8. Design a Cricinfo

Cricinfo is a sports news website exclusively for the game of cricket. The site features live coverage of cricket matches containing ball-by-ball commentary and a database for all the historic matches. The site also provides news and articles about cricket.

Cricinfo

System Requirements

We will focus on the following set of requirements while designing Cricinfo:

1. The system should keep track of all cricket-playing teams and their matches.
2. The system should show live ball-by-ball commentary of cricket matches.
3. All international cricket rules should be followed.
4. Any team playing a tournament will announce a squad (a set of players) for the tournament.
5. For each match, both teams will announce their playing-eleven from the tournament squad.
6. The system should be able to record stats about players, matches, and tournaments.

7. The system should be able to answer global stats queries like, “Who is the highest wicket taker of all time?”, “Who has scored maximum numbers of 100s in test matches?”, etc.
8. The system should keep track of all ODI, Test and T20 matches.

Use Case Diagram

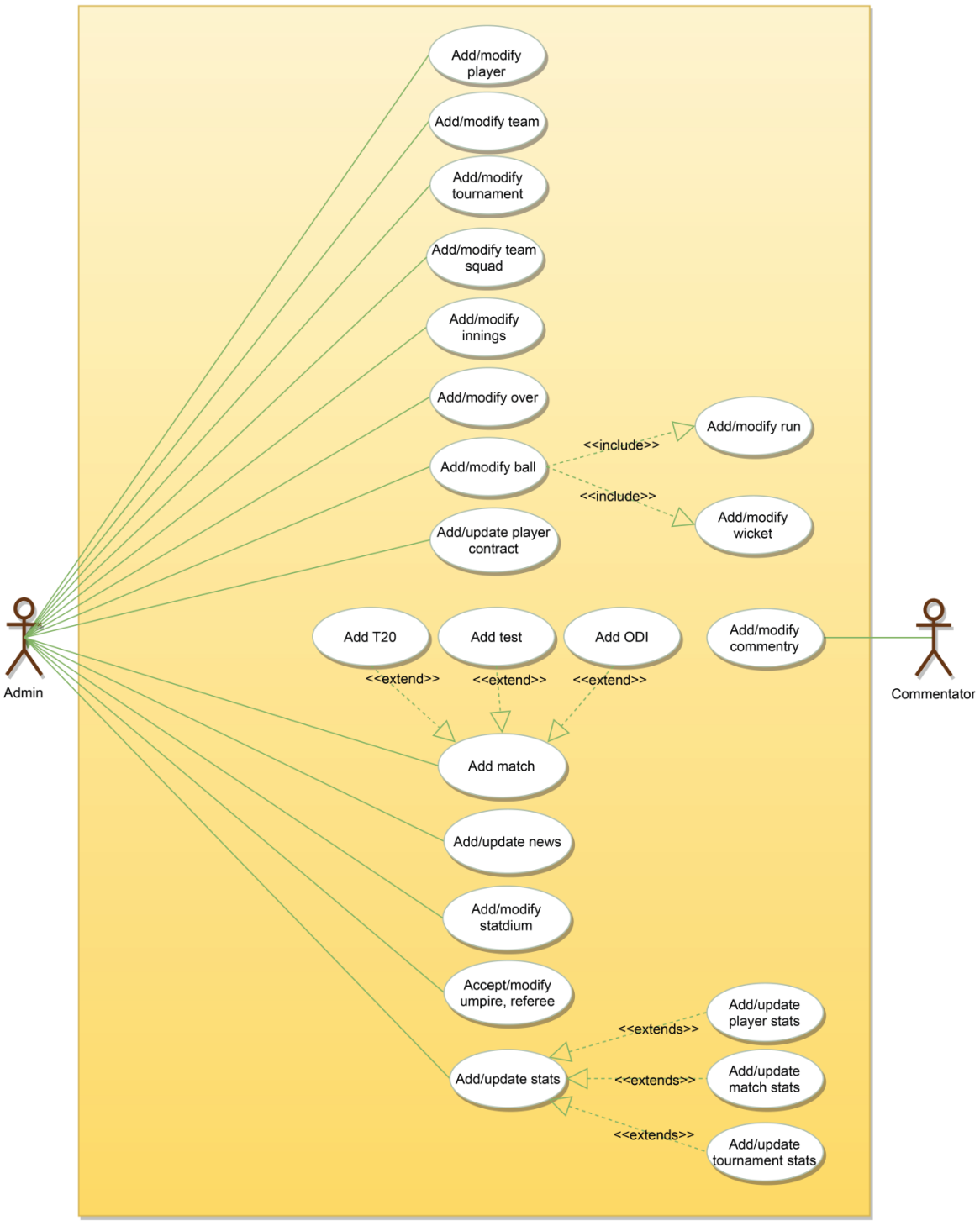
We have two main Actors in our system:

- **Admin**: An Admin will be able to add/modify players, teams, tournaments, and matches, and will also record ball-by-ball details of each match.
- **Commentator**: Commentators will be responsible for adding ball-by-ball commentary for matches.

Here are the top use cases of our system:

- **Add/modify teams and players**: An Admin will add players to teams and keeps up-to-date information about them in the system.
- **Add tournaments and matches**: Admins will add tournaments and matches in the system.
- **Add ball**: Admins will record ball-by-ball details of a match.
- **Add stadium, umpire, and referee**: The system will keep track of stadiums as well as of the umpires and referees managing the matches.
- **Add/update stats**: Admins will add stats about matches and tournaments. The system will generate certain stats.
- **Add commentary**: Add ball-by-ball commentary of matches.

Here is the use case diagram of Cricinfo:

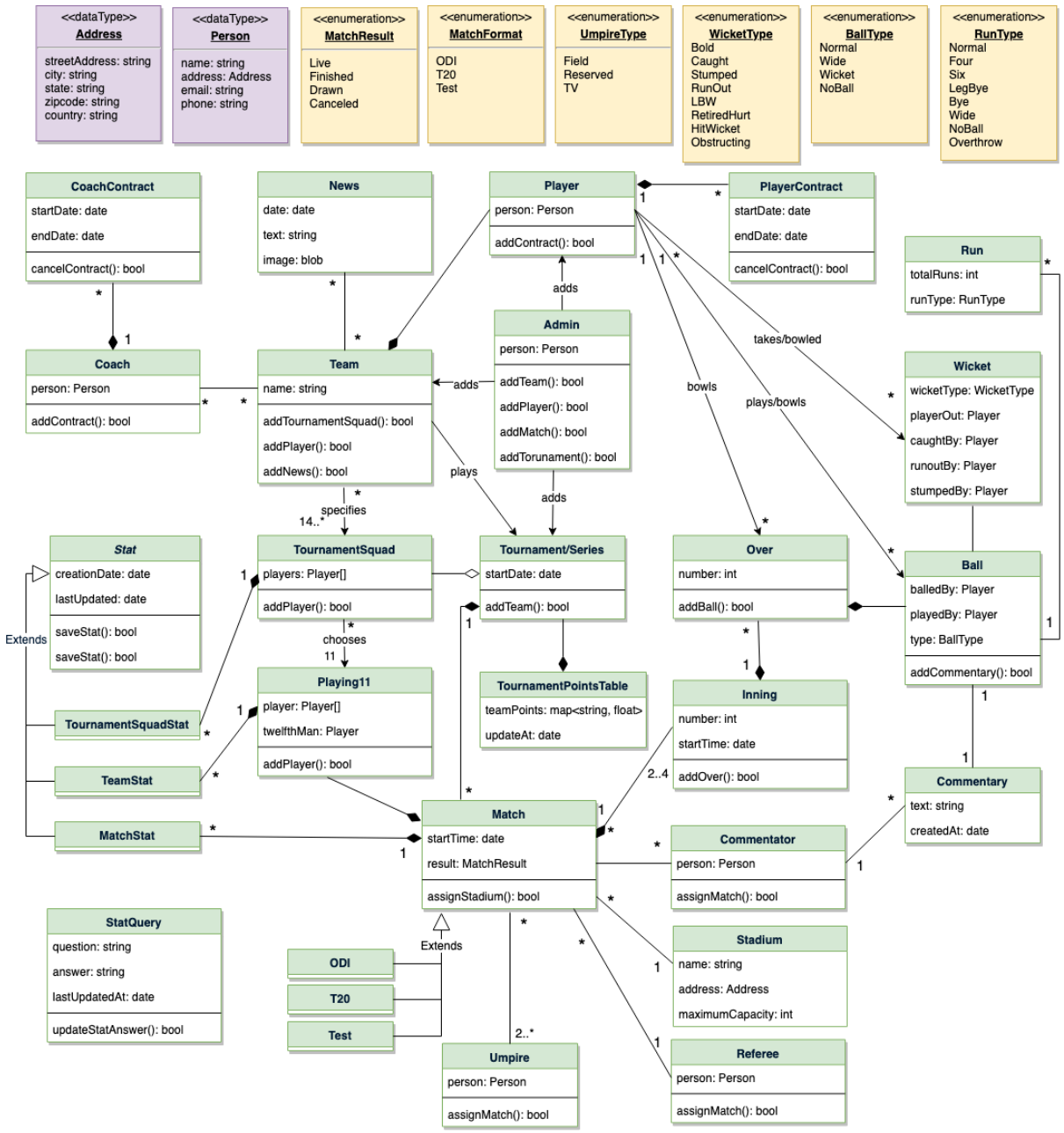


Use Case Diagram for Cricinfo

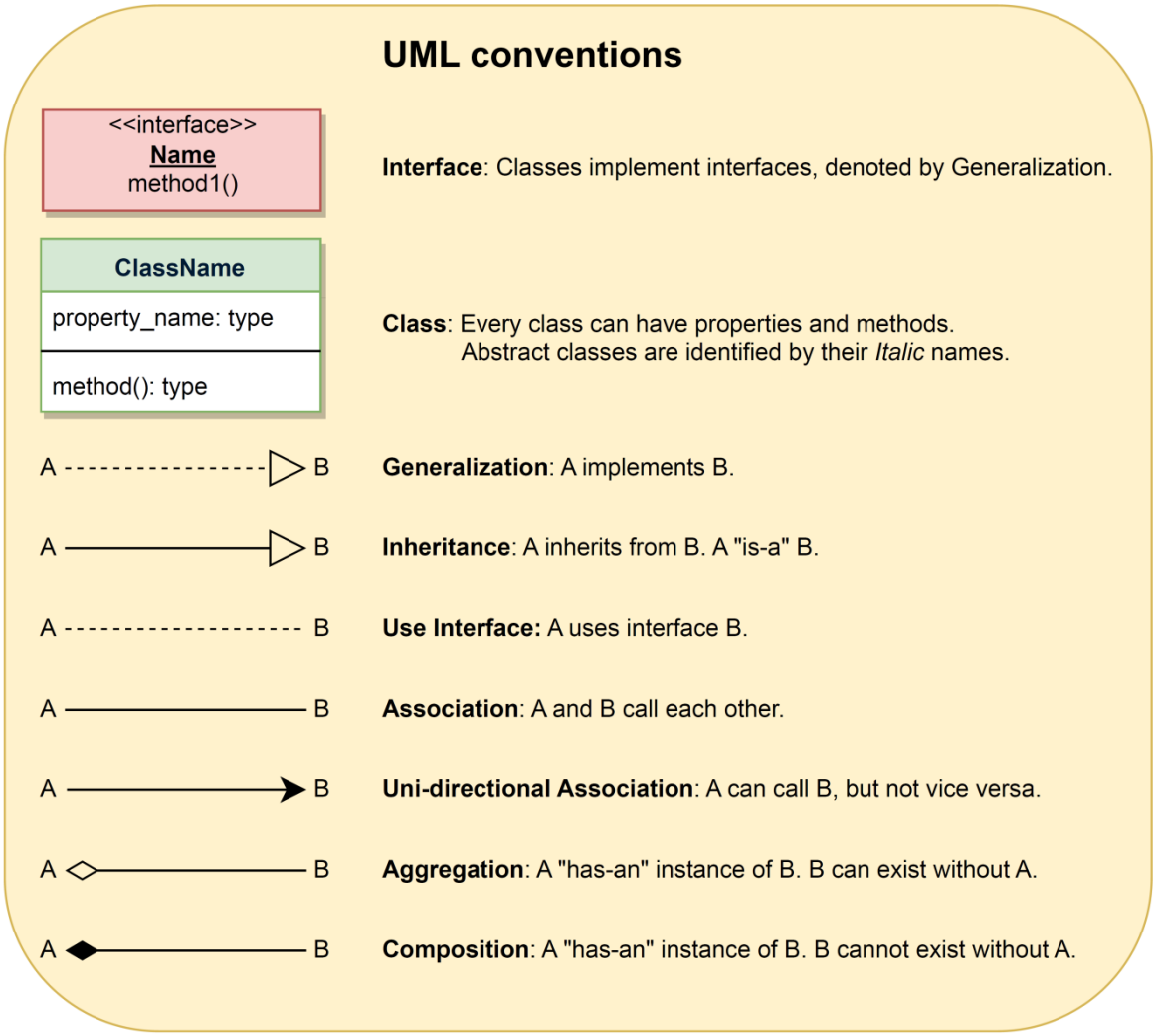
Class Diagram

Here are the main classes of the Cricinfo system:

- **Player**: Keeps a record of a cricket player, their basic profile and contracts.
- **Team**: This class manages cricket teams.
- **Tournament**: Manages cricket tournaments and keeps track of the points table for all playing teams.
- **TournamentSquad**: Each team playing a tournament will announce a set of players who will be playing the tournament. TournamentSquad will encapsulate that.
- **Playing11**: Each team playing a match will select 11 players from their announced tournaments squad.
- **Match**: Encapsulates all information of a cricket match. Our system will support three match types: 1) ODI, 2) T20, and 3) Test
- **Innings**: Records all innings of a match.
- **Over**: Records details about an Over.
- **Ball**: Records every detail of a ball, such as the number of runs scored, if it was a wicket-taking ball, etc.
- **Run**: Records the number and type of runs scored on a ball. The different run types are: Wide, LegBy, Four, Six, etc.
- **Commentator and Commentary**: The commentator adds ball-by-ball commentary.
- **Umpire and Referee**: These classes will store details about umpires and referees, respectively.
- **Stat**: Our system will keep track of the stats for every player, match and tournament.
- **StatQuery**: This class will encapsulate general stat queries and their answers, like “Who has scored the maximum number of 100s in ODIs?” or, “Which bowler has taken the most wickets in test matches?” etc.



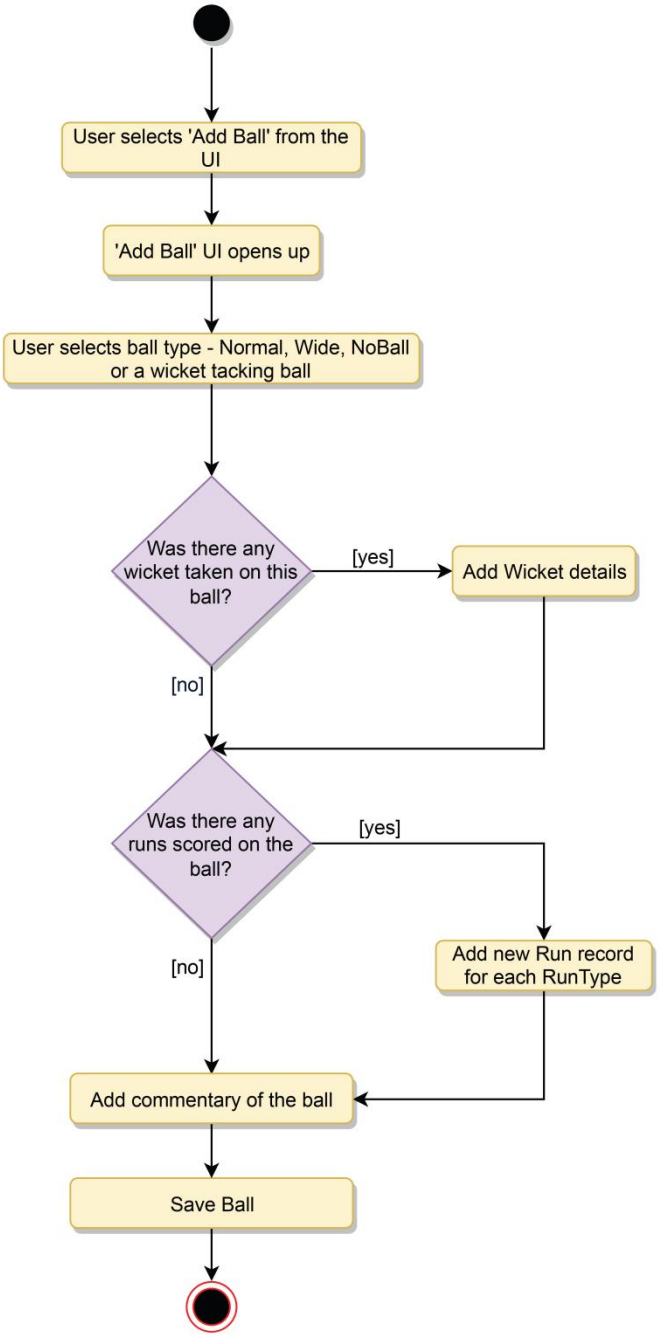
Cricinfo UML



UML for Cricinfo

Activity Diagram

Record a Ball of an Over: Here are the steps to record a ball of an over in the system:



Activity Diagram for Cricinfo

Code

Here is the high-level definition for the classes described above.

Enums, data types, and constants: Here are the required enums, data types, and constants:

```
// Enum representing different types of match formats (ODI, T20, TEST)
enum MatchFormat {
   ODI(1), T20(2), TEST(3);
   private int value;

   MatchFormat(int value) {
      this.value = value;
   }

   public int getValue() {
      return value;
   }
}

// Enum representing possible results of a match (LIVE, FINISHED, DRAWN, CANCELLED)
enum MatchResult {
   LIVE(1), FINISHED(2), DRAWN(3), CANCELLED(4);
   private int value;

   MatchResult(int value) {
      this.value = value;
   }

   public int getValue() {
      return value;
   }
}

// Enum representing different types of umpires (FIELD, RESERVED, TV)
enum UmpireType {
   FIELD(1), RESERVED(2), TV(3);
   private int value;

   UmpireType(int value) {
      this.value = value;
   }
```

```
    public int getValue() {
        return value;
    }
}

// Enum representing different types of wickets (e.g., BOLD, CAUGHT, LBW)
enum WicketType {
    BOLD(1), CAUGHT(2), STUMPED(3), RUN_OUT(4), LBW(5), RETIRED_HURT(6), HIT_WICKET(7),
OBSTRUCTING(8);
    private int value;

    WicketType(int value) {
        this.value = value;
    }

    public int getValue() {
        return value;
    }
}

// Enum representing different types of balls (NORMAL, WIDE, WICKET, NO_BALL)
enum BallType {
    NORMAL(1), WIDE(2), WICKET(3), NO_BALL(4);
    private int value;

    BallType(int value) {
        this.value = value;
    }

    public int getValue() {
        return value;
    }
}

// Enum representing different types of runs (NORMAL, FOUR, SIX, LEG_BYE, BYE, NO_BALL,
OVERTHROW)
enum RunType {
    NORMAL(1), FOUR(2), SIX(3), LEG_BYE(4), BYE(5), NO_BALL(6), OVERTHROW(7);
    private int value;

    RunType(int value) {
        this.value = value;
    }

    public int getValue() {
```

```
        return value;
    }
}

// Class representing an Address with street, city, state, zip code, and country
class Address {
    private String streetAddress;
    private String city;
    private String state;
    private String zipCode;
    private String country;

    public Address(String street, String city, String state, String zipCode, String country) {
        this.streetAddress = street;
        this.city = city;
        this.state = state;
        this.zipCode = zipCode;
        this.country = country;
    }
}

// Class representing a Person with name, address, email, and phone details
class Person {
    private String name;
    private Address address;
    private String email;
    private String phone;

    public Person(String name, Address address, String email, String phone) {
        this.name = name;
        this.address = address;
        this.email = email;
        this.phone = phone;
    }
}

// Class representing a Player, which extends from Person and has contracts
class Player {
    private Person person;
    private List<String> contracts = new ArrayList<>();

    public Player(Person person) {
        this.person = person;
    }
```

```
    public void addContract(String contract) {
        contracts.add(contract);
    }
}

// Class representing an Admin, which extends from Person and has functionalities to add match, team, and tournament
class Admin {
    private Person person;

    public Admin(Person person) {
        this.person = person;
    }

    public void addMatch(String match) {
        // implementation
    }

    public void addTeam(String team) {
        // implementation
    }

    public void addTournament(String tournament) {
        // implementation
    }
}

// Class representing an Umpire, which extends from Person and can assign matches
class Umpire {
    private Person person;

    public Umpire(Person person) {
        this.person = person;
    }

    public void assignMatch(String match) {
        // implementation
    }
}

// Class representing a Referee, which extends from Person and can assign matches
class Referee {
    private Person person;

    public Referee(Person person) {
```

```
        this.person = person;
    }

    public void assignMatch(String match) {
        // implementation
    }
}

// Class representing a Commentator, which extends from Person and can assign matches
class Commentator {
    private Person person;

    public Commentator(Person person) {
        this.person = person;
    }

    public void assignMatch(String match) {
        // implementation
    }
}

// Class representing a Team with name, coach, players, and news
class Team {
    private String name;
    private String coach;
    private List<String> players = new ArrayList<>();
    private List<String> news = new ArrayList<>();

    public Team(String name, String coach) {
        this.name = name;
        this.coach = coach;
    }

    public void addTournamentSquad(String tournamentSquad) {
        // implementation
    }

    public void addPlayer(String player) {
        players.add(player);
    }

    public void addNews(String news) {
        this.news.add(news);
    }
}
```

```
// Class representing a Tournament Squad with players and tournament stats
class TournamentSquad {
    private List<String> players = new ArrayList<>();
    private List<String> tournamentStats = new ArrayList<>();

    public void addPlayer(String player) {
        players.add(player);
    }
}

// Class representing the Playing11 of a team with players and the twelfth man
class Playing11 {
    private List<String> players = new ArrayList<>();
    private String twelfthMan;

    public void addPlayer(String player) {
        players.add(player);
    }
}

// Class representing an Over in a match with a list of balls
class Over {
    private int number;
    private List<String> balls = new ArrayList<>();

    public Over(int number) {
        this.number = number;
    }

    public void addBall(String ball) {
        balls.add(ball);
    }
}

// Class representing a Ball with details about the player who bowled it, the player who faced it,
type, wicket, runs, and commentary
class Ball {
    private String balledBy;
    private String playedBy;
    private BallType type;
    private String wicket;
    private int runs;
    private String commentary;
```

```
    public Ball(String balledBy, String playedBy, BallType type, String wicket, int runs, String
commentary) {
        this.balledBy = balledBy;
        this.playedBy = playedBy;
        this.type = type;
        this.wicket = wicket;
        this.runs = runs;
        this.commentary = commentary;
    }
}

// Class representing a Wicket with details about the type of wicket, player out, and who
dismissed them
class Wicket {
    private WicketType wicketType;
    private String playerOut;
    private String caughtBy;
    private String runOutBy;
    private String stumpedBy;

    public Wicket(WicketType wicketType, String playerOut, String caughtBy, String runOutBy, String
stumpedBy) {
        this.wicketType = wicketType;
        this.playerOut = playerOut;
        this.caughtBy = caughtBy;
        this.runOutBy = runOutBy;
        this.stumpedBy = stumpedBy;
    }
}

// Class representing a Commentary with the text, commentator, and timestamp of creation
class Commentary {
    private String text;
    private String createdBy;
    private LocalDate createdAt;

    public Commentary(String text, String commentator) {
        this.text = text;
        this.createdBy = commentator;
        this.createdAt = LocalDate.now();
    }
}
```

```
// Class representing an Inning with the number, start time, and overs
class Inning {
    private int number;
    private LocalDate startTime;
    private List<Over> overs = new ArrayList<>();

    public Inning(int number, LocalDate startTime) {
        this.number = number;
        this.startTime = startTime;
    }

    public void addOver(Over over) {
        overs.add(over);
    }
}

// Abstract class representing a Match with number, start time, result, teams, innings, umpires, etc.
abstract class Match {
    private int number;
    private LocalDate startTime;
    private MatchResult result;
    private List<String> teams = new ArrayList<>();
    private List<Inning> innings = new ArrayList<>();
    private List<String> umpires = new ArrayList<>();
    private String referee;
    private List<String> commentators = new ArrayList<>();
    private List<String> matchStats = new ArrayList<>();

    public Match(int number, LocalDate startTime, String referee) {
        this.number = number;
        this.startTime = startTime;
        this.result = MatchResult.LIVE;
        this.referee = referee;
    }

    public void assignStadium(String stadium) {
        // implementation
    }

    public void assignReferee(String referee) {
        // implementation
    }
}
```

```
// Class representing an ODI match, inheriting from the Match class
class ODI extends Match {
    public ODI(int number, LocalDate startTime, String referee) {
        super(number, startTime, referee);
    }
}

// Class representing a Test match, inheriting from the Match class
class Test extends Match {
    public Test(int number, LocalDate startTime, String referee) {
        super(number, startTime, referee);
    }
}
```