# Project - Part 1

**Due** Apr 7 by 11:59p.m. **Points** 100 **Submitting** an external tool **Available** Mar 25 at 1a.m. - Apr 7 at 11:59p.m.

## Building a children's soccer team

### The Request

The *British Columbia Soccer Club* has asked you to develop a software solution to build soccer teams for children under ten years old (U10). U10 soccer teams have seven players on the field. The best players are usually selected as the starting lineup and the remaining players are on the bench, ready to substitute the players on the field.

The coaches will use the following 7v7 formation:

- 1 Goalie
- 2 Defenders
- 3 Midfielders
- 1 Forward

For more information, please refer to the 2-3-1 formation on **this website (https://youthsoccer101.net/7v7-formations/)** .

### Software specifications

This software will be used by the head coach as follows.

#### 1. Entering player information

For every player, the head coach will enter the following information:

- Player's first name
- Player's last name
- Player's date of birth (only player's under ten years of age can be part of the team)
- Player's preferred position (goalie, defender, midfielder, forward)
- Skill level, which is a number between 1 and 5 based on the coach's assessment (1 = lowest skill level, 5 = highest skill level)

Each player will also have a jersey number but it will be assigned to them once the team is created.

## 2. Creating the U10 soccer team

Once all the players' information has been entered, your software is ready to build the soccer team. These are the requirements:

- Team size: the team must have a minimum of 10 players and a maximum of 20. If there are less than 10 players, the user must be informed that the team cannot be created unless more players are added. If the team has more than 20 players, the ones with the lowest skill level most be ignored so that we only have 20 players.
- Jersey numbers: Each player gets a number for their jersey. The numbers must be between 1 and 20 (inclusive). Jersey numbers are unique, randomly assigned, and cannot be changed once created.
- Select the starting lineup. The starting lineup is the group of seven players who will start each game. These are usually the most skilled players in the team. If possible, these players will be assigned their preferred positions but there is no guarantee that this will happen. The details of how this starting lineup will be created are up to the software's developer.
- The remaining players are considered to be on the bench. They are not assigned positions.
- The coach should be able to get the following lists:
  - A list of all the players in the team. The following information must be provided for every player: first name, last name, jersey number. The list must be sorted in alphabetical order (last name).
  - A list of the starting lineup: The following information must be provided for every player: first name, last name, jersey number, and position. The list must be sorted by position (goalie, defender, midfielder, forward). Players with the same position should be ordered alphabetically.

# What you need to do for the first part of your project

This project will be built using an MVC architecture. For the first part of this project you will only be designing and implementing the model. The controller and the view will be implemented during the second part of the project.

## Part 1.1 - Design

Before you start to write code, it is a good idea to design your solution. To do this, you need to understand what your program needs to do, decide what classes you will need, and what methods each class will need. Thinking about this early will make the coding process much easier.

Design the model in a way that captures the similarities and accurately represents the relevant data. Create interfaces/classes as you see fit and specify appropriate constructors. You should also be sure to capture what methods and fields those classes have, the visibility of the methods and fields, and the relationships between them.

Create a single PDF containing a UML class diagram that captures all aspects of your design for your model. **Submit this here (https://northeastern.instructure.com/courses/141347/assignments/1793049)** before the deadline that is posted on Canvas.

Remember that design is an iterative process. This means that you can and should continue to refine your design to incorporate feedback as well as changes that are made as you implement your design. Keep an up-to-date design document in the res/ directory of your project.

## Part 1.2 - Implementation

Implement the classes that you specified in Part 1 along with any changes that were suggested during your design review. Do not forget to include your unit tests for all your classes.

Additionally, you should implement a *driver* class (containing a main method) that shows how to use your solution classes and is capable of creating at least one run of your program that communicates to us how to specify commands to run your program to verify that it meets all of the above specifications. This class does *not* need to rely on input from the keyboard.

Recall that we are expecting you to be using Java 11 as is required by Gradescope.

## Part 1.3 - Documentation

We expect your code to be well-commented using well-formed English sentences. The expectations are:

- Each interface and class contains a comment above it explaining specifically what it represents. This should be in plain language, understandable by anybody wishing to use it. Comment above a class should be specific: it should not merely state that it is an implementation of a particular interface.
- Each public method should have information about what this method accomplishes (purpose), the nature and explanation of any arguments, return values and exceptions thrown by it and whether it changes the calling object in any way (contract).
- If a class implements a method declared in an interface that it implements, and the comments in the interface describe this implementation completely and accurately, there is no need to replicate that documentation in the class.
- All comments should be in Javadoc-style.

## Part 1.4. Submission

Your zip file should contain three folders: src/, test/, and res/ (even if empty).

- All your code should be in src/. Keep your code well-organized by using packages.
- All your tests should be in test/. Your tests should be written using JUnit 4 as required by Gradescope.
- Your final design documents should be in res/ in PDF format.
- Submit at least one run of your program in a **simple text file** (in res/). You are free to submit multiple runs, but each run should be in a different file.
- Your zip file should also be free of IDE configuration files, compiled .class files, and other non-essential files that are automatically generated.

Autograder Results

Results | Code

Precheck) Style checking with Checkstyle (40/40)

Checkstyle PASSED

Project - Part 1

● Ungraded

Student
Jinda Zhang

Total Points
- / 100 pts

Autograder Score
- / 40.0

Passed Tests
Precheck) Style checking with Checkstyle (40/40)