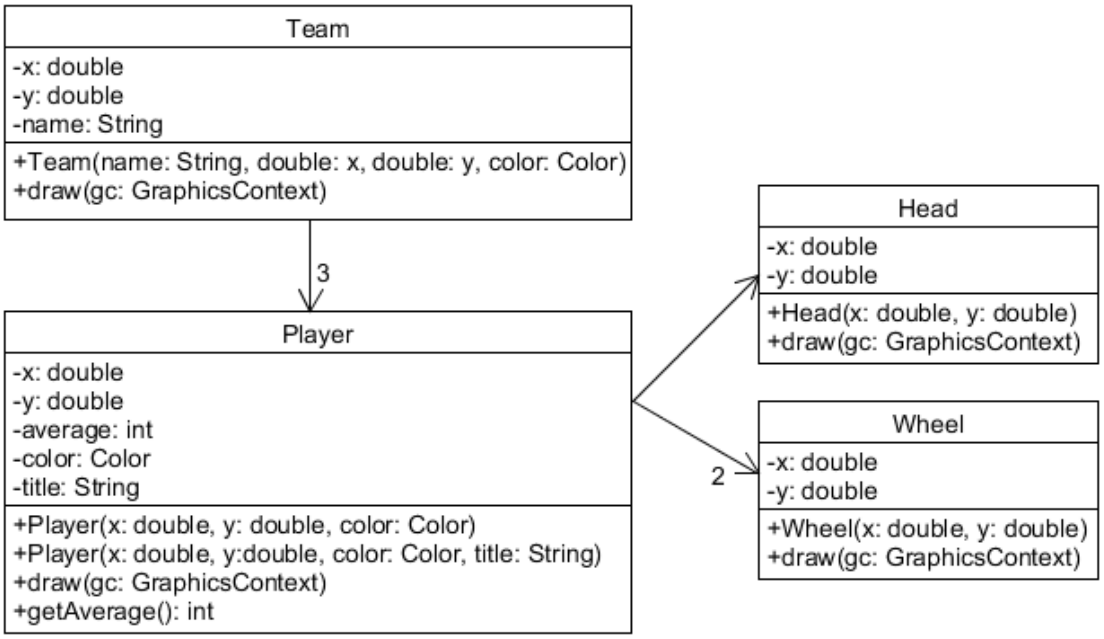
A team has a name and contains 3 players of the same color, each with a random number integer representing their win percentage (0-99). A player has a head and two wheels. The referee is a single player. The ref is not part of a team, but it was drawn from one of them, selected randomly at run time, and its color and title show that.

Here's an example of what the output of your program might look like. Yours SHOULD NOT look exactly like this. Your colors, players, names, and drawings should be different.



Code Structure

The UML class diagram below shows the classes you are to develop for the **model**, and how they relate to one another. Below the diagram are some notes on each of the classes.



Head

A Head is an oval, rectangle, or other head-like shape with a neck shape attached. Its x and y location determine its top left corner and the neck is drawn relative to that.

Wheel

A Wheel is a rectangle, oval or other wheel-like shape. Its x and y location determine its top left corner.

Player

A Player is a body shape with its win percentage displayed within it. Your players SHOULD NOT look like the picture – make them unique. The Players in each Team have the empty string as a title, but the referee has a non-empty title as shown. A Player's x and y location determine the top left corner of its body shape, and it contains two Wheels and a Head, appropriately sized, somewhere within or around its body.

The first constructor creates a regular player with an empty title and a random win percentage. The second constructor is used to create the referee with a non-empty title and a win percentage of 100. Player constructors create Head and Wheel objects with placement based on the Player's own location. The draw method draws the body and title for the Player, then calls the Head and Wheel draw methods.

Team

A Team has three Players of the same color. The Players in a Team are drawn in a straight line from left to right, spaced out nicely. The location of a Team corresponds to the location its first Player. The Team constructor creates the Players. When a Team draws itself, it calls the Player draw methods, then it draws text underneath with the name of the Team and its average win percentage.

ThreeTeams

Finally, you should create a ThreeTeams class (not shown in the diagram) to serve as the **view** for the model described above. This class will be based on `FXGraphicsTemplate`. It should create three Teams and a Referee from one of the teams (selected randomly) and draw them. If done correctly, this should only take a few lines of code (to create and draw each object).

Implementation and Documentation

Make sure you implement the UML diagrams exactly as shown.

Your domains don't have to look exactly like the example, but they should not always look the same on every run. At a minimum, the win percentages and referee color and title should be determined randomly. You could also try to set the Player sizes randomly, but this is trickier and is not required.

Make sure you comment to JavaDoc standards and follow the **Documentation Standards** on Canvas.

Going Further

If you are feeling creative, you could add other elements to this program (more complicated Player objects; more complicated Teams with other types of buildings; different styles of Players; Players with randomly determined sizes; animated elements; sounds; etc.). You can raise the multiplicities on the associations (i.e. have 4 Wheels per Player or 5 Players per Team), and you can add more classes (an Eye class, Weapon class, etc.) But don't remove any methods, fields, or associations from the core class diagrams, and don't remove any elements from the output. In other words, you can build on what's required, but you can't change or remove any basic requirements.

It might be a better model to have Competition object that creates and contains the three Team objects and the Referee object. Then ThreeTeams just creates a single Competition object and tells it to draw itself. Feel free to implement that change as well if you like it.

Handing In

See Canvas for the Due Date, but it would be a very good idea to get most of this assignment done before the test, since every class you write helps you understand Java and Object-Oriented programming more deeply.

See the due date and time on Canvas.

Hand in by submitting to the drop box the 5 java files with a txt file extension added on the end to the file. For example you file should look like: **FileName.java.txt**

Evaluation

Your assignment will be evaluated for performance (40%), structure (40%), and documentation (20%) using the rubric in the Canvas Assignment.

Helpful Example

See the SquareDonut and related classes in this week's example code, and relate it to the UML diagram shown at right.