eSports Tournament Management System

We want to develop an application to manage an **eSports tournament** using object-oriented programming. To do this, we will create an IntelliJ project called **eSportsTournament**. Inside the source folder, you must define two packages:

- **tournament.main**: This package will contain the main class of the system, called `Main`, and an additional class called `TournamentManager`.
- **tournament.data**: This package will store all the classes and interfaces needed to manage the tournament data.
- **tournament.comparators**: This package will store the comparators used for showing information ordered.
- **tournament.exceptions**: This package will store the personalized exception we will use.

1. Class Structure

Abstract Class: Tournament

Each tournament has a **name** and an **associated game** (e.g., "League of Legends", "CS:GO", "Valorant")and a **prize** (amount in euros). Define a **parameterized constructor**, the corresponding **getters and setters**, and override the `toString()` method to return the tournament name along with the game.

Example output:
`Tournament – Masters League (League of Legends) – Prize: 6000€`

From this abstract class, we will define three different tournament types:

1.1. Tournament Subclasses

- **`IndividualTournament`**: Represents tournaments where players compete individually.
- **`TeamTournament`**: Represents tournaments where players compete in teams. It must include an attribute `playersPerTeam`.
- **`MixedTournament`**: Represents tournaments that allow both individual and team participation. It must include an attribute `gameMode` (can be "1v1" or "5v5").

Each of these classes must override `toString()` to display the tournament type, its name, game, and additional attributes.

Example output:
`IndividualTournament – Masters Solo (CS:GO) – Prize: 5000€`

1.2. Abstract Class: Participant

Each participant has:

- `name`

Implement getters, setters and constructor.

1.2.1. Player Class

Each player has the following aditiontal attributes:

- `level` (a number between 1 and 100)
- `ranking` (a floating-point number representing their score)

Override `toString()` to display the username, level, and ranking.

Example output:
`Player: ShadowReaper – Level: 85 – Ranking: 2400.5`

1.2.2. Team Class

Each team has as addtional attribute:

- An **array of players** with a minimum of 2 and a maximum of 5 players.

Define a method `addPlayer(Player p)` to add a player to the team if it has not reached the limit. If we cannot add a new player because there are already 5 components in the team, we will throw a **personalized exception** called FullTeamException that will be catched in Main Class. Define also constructor, getters and setters and ToString().

Example output:
`Team Phoenix – Members: 3/5: – Player: ShadowReaper – Level: 85 – Ranking: 2400.5 – Player: AceGamer – Level: 100 – Ranking: 2500`

1.3. Match Class

Each match has:

- An **associated tournament**
- Two **participants** (players or teams) (Participant1 and Participant2)
- A `result` (default value: "Pending")

Define a method `setResult(String result)` to update the match result.

Example output:
`Match in Masters League (LoL) – ShadowReaper vs. FireDragon – Result: Pending`

---

2. Tournament Logic

In the `TournamentManager` class, implement the following:

- An **array of registered players** (minimum of 10).
- An **array of registered teams** (minimum of 5).

- An **array of tournaments** with at least 3 tournaments of different types.
- An **array of matches** generated randomly.

Additionally, the class must include the following methods and every other method you consider useful:

- `void initialize()` → Populates the arrays with predefined data.
- `Player findPlayer(String username)` → Searches for a player by username.
- `Team findTeam(String teamName)` → Searches for a team by name.
- `void showTournaments()` → Displays all available tournaments.
- `void showPlayerRanking()` → Displays players sorted by ranking.
- `void showTeamRanking()` → Displays teams sorted by average player ranking.

---

3. Main Menu in `Main` Class

The menu should allow the user to:

1. **View available tournaments ordered by name**
2. **View players information ordered by ranking and name**
3. **View teams information ordered by ranking**
4. **Add a new player to a team**
5. **Find an exact player by name**
6. **Find players**
7. **Find teams**
8. **Show all the matches ordered by tournament name**
9. **Update the result of the matches pending**
10. **Exit**

Every ordered list (Options 1,2,3,8)

For each of these options you need to:

- Use a class that implements Comparator
- Use an anonymous class
- Use a lambda expression (only Mari Chelo's group)

The result of these options will be shown 2 times (3 for Mari Chelo's group)

Options 6 and 7

We have to ask the user a text and then we will show all the players (or teams) that have some coincidence in their name with the text.

Update Results (Option 9)

The `inputResult()` method will show in console all the pending matches. After that, the user will pick one of them and then we will ask them the result of that match, and we will update the result attribute.

Error handling

Deal with every possible error caused by a not adecuate input from the user. Ranking below 0, etc...

5. Deliverables

Deliver the complete project compressed in a ZIP file. Ensure that you include good comments in the code and follow best coding practices.