Code for Cricinfo

Write object-oriented code to implement the design of the Cricinfo problem.

We'll cover the following

We've gone over the different aspects of Cricinfo and observed the attributes attached to the problem using various UML diagrams. Let's now explore the more practical side of things, where we will work on implementing the Cricinfo problem using multiple languages. This is usually the last step in an object-oriented design interview process.

We have chosen the following languages to write the skeleton code of the different classes present in Cricinfo:

- Java
- C#
- Python
- C++
- JavaScript

Cricinfo classes

In this section, we will provide the skeleton code of the classes designed in the class diagram lesson.

Note: For simplicity, we are not defining getter and setter functions. The reader can assume that all class attributes are private and accessed through their respective public getter methods and modified only through their public method functions.

Constants

The following code defines the various enums and custom data types used in the Cricinfo design:

Note: JavaScript does not support enumerations. We'll usethe `Object.freeze()` method as an alternative. It freezes an object and prevents further modifications.

```
public class Address {
  private int zipCode;
  private String streetAddress;
  private String city;
  private String state;
  private String country;

}

enum MatchResult {
  LIVE,
  BAT_FIRST_WIN,
  FIELD_FIRST_WIN,
  DRAW,
  CANCELLED
}

enum UmpireType {
  FIELD,
  RESERVED,
  THIRD_UMPIRE
}

enum WicketType {
  BOLD,
  CAUGHT,
  RUNOUT,
  STUMPED,
  LBW,
  RETIRED_HURT,
  HIT_WICKET,
  ...
```

Constant definitions

Admin, player, coach, and umpire

The definitions of the `Admin`, `Player`, `Coach` and `Umpire` classes are as follows:

```
public class Admin {
  public boolean addPlayer(Player player);
  public boolean addTeam(Team team);
  public boolean addMatch(Match match);
  public boolean addTournament(Tournament tournament);
  public boolean addStats(Stat stats);
  public boolean addNews(News news);
}

public class Player {
  private String name;
  private int age;
  private int country;
  private PlayerPosition position;
  private List<Team> teams;
  private PlayerStat stat;
}

public class Coach {
  private String name;
  private int age;
  private int country;
  private List<Team> teams;
}

public class Umpire {
  private String name;
  private int age;
  private int country;
  ...
```

The Admin, Player, Coach, and Umpire classes

Run, ball, wicket, over, and innings

In cricket, the mandatory concepts can be of five types: run, ball, wicket, over, and innings. To store information about these identities, we defined the `Run`, `Ball`, `Wicket`, `Over`, and `Innings` classes as shown below:

```
public class Run {
  private int totalRuns;
  private RunType type;
  private Player scoredBy;
}

public class Ball {
  private Player bowledBy;
  private Player playedBy;
  private BallType type;
  private List<Run> runs;
  private Wicket wicket;

  public boolean addCommentary(Commentary commentary);
}

public class Wicket {
  private WicketType type;
  private Player playerOut;
  private Player bowledBy;
  private Player caughtBy;
  private Player runoutBy;
  private Player stumpedBy;
}

public class Over {
  private int number;
  private Player bowler;
  private List<Ball> balls;
  private List<Ball> balls;
  ...
```

Implementation of the Run, Ball, Wicket, Over, and Innings classes

Match

The `Match` is an abstract class representing matches in Cricinfo. Matches can be of three types: `T20`, `Test`, and `ODI`. The implementation of these classes is given below:

```
public abstract class Match {
  private Date startTime;
  private MatchResult result;
  private int totalOvers;
  private List<Playing11> team;
  private List<Innings> innings;
  private Playing11 tossWin;
  private Map<Umpire, UmpireType> umpires;
  private Stadium stadium;
  private List<Commentator> commentators;
  private List<MatchStat> stats;

  public abstract boolean assignStadium(Stadium stadium);
  public abstract boolean assignUmpire(Umpire umpire);
}

public class T20 extends Match {
  public boolean assignStadium(Stadium stadium);
  public boolean assignUmpire(Umpire umpire);
}

public class Test extends Match {
  public boolean assignStadium(Stadium stadium);
  public boolean assignUmpire(Umpire umpire);
}

public class ODI extends Match {
  public boolean assignStadium(Stadium stadium);
  public boolean assignUmpire(Umpire umpire);
}
```

Match and its derived classes

Team, tournament squad, and playing11

The definitions of the `AdminTeam`, `TournamentSquad`, and `Playing11` classes are as follows:

```
public class Team {
  private String name;
  private List<Player> players;
  private Coach coach;
  private List<News> news;
  private TeamStat stats;

  public boolean addSquad(TournamentSquad squad);
  public boolean addPlayer(Player player);
  public boolean addNews(News news);
}

public class TournamentSquad {
  private List<Player> players;
  private Tournament tournament;
  private List<TournamentStat> stats;

  public boolean addPlayer(Player player);
}

public class Playing11 {
  private List<Player> players;

  public boolean addPlayer(Player player);
}
```

The Team, TournamentSquad, and Playing11 classes

Tournament, points table, and stadium

The definitions of the `Tournament`, `PointsTable`, and `Stadium` classes are as follows:

```
public class Tournament {
  private Date startDate;
  private List<TournamentSquad> teams;
  private List<Match> matches;
  private PointsTable points;

  public boolean addTeam(TournamentSquad team);
  public boolean addMatch(Match match);
}

public class PointsTable {
  private Map<MatchString, Float> teamPoints;
  private Map<MatchTeam, MatchResult> matchResults;
  private Tournament tournament;
  private Date lastUpdated;
}

public class Stadium {
  private String name;
  private Address location;
  private int seatCapacity;
}
```

The Tournament, PointsTable, and Stadium classes

Commentator, commentary, and news

The definitions of the `Commentator`, `Commentary`, and `News` classes are as follows:

```
public class Commentator {
  private String name;

  public boolean assignMatch(Match match);
}

public class Commentary {
  private String text;
  private Date createdAt;
  private Commentator commentator;
}

public class News {
  private Date date;
  private String text;
  private List<byte> image;
  private Team team;
}
```

Commentator, Commentary, and News classes

Statistics

The `Stat` is an abstract class which represents the statistics in the Cricinfo. Statistics can be of three types: `PlayerStat`, `MatchStat`, and `TeamStat`. These classes are represented below:

```
public abstract class Stat {
  public abstract boolean updateStats();
}

public class PlayerStat extends Stat {
  private int matches;
  private int totalRuns;
  private int boundaryCount;
  private int totalMatchesPlayed;
  private int totalSR;
  private int totalWicketStats;

  public boolean updateStats();
}

public class MatchStat extends Stat {
  private double winPercentage;
  private Player topRunner;
  private Player topBowler;

  public boolean updateStats();
}

public class TeamStat extends Stat {
  private int totalGames;
  private int totalPlays;
  private int totalWinnings;

  public boolean updateStats();
}
```

Stat and its derived classes

Wrapping up

We've explored the complete design of Cricinfo in this chapter. We've looked at how Cricinfo can be visualized using various UML diagrams and designed using object-oriented principles and design patterns.