CS21120: Algorithm Design and Data Structure Assignment

Zeyad S M Moustafa

Zes3@aber.ac.uk

Table of Contents

Introduction

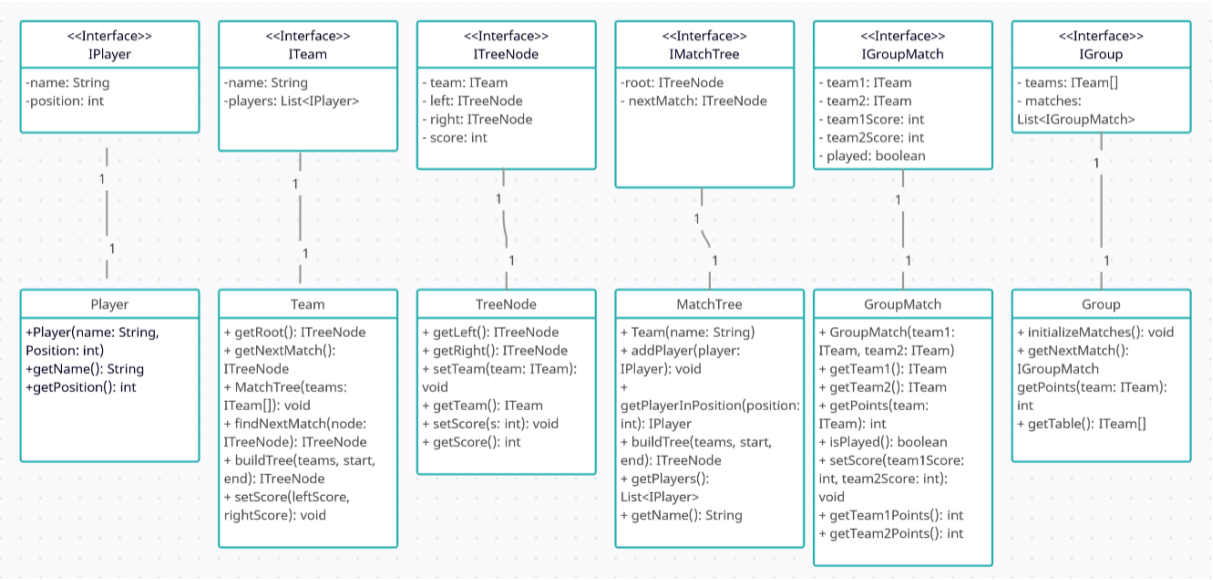
In the scope of this assignment, the goal was to design and implement a system for managing a team competition following the single-elimination format. This format, commonly known as "knockout" involves each round consisting of pairs of players or teams, with winners advancing to the next round. The implemented system comprises several Java classes conforming to provided interfaces (IPlayer, ITeam, IMatchTree, ITreeNode, IGroup, IGroupMatch).

This report outlines the development process and functionality of each class, discussing the challenges encountered, solutions implemented, and observance to the specified interfaces. The final part of the assignment involves extending the system to include a "group stage" where teams are divided into groups, playing each other before proceeding to the knockout stage.

The implemented classes include Player, Team, TreeNode, MatchTree, GroupMatch, Group, and Main. Each class serves a specific purpose in modelling the competition structure and interactions between players and teams. The report delves into the details of each class, highlighting design choices, algorithmic considerations, and positive implementations.

Design

Class Diagram



Tasks

I started the assignment by thoroughly reviewing the assignment brief, carefully understanding the instructions provided for each task. My next step involved setting up a new IntelliJ project, and afterwards, I organized the project structure by creating three distinct packages. This precise approach aimed to ensure clarity and maintainability throughout the development process.

With the base established, I systematically implemented each class, following closely the guidelines outlined in the assignment brief. The process involved focusing on one class at a time, progressively implementing the necessary functionality while referring to the specified interfaces and testing.

Task 1: Player

The implementation of this part proceeded smoothly without encountering significant difficulties. Adhering closely to the assignment brief and the IPlayer interface guidelines, the Player class was successfully implemented. The constructor validation checks and the implementation of essential methods, such as getName and getPosition, proceeded seamlessly. No major challenges were faced during this phase, and I anticipate receiving full marks for this task.

Task2: The Team Class

The implementation of the Team class proceeded smoothly. I followed the specifications of the ITeam interface, constructing the class with a name attribute and a List<IPlayer> for player storage. I picked for a List<IPlayer> due to its dynamic nature, allowing for flexibility in team compositions. The addPlayer, getPlayerInPosition, and getPlayers methods were implemented to facilitate the necessary functionalities. In terms of worst-case big-O time complexity:

Time Complexity:

- **The addPlayer method**: O(n) due to the linear search within the list.
- **The getPlayerInPosition method**: O(n) as it involves a linear search through the list.
- **The getPlayers method**: O(1) since it returns a copy of the player list, taking constant time.

This choice of data structure aims to balance efficient operations for adding and retrieving players while considering the expected size of teams in a competition setting. The implementation of the Team class posed no significant challenges.

Task3: Constructing the Match Tree

The implementation of the binary tree structure for organizing matches involved the creation of two distinct classes: TreeNode and MatchTree. The goal was to fit to the provided interfaces, namely ITreeNode and IMatchTree, respectively.

Time Complexity:

- **MatchTree Method:**
  - **Worst Case for Constructor :** O(n log n)
  - **Explanation**: The buildTree method employs a recursive approach to divide the array of teams into subarrays, constructing the binary tree in a logarithmic fashion. The overall time complexity is O(n log n) for the constructor, where n is the number of teams. The recursive process involves dividing the teams at each level of the tree.

Overall Impressions:

The implementation of the binary tree structure went smoothly. Unit tests were helpful in ensuring the correctness of both the TreeNode and MatchTree classes. The recursive construction of the tree was effective in handling various team scenarios. The time complexity of the constructor is reasonable for the given task, and the structure adheres to the requirements.

Task 4: Retrieving and Scoring Matches

getNextMatch() method

The getNextMatch() method is implemented to return the node representing the next match to be played. The conditions for a match to be playable include:

- The team has not been set.
- Both left and right child nodes exist.
- Both left and right child nodes have teams set.

Implementation*:*

1. The method recursively traverses the tree to find the deepest node that meets the playability conditions.
2. It returns null if all matches have been played.

setScore(int leftScore, int rightScore) Method:

The setScore method sets the scores for the match retrieved by getNextMatch. It throws a RuntimeException if there is no next match. The scores are set in the left and right child nodes, and the team of the next match is set to the winner.

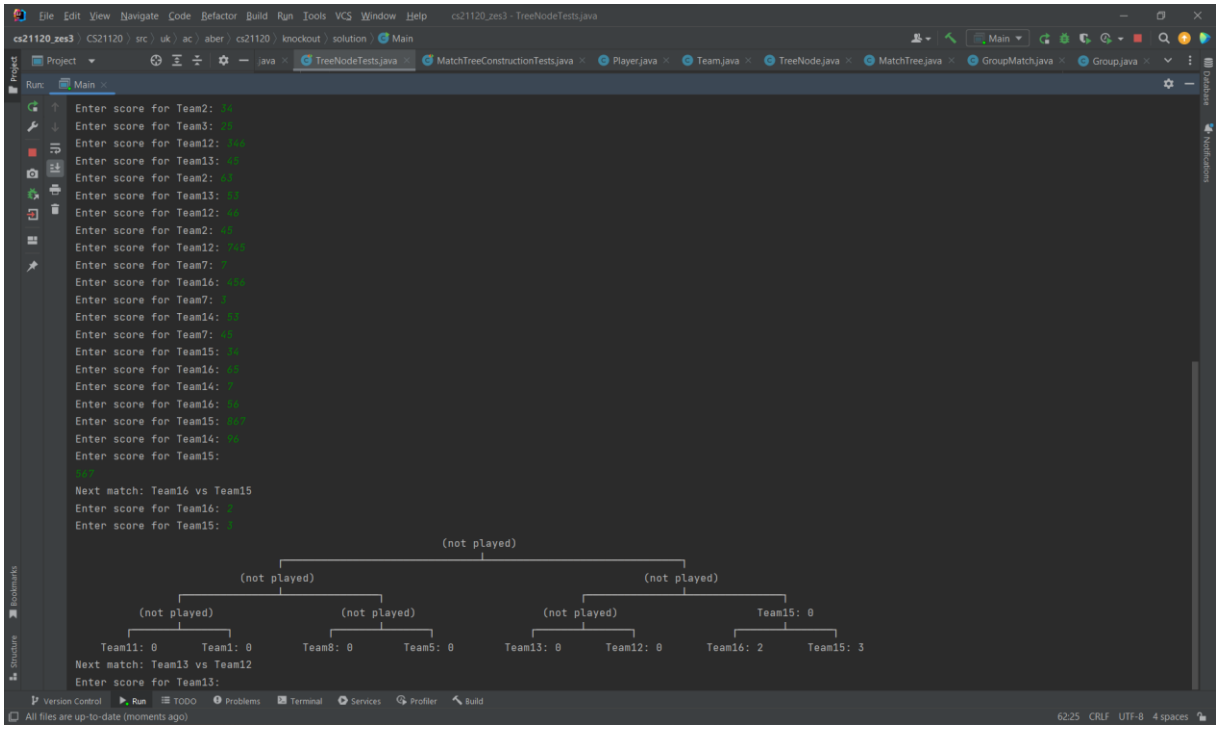
Implementation*:*

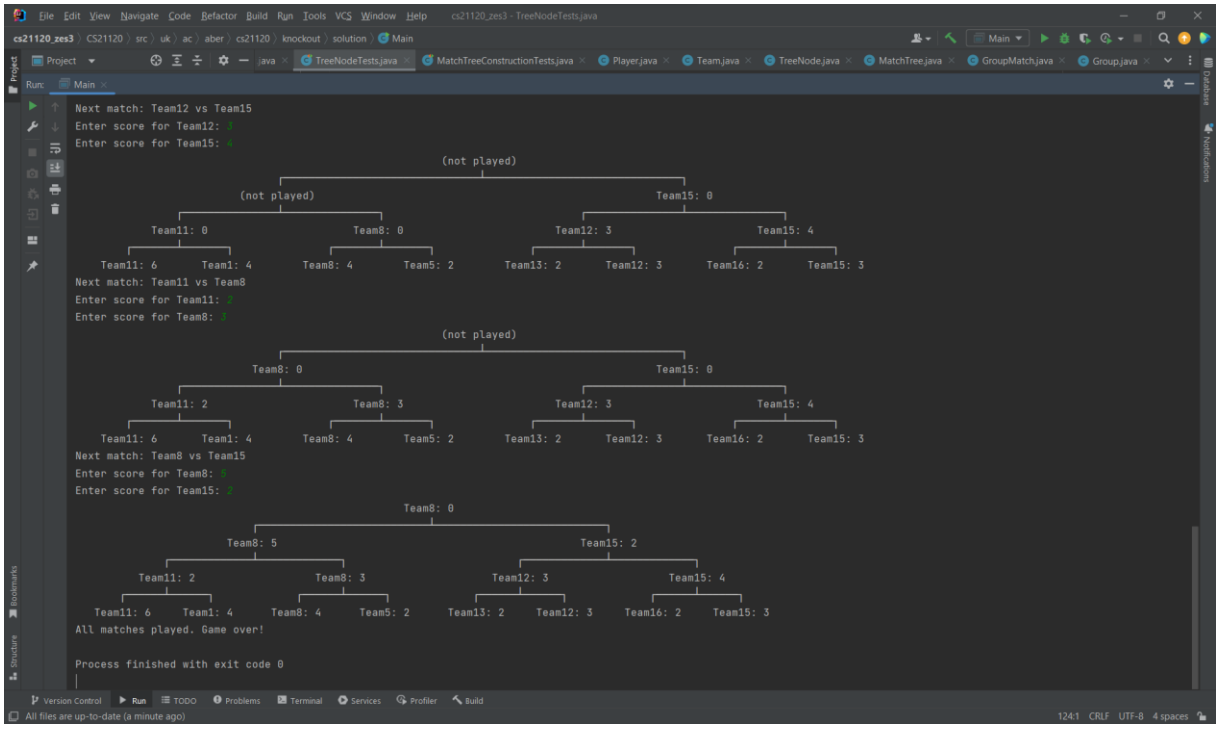
1. It checks if there is a next match available; otherwise, it throws a RuntimeException.
2. Scores are set in the left and right child nodes.
3. The team of the next match is set based on the winner.

Main Program:

The main program initializes a tree from an array of teams, simulating a manual testing environment. It displays each match, prompts the user for scores, sets the scores in the tree, and exits when all matches have been played. The TreePrinter class is used to display the tree at each stage.

Screenshots





Overall Impressions:

The implementation of both methods and the main program went smoothly. Unit tests provided confidence in the correctness of the code. The main program facilitated interactive testing, and the use of TreePrinter allowed visual confirmation of the tree's state at each stage. The program's output during manual testing matched the expected behaviour.

Task 5: The Group Stage

Challenges Faced:

In the implementation of the group stage, one notable challenge was the need to navigate through the existing interfaces and test cases without being able to modify them. This constraint required careful consideration of available methods and a thorough understanding of their functionality to implement the required features accurately.

Implementation:

GroupMatch Class*:*

The GroupMatch class implements the IGroupMatch interface, representing matches between two teams within a group. Key methods include those for getting and setting scores, as well as calculating points gained by each team.

- **Constructor (GroupMatch(ITeam team1, ITeam team2))**
  - Initializes a GroupMatch instance with two participating teams.

Group Class*:*

The Group class implements the IGroup interface, representing a group of teams in the group stage. This class manages the teams and organizes the matches they play.

- **Constructor (Group(ITeam[] teams))**
  - Initializes a Group instance with an array of teams.
- **Playing Matches (void playMatches(Scanner scanner))**
  - Utilizes user input to play matches between teams in the group.
  - Ensures that each team plays every other team once in a mini tournament.
- **Determining Finalists (List<ITeam> determineFinalists())**
  - Calculates the points for each team based on match outcomes.
  - Resolves ties randomly if teams have the same number of points.
  - Selects the top two teams as finalists.

Match Generation and Tie Resolution:

- **Match Sequence:**
  - Teams were randomly divided into four groups, simulating a mini tournament within each group.
  - User input facilitated the playing of matches, generating a sequence based on the round-robin format.
- **Tie Resolution:**
  - Ties were resolved randomly, aligning with the specified assignment instructions.
  - In instances where teams shared the same point total, a random selection was made to determine the order of finalists.

Calculating Matches Required:

For the group stage with 16 teams and 4 groups, the calculation for the number of matches required was simplified. Then each team will play the other teams in the group once so during the group stages each team will play 3 games.

Self-Evaluation

The most challenging aspect was dealing with the constraints of not being able to modify existing interfaces in Task 5. Navigating through available methods and understanding their functionality required a comprehensive examination. However, I believe I handled this challenge effectively. Overall, the implementation of various classes and functionalities demonstrates a solid understanding of the concepts. I spot the need for further optimization and readability developments, but I would expect to receive a grading of 80% on this assignment as all tasks have been completed.