CPSC 219: Introduction to Computer Science for Multidisciplinary Studies II

Project – Demo 2 Guidelines

Weight: 10%

Demo grading points

- Program is object-oriented (10)
    - Program is launched from main()
    - Menu and helper functions can be static
    - However now data must be stored in OO data structures
        - (java data structures like Arrays/ArrayList/HashSets/HashMaps can be used, but the majority of data storage should be in objects, with these just used to organize them)
    - proper use of inheritance, abstract, protected, public, private
    - proper use of final (constants)
    - proper use of static
    - proper use of interfaces, enums
    - proper usage of toString
- Proper usage of hashCode, equals, compareTo (5)
    - Need to be able to demonstrate how completing at least one of the above gives you a feature to your objects that you use
    - Like sorting an ArrayList, or using a HashMap with an object of your own
- Sketch of UML of program structure (5)
    - Does not have to give internal details about internal java classes
    - Make a class box diagram for each data object and gives details for those
    - Make a connection diagram between the project class name boxes with the arrows to show association/composition/aggregation/inheritance/etc.
- Menu still exists (5)
    - Has at least same complexity
    - Can still view all the stored data as well as the other 4 features
- Save/load data to file (5)
    - Program should have menu option which will save data to a (comma separate value) .csv file
    - Program should have menu option which will load data from a (comma separate value) .csv file
    - Program should be able to be run with command line argument that will start program using previous data saved to a (comma separate value) .csv file
- JUnit Testing (out of 5)
    - Unit tests for functions (non-input/non-output/non-menu) functions
- Gitlab Usage (out of 10)
    - Gitlab account exists, private project exists, at least 1 commit, small commits, both partners have a commit, (5) regular commits

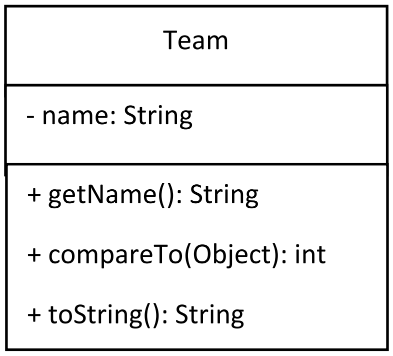
- Style/Commenting (out of 5)
    - Name/Date/Tutorial, Functions commented, Javadoc, Inline commenting, doesn't use inline conditionals, limited magic numbers, don't change function names, don't change filenames, etc.
- Partnership penalties
    - During the demo the TA will at times ask different members of your group (partnership) to describe how something works. Each partner should expect to have contributed to unit testing, git commits, commenting, and program functionality. In general, the penalties will be
        - -5 Partner is judged to have not contributed in one area (ex. uml, git, junit, commenting, code)
        - -10 Partner didn't contribute in two areas
        - -15 partner didn't contribute in three areas
        - -20 partner didn't contribute in all four areas
    - Contribution penalties are recorded separately for each student. Judgement is made by TA on basis of students being able to explain something about part of code that is being viewed in more detail than just re-iterating the readable syntax.

Example program. I decide to make a CWHL (ice hockey) statistics tracking program. I am going to make a

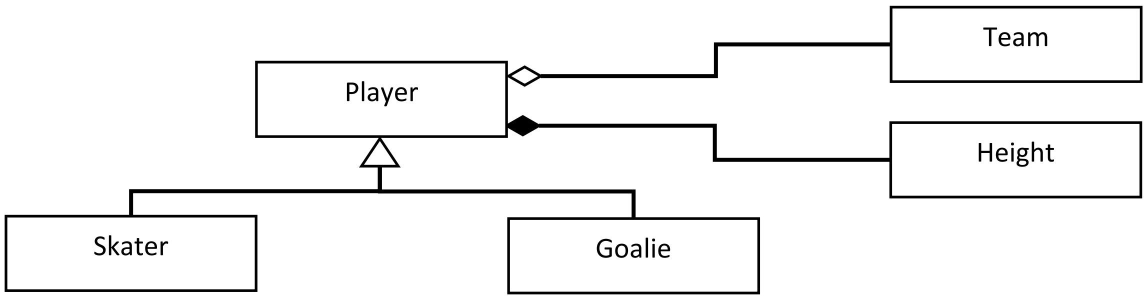
- **Team** object
    - Team has String name and other interesting info
- **Player** object
    - Player has **Team** team, String first name, String last name, Date birth date, Character position, String hometown, Integer jersey number, Integer weight, **Height** height
    - Note in my world I've decided goalies can't have shots on goal/goals, and skaters can't have saves (this is a choice is not necessarily an accurate model of stats that could be tracked but what I've decided I want to track)
- **Height** object
    - Height has Integer feet, Integer inches
- **Goalie** object extending **Player** (inheritance)
    - has Integer saves, shots on goal and calculated double GAA and double save percentage
- **Skater** object extending **Player** (inheritance)
    - has Integer goals, assists, shots and calculated Integer points, double shooting percentage
- ArrayList of **Teams**
    - Use compareTo to make sortable by name
- HashMap that lets me get **Player** object by using a **Player** to lookup
    - I will use hashCode/equals when getting a **player** (I will make a dummy **Player** with team/jersey from user input and use this as a key as players will be unique based on team and jersey number)
    - Could store a separate **Player** and **Goalie** map depending on what makes data easier to use.
- Likely a separate ArrayList for keep the list of all **Players** to process

- I will use compareTo to sort **Players** by a default team, jersey number order
- I will have secondary Comparators that allow me to sort by other features like certain stats
- Again might use separate **Player** and **Goalie** list

Example of class diagram for a simplistic Team. I would make one for Skater, Goalie, Player, Height, and Team.



Example UML relationship diagram of my objects. There would be more detail with a Main, Menu, and likely file Reader/Writer classes I'd make to complete this program)



Skater and **Goalie** inherit from **Player**

Player has-a **Team**

Height is a part-of **Player**

Previous menu options listed

Track basic data

1. add a Team
2. add a player to a team with a name, birthdate, position, and jersey number (use lookup using dummy Player to see if there is overlapping Players already)

Add additional data

3. add a goal to a player (use lookup using dummy Player) will add a shot as well
4. add an assist to a player (use lookup using dummy Player) will add a shot as well
5. add a save to a goalie (use lookup using dummy Player)
6. add a shot on goal to a goalie (use lookup using dummy Player)

Output General

7. ask for all players to be printed (use default compareTo sort)

Output Special

8. ask for the top 5 goal scorers (using comparator to sort)
9. ask for the top goalie in save percentage (using comparator to sort)
10. recommend line up of 2 defenceman, 1 goalie, and 3 forwards based on being top players (using comparator to sort)
11. list of players over a certain age (using comparator to sort)

File I/O

12. Save data
13. Load data