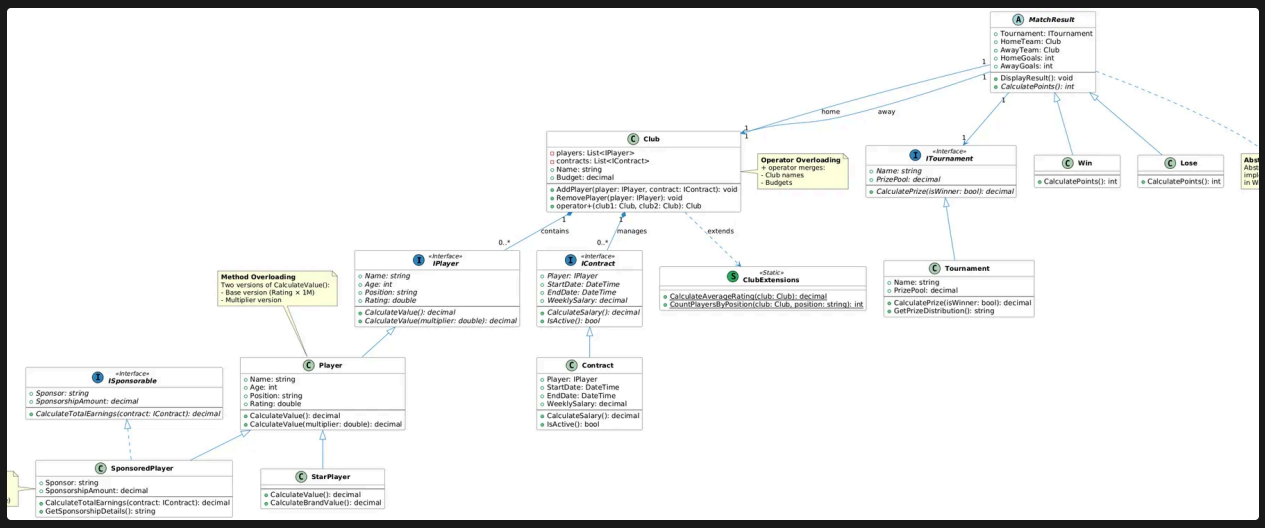
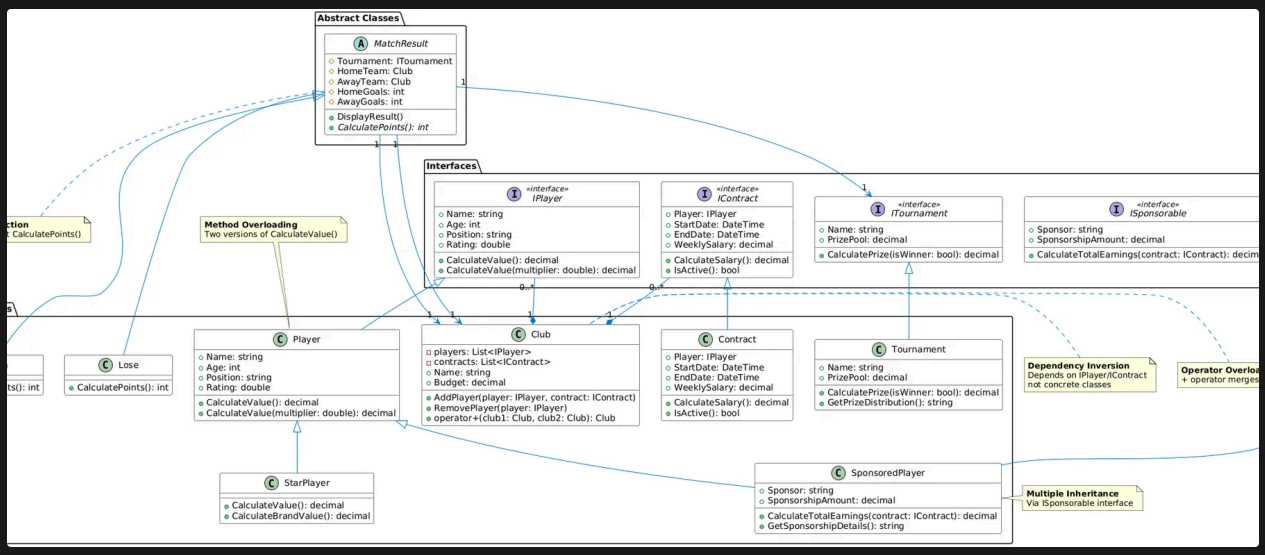
Lab 5

First UML:



Second UML:



1. Develop a system to manage :

- **Players** (basic and star players with different valuation rules)
- **Contracts** (salary calculations and active status tracking)
- **Tournaments** (prize money distribution logic)
- **Match Results** (point calculation for wins/losses)

- **Club Operations** (budget management, player transfers)

2. Detailed Component Specifications:

2.1 Player Management

Player Interface (`IPlayer`)

- **Properties:**
  - Name (string, readable/writable)
  - Age (integer, readable/writable)
  - Position (string, readable/writable)
  - Rating (decimal, 0.0-10.0 scale, readable/writable)
- **Methods:**
  - `CalculateValue()` → Returns market value as decimal (Rating × 1,000,000)
  - `CalculateValue(multiplier)` → Overloaded version with custom multiplier

Player Types

1. **Base Player**
   - Implements `IPlayer` directly
   - No additional properties/methods
2. **Star Player** (inherits from base `Player`)
   - Overrides `CalculateValue()` to return **double** the base value
   - Adds `CalculateBrandValue()` → Returns 30% of market value
3. **Sponsored Player** (inherits from `Player` , implements `ISponsorable`)
   - **Additional Properties:**
     - Sponsor name (string)
     - Annual sponsorship amount (decimal)
   - **Methods:**
     - `CalculateTotalEarnings(contract)` → Returns salary + sponsorship

2.2 Contract Management

Contract Interface (`IContract`)

- **Properties:**
  - Associated player (`IPlayer` , readable/writable)
  - Start/end dates (DateTime)
  - Weekly salary (decimal)
- **Methods:**
  - `CalculateSalary()` → Returns annual salary (weekly × 52)
  - `IsActive()` → Returns boolean (true if current date is within contract period)

2.3 Tournament System

Tournament Interface (`ITournament`)

- **Properties:**
  - Name (string)
  - Total prize pool (decimal)
- **Methods:**
  - `CalculatePrize(isWinner)` → Returns 60% of pool for winners, 40% for losers

2.4 Match Results

Abstract Base Class (`MatchResult`)

- **Properties:**
  - Tournament (`ITournament`)
  - Home/Away teams (`Club` objects)
  - Goal counts (integers)

- **Methods:**
  - `DisplayResult()` → Prints scoreline (e.g., "Team A 2-1 Team B")
  - **Abstract** `CalculatePoints()` → Must be implemented by subclasses

Concrete Implementations

1. **Win** → Returns 3 points
2. **Loss** → Returns 0 points

2.5 Club Operations

Club Class

- **Properties:**
  - Name (string)
  - Budget (decimal, privately modifiable)
  - Private lists for players (`IPlayer`) and contracts (`IContract`)
- **Methods:**
  - `AddPlayer(player, contract)` → Adds to collections, deducts annual salary from budget
  - `RemovePlayer(player)` → Removes player, refunds 50% of remaining salary
- **Operator Overloading:**
  - `+` operator merges two clubs: combines budgets, concatenates names

Extension Methods (for Club)

1. `CalculateAverageRating()` → Returns mean player rating (decimal)
2. `CountPlayersByPosition(position)` → Returns integer count of players in specified position

3. Technical Requirements

3.1 Object-Oriented Principles

- **Inheritance:**
  - `StarPlayer` and `SponsoredPlayer` derive from base `Player`
- **Abstraction:**
  - `MatchResult` defines abstract `CalculatePoints()` for polymorphic behavior
- **Interfaces:**
  - All core components depend on abstractions (`IPlayer`, `IContract`, etc.)

3.2 Advanced C# Features

- **Method Overloading:**
  - `Player.CalculateValue()` has two versions (with/without multiplier)
- **Operator Overloading:**
  - Club merger via `+` operator
- **Extension Methods:**
  - Add functionality to `Club` without inheritance

3.3 SOLID Compliance

- **Dependency Inversion:**
  - `Club` depends on `IPlayer` / `IContract`, not concrete implementations
- **Open-Closed Principle:**
  - Extendable via new player types (e.g., `SponsoredPlayer`) without modifying `Club`