Class Diagram for Cricinfo Understand how to create a class diagram for Cricinfo using the bottom-up approach. We'll cover the following Components of Cricinfo Admin • Run, ball, and wicket Over and innings Match Stadium • Player, coach, and umpire • Team, tournament squad, and playing eleven • Tournament and points table Stats Commentator and commentary News Enumerations Custom data type Relationship between classes Association One-way association Two-way association Aggregation Composition Inheritance Class diagram of Cricinfo Design pattern Al-powered trainer In this lesson, we'll identify and design the classes, abstract classes, and interfaces based on the requirements we have previously gathered from the interviewer in our Cricinfo system. **Components of Cricinfo** As mentioned earlier, we will design the Cricinfo system using a bottom-up approach. **Admin** The Admin class is responsible for managing the system as well as adding and modifying updates. The representation of the class is shown below: Admin - addPlayer() : bool addTeam(): bool - addMatch(): bool - addTournament(): bool - addStats(): bool - addNews(): bool The Admin class ∵ R8: Cricinfo R8: The admin of the system should be able to add tournaments, matches, teams, players, and news to the system. Run, ball, and wicket The Run class records the number and type of runs scored on a ball. The Ball class records every detail of a ball, such as the number of runs scored, if it was a wickettaking ball, etc. The Wicket class records the details of the wicket, including its type, the player that bowled, and the player that was declared out. The mentioned classes are shown below: Run Ball Wicket totalRuns : int - balledBy : Player - type : WicketType - type : RunType - playedBy: Player - playerOut: Player - scoredBy : Player - balledBy : Player - type : BallType - runs : Run {list} - caughtBv : Plaver - wicket : Wicket - runoutBy : Player - stumpedBy : Player + addCommentary() The Run. Ball. and Wicket classes ∵ R2: Cricinfo **R2:** The system should be able to track all scores or wickets that occurred for each ball. The system should also provide a live commentary for every ball. Over and innings The Over class represents all the details of an over of the innings. The Innings class represents the details of a match innings. The two classes are shown below: **Innings** Over - bowling : Playing11 - number · int - bowler : Player - batting : Playing11 - startTime : date/time - totalScore: int - endTime : date/time - balls : Ball {list} - totalScore : int + addBall(): bool - totalWickets : int - overs : Over {list} + addOver() : bool The Over and Innings classes Match The Match class is an abstract class that has three child classes that represent the types of matches that can take place. • The Test class The ODI class The T20 class The class diagram is shown below: <<Abstract>> Match - startTime: date/time - result : MatchResult - totalOvers : int - teams : Playing11 {list} - innings : Inning {list} - tossWin : Playing11 - umpires : Map<Umpire, UmpireType> - stadium : Stadium commentators : Commentator {list} - stats : MatchStat {list} + assignStadium(Stadium): bool + assignUmpire(Umpire) : bool extends T20 Test The Match and its derived classes ∵റ്റ்∙ R3: Cricinfo R3: The system should be able to keep track of all matches—Test, T20, and ODI matches. **Stadium** The Stadium class represents the information about a stadium, including its name, address, and capacity. The UML representation of this class is given below: Stadium name : string - location : Address - maxCapacity : int The Stadium class Player, coach, and umpire The Player class includes the information of a player and their statistics. The Coach class contains the information of a coach. The Umpire class contains the information of an umpire. The three classes are shown below: **Player** Coach **Umpire** - name : string - name : string - name : string - age : int - age : int - age : int - country : int - country : int - country : int - position : PlayingPosition - teams : Team {list} - teams : Team {list} + assignMatch(): bool - stat : PlayerStat The Player, Coach, and Umpire classes ∵⇔ R1: Cricinfo **R1:** The system should be able to track the stats of all players, teams, and matches. Team, tournament squad, and playing eleven The Team class represents the information about a cricket team, including the list of players, the team coach, and any news related to the team. The TournamentSquad class represents the team members participating in a tournament. The Playing11 class represents the squad members playing in a match. The class diagram for these classes is given below: **TournamentSquad** Playing11 Team name : string - players : Player {list} - players : Player {list} - players : Player {list} - tournament : Tournament + addPlayer(Player) : bool - coach : Coach - stats : TournamentStat - news : News {list} + addPlayer(Player) : bool - stats : TeamStat + addSquad(TournamentSquad): bool + addPlayer(Player) : bool + addNews(News) : bool The Team, TournamentSquad, and Playing11 classes ∵்; R6 and R7: Cricinfo R6: All teams should be able to select some players that will participate in the tournament. R7: For every match, the teams must select 11 players to play on the field, known as the playing eleven. Tournament and points table The Tournament class contains information about a cricket tournament. The PointsTable class shows the accumulated points and match results of the teams that play in the tournament. These classes are shown below: **PointsTable** Tournament teamPoints : Map<string, float> - startDate : date/time - teams : TournamentSquad {list} - matchResults : Map<Team, MatchResult> matches : Match {list} - tournament : Tournament - points : PointsTable - lastUpdated : date/time + addTeam(): bool + addMatch(): bool The Tournament and PointsTable classes ∵⇔ R4: Cricinfo R4: The system should be able to keep track of ongoing and previous tournaments. The system should also be able to show a points table for all teams participating in a tournament. **Stats** The Stat class is an abstract class that extends to PlayerStat, TeamStat, and MatchStat classes. These classes contain important statistics. The UML representation is shown below: <<Abstract>> Stat + updateStats() : bool extends **PlayerStat TeamStat MatchStat** - winPercentage : double - totalSixes : int - ranking : int - totalFours : int - bestScore : int - topBatsman : Player - bestWicketCount : int topBowler : Player - totalReviews : int - totalMatchesPlayed : int - total100s : int - totalHattricks : int Stat and its derived classes ∵ R1 and R5: Cricinfo R1: The system should be able to track the stats of all players, teams, and matches. R5: The system should be able to show the result of all previous televised matches. **Commentator and commentary** The Commentator class records the information about the commentator. The Commentary class contains information about the commentary for every ball of an over. The two classes are shown below: Commentator Commentary name : string text : string - createdAt : date/time + assignMatch() : bool commentator : Commentator The Commentator and Commentary classes ∵⇔ R2: Cricinfo R2: The system should be able to track all scores or wickets that occurred for each ball. The system should also provide a live commentary for every ball. News The News class holds the news updates of a team. The definition of this class is given below: News - date : date/time text : string - image : byte {list} team : Team The News class **Enumerations** The enumerations required in the Cricinfo system are listed below: MatchResult: This records the result of a match— a win, loss, canceled, or drawn. UmpireType: This records the type of umpire—field umpire, third umpire, or reserved. WicketType: This records the type of the wicket—stumped, bold, caught, etc. BallType: This records the type of ball played—a regular delivery, wide, no ball, or wicket. RunType: This records the type of run scored—a regular run, four, six, wide, etc. PlayingPosition: This records the playing position of a player—batsman, bowler, and all-rounder. <<Enumeration>> <<Fnumeration>> <<Enumeration>> RunType **MatchResult** WicketType Normal Live Bold Four BatFirstWin Caught Six Stumped FieldFirstWin Wide RunOut Draw LegBye Canceled Lbw Bye RetiredHurt NoBall HitWicket Overthrow Obstruction Handled <<Fnumeration>> <<Enumeration>> <<Enumeration>> **PlayingPosition BallType UmpireType** Normal Wide Reserved Bowling AllRounder **ThirdUmpire** NoBall Wicket **Enums in Cricinfo Custom data type** We need to create a custom data type, Address, that will store the physical location of any place. **Address** - zipCode : int - streetAddress : string - city : string state : string - country : string The Address custom data type Relationship between classes Now, we will discuss the relationships between the classes we have defined above in our Cricinfo system. **Association** The class diagram has the following association relationships: One-way association • The Admin class has a one-way association with the Player, Team, Match, and Tournament classes. The Player class has a one-way association with the Run, Ball, Wicket, and Over classes. • The Team class has a one-way association with the TournamentSquad and Tournament classes. • The TournamentSquad class has a one-way association with the Playing11 class. Two-way association The Ball class is associated with the Run, Wicket, and Commentary classes. The Team class is associated with the Coach and News classes. The Commentary class is associated with the Commentator class. The Match class is associated with the Umpire, Commentator, and Stadium classes. Admin adds adds adds Player Tournament Match Team Run **Umpire TournamentSquad** \rightarrow scores chooses **Stadium** Playing11 Ball Coach Wicket Commentator News takes Over Commentary plays The association relationships between classes Aggregation The class diagram has the following aggregation relationships: • The Tournament class contains the TournamentSquad class. **Tournament TournamentSquad** The aggregation relationship between classes Composition The class diagram has the following composition relationships: The Player class is composed of the PlayerStat class. The Team class is composed of the Player and TeamStat classes. The Tournament class is composed of the Match and PointsTable classes. The Match class is composed of the Playing 11, Innings, and MatchStat classes. The Innings class is composed of the Over class. The Over class is composed of the Ball class. Team **Player PlayerStat TeamStat Tournament PointsTable MatchStat** Match Playing11 Ball **Innings** Over The composition relationships between classes **Inheritance** The class diagram has the following inheritance relationships: The ODI, Test, and T20 classes are derived from the Match class. The TeamStat, MatchStat, and PlayerStat classes are derived from the Stat class. **Note:** We have already discussed the inheritance relationship between classes in the component section above one by one. Class diagram of Cricinfo Here's the complete class diagram for Cricinfo: TournamentSquad Admin <<Abstract>> Playing11 players : Player {list} name: string players : Player {list} age : int country : int teams : Team {list} addPlayer() : bool stats: TournamentStat + updateStats() : bool addPlayer(Player) : bool addMatch(): bool addTournament(): bool + addPlayer(Player) : bool Tournament TeamStat PlayerStat MatchStat startDate : date/time winPercentage : double topBatsman : Player ranking : int bestScore : int bestWicketCount : int totalSixes : int teams: TournamentSquad (list) PointsTable totalFours : int totalReviews : int topBatsman : Player topBowler : Player matches : Match {list} teamPoints: Map<string, float> totalMatchesPlayed: int total100s : int totalHattricks : int Innings Commentator Stadium bowling : Playing11 batting : Playing11 startTime : date/time name : string
players : Player {list}
coach : Coach startTime: date/time
result : MatchResult
totalOvers : int
teams : Playing11 {list}
innings : Inning {list}
tossWin : Playing11
umpires : Map<Umpire, UmpireType>
stadium : Stadium location : Address maxCapacity : int assignMatch(): bool news : News {list} endTime : date/time stats : TeamStat totalScore : int totalWickets : int Umpire addSquad(TournamentSquad) : bool · addPlayer(Player) : bool · addNews(News) : bool Commentary name : string + addOver() : bool commentators : Commentator {list} stats : MatchStat {list} age : int text : string createdAt : date/time country : int assignStadium(Stadium) : bool assignUmpire(Umpire) : bool News assignMatch(): bool date : date/time text : string image : byte {list} team : Team extends Odi T20 Test Over Player Ball balledBy : Player type : WicketType name : string playedBy: Player type : RunType scoredBy : Player playerOut: Playe country : int type : BallType balledBv : Plaver totalScore: int position : PlayingPosition teams : Team {list} stat : PlayerStat caughtBy : Player runoutBy : Player balls : Ball {list} stumpedBy : Player addBall() : bool The class diagram of Cricinfo Design pattern In the Cricinfo system, we need to create different types of matches, tournaments, and squads at runtime. To do this, we can use the Factory design pattern. This pattern provides a way to create objects without specifying the exact class of object that will be created. Al-powered trainer At this stage, everything should be clear. If you encounter any confusion or ambiguity, feel free to utilize the interactive AI-enabled widget below to seek clarification. This tool is designed to assist you in strengthening your understanding of the concepts. Powered by Al 20 Prompts Remaining Prompt Ai Wiaget