Projects

LIVE CRICKET SCORE APP

LEARNING OBJECTIVES

The objectives of this Cricket Score application is:

1. Create a simple console-based application in Java
2. Access the CricBuzz Web Service APIs to get the score data
3. Primary use of java networking API (java.net) for accessing a web service
4. Use Java API for XML Processing (JAXP) is for processing/parsing XML data (https://docs.oracle.com/javase/tutorial/jaxp/TOC.html)
5. To use decision-making constructs for applying application logic
6. Demonstrate the inheritance and polymorphism principles

INTRODUCTION

Live Cricket Score App, is a simple console-based application.

This application uses the CricBuzz API to get the desired details.

The API exposes the match data in XML format (Extensible Markup Language, which is commonly used to describe Data). XML is used to pass data between applications, often those applications are written in different programming languages, hence, XML gives us a common data format.

The application starts by providing the User its Menu options and asking for a choice.

For Choice=1, the application lists the currently ongoing matches along with their ID (identifier).

Choice=2, is for the case where the User already knows the ID of the match and is interested to know the score. After providing Choice=2, User is again asked for the ID of the match. Upon providing a valid ID, the application prints the score of that match on console.

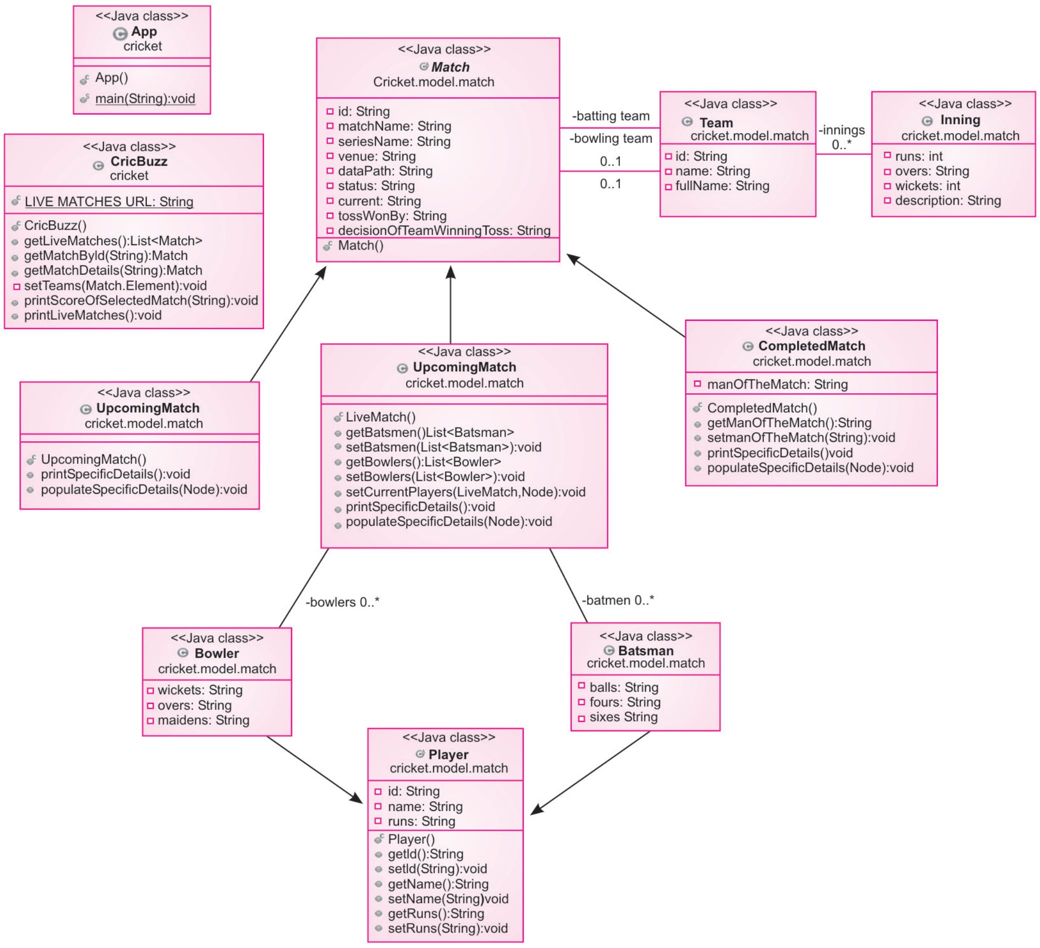
This application maps the XML and creates classes for ease of use.

The classes defined in this application are **Match, Team and Inning**.

For choice =2 and given matchID, the match can be in 1 of the 3 possible states (Live, Upcoming or Completed), and its display logic will differ. For example, in completed match, we display the man-of-the-match; in live match, we display the current playing batsmen and bowlers. This is handled via inheritance and polymorphism.

CRICKET APP PROGRAM

Class Diagram



Source Code

The source code is divided into different classes for the ease of use and modularity.

App.java

```
package cricket;
import java.util.Scanner;

/**
 * Entry point of application, Main method
 *
 */
public class App {
    public static void main(String[] args) throws Exception {
        CricBuzz c = new CricBuzz();
        Scanner sc = new Scanner(System.in);

        System.out.println("Menu \n" + "1. Get Details of Live Matches \n"
            + "2. If you already know matchID, press 2 \n" + " Enter your
            input ...");
        int choice = sc.nextInt();

        switch (choice) {
        case 1:
            // If choice = 1, print summary of live matches and exit
            c.printLiveMatches();
            break;
        case 2:
            /*
             * If choice =2, ask user to enter the ID of desired match
             * If ID is correct,
             * print the detailed score of the match
             */
            System.out.println("Enter the ID of the match, to see its
            score ...");
            String matchId = sc.next();
            c.printScoreOfSelectedMatch(matchId);
            break;
        default:
            throw new IllegalArgumentException(String.format("Choice %d is
            invalid", choice));
        }
        System.out.println("********************END OF
        PROGRAM*************************");
    }
}
```

CricBuzz.java

```
package cricket;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

import cricket.model.match.Match;
import cricket.model.match.UpcomingMatch;
import cricket.model.team.Team;

/**
 * Class where all the work happens.
 * <ul>
 *   <li>1. Access the CricBuzz API to get livematches/score of desired
     match
 *   <li>2. Parses the XML content using JAXP
 *   <li>3. Also maps the content to appropriate model class, so that
     objects are
 *       created with match details by parsing XML
 *   <li>4. Finally it has methods to display the details to console too
 * </ul>
 *
 */
public class CricBuzz {
    private static final String LIVE_MATCHES_URL = "http://synd.cricbuzz.
    com/j2me/1.0/livematches.xml";

    /**
     * 1. Consumes the live match API
     * 2. Parses the result and converts #{Match} objects from the same
     * 3. Finally returns List of live matches
     *
     * @return List of live matches
     * @throws IOException
     */
    public List<Match> getLiveMatches() throws IOException {
        List<Match> matches = new ArrayList<>();
        try {
            DocumentBuilderFactorydbFactory = DocumentBuilderFactory.
            newInstance();
            DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
            Document doc = dBuilder.parse(LIVE_MATCHES_URL);
            NodeList nList = doc.getElementsByTagName("match");

            for (int i = 0; i < nList.getLength(); i++) {
                Node matchNode = nList.item(i);
                matches.add(Match.getMatchObjectFromXML(matchNode));
            }
        } catch (Exception e) {
            System.out.println("Error in fetching LIVE_MATCHES");
            e.printStackTrace();
        }
        return matches;
    }

    /**
     * 1. Searches for a particular match
     * 2. Parses the XML and creates the #{Match} object
     * 3. Returns the Match object
     *
     * @param matchID - ID of desired match (Must be valid)
     * @return - #{Match} object containing all the details
     * @throws IOException
     */
    public Match getMatchById(String matchID) throws IOException {
        List<Match> liveMatches = getLiveMatches();
        for (Match m : liveMatches) {
            if (m.getId().equals(matchID)) {
                return m;
            }
        }
        throw new IllegalArgumentException(String.format("No match found
        for ID : %s", matchID));
    }

    /**
     * This method provides Match details in a greater depth
     * ie Teams, Scores, Innings
     *
     *
     * @param matchID
     * @return
     * @throws Exception
     */
    public Match getMatchDetails(String matchID) throws Exception {
        Match match = getMatchById(matchID);
        final String commentryURL = match.getCommentaryURL();
        DocumentBuilderFactory dbFactory = DocumentBuilderFactory.
        newInstance();
```

```
          DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
          Document doc = dBuilder.parse(commentryURL);
          Node matchNode = doc.getElementsByTagName("match").item(0);
          Element matchElement = (Element) matchNode;

          setTeams(match, matchElement);
          match.populateSpecificDetails(matchNode);
          return match;
     }

     /**
      * Determines based on toss and decision post toss
      * that which team is batting and the one which is bowling
      * @param match
      * @param matchElement
      */
     private void setTeams(final Match match, Element matchElement) {
          Element scores = (Element) matchElement.getElementsByTag
          Name("mscr").item(0);
          Node team1Node = matchElement.getElementsByTagName("Tm").item(0);
          Node team2Node = matchElement.getElementsByTagName("Tm").item(1);

          Team team1 = Team.generateTeamObject(team1Node);
          Team team2 = Team.generateTeamObject(team2Node);
          if(match instanceof UpcomingMatch) {
               // we dont know batting and bowling teams. Set in random order
               match.setBattingTeam(team1);
               match.setBowlingTeam(team2);
               return;
          }

          Node battingTeamNode = scores.getElementsByTagName("btTm").
          item(0);
          Node bowlingTeamNode = scores.getElementsByTagName("blgTm").
          item(0);

          if(match.getTossWonBy().equalsIgnoreCase(team1.getFullName())) {
               // TEAM 1 won the toss
               if(match.getDecisionOfTeamWinningToss().equalsIgnoreCase
               ("Fielding")) {
                    match.setBattingTeam(team2);
                    match.setBowlingTeam(team1);
               } else {
                    match.setBattingTeam(team1);
                    match.setBowlingTeam(team2);
                }
                team1.setInnings(battingTeamNode);
                team2.setInnings(bowlingTeamNode);
            } else {
                // TEAM 2 won the toss
                if(match.getDecisionOfTeamWinningToss().equalsIgnoreCase
                ("Fielding")) {
                    match.setBattingTeam(team1);
                    match.setBowlingTeam(team2);
                } else {
                    match.setBattingTeam(team2);
                    match.setBowlingTeam(team1);
                }
                team2.setInnings(battingTeamNode);
                team1.setInnings(bowlingTeamNode);
            }
        }

        /**
         * Prints the score of selected Match in detail
         * @param matchId
         * @throws Exception
         */
        public void printScoreOfSelectedMatch(String matchId) throws Exception
{
            Match m = getMatchDetails(matchId);
            m.printMatchDetails();
        }

        /**
         * Prints the sumary of live on going matches
         * @throws IOException
         */
        public void printLiveMatches() throws IOException {
            System.out.println("=============LIVE MATCHES===============");
            List<Match> matches = getLiveMatches();
            int i = 1;
            for (Match m : matches) {
                m.printMatchSummary(i++);
            }
            System.out.println("=======================================");
        }
}
```

Constants.java

```
package cricket;

/**
 * Class containing all the constants needed by entire application
 *
 * @author hemant
 *
 */
public class Constants {

    public static final class MATCH_XML {
        public static final String ID = "id";
        public static final String DESC = "mchdesc";
        public static final String NUMBER = "mnum";
        public static final String SERIES = "srs";
        public static final String GROUND = "grnd";
        public static final String CITY = "vcity";
        public static final String COUNTRY = "vcountry";
        public static final String CURRENT_STATE = "state";
        public static final String STATUS = "status";
        public static final String DATAPATH = "datapath";
        public static final String TEAM_WINNING_TOSS = "TW";
        public static final String DECISION_OF_TEAM_WINNING_TOSS =
        "decisn";
        public static final String CURRENT = "mchState";
        public static final String MOM_PARENT = "manofthematch";
        public static final String MOM_CHILD = "mom";

        // live match
        public static final String BATSMEN = "btsmn";
        public static final String BOWLERS = "blrs";

    }

    public static final class INNING_XML {
        public static final String RUNS = "r";
        public static final String WICKETS = "wkts";
        public static final String DESC = "desc";
        public static final String OVERS = "ovrs";
    }

    public static final class TEAM_XML {
        public static final String ID = "id";
        public static final String FULL_NAME = “Name”;
        public static final String NAME = “sName”;
        public static final String INNINGS = “Inngs”;
    }

    public static final class PLAYER {
        public static final String ID = “id”;
        public static final String NAME = “sName”;
        public static final String RUNS = “r”;
    }

    public static final class BATSMAN {
        public static final String BALLS = “b”;
        public static final String FOURS = “frs”;
        public static final String SIXES = “sxs”;
    }

    public static final class BOWLER {
        public static final String WICKETS = “wkts”;
        public static final String OVERS = “ovrs”;
        public static final String MAIDENS = “mdns”;
    }
}
```

Match.java

```
package cricket.model.match;
import org.w3c.dom.Element;
import org.w3c.dom.Node;

import cricket.Constants;
import cricket.model.team.Team;

/**
 * Class representing a Cricket match
 * It has following primary attributes
 *
 * 1. Match ID - unique identifier
 * 2. Name, SeriesName
 * 3. Venue
 * 4. DataPath - URI to fetch exact details
 * 5. status - current status
 *
 */
public abstract class Match {
    private String id;
    private String matchName;
    private String seriesName;
    private String venue;
    private String dataPath;
    private String status;
    protected String current;
    private String tossWonBy;
    private String decisionOfTeamWinningToss;

    private Team battingTeam;
    private Team bowlingTeam;

    public String getId() {
         return id;
    }

    public void setId(String id) {
         this.id = id;
    }

    public String getMatchName() {
         return matchName;
    }

    public void setMatchName(String matchName) {
         this.matchName = matchName;
    }

    public String getSeriesName() {
         return seriesName;
    }

    public void setSeriesName(String seriesName) {
         this.seriesName = seriesName;
    }

    public String getVenue() {
         return venue;
    }

    public void setVenue(String venue) {
         this.venue = venue;
```

```
}
public String getDataPath() {
     return dataPath;
}

public void setDataPath(String dataPath) {
     this.dataPath = dataPath;
}

public String getStatus() {
     return status;
}

public void setStatus(String status) {
     this.status = status;
}

public String getCommentaryURL() {
     return dataPath + "commentary.xml";
}

public Team getBattingTeam() {
     return battingTeam;
}

public void setBattingTeam(Team battingTeam) {
     this.battingTeam = battingTeam;
}

public Team getBowlingTeam() {
     return bowlingTeam;
}

public void setBowlingTeam(Team bowlingTeam) {
     this.bowlingTeam = bowlingTeam;
}

public String getCurrent() {
     return current;
}

public void setCurrent(String current) {
     this.current = current;
```

```
}
public String getTossWonBy() {
     return tossWonBy;
}

public void setTossWonBy(String tossWonBy) {
     this.tossWonBy = tossWonBy;
}

public String getDecisionOfTeamWinningToss() {
     return decisionOfTeamWinningToss;
}

public void setDecisionOfTeamWinningToss(String
decisionOfTeamWinningToss) {
     this.decisionOfTeamWinningToss = decisionOfTeamWinningToss;
}

@Override
public int hashCode() {
     final int prime = 31;
     int result = 1;
     result = prime * result + ((id == null) ? 0 : id.hashCode());
     return result;
}

@Override
public boolean equals(Object obj) {
     if (this == obj)
          return true;
     if (obj == null)
          return false;
     if (getClass() != obj.getClass())
          return false;
     Match other = (Match) obj;
     if (id == null) {
          if (other.id != null)
               return false;
     } else if (!id.equals(other.id))
          return false;
     return true;
}
```

```
@Override
public String toString() {
     return "Match [id=" + id + ", matchName=" + matchName + ",
     seriesName=" + seriesName + ", venue=" + venue
               + ", dataPath=" + dataPath + ", status=" + status + "]";
}

/**
 * Static function to create a Match object by parsing the XML Node
 * @param matchNode
 * @return
 */
public static Match getMatchObjectFromXML(Node matchNode) {
     Element e = (Element) matchNode;
     Element state = (Element) e.getElementsByTagName(Constants.MATCH_
     XML.CURRENT_STATE).item(0);


     String current = state.getAttribute(Constants.MATCH_XML.CURRENT);
     Match match = createSpecificMatch(current);

     match.setId(e.getAttribute(Constants.MATCH_XML.ID));
     match.setMatchName(e.getAttribute(Constants.MATCH_XML.DESC) + "( "
     + e.getAttribute(Constants.MATCH_XML.NUMBER) + " )");
     match.setSeriesName(e.getAttribute(Constants.MATCH_XML.SERIES));
     match.setVenue(e.getAttribute(Constants.MATCH_XML.GROUND) + ", " +
     e.getAttribute(Constants.MATCH_XML.CITY) + ", "
               + e.getAttribute(Constants.MATCH_XML.COUNTRY));


     match.setStatus(state.getAttribute(Constants.MATCH_XML.STATUS));
     match.setTossWonBy(state.getAttribute(Constants.MATCH_XML.TEAM_
     WINNING_TOSS));
     match.setDecisionOfTeamWinningToss(state.getAttribute(Constants.
     MATCH_XML.DECISION_OF_TEAM_WINNING_TOSS));
     match.setDataPath(e.getAttribute(Constants.MATCH_XML.DATAPATH));
     return match;
}

/**
 * Print match in detailed fashion
 */
public void printMatchDetails() {
     System.out.println("==================MATCH
     DETAILS==========================”);
     System.out.println(“Series = “ + seriesName);
     System.out.println(“Match = “+ matchName);
     System.out.println(“Status = “ + status);
     System.out.println(“Current State = “ + current);
     System.out.println(“Toss Won By = “ + tossWonBy);
     System.out.println(“Decision of “ + tossWonBy + “ is “ +
     decisionOfTeamWinningToss);

     System.out.println();
     System.out.println(“Batting team”);
     System.out.println(“---------------”);
     System.out.println(battingTeam);
     System.out.println(“---------------”);
     System.out.println(“Bowling team”);
     System.out.println(“---------------”);
     System.out.println(bowlingTeam);
     System.out.println(“---------------”);

     printSpecificDetails();
     System.out.println(“===========================================”);
}

public abstract void populateSpecificDetails(Node matchNode);

/**
 * Prints specific details
 * Implementation is decided by the subclasses
 */
public abstract void printSpecificDetails();

/**
 * Just prints the match summary
 * @param index
 */
public void printMatchSummary(int index) {
     System.out.println(String.format(“%d.  ID  =  %s,  Name:  %s  %s”,
     index, id, seriesName, matchName));
}

/**
 * Based on the state, create specific Match object
 * @param current
 * @return
```

```
      */
     private static Match createSpecificMatch(String current) {
          if(current.equalsIgnoreCase("inprogress")) {
               return new LiveMatch();
          } else if(current.equalsIgnoreCase("complete")) {
               return new CompletedMatch();
          } else {
               // match has not started yet
               return new UpcomingMatch();
          }
     }
}
```

CompletedMatch.java

```
package cricket.model.match;

import org.w3c.dom.Element;
import org.w3c.dom.Node;

import cricket.Constants;

/**
 * Subclass of Match which is currently completed This will override the
base
 * class behaviour (Eg : display match completion stats) as well as extend
it
 * (add more attributes indicating)
 *
 * @author hemant
 *
 */
public class CompletedMatch extends Match {
     public CompletedMatch() {
          current = "complete";
     }

     private String manOfTheMatch;

     public String getManOfTheMatch() {
          return manOfTheMatch;
     }

     public void setManOfTheMatch(String manOfTheMatch) {
          this.manOfTheMatch = manOfTheMatch;
```

```
    }

    /**
     *
     * Prints details which are specific to Completed Match Here they are 1.
       Man Of
     * the Match
     *
     */
    @Override
    public void printSpecificDetails() {
        System.out.println("Man Of the Match =" + manOfTheMatch);
    }

    /**
     * Populates the details for the COMPLETED MATCH
     * ie ManOfTheMatch
     */
    @Override
    public void populateSpecificDetails(Node matchNode) {
        Element matchElement = (Element) matchNode;
        Element momElement = (Element)
        matchElement.getElementsByTagName(Constants.MATCH_XML.MOM_PARENT).
        item(0);
        if(null != momElement) {
            Element momChild = (Element)
            momElement.getElementsByTagName(Constants.MATCH_XML.MOM_
            CHILD).item(0);
            if(null != momChild) {
                this.setManOfTheMatch(momChild.getAttribute("Name"));
            }
        }
    }
}
```

LiveMatch.java

```
package cricket.model.match;

import java.util.ArrayList;
import java.util.List;

import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import cricket.Constants;
import cricket.model.player.Batsman;
import cricket.model.player.Bowler;

/**
 * Subclass of Match which is currently ongoing This will override the base
 * class behaviour (Eg : display match stats) as well as extend it (add
   more
 * attributes indicating the live score)
 *
 * @author hemant
 *
 */
public class LiveMatch extends Match {

    private List<Batsman> batsmen = new ArrayList<>();
    private List<Bowler> bowlers = new ArrayList<>();

    public LiveMatch() {
        current = "inprogress";
    }

    public List<Batsman> getBatsmen() {
        return batsmen;
    }

    public void setBatsmen(List<Batsman> batsmen) {
        this.batsmen = batsmen;
    }

    public List<Bowler> getBowlers() {
        return bowlers;
    }

    public void setBowlers(List<Bowler> bowlers) {
        this.bowlers = bowlers;
    }

    public void setCurrentPlayers(LiveMatch match, Node matchNode) {
        Element e = (Element) matchNode;
        NodeList  batsmenNodes  =  e.getElementsByTagName(Constants.MATCH_
        XML.BATSMEN);

        for (int i = 0; i < batsmenNodes.getLength(); i++) {
            Node batsmanNode = batsmenNodes.item(i);
            this.batsmen.add(Batsman.getFromXML(batsmanNode));
        }
        NodeList bowlerNodes = e.getElementsByTagName(Constants.MATCH_XML.
        BOWLERS);

        for (int i = 0; i < bowlerNodes.getLength(); i++) {
            Node bowlerNode = bowlerNodes.item(i);
            this.bowlers.add(Bowler.getFromXML(bowlerNode));
        }
    }

    /**
     * Prints details which are specific to Live Match
     * Here they are
     * 1. Current batsmen at crease and their score stats
     * 2. Current bowlers at crease and their stats
     */
    @Override
    public void printSpecificDetails() {
        System.out.println("--------Current batsmen at crease --------");
        final String format = "%-30s%-15s%-15s%-15s%-15s %n";
        System.out.printf(format,  "Name",  "Runs",  "Balls",  "Fours",
        "Sixes");
        for(Batsman bat : batsmen) {
            System.out.printf(format,
                    bat.getName(), bat.getRuns(), bat.getBalls(), bat.
                    getFours(), bat.getSixes());
        }

        System.out.println("--------Current bowlers at crease --------");
        System.out.printf(format,  "Name",  "Overs",  "Runs",  "Wickets",
        "Maidens");
        for(Bowler bowl : bowlers) {
            System.out.printf(format,
                    bowl.getName(),   bowl.getOvers(),   bowl.getRuns(),
                    bowl.getWickets(), bowl.getMaidens());
        }

    }

    /**
     * Populate specific attributes for a Live match
     * ie list of current batsmen and bowlers
```

```
      */
     @Override
     public void populateSpecificDetails(Node matchNode) {
          setCurrentPlayers(this, matchNode);
     }
}
```

UpcomingMatch.java

```
package cricket.model.match;

import org.w3c.dom.Node;

/**
 * Subclass of Match which is currently upcoming
 * This will override the base
 *
 * @author hemant
 *
 */
public class UpcomingMatch extends Match {
     public UpcomingMatch() {
          current = "upcoming";
     }

     /**
      * Prints details which are specific to UPCOMING Match
      */
     @Override
     public void printSpecificDetails() {
          // Nothing specific here
     }

     /**
      * Populates the details for the UPCOMING MATCH
      */
     @Override
     public void populateSpecificDetails(Node matchNode) {
          // Nothing specific here
     }
}
```

Player.java

```
package cricket.model.match;

import org.w3c.dom.Node;
```

```
/**
 * Subclass of Match which is currently upcoming
 * This will override the base
 *
 * @author hemant
 *
 */
public class UpcomingMatch extends Match {
    public UpcomingMatch() {
        current = "upcoming";
    }

    /**
     * Prints details which are specific to UPCOMING Match
     */
    @Override
    public void printSpecificDetails() {
        // Nothing specific here
    }

    /**
     * Populates the details for the UPCOMING MATCH
     */
    @Override
    public void populateSpecificDetails(Node matchNode) {
        // Nothing specific here
    }
}
```

Batsman.java

```
package cricket.model.player;

import org.w3c.dom.Element;
import org.w3c.dom.Node;

import cricket.Constants;

public class Batsman extends Player {

    private String balls;
    private String fours;
    private String sixes;

    public String getBalls() {
        return balls;
    }

    public void setBalls(String balls) {
        this.balls = balls;
    }

    public String getFours() {
        return fours;
    }

    public void setFours(String fours) {
        this.fours = fours;
    }

    public String getSixes() {
        return sixes;
    }

    public void setSixes(String sixes) {
        this.sixes = sixes;
    }

    public static Batsman getFromXML(Node batsmanNode) {
        Element e = (Element) batsmanNode;
        Batsman bats = new Batsman();
        bats.setId(e.getAttribute(Constants.PLAYER.ID));
        bats.setName(e.getAttribute(Constants.PLAYER.NAME));
        bats.setRuns(e.getAttribute(Constants.PLAYER.RUNS));
        bats.setBalls(e.getAttribute(Constants.BATSMAN.BALLS));
        bats.setFours(e.getAttribute(Constants.BATSMAN.FOURS));
        bats.setSixes(e.getAttribute(Constants.BATSMAN.SIXES));
        return bats;
    }

    @Override
    public String toString() {
        return "Batsman [balls=" + balls + ", fours=" + fours + ", sixes="
        + sixes + ", ID=" + getId()
                 + ", Name=" + getName() + ", Runs=" + getRuns() + "]";
    }


}
```

Bowler.java

```
package cricket.model.player;

import org.w3c.dom.Element;
import org.w3c.dom.Node;

import cricket.Constants;

public class Bowler extends Player {

    private String wickets;
    private String overs;
    private String maidens;

    public String getWickets() {
        return wickets;
    }

    public void setWickets(String wickets) {
        this.wickets = wickets;
    }

    public String getOvers() {
        return overs;
    }

    public void setOvers(String overs) {
        this.overs = overs;
    }

    public String getMaidens() {
        return maidens;
    }

    public void setMaidens(String maidens) {
        this.maidens = maidens;
    }

    public static Bowler getFromXML(Node batsmanNode) {
        Element e = (Element) batsmanNode;
        Bowler bowler = new Bowler();
        bowler.setId(e.getAttribute(Constants.PLAYER.ID));
        bowler.setName(e.getAttribute(Constants.PLAYER.NAME));
        bowler.setRuns(e.getAttribute(Constants.PLAYER.RUNS));
        bowler.setOvers(e.getAttribute(Constants.BOWLER.OVERS));
        bowler.setMaidens(e.getAttribute(Constants.BOWLER.MAIDENS));
        bowler.setWickets(e.getAttribute(Constants.BOWLER.WICKETS));
        return bowler;
    }

    @Override
    public String toString() {
        return "Bowler [wickets=" + wickets + ", overs=" + overs + ",
        maidens=" + maidens + ", getId()= +" getId() + "
                + ", getName()=" + getName() + ", getRuns()=" + getRuns()
                + "]";
    }
}
```

Team.java

```
package cricket.model.team;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

import cricket.Constants;

/**
 * Class representing a cricket team For simplicity it contains
 * 1. Name of the team
 * 2. List of innings played by the team
 *
 */
public class Team {

    private String id;
    private String name;
    private String fullName;
    private List<Inning> innings = new ArrayList<>();

    public String getName() {
        return name;
    }

    public void setName(String name) {
         this.name = name;
    }

    public List<Inning> getInnings() {
         return innings;
    }

    public void setInnings(List<Inning> innings) {
         this.innings = innings;
    }

    public String getFullName() {
         return fullName;
    }

    public void setFullName(String fullName) {
         this.fullName = fullName;
    }

    public String getId() {
         return id;
    }

    public void setId(String id) {
         this.id = id;
    }

    @Override
    public String toString() {
         StringBuilder builder = new StringBuilder();
         builder.append("Name : " + name).append("\n");
         for (Inning inn : innings) {
              builder.append(inn.toString()).append("\n");
         }
         return builder.toString();
    }

    @Override
    public int hashCode() {
         final int prime = 31;
```

```
        int result = 1;
        result = prime * result + ((id == null) ? 0 : id.hashCode());
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Team other = (Team) obj;
        if (id == null) {
            if (other.id != null)
                return false;
        } else if (!id.equals(other.id))
            return false;
        return true;
    }

    /**
     * Static function to create a Team object by parsing the XML Node
     *
     * @param node
     * @return
     */
    public static Team generateTeamObject(Node teamNode) {
        Element e = (Element) teamNode;
        Team team = new Team();

        team.setId(e.getAttribute(Constants.TEAM_XML.ID));
        team.setName(e.getAttribute(Constants.TEAM_XML.NAME));
        team.setFullName(e.getAttribute(Constants.TEAM_XML.FULL_NAME));
        return team;
    }

    public void setInnings(Node teamNode) {
        Element e = (Element) teamNode;
        NodeList inningNodes = e.getElementsByTagName(Constants.TEAM_XML.
        INNINGS);

        for (int i = 0; i < inningNodes.getLength(); i++) {
                Node inningNode = inningNodes.item(i);
                this.innings.add(Inning.getInningFromXML(inningNode));
            }
            Collections.sort(this.innings);
        }
}
```

Inning.java

```
package cricket.model.team;
import org.w3c.dom.Element;
import org.w3c.dom.Node;

import cricket.Constants;

/**
 * Class representing a cricket inning detail
 * The major attributes are runs, overs, wickets and description (if any-
   first innings/second innings)
 *
 */
public class Inning implements Comparable<Inning> {

    private int runs;
    private String overs;
    private int wickets;
    private String description;

    public int getRuns() {
        return runs;
    }

    public void setRuns(int runs) {
        this.runs = runs;
    }

    public String getOvers() {
        return overs;
    }

    public void setOvers(String overs) {
        this.overs = overs;
    }

    public int getWickets() {
        return wickets;
    }

    public void setWickets(int wickets) {
        this.wickets = wickets;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    @Override
    public String toString() {
        return String.format("%s ===> runs=%s, overs=%s, wickets=%s",
        description, runs, overs, wickets);
    }

    /**
     * Static function to create a Inning object by parsing the XML Node
     * @param node
     * @return
     */
    public static Inning getInningFromXML(Node node) {
        Element e = (Element) node;
        Inning inning = new Inning();
        inning.runs = Integer.valueOf(e.getAttribute(Constants.INNING_XML.
        RUNS));
        inning.description = e.getAttribute(Constants.INNING_XML.DESC);
        inning.overs = e.getAttribute(Constants.INNING_XML.OVERS);
        inning.wickets = Integer.valueOf(e.getAttribute(Constants.INNING_
        XML.WICKETS));
        return inning;
    }

    /**
     * CompareTo is used to define natural ordering of elements in a
       collection
     * A team can have multiple innings (Eg : for a test match)
     * In this case, we need to order the innings
     *
```

```
      *
      * Here innings will be sorted by their description
      *
      * Eg : we have multiple Inning objects in a List with desc as (Inn1,
        Inn3, Inn2)
     *
     * On calling #{Collections.sort(innings);}, the order of objects will
       be [Inn1, Inn2, Inn3]
      *
      */
     @Override
     public int compareTo(Inning o) {
          return this.description.compareTo(o.description);
     }
}
```

RUNNING THE APPLICATION

The application classes are packaged into different packages for modularization.

To run the Live Cricket Score application, we have to perform following steps:

1. Make sure you have Java installed on your system (JDK 1.8 preferably)
2. Go to the parent folder and organize the files in following hierarchy

```
hemant@~/PROJECTS/CRICKET$ tree cricket/
cricket/
├── App.java
├── Constants.java
├── CricBuzz.java
└── model
    ├── match
    │   ├── CompletedMatch.java
    │   ├── LiveMatch.java
    │   ├── Match.java
    │   └── UpcomingMatch.java
    ├── player
    │   ├── Batsman.java
    │   ├── Bowler.java
    │   └── Player.java
    └── team
        ├── Inning.java
        └── Team.java

4 directories, 12 files
hemant@~/PROJECTS/CRICKET$
```

3. Go to the folder location and compile all the .java files. This can be done using wildcard. Class files of all the classes will get created.

```
hemant@~/PROJECTS/CRICKET$ javac cricket/*.java
```

4. After successful compilation of all class files, run the following command:

```
hemant@~/PROJECTS/CRICKET$ java cricket/App
```

4.1 Output in case of Choice =1 (Summary of all live matches will be printed)

```
hemant@~/PROJECTS/CRICKET$ java App
Menu
1. Get Details of Live Matches
2. If you already know matchID, press 2
 Enter your input ...
1
=============LIVE MATCHES===============
1. ID = 4, Name: Pakistan Women tour of Sri Lanka, 2018 ( 1st T20I )
2. ID = 20124, Name: Pakistan Women tour of Sri Lanka, 2018 ( 1st T20I )
3. ID = 19872, Name: Womens T20I Tri-Series in India, 2018 ( 5th Match )
=======================================
********************END OF PROGRAM************************
```

4.2 Output in case of Choice=2, but invalid matchID

```
hemant@~/PROJECTS/CRICKET$ java App
Menu
1. Get Details of Live Matches
2. If you already know matchID, press 2
 Enter your input ...
2
Enter the ID of the match, to see its score ...
123
Exception in thread "main" java.lang.IllegalArgumentException: No match
found for ID : 123
     at CricBuzz.getMatchById(CricBuzz.java:70)
     at CricBuzz.getMatchDetails(CricBuzz.java:83)
     at CricBuzz.printScoreOfSelectedMatch(CricBuzz.java:106)
     at App.main(App.java:30)
```

4.3 Detailed Match Score for Choice=2 and CompletedMatch

```
hemant@~/PROJECTS/CRICKET$ java cricket/App
Menu
1. Get Details of Live Matches
2. If you already know matchID, press 2
 Enter your input ...
```

```
2
Enter the ID of the match, to see its score ...
1
==================MATCH DETAILS==========================
Series = Indian Premier League, 2018
Match = ( 30th Match )
Status = Chennai won by 13 runs
Current State = complete
Toss Won By = Delhi
Decision of Delhi is Fielding

Batting team
---------------
Name : CSK
Inns ===> runs=211, overs=20, wickets=4

---------------
Bowling team
---------------
Name : DD
Inns ===> runs=198, overs=20, wickets=5

---------------
Man Of the Match =Watson
=========================================================
********************END OF PROGRAM************************
hemant@~/PROJECTS/CRICKET$
```

4.4 Detailed Match Score for Choice=2 and UpcomingMatch

```
hemant@~/PROJECTS/CRICKET$ java cricket/App
Menu
1. Get Details of Live Matches
2. If you already know matchID, press 2
 Enter your input ...
2
Enter the ID of the match, to see its score ...
3
==================MATCH DETAILS==========================
Series = Indian Premier League, 2018
Match = ( 31st Match )
Status = Starts on May 01 at 14:30 GMT
```

```
Current State = upcoming
Toss Won By =
Decision of  is

Batting team
---------------
Name : RCB

---------------
Bowling team
---------------
Name : MI

---------------
=========================================================
********************END OF PROGRAM************************
hemant@~/PROJECTS/CRICKET$
```

4.5 Detailed Match Score for Choice=2 and LiveMatch

```
hemant@~/PROJECTS/CRICKET$ java cricket/App
Menu
1. Get Details of Live Matches
2. If you already know matchID, press 2
 Enter your input ...
2
Enter the ID of the match, to see its score ...
1
==================MATCH DETAILS==========================
Series = Indian Premier League, 2018
Match = ( 30th Match )
Status = Chennai won by 13 runs
Current State = inprogress
Toss Won By = Delhi
Decision of Delhi is Fielding

Batting team
---------------
Name : CSK
Inns ===> runs=211, overs=20, wickets=4

---------------
```

```
Bowling team
---------------
Name : DD
Inns ===> runs=198, overs=20, wickets=5

---------------
--------Current batsmen at crease --------
Name                 Runs      Balls          Fours          Sixes
Vijay Shankar*        54        31              1              5
Rahul Tewatia          3         4              0              0
--------Current bowlers at crease --------
Name             Overs          Runs           Wickets        Maidens
DJ Bravo*         3              43             0              0
Lungi Ngidi       4              26             1              0
=========================================================
********************END OF PROGRAM************************
hemant@~/PROJECTS/CRICKET$
```