### **TEAM MANAGEMENT SYSTEM IN THE CHAMPIONS LEAGUE**

**PHASE 1: IDENTIFICATION OF THE PROBLEM**

**Identification of Needs and Symptoms**

* **Real needs:** The UEFA organizers require an automated solution to manage the teams, matches, and rankings of the Champions League efficiently.
* **Conditions:** The system must allow for team and match registration, manage rankings efficiently, and have the capability to undo errors in actions quickly and easily.
* **Symptoms:** The lack of an automated and efficient system to handle the vast amount of data generated by each match, team rankings, and real-time updates.

**Problem Definition**

UEFA needs a software system to manage the teams, matches played, and rankings of teams in the Champions League, ensuring that the information is processed efficiently, accurately, and is easily accessible.

**SPECIFICATION OF REQUIREMENTS**

**FUNCTIONAL REQUIREMENTS**

1. Team Registration: a. Allow the registration of teams with their respective attributes in the system.
2. Match Registration: a. Allow registering the results of the matches played between teams.
3. Undo Action: a. Use the stack to revert the last action executed in the system.
4. Match Management: a. Organize the matches in a queue to establish the order of play for each matchday.
5. Ranking and Standings: a. Manage the ranking of teams in the Champions League using a priority queue based on the points earned and UEFA coefficient.

**NON-FUNCTIONAL REQUIREMENTS**

1. **Efficiency:** The system must guarantee fast response times, especially in querying and updating the data for teams and matches. Search, insert, and update operations should be performed optimally using data structures like hash tables and priority queues.
2. **Scalability:** The system must be capable of managing a growing number of teams, matches, and rankings without significant performance degradation.
3. **Reliability:** The system must be reliable with minimal failures. In case of errors, it should allow users to undo the last action performed.
4. **Security:** The system must ensure that only authorized users can access and modify the team and match records. User authentication should be implemented for data management operations.
5. **Maintainability:** The system should be designed to facilitate long-term updates and maintenance, with a modular architecture and proper documentation.

**PROCESS REQUIREMENTS**

1. **Team and Match Registration:** The registration process should be intuitive, and data for each team and match should be validated before being stored.
2. **Ranking Update:** The ranking update must happen automatically after each match is registered. Points and coefficients for teams should be recalculated and the ranking reordered in real-time.
3. **Undo Actions:** The system must allow the last user action to be undone. When an action is undone, the data should be immediately updated without compromising the integrity of the system.
4. **User Interface (UI):** Users should have access to all main functionalities through an easy-to-use interface, allowing them to manage teams, register matches, view rankings, and undo actions.

**PHASE 2: COLLECTION OF NECESSARY INFORMATION**

**SOURCES OF INFORMATION**

1. UEFA Regulations and Rules:
   * Data on teams: Requirements for team registration (such as titles won, country, UEFA coefficient).
   * Data on matches: Match results, teams involved, match dates, and goals scored.
   * Ranking criteria: How the ranking system works (points, coefficients, etc.).
2. Data Structures:
   * Hash Tables: For storing and quickly accessing information about teams and matches.
   * Stacks: To track actions that can be undone (team registration, match results).
   * Queues: To manage the order of matches (FIFO processing).
   * Priority Queues: To maintain the ranking of teams based on points and UEFA coefficient.
3. Software Requirements:
   * Programming Language: Java (based on the task requirements).
   * Database or File System: If applicable, to store persistent data.
   * Libraries: Use of standard libraries for implementing hash tables, queues, and stacks.

**Key Elements for the System**

1. Teams:
   * Attributes: Team name, country, titles, and UEFA coefficient.
   * Data Storage: Hash table for fast lookup by team name.
   * Functionality: Ability to add, update, and remove teams efficiently.
2. Matches:
   * Attributes: Home team, away team, goals scored by each team, and match date.
   * Data Storage: Queue for FIFO processing of match registration.
   * Functionality: Ability to record match results and process matches in the order they were added.
3. Ranking:
   * Attributes: Points and UEFA coefficient for each team.
   * Data Storage: Priority queue to keep teams ordered by performance.
   * Functionality: Update rankings after each match and sort teams by points and UEFA coefficient.
4. Undo Actions:
   * Attributes: A record of all actions taken (team addition, match registration).
   * Data Storage: Stack to manage the undo operation.
   * Functionality: Ability to revert the last action performed by the user.

**Additional Information**

* External Research:
  + Looking at similar systems (e.g., other sports management systems) to understand the best practices.
  + Reviewing how rankings are typically calculated in other football leagues to ensure accuracy.
* Stakeholder Input:
  + Communication with stakeholders (like the UEFA organizers) to gather any additional data they might need for the system.

**PHASE 3: SEARCH FOR CREATIVE SOLUTIONS**

**Brainstorming**

In this phase, we will generate potential solutions to solve the problem of managing teams, matches, and rankings in the UEFA Champions League. Here are some possible ideas:

1. Modular System Architecture:
   * Modules: Create independent modules for each function: Team Registration, Match Registration, Ranking Management, and Undo Function.
   * Each module would handle a specific part of the system (team data, match data, ranking data, etc.), making the system easier to maintain and extend.
2. User Interface:
   * Create a CLI (Command Line Interface) for easy interaction.
   * Alternatively, we could design a Graphical User Interface (GUI) using Java Swing for a more user-friendly experience.
3. Data Structures:
   * Hash Table for storing team information efficiently (quick access to teams).
   * Queue for storing matches in the order they were registered (FIFO).
   * Priority Queue (Max-Heap) for maintaining the ranking of teams based on points and UEFA coefficients.
   * Stack for undoing the last action performed, whether it’s adding a team or recording a match result.
4. Automated Ranking Update:
   * Each time a match is registered, the ranking should be automatically updated based on the results (goals, points, and UEFA coefficient).
   * The ranking will be dynamically adjusted to reflect the latest match results.
5. Undo System:
   * Allow users to revert the last action (team registration, match result entry, etc.) by using the stack data structure.
   * This can be useful in case of mistakes or incorrect data entry.
6. Data Validation and Error Handling:
   * Implement checks to ensure that data entered (team information, match scores, etc.) is valid.
   * If there’s any invalid data, the system should give an error message and prevent the action from being finalized until corrected.

**Review Lists**

* Team Data: Include the team’s name, country, number of titles, and UEFA coefficient.
* Match Data: Home and away teams, goals scored, and match date.
* Ranking: Points, UEFA coefficient, and team position.
* Undo History: Actions that can be undone, such as adding/removing teams, updating match results, or recalculating rankings.

**List of Attributes**

We can list the attributes of each module:

* Team Module:
  + Name
  + Country
  + Titles won
  + UEFA coefficient
  + Points
* Match Module:
  + Home team
  + Away team
  + Goals scored by both teams
  + Match date
* Ranking Module:
  + Points
  + UEFA coefficient
  + Current rank position

**Forced Relationship (Integration of Modules):**

To push creativity further, we can force a relationship between two seemingly unrelated components to spark innovative solutions. For example:

* **Team Registration + Match Registration**: Each match involves two teams, so every time a match is registered, the teams’ details must be checked or updated (e.g., updating the number of games played).
* **Undo System + Ranking Update**: When a team or match is undone, the rankings should be recalculated automatically based on the previous state.

**SOLUTION ALTERNATIVES**

|  |  |  |  |
| --- | --- | --- | --- |
| Alternativa | Main Structure | Pros | Cons |
| A | Use basic arrays and linked lists | Simple to implement | Slow searching and updating |
| B | Hash table, Queue, Priority Queue, Stack | Fast acces sorting and undo features | Slightly more complex to implement |

**Evaluation Criteria for Solutions**

* **Accuracy**: Does the solution accurately handle ranking updates, match results, and team registration?
* **Efficiency**: Does the solution perform well with large numbers of teams and matches?
* **Scalability**: Can the system handle a growing number of teams and matches without performance issues?
* **Ease of Implementation**: How easy is it to implement this solution with the given resources?

**Chosen Solution**

We will proceed with **Alternative B** (using **Hash Table**, **Queue**, **Priority Queue**, and **Stack**) as it provides the best balance of speed, flexibility, and ease of implementation.

**PHASE 4 : TRANSITION TO PRELIMINARY DESIGNS**

**Design of Data Structures**

We will define the **data structures** and outline how each part of the system will interact with others:

**Team Module (Team Registration)**

* **Attributes**:
  + Name (String)
  + Country (String)
  + Titles (int)
  + UEFA Coefficient (double)
  + Points (int)
* **Methods**:
  + addTeam(): Adds a new team to the hash table.
  + getTeam(): Retrieves team information by name.
  + updateTeam(): Updates team details (e.g., after a match).
* **Data Storage**: This will be stored in a **Hash Table** for fast access by team name.

**Match Module (Match Registration)**

* Attributes:
  + Home Team (Team)
  + Away Team (Team)
  + Home Goals (int)
  + Away Goals (int)
  + Date (String)
* Methods:
  + registerMatch(): Registers match results (home and away team, scores, and date).
  + getMatchResults(): Retrieves match results for a specific date or team.
* Data Storage: This will be stored in a Queue, following the FIFO (First-In, First-Out) principle to manage matches in the order they are registered.

**Ranking Module (Ranking and Standings)**

* Attributes:
  + Team (Team)
  + Points (int)
  + UEFA Coefficient (double)
* Methods:
  + updateRanking(): Updates the ranking after each match, based on points and UEFA coefficient.
  + getTopTeams(): Retrieves the top teams based on current rankings.
* Data Storage: This will be stored in a Priority Queue (Max-Heap) to ensure that the highest-ranked teams are always prioritized.

**Undo Module (Undo Action)**

* Attributes:
  + Action Type (String) - E.g., "Add Team", "Register Match"
  + Action Data (Object) - Details of the action to be undone.
* Methods:
  + undoLastAction(): Pops the last action from the stack and reverts it.
* Data Storage: This will be stored in a Stack to keep a history of performed actions.

**Data Flow and Interaction**

**Team Registration (Tabla Hash)**

Process Flow:

1. Input: User provides team name, country, number of titles, and UEFA coefficient.
2. Action: The system will validate the team’s information.
3. Storage: The system stores the team’s data in a Hash Table for fast retrieval using the team name as the key.
4. Output: Confirmation message that the team has been successfully added.

Example:

* Input: "PSG", "France", 5 titles, 4.2 UEFA coefficient
* Stored in: Hash table for fast lookups by team name.

**Match Registration (Pila)**

Process Flow:

1. Input: User inputs home team, away team, goals scored by both teams, and match date.
2. Action: The match details are registered and stored in a Queue to maintain the order of match registration (FIFO).
3. Storage: The match is added to the queue.
4. Output: The match is confirmed as registered and the ranking system is ready for updates.

Example:

* Input: "PSG" vs. "Barcelona", PSG 3 - 1 Barcelona, Date: 2025-04-01
* Stored in: Queue for FIFO processing.

**Undo Action (Pila)**

Process Flow:

1. Input: User performs an action (add team, register match).
2. Action: The system stores the action in a Stack for later reversal.
3. Storage: The last performed action is saved in the stack.
4. Output: When needed, the last action can be undone by popping from the stack, reversing the changes made (e.g., removing a team or match).

Example:

* Input: "Remove PSG from the system"
* Stored in: Stack as the latest action.
* Undo: "PSG is re-added to the system" after popping the stack.

**Match Management (Cola)**

Process Flow:

1. Input: Matches are automatically queued based on the order in which they are registered.
2. Action: The system processes the next match in the queue when it's time to play.
3. Storage: Matches are stored in a Queue and processed in First In, First Out (FIFO) order.
4. Output: The next scheduled match is processed, and the system moves to the next match in line.

Example:

* Input: "PSG vs. Barcelona" is registered.
* Stored in: Queue.
* Output: The match will be processed in the order it was added to the queue.

**Ranking and Standings (Cola de Prioridad)**

Process Flow:

1. Input: Match results are recorded.
2. Action: The priority queue automatically updates the ranking of the teams based on their points and UEFA coefficient.
3. Storage: Teams are sorted by their points and coefficient, with the highest-ranked teams at the top.
4. Output: Updated team rankings are displayed.

Example:

* Input: PSG wins 3 - 1 over Barcelona.
* Action: Points are awarded (PSG +3 points).
* Output: PSG’s ranking is updated in the priority queue.

**PHASE 5: EVALUATION AND SELECTION OF THE BEST SOLUTION**

**Criteria for Evaluating Solutions**

We will use the same criteria to evaluate the proposed solutions for the **Champions League** Team Management System:

A. Accuracy

* Definition: How well the solution delivers the expected results (team registration, match registration, ranking updates).
  + Exact: The solution delivers precise results.
  + Approximate: The solution provides an approximate result.

B. Efficiency

* Definition: How well the solution handles the data processing speed and resource usage.
  + Constant: The time complexity doesn’t grow with the number of teams or matches.
  + Greater than Constant: The time complexity increases linearly or logarithmically.
  + Logarithmic: Operations are efficient, even with large datasets.
  + Linear: Time complexity increases directly with the number of teams or matches.

C. Completeness

* Definition: How well the solution covers all possible cases and handles them correctly.
  + All: The solution handles every possible case.
  + Some: The solution covers most cases but not all.
  + None: The solution doesn’t handle many cases correctly.

D. Ease of Implementation

* Definition: How easy it is to implement the solution based on the technology stack and required resources.
  + Compatible with modern computing: The solution uses basic operations that are supported by modern computing systems.
  + Not fully compatible: The solution has limitations or compatibility issues with basic operations.

**Solution Alternatives**

We will evaluate the same three alternatives that we considered earlier:

**Alternative A: Basic Arrays and Linked Lists**

* Structure: Uses simple arrays and linked lists to store data.
* Pros:
  + Easy to implement.
  + Simple to understand.
* Cons:
  + Not efficient for large datasets (slow search, update, and deletion operations).
  + Not scalable for a growing number of teams or matches.

**Alternative B: Hash Table + Queue + Priority Queue + Stack**

* Structure: Uses a Hash Table for fast team lookups, Queue for match order, Priority Queue for rankings, and Stack for undo functionality.
* Pros:
  + Highly efficient for all required operations (quick lookup, insertion, deletion, sorting).
  + Scalable to handle large numbers of teams and matches.
  + Flexible with undo functionality and real-time ranking updates.
* Cons:
  + More complex to implement, requiring a good understanding of data structures.
  + Slightly higher overhead compared to simpler solutions.

EVALUATION MATRIX

|  |  |  |
| --- | --- | --- |
| Criteria | Alternative A | Alternative B |
| Accuracy | Approximate (1) | Exact (2) |
| Efficiency | Lineal (1) | Constant (4) |
| Completeness | Some (2) | All (3) |
| Ease of implementation | Easy (2) | Medium (3) |
| Total | 6 | 12 |

Analysis:

* Alternative B (Hash Table + Queue + Priority Queue + Stack) stands out as the best solution because it has the highest score, especially in accuracy, efficiency, and completeness.
* Alternative A (Arrays + Linked Lists), while easy to implement, is not efficient enough for large datasets and lacks scalability.

We choose Alternative B as the best solution because it provides high efficiency and scalability and accurate results for team registration, match processing, and ranking updates.