Supervised by : Dr.Laith Shehab

Prepared by : Ahmad Al-bishawi & Mohammad Al-shareef

Documentation for Football Team Management System

Package :

- `package main1;`

Imports :

- `import java.util.ArrayList;`
- `ArrayList` is used in the `Team` class to maintain a list of `Player` objects.

- `import java.util.Scanner;`
- In the `Main` class, `Scanner` is used to read user input for team details, player details, coach details, stadium details, and other menu choices.

Classes :

1. **Player (Main Class)**

- Represents a generic player with attributes like name, salary, contract length, and jersey number.
- Abstract method: `getPlayerType()` which must be implemented by subclasses.
- Constructor: Initializes name, salary, contract length, and jersey number.
- Getters: Provide access to the player's attributes.

1. **Goalkeeper (Derived Class)**
   - Extends `Player`.
   - Additional attribute: `cleanSheets`.
   - Constructor: Initializes player's attributes along with clean sheets.
   - Getter: `getCleanSheets()`.
   - Overrides `getPlayerType()`: Returns "Goalkeeper".

2. **Defender (Derived Class)**
   - Extends `Player`.
   - Additional attribute: `tackles`.
   - Constructor: Initializes player's attributes along with tackles.
   - Getter: `getTackles()`.
   - Overrides `getPlayerType()`: Returns "Defender".

3. **Midfielder (Derived Class)**
   - Extends `Player`.
   - Additional attribute: `assists`.
   - Constructor: Initializes player's attributes along with assists.
   - Getter: `getAssists()`.
   - Overrides `getPlayerType()`: Returns "Midfielder".

4.**Forward** (**Derived Class)**
   - Extends `Player`.
   - Additional attribute: `goals`.
   - Constructor: Initializes player's attributes along with goals.
   - Getter: `getGoals()`.
   - Overrides `getPlayerType()`: Returns "Forward".

Other Classes :

6. **InvalidPlayerTypeException**

- Custom exception for handling invalid player type errors.

7. **InvalidInputException**

- Custom exception for handling invalid input errors.

8. **Coach**

- Represents a coach with attributes like `name`, `nationality`, and `age`.

9. **Stadium**

- Represents a stadium with attributes like `name`, `location`, and `capacity`.

10. **Team**

- Represents a team with attributes like `name`, `owner`, `value`, `coach`, `stadium`, `cups`, and a list of `players`.

Main Class for Execution:

11. **Main**

- The entry point of the program.
- Manages user input and interaction, allowing the user to add players, coach, and stadium to the team, set the number of cups, and view team details.

This structure ensures that common attributes and behaviors are defined in the `Player` class, while specific details and behaviors are implemented in the respective derived classes (`Goalkeeper`, `Defender`, `Midfielder`, `Forward`).

Program Flow:

1. **Initialization**

The program starts by greeting the user and prompting for team details.

2. **Main Menu**

- Options provided:

1. Add Player
2. Add Coach
3. Add Stadium
4. Add Cups
5. View Team Details
6. Exit

3. **Add Player**

- Prompts for player details including type.

4. **Add Coach**

- Prompts for coach details and sets the coach for the team.

5. **Add Stadium**

- Prompts for stadium details and sets the stadium for the team.

6. **Add Cups**

- Prompts for the number of cups won and updates the team's record.

7. **View Team Details**

- Displays all team details including players, coach, stadium, and number of cups.

8. **Exit**

- Exits the program.

Functional Requirements :

1. **User Input Management**
2. **Player Management**
3. **Coach Management**
4. **Stadium Management**
5. **Team Management**
6. **Custom Exception Handling**:

- The system should handle invalid player types using `InvalidPlayerTypeException`.
- The system should handle invalid inputs using `InvalidInputException`.

7. **Program Execution Flow**:

- The system should provide a main menu with options to add a player, coach, stadium, cups, view team details, and exit the program.
- The system should display team details including players, coach, stadium, and the number of cups won when requested.

Non-Functional Requirements :

1. **Usability**:

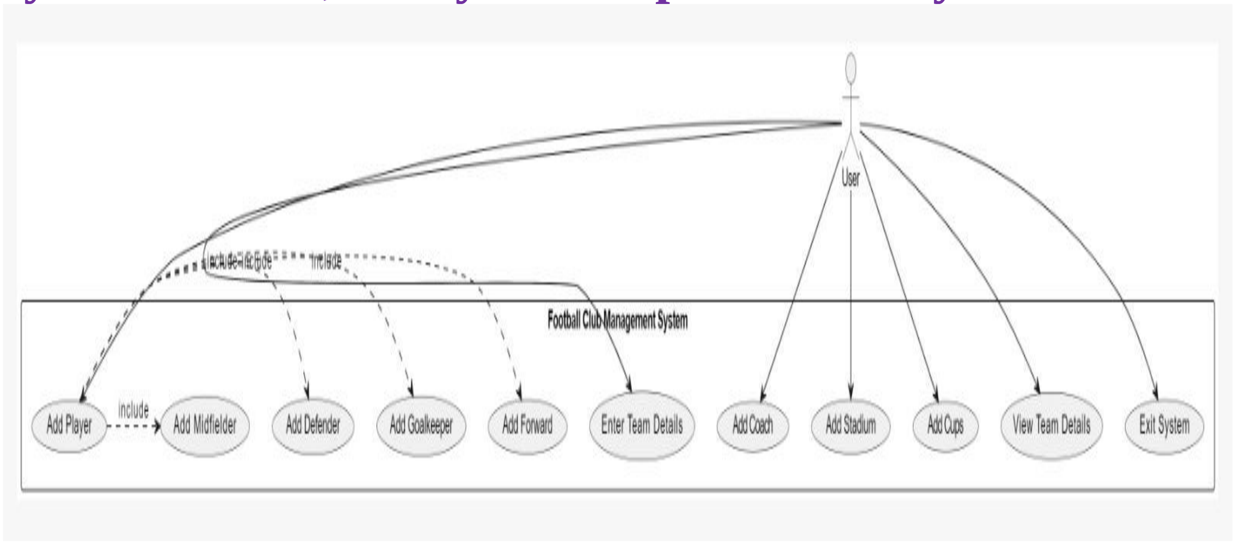
- The user interface (console-based) should be intuitive and easy to navigate.

2. **Performance**:

- The system should quickly process user inputs and display outputs without noticeable delay.

3. **Security**

This is a simple use case diagram to show how the system works , the system depends mainly on the user.



NOTE : *You can find how to use the system with GUI in the presentation and thank you ☺*