UML Class Structure Outline

This system consists of multiple components, categorized as follows.

Visibility Modifiers:

- \+ **Public** → Accessible by all classes.
- \- **Private** → Accessible only within the class.
- # **Protected** → Accessible within the class and subclasses.
- ~ **Package-Private** → Accessible within the same package.

---

1. Core Classes

1.1 GeneralStats (Abstract Parent Class)

Responsibilities:

- Acts as a **base class** for all game statistics.
- Contains **universal statistics** applicable to all board games.
- Provides generic methods (win(), lose(), tie()) that child classes override.
- Handles **MMR updates** and **rank updates** dynamically.

Attributes (Instance Variables):

- # playerId: String
- # wins: int
- # losses: int
- # ties: int
- # gamesPlayed: int
- # mmr: int
- # rank: Rank **(NEW: Stores player's current rank)**

Methods:

- + win(): void
- + lose(): void
- + tie(): void
- + get_mmr(): int
- + get_wins(): int
- + get_losses(): int
- + get_ties(): int

- # updateMMR(win: boolean): void (abstract) **(Calls updateRank())**
- # updateRank(): void **(NEW: Adjusts rank after MMR change)**

1.2 Game-Specific Stats (Child Classes)

Each game (Connect4, Tic Tac Toe, Checkers) extends GeneralStats, adding game-specific attributes and logic.

1.2.1 Connect4Stats

Attributes:

- - piecesDropped: int

Methods:

- + recordMove(column: int): void
- + getMoves(): int
- # updateMMR(win: boolean): void

1.2.2 TicTacToeStats

Attributes:

- - movesMade: int

Methods:

- + recordMove(): void
- + getMoves(): int
- # updateMMR(win: boolean): void

1.2.3 CheckersStats

Attributes:

- - piecesCaptured: int
- - movesMade: int

Methods:

- + recordMove(): void

- + getMoves(): int
- # updateMMR(win: boolean): void

---

2. Ranking System

2.1 Rank (Enum)

Responsibilities:

- Stores **universal ranking tiers**.
- Ensures **consistent ranking levels across all games**.

Values (Enum Constants):

- + BRONZE
- + SILVER
- + GOLD
- + PLATINUM
- + DIAMOND
- + MASTER
- + GRANDMASTER

Methods:

- + getRank(mmr: int, minMMR: int, maxMMR: int): Rank
    - Takes **mmr**, **minMMR**, and **maxMMR** to compute the **player's rank dynamically**.

---

3. Player Profile

3.1 Player

Responsibilities:

- Represents a **player profile** storing statistics for multiple games.

Attributes:

- + playerId: String
- + connect4Stats: Connect4Stats
- + ticTacToeStats: TicTacToeStats

- + checkersStats: CheckersStats

Methods:

- + getStats(gameType: String): GeneralStats

---

4. Matchmaking System

4.1 MatchmakingSystem

Responsibilities:

- Handles **rank-based queue system** for matchmaking.
- Moves players **down to lower-ranked queues** after a wait time.

Attributes:

- - queues: Map<int, List<Player>> (14 ranks, 2 queue pairs each)

Methods:

- + addPlayerToQueue(player: Player): void
- + matchPlayers(): List<Pair<Player, Player>>
- + waitAndMoveDown(player: Player): void
- + removePlayer(player: Player): void

---

5. Leaderboard System

5.1 LeaderboardManager

Responsibilities:

- Tracks and sorts **player statistics**.
- Stores data in **CSV or simulated database**.

Attributes:

- - leaderboardData: List<Player>

Methods:

- + updatePlayer(player: Player): void
- + getTopPlayers(by: String, topN: int): List<Player>

---

6. UML Relationship Types

Using **UML standards** and referencing the **provided images**, the relationships between classes will follow:

1. Inheritance (Generalization)

- GeneralStats **(Abstract Parent Class) → Connect4Stats, TicTacToeStats, CheckersStats** (Concrete Subclasses).
- MatchmakingSystem and LeaderboardManager **operate separately** but interact with Player.

2. Aggregation/Composition

- Player **composes** multiple GameStats objects. These stats cannot exist without a player
- LeaderboardManager **aggregates** multiple Players, to collect info on each of their stats.

3. Dependency

- GeneralStats **depends** on Rank for the updateRank()
- MatchmakingSystem also depends on Player, as it needs to move around, players, and match them up using their stats information.