# Tournament Management System

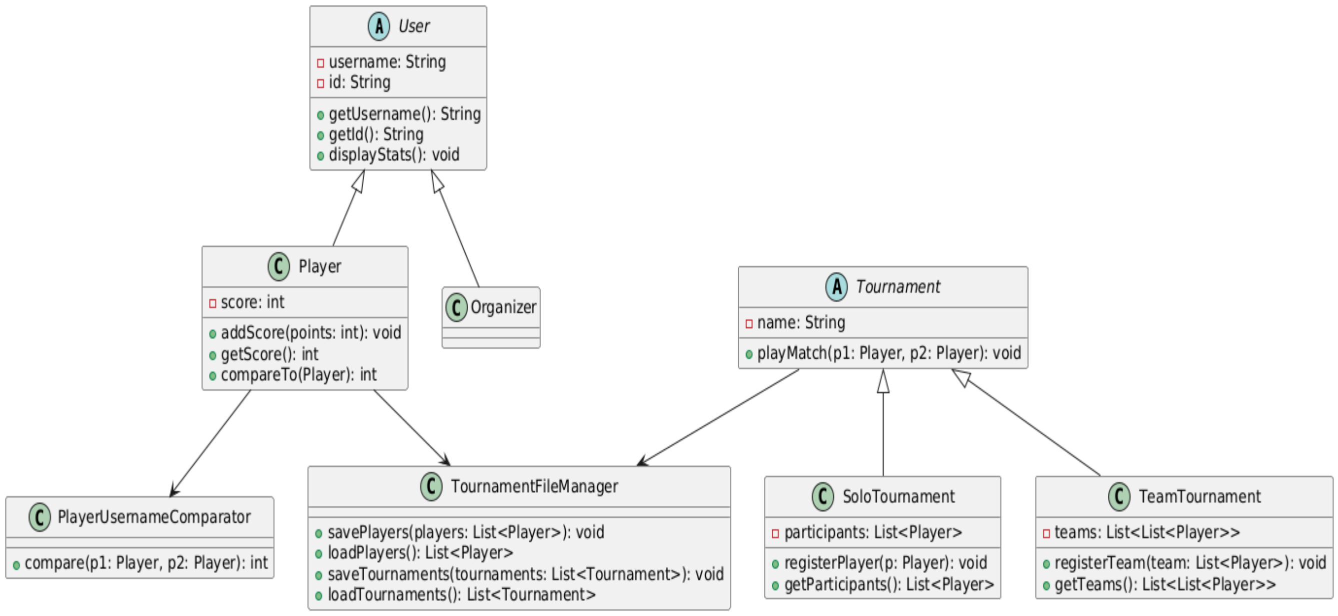
**Name:** Lenny Manset
**Course:** 420-SF2-RE Data Structures and Object-oriented Programming
**Instructor:** Yi Wang
**Date:** May 11th 2025

**Table of Contents**

## Project Description

**Scenario:**
This project simulates a **Tournament Management System** where organizers can manage tournaments, and players can participate in matches. The system allows for both solo and team tournaments, keeping track of participants, scores, and basic match logic.

**Design Paradigm / Functionalities:**



A User
username: String
id: String
getUsername(): String
getId(): String
displayStats(): void
C Player
score: int
addScore(points: int): void
getScore(): int
compareTo(Player): int
C Organizer
A Tournament
name: String
playMatch(p1: Player, p2: Player): void
C PlayerUsernameComparator
compare(p1: Player, p2: Player): int
C TournamentFileManager
savePlayers(players: List<Player>): void
loadPlayers(): List<Player>
saveTournaments(tournaments: List<Tournament>): void
loadTournaments(): List<Tournament>
C SoloTournament
participants: List<Player>
registerPlayer(p: Player): void
getParticipants(): List<Player>
C TeamTournament
teams: List<List<Player>>
registerTeam(team: List<Player>): void
getTeams(): List<List<Player>>

- **Class hierarchy:**
  - User (abstract) ➔ Player, Organizer
  - Tournament (abstract) ➔ SoloTournament, TeamTournament
- **Interface:**

  - Borrowable was adapted as MatchPlayable (if applicable in your code base).
- **Polymorphism:**
  - playMatch() is overridden differently in SoloTournament and TeamTournament.
- **Data structures:**
  - ArrayList, List<List<Player>>, etc., to manage players and teams.
- **File handling:**
  - Save and load players.txt and tournaments.txt.
- **Comparable & Comparator:**
  - Player implements Comparable<Player> for sorting by score.
  - PlayerUsernameComparator sorts by username alphabetically.
- **JUnit Testing:**
  - Full unit tests covering players, tournaments, file I/O, and edge cases.
- **Null Safety:**
  - Full handling of null/empty players, teams, and scores.
- **Git Repository:**
  - Version-controlled via Git, structured with /doc and code folders.

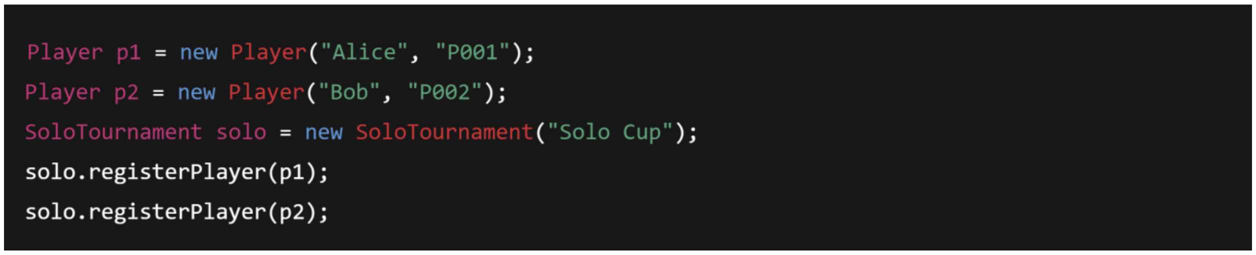
---

**Program Features and Screenshots**

**Register Players and Teams:**

- Players can be created with a username and ID.

  Input



```
Player p1 = new Player("Alice", "P001");
Player p2 = new Player("Bob", "P002");
SoloTournament solo = new SoloTournament("Solo Cup");
solo.registerPlayer(p1);
solo.registerPlayer(p2);
```

Output

```
Player 'Alice' registered to Solo Tournament 'Solo Cup'.
Player 'Bob' registered to Solo Tournament 'Solo Cup'.
```

- Teams are registered as lists of players.

  Input

```
Player t1 = new Player("Team1_Player1", "T001");
Player t2 = new Player("Team1_Player2", "T002");
List<Player> team = Arrays.asList(t1, t2);
TeamTournament teamTournament = new TeamTournament("Duo Clash");
teamTournament.registerTeam(team);
```

  Output

```
Team of 2 players registered to Team Tournament 'Duo Clash'.
```

**Play Matches:**

- Solo matches award 10+5 points to the player(s).

  Input

```
solo.playMatch(p1, p2);
```

  Output

```
Match Result: Alice receives 15 points, Bob receives 5 points.
```

```
p1.getScore() // returns 15
p2.getScore() // returns 5
```

- Team matches award 5+5 points to team captains.

  Input

```
teamTournament.playMatch(t1, t2);
```

Output

```
Team Match: Captains Team1_Player1 and Team1_Player2 awarded 10 points each.
```

```
t1.getScore() // returns 10
t2.getScore() // returns 10
```

**Example console output:**

```
Solo match between Alice and Bob: Alice +15 points, Bob +5 points
Team match between Team1 and Team2: Captains awarded 10 points each
```

**Save & Load:**

- Players and tournaments are saved to files (players.txt, tournaments.txt).
- Supports safe loading with missing/invalid data gracefully handled.

Example file

```
Alice,P001,15
Bob,P002,5
Charlie,P003,
```

```
TournamentFileManager tfm = new TournamentFileManager();
List<Player> players = tfm.loadPlayers();
```

```
Loaded player: Alice with score 15
Loaded player: Bob with score 5
Missing or blank score for: Charlie, setting score to 0
```

**Sorting:**

- Players are sorted by score (descending).
- Alternative sorting by username using PlayerUsernameComparator.

**Example output:**

```
Players sorted by score:
- Alice: 25 pts
- Bob: 10 pts

Players sorted by name:
- Alice
- Bob
```

**Unit Testing:**

- Tests for adding scores, registering participants, file I/O, comparator logic, and null safety.

---

**Challenges**

**Null Handling:**

- Early tests and matches failed with NullPointerException when players were null.
- Fixed by adding if (p1 != null) guards everywhere in match logic.

**File Path Issues:**

- Test files (Player.csv, etc.) were mismatched with production files (players.txt).
- Standardized file paths across code and tests.

**Score Parsing:**

- Crashes occurred when parsing empty or malformed score strings.
- Added strong guards in TournamentFileManager to skip bad data and default scores to 0.

**Deserialization of Tournaments:**

- I Initially left as a placeholder, which led to problems. After  expanding it  later it allowed me to  parse tournament types properly.

---

**Learning Outcomes**

- **Polymorphism & Inheritance:**
  - Stronger understanding of how class hierarchies and overridden methods work.
- **File Handling:**
  - Learned to handle I/O with robust error-checking (null-safe, malformed data).
- **Unit Testing:**
  - Designed reliable tests that cover both valid and edge cases (null/empty).
- **Defensive Coding:**
  - Improved ability to anticipate and handle bad input (nulls, blanks, invalid numbers).
- **Version Control & Project Structure:**
  - Practiced working with Git, organized project with a /doc folder for reports.