EECS2030F: LAB 3 Due: Oct. 14, 2025 - 11:59 pm

The purpose of this lab is to ensure that you practice

A) creating a correct aggregation and composition relationship between objects.

B) implementing static factory methods.

C) Understanding object lifecycle and ownership in complex systems.

1. Lab Policies

a) **Academic Integrity**: Submit your own work. Do not copy code from classmates or online sources. All violations will be reported as academic misconduct.

b) **Submission Format**: Submit only the file: **League.java**. Do not upload ZIP files or project folders.

c) **Work Environment**: Complete the methods and test your submission by using tester file**.** Your code should not contain a main method and Scanner Class objects and its methods. Ensure your code compiles without errors.

d) **Deadline:** Submit your **League.java** file to the eClass course page by **Tuesday, October 14 (11:59pm).** No late submissions are accepted. Email submissions are not accepted.

e) Don't change the name of the file as we will not be able to test your code with our tester**.** It needs to be **League.java** (no case change)

f) Your lab assignment is not graded during the weekly lab sessions scheduled. The lab sessions are meant to get your questions answered from TAs.

2. Downloading and importing the Starter Project

You have already done this step in previous labs. Follow the below listed steps to set up your working environment by:

a) Download the Eclipse Java project archive file from eClass: EECS2030_Lab3.zip

b) Launch Eclipse and browse to EECS2030-workspace (for instance or your own created workspace).

c) In Eclipse:

- Choose File->Import
- Under General, choose Existing Projects into workspace
- Choose Select archive file. Browse your compressed zip folder and attach it.

- Make sure that the EECS2030_Lab3 box is checked under Projects and you don't have the same project already in the workspace. Then Finish.
- You should see two files, one is called **League.java** and one **LeagueTest.java**.

3. Important Notes:

To practice testing, we have only provided a set of incomplete test cases. Please make sure to add enough test cases to the tester that thoroughly tests your code. Look at the tester code, **LeagueTest.java**, and add more test cases to ensure comprehensive testing of your code. To do this, you can copy one of the methods, change the name of the method to avoid having a duplicate method name, and modify the body of the method to test your code with your selected input.

4. Javadoc generation

The Javadoc has been written for you. All you need to do is generate it as an HTML file to make navigation easier. To do this, click on the lab3 package, select Project -> Generate Javadoc. It will ask you for the location where you want to store the documentation. You can choose the default location or select your own folder. Enter the path, and then click on Finish. If you look at the location where you stored the documentation, you'll see a file called index.html. Clicking on this file will display the project documentation in your browser.

If you have any doubts on the Javadoc generation, please review lecture slides of week2.

5. Programming Tasks for this Lab

In this lab, you are going to implement the backbone of a Sports League Management System, focusing on the objects and their relationships. This league allows participants to create virtual teams by drafting real professional athletes. Participants compete based on the statistical performance of these athletes in actual games.

League Structure

This sports league is a competitive environment where 8 to 12 teams compete against each other throughout a season. Each league is managed by a commissioner who is responsible for setting up the league settings, resolving disputes between team owners, and ensuring fair play throughout the season. The league operates under a specific scoring format that determines how player

performances translate into fantasy points. This scoring format is established when the league is created and remains consistent for all teams throughout the entire season.

Team Structure

Each team in the league is owned and managed by a participant who acts as the team owner. The owner is responsible for making all strategic decisions regarding their team, including drafting players, setting lineups, and making trades. Every team has a unique team name chosen by the owner to represent their franchise, along with a logo color that provides visual identity within the league. The core of each team is its roster, which consists of professional athletes from real-world sports leagues. These athletes are the players who generate points for the team based on their actual performance in professional games. Throughout the season, teams accumulate points as their rostered athletes perform in real games, with these points being calculated according to the league's scoring format. The team's success depends entirely on the owner's ability to select high-performing athletes and make strategic decisions about which players to start each week.

Athlete Structure

Athletes in the fantasy sports system are professional players from major sports leagues such as the NFL (National Football League), NBA (National Basketball Association), NHL (National Hockey League), and other professional sports organizations. Each athlete is defined by several key characteristics that determine their value and usage in fantasy leagues. Every athlete has a specific position designation such as Quarterback, Running Back, Wide Receiver, Tight End, Kicker, or Defense, which determines where they can be played in a team's lineup. The system tracks detailed statistics for each athlete, which are updated on a weekly basis to reflect their most recent performances. These statistics include various performance metrics such as touchdowns scored, yards gained, receptions made, and other sport-specific achievements that contribute to their fantasy point totals.

Scoring Format

There are three primary scoring formats used in fantasy sports, each with its own point allocation system. The Standard scoring format awards 6 points for each touchdown scored by a player and 3 points for field goals made by kickers, with an additional 0.1 points awarded for every yard gained (whether rushing, receiving, or passing). The PPR (Points Per Reception) scoring format includes all the Standard scoring rules but adds an extra incentive for pass-catching players by awarding 1 full

point for every reception made, making wide receivers and pass-catching running backs more valuable. The Half-PPR scoring format takes a middle-ground approach, applying all Standard scoring rules while adding 0.5 points for each reception, creating a balanced system that values both yardage and receptions without over-emphasizing either statistic. The chosen scoring format fundamentally shapes draft strategy and roster management throughout the season.

Business Rules

The league operates under a specific set of business rules that govern team composition and league structure. Each team must maintain a roster consisting of a minimum of 10 athletes and a maximum of 16 athletes, ensuring that teams have enough depth to handle injuries. Within a single league, roster exclusivity is enforced, meaning that an athlete can only appear on one team's roster at any given time, preventing multiple owners from benefiting from the same player's performance. The league itself has size constraints, requiring a minimum of 8 teams to create sufficient competitive balance and a maximum of 12 teams to ensure adequate player availability during the draft. Once a league is created, its scoring format is permanently set and applies uniformly to all teams within that league, ensuring that every team owner is competing under identical rules and point calculations. These business rules create a fair, balanced, and competitive environment where strategic roster management and player evaluation skills determine success rather than arbitrary advantages or inconsistent scoring systems.

For this lab, you are going to implement only the constructors, static factory methods, setter and getter methods for the classes that are shown in the UML below. It is important that the relationships (i.e., aggregations and composition) are implemented correctly. Please note that there are some methods in this UML that we are not asking you to implement. They are only there to make the objects more understandable and the relationship clearer.

Please make sure that you read the UML carefully for private/public features and aggregation/composition relationships.

UML Diagram: See the Sports League UML (Provided)

Task 1: Implementing ScoringFormat class

For this task, you are required to implement the overloaded and copy constructor in addition to the static factory methods and setter and getter methods. The full documentation of the class and methods can be found in the starter code.

Important Design Note: ScoringFormat has private constructors. Objects can ONLY be created using the three static factory methods. This design enforces that only valid, pre-configured scoring formats can exist.

Task 2: Implementing League class

Three types of constructors (default, overloaded, and copy) and all setter and getter methods plus static factory methods are required. This is the main class.

Task 3: Implementing Athlete class

Two types of constructors (overloaded and copy) and all setter and getter methods plus static factory methods are required for this task.

Task 4: Implementing Team class

Two types of constructors (overloaded and copy) and all setter and getter methods plus static factory methods are required.

6. Testing your code

A test case file has been provided to evaluate your methods. Your implementation should pass all the test cases if it correctly handles both normal and edge cases. **Do not modify or update the provided test cases in any way. You should create your own test cases too to thoroughly test your code.**

7. Submit

Submit only one file named " **League.java** " via eClass by clicking on the lab link.

8. Marking Schema

You are graded based on the correctness of your code. For each method, there will be comprehensive test cases to examine the correctness of the code. All test cases carry the same weight.

Please note that in all the labs, even if the test cases are provided, we will test your code with a different set of test cases to ensure that you have tested your code completely.

Note:

- The program that does not compile will get zero marks.
- Don't change the method headers. The test cases will fail if you change the method headers.
- Your submission should strictly have the name as **League.java**, otherwise it can not be used to test your assignment. It will have zero grades.
- No resubmissions are allowed. Therefore, please make sure you submit the correct file within due date.

****