Object-Oriented Programming – Exercises Week 11

This is the final week of programming exercises. We will be programming an Android application once again!

This will be the most complicated exercise you have to program on this course. So, get ready!

Our goal is to create a Soccer Team Management App that utilizes generics, iterators, and lambda functions. Your app will display lists of teams, players, and matches using hardcoded data, with sophisticated filtering and sorting capabilities.

The learning goal for the exercises

1. Implementing and using generic classes in Android
2. Working with Java's Iterator pattern
3. Using lambda expressions for efficient data processing
4. Applying object-oriented principles in Android development

On returning the exercise to Moodle

This time, the exercises must be submitted to Moodle for peer review, not to CodeGrade for automatic evaluation.

You must submit two things:

1) Link to a public GitHub (or any other public Git) repository containing your solution code
2) Link to a video, where you explain your code and demonstrate your solution

Both of these must be submitted. No late submissions are accepted. You are not allowed to change your Git code after the submission. Having commits after the deadline leads to automatic 0 points from the exercises.

Points Received

2 points for proper implementation (minus points for missing something or unclear explanations)

1 points for evaluating two other solutions

Requirements

1. Generic Data Repository

- Create a generic `Repository<T>` class that can manage collections of any entity type
- Implement methods for adding, retrieving, and filtering items
- Use proper bounded type parameters where appropriate (e.g., `<T extends DataModel>`)
- Include error handling for invalid data operations

2. Soccer Entity Classes

- Create model classes for `Team`, `Player`, and `Match` that extend a common `SoccerEntity` interface
- Implement custom iterators for collections of these entities (e.g., `TeamIterator`, `PlayerIterator`)
- Include proper validation and error checking for all fields

3. Data Provider with Generics

- Create a `DataProvider<T>` class that serves hardcoded data for the application
- Implement different methods to retrieve teams, players, and matches
- Use Java generics to make the utility reusable across different entity types

4. Lambda Function Implementation

- Implement search/filter functionality using lambdas
- Create sort operations using Comparator with lambda expressions
- Use streams API with lambda expressions to transform data collections
- Implement click handlers and callbacks with lambda expressions

Implementation Guidelines

1. Use a clean architecture approach with proper separation of concerns
2. Include at least 2 uses of lambda expressions in different contexts
3. Create at least 1 custom iterator implementations
4. Implement at least 2 generic classes or methods
5. Use the provided sample data or create your own

Main Activity

Create a MainActivity that:

1. Sets up a simple TabLayout with fragments for Teams, Players, and Matches
2. Implements a search bar that uses lambdas for filtering content
3. Includes sorting options that leverage lambda expressions
4. Demonstrates the use of your generic repository classes

Expected Output

Your app should:

1. Display lists of soccer teams, players, and matches
2. Provide filtering capabilities using lambda expressions
3. Allow sorting of data using lambda-based comparators
4. Demonstrate the use of custom iterators for data traversal

Evaluation Criteria

Your solution will be evaluated based on:

1. Correct implementation of generics
2. Proper usage of iterators
3. Effective use of lambda expressions
4. Clean architecture and code organization
5. UI design and user experience
6. Video presentation explaining your implementation choices

Example data

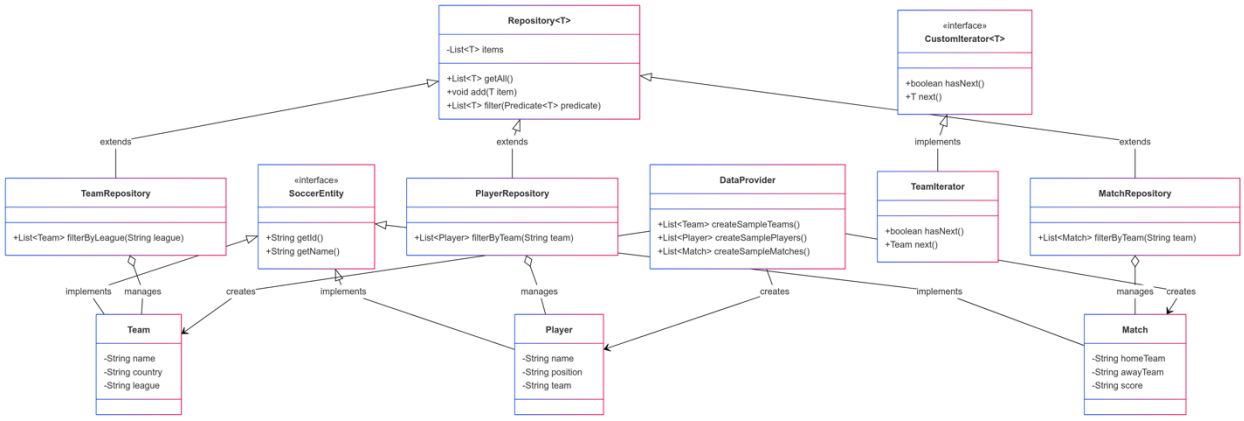
```
// Example Teams Data
List<Team> createSampleTeams() {
    List<Team> teams = new ArrayList<>();
    teams.add(new Team("FC Barcelona", "Spain", "La Liga", "Camp Nou", 1899));
    teams.add(new Team("Manchester United", "England", "Premier League", "Old Trafford", 1878));
    teams.add(new Team("Bayern Munich", "Germany", "Bundesliga", "Allianz Arena", 1900));
    teams.add(new Team("Juventus", "Italy", "Serie A", "Allianz Stadium", 1897));
    teams.add(new Team("Paris Saint-Germain", "France", "Ligue 1", "Parc des Princes", 1970));
    teams.add(new Team("Ajax Amsterdam", "Netherlands", "Eredivisie", "Johan Cruyff Arena",
1900));
    teams.add(new Team("River Plate", "Argentina", "Primera División", "El Monumental", 1901));
    teams.add(new Team("Flamengo", "Brazil", "Brasileirão", "Maracanã", 1895));
    return teams;
}

// Example Players Data
List<Player> createSamplePlayers() {
    List<Player> players = new ArrayList<>();
    players.add(new Player("Lionel Messi", 34, "Argentina", "Forward", "FC Barcelona", 10));
    players.add(new Player("Cristiano Ronaldo", 36, "Portugal", "Forward", "Juventus", 7));
    players.add(new Player("Robert Lewandowski", 32, "Poland", "Forward", "Bayern Munich", 9));
    players.add(new Player("Kevin De Bruyne", 29, "Belgium", "Midfielder", "Manchester City",
17));
    players.add(new Player("Virgil van Dijk", 30, "Netherlands", "Defender", "Liverpool", 4));
    players.add(new Player("Manuel Neuer", 35, "Germany", "Goalkeeper", "Bayern Munich", 1));
    players.add(new Player("Kylian Mbappé", 22, "France", "Forward", "Paris Saint-Germain", 7));
    players.add(new Player("Erling Haaland", 20, "Norway", "Forward", "Borussia Dortmund", 9));
    players.add(new Player("Bruno Fernandes", 26, "Portugal", "Midfielder", "Manchester United",
18));
    players.add(new Player("Joshua Kimmich", 26, "Germany", "Midfielder", "Bayern Munich", 6));
    players.add(new Player("Jan Oblak", 28, "Slovenia", "Goalkeeper", "Atletico Madrid", 13));
    players.add(new Player("Neymar Jr.", 29, "Brazil", "Forward", "Paris Saint-Germain", 10));
    return players;
}

// Example Matches Data
List<Match> createSampleMatches() {
    List<Match> matches = new ArrayList<>();
    matches.add(new Match("FC Barcelona", "Real Madrid", "2-1", "La Liga", "2023-04-10", "Camp
Nou"));
    matches.add(new Match("Manchester United", "Liverpool", "0-3", "Premier League", "2023-03-
15", "Old Trafford"));
    matches.add(new Match("Bayern Munich", "Borussia Dortmund", "4-2", "Bundesliga", "2023-04-
01", "Allianz Arena"));
    matches.add(new Match("Juventus", "AC Milan", "1-1", "Serie A", "2023-03-20", "Allianz
Stadium"));
    matches.add(new Match("Paris Saint-Germain", "Lyon", "3-0", "Ligue 1", "2023-04-05", "Parc
des Princes"));
    matches.add(new Match("FC Barcelona", "Bayern Munich", "0-3", "Champions League", "2023-02-
28", "Camp Nou"));
    matches.add(new Match("Manchester City", "Paris Saint-Germain", "2-1", "Champions League",
"2023-03-08", "Etihad Stadium"));
    matches.add(new Match("Liverpool", "Ajax Amsterdam", "1-0", "Champions League", "2023-03-01",
"Anfield"));
    return matches;
}
```

Example Class Diagram

Here is an a simplified example class diagram on how to get started. Feel free to do it in a different way, just as long as it implements the requirements



(See the bigger version in the course Moodle page)