**A Presentation Report for OOPS THROUGH JAVA LABORATORY (22CS306PC)**

**On**

**CRICKET SCORER**

**Submitted**

**to**

**CMR Technical Campus, Hyderabad**

**In Partial fulfillment for the requirement of the Award of the Degree of**

#### BACHELOR OF TECHNOLOGY

##### in

#### COMPUTER SCIENCE & ENGINEERING

#### By

**MOHAMMAD SUBHAN**

**(227R1A05G3)**

**Under the esteemed guidance of**

## MS.SWATHI

## Assistant Professor



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**CMR TECHNICAL CAMPUS**

***An UGC Autonomous Institute***

**Accredited by NBA & NAAC with A Grade**

**(Approved by AICTE, Affiliated to JNTU, Hyderabad)**

**Kandlakoya (V), Medchal (M), Hyderabad-501 401**

**(2023-2024)**



**CERTIFICATE**

This to certify that, the Presentation entitled is CRICKET SCORER submitted by **Mohammad subhan** bearing the Roll Number **227R1A05G3** of **B.Tech Computer Science and Engineering**, In Partial fulfillment for the requirement of the Presentation and for the award of the **Degree of Bachelor of Technology** during the academic year 2023-24.

**Subject Faculty**

MS.SWATHI

**Name of the student : MOHAMMAD SUBHAN**

**Roll No : 227R1A05G3**

**Year : B.Tech I/II/III/IV**

**Semester : I/II**

**Branch : computer science and engineering**

**Name of the Subject : OOPS THROUGH JAVA LABORATORY**

**Name of the Topic : CRICKET SCORER**

**Date of Viva-voice :**

**Signature of the student:**

**Remarks or comments by faculty :**

**Name of the Faculty : G Swathi**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Subject Viva-voice/ppt/presentation(pp)/case study(cs)** | | | | |
| **Name of the topic &concept** | **Presentation on Topic** | **Viva-voice on topic** | **Total**  **Marks** | **Final**  **Marks** |
| **15** | **15** | **10** | **40** | **5** |
|  |  |  |  |  |

**Signature of the Faculty :**

**Remarks or comments by faculty :**

**Name of the Faculty : G Swathi**

**Signature of the Faculty :**

ABSTRACT

An abstract cricket scorer in Java is a program that helps keep track of the score, players, and other game details during a cricket match. It's like having a virtual scorekeeper that automatically updates the score as the game progresses.

To create an abstract cricket scorer, you would typically design classes and methods to handle various aspects of the game. For example, you might have a "Player" class to store information about each player, such as their name and statistics. You could also have a "Scoreboard" class to keep track of the runs, wickets, and overs.

The scorer would include methods to update the score whenever runs are scored, wickets are taken, or overs are bowled. It would also handle things like switching the batting team, calculating the run rate, and displaying the current status of the game.

By writing an abstract cricket scorer in Java, you can create a flexible and reusable tool that can be used for different cricket matches. It's a fun project that combines the excitement of cricket with the power of programming.

Table of contents

1. INTRODUCTION
2. LITERATURE SURVEY
3. ANALYSIS AND DESIGN
4. IMPLEMENTATION
5. RESULT
6. SCOPE
7. CONCLUSION
8. REFERENCE

**INTRODUCTION**

A cricket scorer is a tool or program that helps keep track of the score, players, and other important details during a cricket match. It's like having a virtual scorekeeper that automatically updates the score as the game progresses.

With a cricket scorer, you can easily monitor the runs, wickets, and overs of both teams. It allows you to record each player's performance, such as their batting and bowling statistics. You can also keep track of important milestones like centuries, half-centuries, and bowling figures.

A cricket scorer can be a standalone program, a mobile app, or even a web-based tool. It's designed to make scoring easier and more efficient, ensuring accuracy and providing valuable insights into the game.

Whether you're a player, coach, or simply a cricket enthusiast, a cricket scorer can be a handy companion to enhance your cricketing experience.

**LITERATURE SURVEY**

1. Manuka Maduranga Hatharasinghe et al. used simulation based approach, Team Composition Approach and brought about a conclusion that a high accuracy scores based model can be developed accurately to predict outcome of cricket matches.

2. Dr.B.Santhosh Kumar, T.Daniya,Dr.J.Ajayan proposed a Paper on Detecting Third Umpire Decisions & Automated Scoring System of Cricket using Image Pre-Processing and Convolutional Neural Network.

3. Harshit Barot, Arya Kothari et al. had done Analysis and Prediction for the Indian Premier League. The methodology that they used were Pre-Processing And Feature Extraction Analysis on the basis of The Toss Factor, Bat and Win or Chase and Win, Targets chased, Runs Scored by teams per over on an average ,Batting Index, Bowling Index.

4. A.N.Wickramasinghe, Roshan D.Yapa created Cricket Match Outcome Prediction Using using R-Studio by accessing Twitter API network analysis. In the paper, According to the logistic models that have created, combination of degree and betweenness centrality measurements have the highest accuracy (92%).

5. Jacob Perricone has predicted Results for Professional Basketball Using NBA API Data using K-nearest neighbours with 10 neighbours and 30 leafs ,Neural Networks, Logistic Regression , Support Vector Machine with the linear, rbf, and sigmoid kernels.

ANALYSIS AND DESIGN

Designing and analyzing a cricket scorer in Java can be a fascinating project. To get started, you can create classes to represent players, teams, and matches. These classes will have attributes and methods to handle different aspects of the game.

For example, the Player class can have attributes like name, runs scored, balls faced, and wickets taken. The Team class can have attributes like team name, total runs, total wickets, and overs played. The Match class can have attributes like the two teams playing, current innings, and match result.

To keep track of scores, you can implement methods to increment runs, update balls faced, calculate run rates, and determine required run rates. You can also handle user input to record actions during the game, such as runs scored, wickets taken, and overs bowled.

To display the results, you can create methods to print the scorecard, show the current state of the game, and display the match result in a user-friendly manner.

Remember to break down the project into smaller tasks, test your code as you go, and make adjustments as needed. Planning and designing your code before implementation can help you stay organized.

**IMPLEMENTATION**

import javax.swing.\*;

import javax.swing.table.DefaultTableModel;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.util.ArrayList;

import java.util.List;

class Player {

String name;

int score;

int ballsPlayed;

int fours;

int sixes;

Player(String name) {

this.name = name;

this.score = 0;

this.ballsPlayed = 0;

this.fours = 0;

this.sixes = 0;

}

void addScore(int runs, int balls, int fours, int sixes) {

this.score += runs;

this.ballsPlayed += balls;

this.fours += fours;

this.sixes += sixes;

}

double getStrikeRate() {

if (ballsPlayed == 0) {

return 0.0;

}

return (score \* 100.0) / ballsPlayed;

class Bowler {

String name;

int runsGiven;

int wicketsTaken;

int oversBowled;

Bowler(String name, int runsGiven, int wicketsTaken, int oversBowled) {

this.name = name;

this.runsGiven = runsGiven;

this.wicketsTaken = wicketsTaken;

this.oversBowled = oversBowled;

}

double getEconomy() {

if (oversBowled == 0) {

return 0.0;

}

return runsGiven / (oversBowled \* 1.0);

class TeamTotal {

int totalRuns;

int totalWickets;

int totalOvers;

int highestScore;

int highestWickets;

String highestScorer;

String highestWicketTaker;

TeamTotal() {

this.totalRuns = 0;

this.totalWickets = 0;

this.totalOvers = 0;

this.highestScore = -1;

this.highestWickets = -1;

this.highestScorer = "";

this.highestWicketTaker = "";

class Summary {

List<TeamTotal> teamTotals;

List<String> teamNames;

Summary(List<TeamTotal> teamTotals, List<String> teamNames) {

this.teamTotals = teamTotals;

this.teamNames = teamNames;

}

JTable generateSummaryTable() {

String[] columnNames = {"Team", "Total Runs", "Total Wickets", "Highest Scorer", "Highest Score", "Highest Wicket-Taker", "Highest Wickets"};

Object[][] data = new Object[teamTotals.size()][columnNames.length];

for (int i = 0; i < teamTotals.size(); i++) {

TeamTotal teamTotal = teamTotals.get(i);

data[i][0] = teamNames.get(i);

data[i][1] = teamTotal.totalRuns;

data[i][2] = teamTotal.totalWickets;

data[i][3] = teamTotal.highestScorer;

data[i][4] = teamTotal.highestScore;

data[i][5] = teamTotal.highestWicketTaker;

data[i][6] = teamTotal.highestWickets;

}

return new JTable(new DefaultTableModel(data, columnNames));

}

String determineWinningTeam() {

int maxRuns = -1;

int winningTeam = -1;

for (int i = 0; i < teamTotals.size(); i++) {

if (teamTotals.get(i).totalRuns > maxRuns) {

maxRuns = teamTotals.get(i).totalRuns;

winningTeam = i;

} else if (teamTotals.get(i).totalRuns == maxRuns) {

if (teamTotals.get(i).totalWickets <

winningTeam = i;

} }

}

return (winningTeam != -1) ? teamNames.get(winningTeam) : "Draw";

class Team {

String name;

List<Player> players;

List<Bowler> bowlers;

Team(String name, List<Player> players) {

this.name = name;

this.players = players;

this.bowlers = new ArrayList<>();

}

class ScoreboardApp {

private JFrame frame;

private JPanel panel;

private List<JTable> battingTables;

private List<JTable> bowlingTables;

private List<DefaultTableModel> battingModels;

private List<DefaultTableModel> bowlingModels

public ScoreboardApp() {

initialize();

}

private void initialize() {

frame = new JFrame("Scoreboard App");

frame.setBounds(100, 100, 1200, 600);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

panel = new JPanel();

frame.getContentPane().add(panel, BorderLayout.CENTER);

panel.setLayout(new GridLayout(4, 1));

teams = new ArrayList<>();

battingTables = new ArrayList<>();

bowlingTables = new ArrayList<>();

battingModels = new ArrayList<>();

for (int i = 0; i < 2; i++) {

int numberOfPlayers = Integer.parseInt(JOptionPane.showInputDialog("Enter the number of players for Team " + (i + 1) + ":"));

List<Player> players = new ArrayList<>();

for (int j = 0; j < numberOfPlayers; j++) {

players.add(new Player(JOptionPane.showInputDialog("Enter name of Player " + (j + 1) + " for Team " + (i + 1) + ":")))

}

Team team = new Team(JOptionPane.showInputDialog("Enter name of Team " + (i + 1) + ":"), players);

teams.add(team);

teamNames.add(team.name);

String[] battingColumnNames = {"Player", "Runs", "Balls", "4s", "6s", "Strike Rate"};

DefaultTableModel battingModel = new DefaultTableModel(battingColumnNames, 0);

battingModels.add(battingModel);

String[] bowlingColumnNames = {"Bowler", "Overs", "Runs Given", "Wickets", "Economy"};

DefaultTableModel bowlingModel = new DefaultTableModel(bowlingColumnNames, 0);

bowlingModels.add(bowlingModel);

JTable battingTable = new JTable(battingModel);

JTable bowlingTable = new JTable(bowlingModel);

battingTables.add(battingTable);

bowlingTables.add(bowlingTable);

JScrollPane battingScrollPane = new JScrollPane(battingTable);

JScrollPane bowlingScrollPane = new JScrollPane(bowlingTable);

panel.add(createTablePanel(battingScrollPane, team.name + " - Batting"));

panel.add(createTablePanel(bowlingScrollPane, team.name + " - Bowling"));

}

for (int i = 0; i < 2; i++) {

teamTotals.add(new TeamTotal());

}

JButton buttonAddScore = new JButton("Add Score");

panel.add(buttonAddScore);

buttonAddScore.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

addScore();

updateScoreboard();

private JPanel createTablePanel(JScrollPane scrollPane, String title) {

JPanel tablePanel = new JPanel(new BorderLayout());

JLabel titleLabel = new JLabel(title, SwingConstants.CENTER);

tablePanel.add(titleLabel, BorderLayout.NORTH);

tablePanel.add(scrollPane, BorderLayout.CENTER);

return tablePanel;

}

private void addScore() {

for (int i = 0; i < teams.size(); i++) {

Team team = teams.get(i);

for (Player player : team.players) {

String runsInput = JOptionPane.showInputDialog("Enter runs scored by " + player.name + ":");

String ballsInput = JOptionPane.showInputDialog("Enter balls played by " + player.name + ":");

String foursInput = JOptionPane.showInputDialog("Enter number of 4s by " + player.name + ":");

String sixesInput = JOptionPane.showInputDialog("Enter number of 6s by " + player.name + ":");

try {

int runs = Integer.parseInt(runsInput);

int balls = Integer.parseInt(ballsInput);

int fours = Integer.parseInt(foursInput);

int sixes = Integer.parseInt(sixesInput);

player.addScore(runs, balls, fours, sixes);

teamTotals.get(i).totalRuns += runs;

teamTotals.get(i).totalOvers += balls / 6;

if (runs > teamTotals.get(i).highestScore) {

teamTotals.get(i).highestScore = runs;

teamTotals.get(i).highestScorer = player.name;

}

} catch (NumberFormatException ex) {

JOptionPane.showMessageDialog(frame, "Invalid input. Please enter valid integers.");

}

}

for (Player selectedPlayer : team.players) {

String oversBowledInput = JOptionPane.showInputDialog("Enter overs bowled by " + selectedPlayer.name + ":");

String runsGivenInput = JOptionPane.showInputDialog("Enter runs given by " + selectedPlayer.name + ":");

String wicketsTakenInput = JOptionPane.showInputDialog("Enter wickets taken by " + selectedPlayer.name + ":");

try {

int oversBowled = Integer.parseInt(oversBowledInput);

int runsGiven = Integer.parseInt(runsGivenInput);

int wicketsTaken = Integer.parseInt(wicketsTakenInput);

team.bowlers.add(new Bowler(selectedPlayer.name, runsGiven, wicketsTaken, oversBowled)); teamTotals.get(i).totalWickets += wicketsTaken;

if (wicketsTaken > teamTotals.get(i).highestWickets) {

teamTotals.get(i).highestWickets = wicketsTaken;

teamTotals.get(i).highestWicketTaker = selectedPlayer.name;

}

} catch (NumberFormatException ex) {

JOptionPane.showMessageDialog(frame, "Invalid input. Please enter valid integers.");

}

private void updateScoreboard() {

for (int i = 0; i < teams.size(); i++)

Team team = teams.get(i);

DefaultTableModel battingModel = battingModels.get(i);

battingModel.setRowCount(0);

for (Player player : team.players) {

Object[] battingRowData = {

player.name,

player.score,

player.ballsPlayed,

player.fours,

player.sixes,

player.getStrikeRate()

};

battingModel.addRow(battingRowData);

}

}

for (int i = 0; i < teams.size(); i++) {

Team team = teams.get((i + 1) % teams.size())

DefaultTableModel bowlingModel = bowlingModels.get(i);

bowlingModel.setRowCount(0);

for (Bowler bowler : team.bowlers) {

Object[] bowlingRowData = {

bowler.name,

bowler.oversBowled,

bowler.runsGiven,

bowler.wicketsTaken,

bowler.getEconomy()

};

bowlingModel.addRow(bowlingRowData);

displaySummary();}

private void displaySummary() {

Summary summary = new Summary(teamTotals, teamNames);

JTable summaryTable = summary.generateSummaryTable();

JScrollPane summaryScrollPane = new JScrollPane(summaryTable)

summaryText += "Team " + (i + 1) + ": " + teamTotal.totalRuns + " runs, " + teamTotal.totalWickets + " wickets\n";

}

summaryText += "-----------------------------------------\n";

summaryText += "Highest Scorers:\n";

for (int i = 0; i < teams.size(); i++) {

TeamTotal teamTotal = teamTotals.get(i);

summaryText += "Team " + (i + 1) + ": " + teamTotal.highestScorer + " (" + teamTotal.highestScore + " runs)\n";

}

summaryText += "-----------------------------------------\n";

summaryText += "Highest Wicket-Takers:\n";

for (int i = 0; i < teams.size(); i++) {

TeamTotal teamTotal = teamTotals.get(i);

summaryText += "Team " + (i + 1) + ": " + teamTotal.highestWicketTaker + " (" + teamTotal.highestWickets + " wickets)\n";}

JOptionPane.showMessageDialog(frame, summaryText, "Game Summary", JOptionPane.INFORMATION\_MESSAGE) }

public void setVisible(boolean visible) {

frame.setVisible(visible);}

public static void main(String[] args) {

SwingUtilities.invokeLater(new Runnable() {

public void run() {

try {

ScoreboardApp window = new ScoreboardApp();

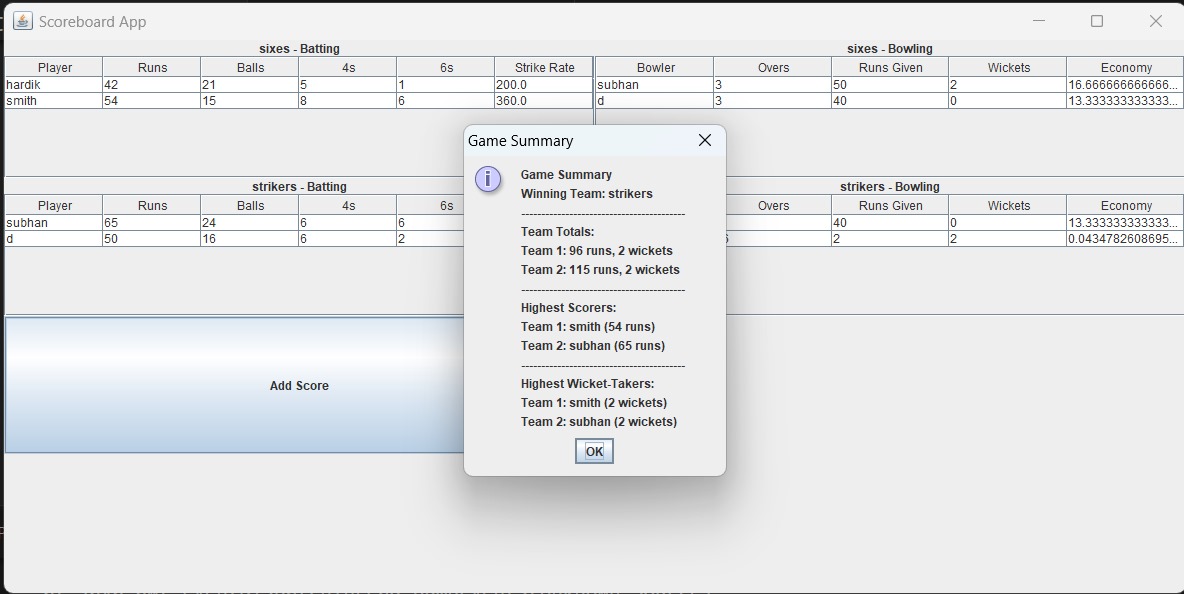
window.setVisible(true);

} catch (Exception e) {

e.printStackTrace();

}

}

});

SCOPE

1. **Local Cricket Matches:**
   * Local cricket tournaments and matches at various levels can benefit from a Java-based scorer. It can streamline the scoring process, reduce errors, and provide quick and accurate updates for players, coaches, and spectators.
2. **School and College Tournaments:**
   * Educational institutions organizing cricket tournaments can utilize a Java scorer to manage and record match scores. The program can be customized to fit the specific rules and formats of school or college-level competitions.
3. **Club Cricket:**
   * Cricket clubs organizing regular matches can adopt a Java scorer to maintain records, analyze player performance, and keep track of team statistics. The program can contribute to a more professional and organized approach to club-level cricket.
4. **Professional Cricket Leagues:**
   * Professional and semi-professional cricket leagues can employ a Java scorer for official matches. The program's scalability allows it to handle the complexities of various formats, including T20, ODI, and Test matches.
5. **Online Platforms:**
   * Integration with online platforms can extend the reach of a Java scorer, allowing users to follow live updates, access historical match data, and engage with cricket statistics through web or mobile applications.
6. **Cricket Analytics:**
   * The collected data can be utilized for cricket analytics, offering insights into player and team performance. This could be valuable for coaches, analysts, and cricket enthusiasts interested in in-depth statistical analysis.

Conclusion

1. **Automation of Scorekeeping:**
   * The Java program automates the process of scorekeeping, eliminating manual errors and ensuring accurate and reliable score calculations.
2. **Real-time Updates:**
   * The scorer provides real-time updates on the match progress, keeping both players and spectators informed about the latest scores, wickets, and other relevant statistics.
3. **User-Friendly Interface:**
   * A well-designed graphical user interface (GUI) can enhance the user experience, making it easy for scorers to input data and for viewers to follow the match without confusion.
4. **Scalability:**
   * The program can be expanded to accommodate various match formats, including Test matches, One Day Internationals (ODIs), and Twenty20 (T20) games, showcasing its flexibility and scalability.
5. **Data Persistence:**
   * The scorer can store match data persistently, allowing users to retrieve historical information and analyze player or team performance over time.
6. **Statistical Analysis:**
   * The scorer can potentially include statistical analysis features, such as run rates, batting averages, and bowling averages, providing deeper insights into player and team performance.
7. **Adherence to Cricket Rules:**
   * The program follows the rules and conventions of cricket, ensuring that scores are calculated accurately based on runs scored, wickets taken, and other parameters defined by the sport.
8. **Customization:**
   * The scorer may allow customization based on specific league or tournament rules, catering to the diverse needs of different cricket competitions.

REFERENCE

Harshit Barot, Arya Kothari et al. had done Analysis and Prediction for the Indian Premier League. The methodology that they used were Pre-Processing And Feature Extraction Analysis on the basis of The Toss Factor, Bat and Win or Chase and Win, Targets chased, Runs Scored by teams per over on an average ,Batting Index, Bowling Index.

4. A.N.Wickramasinghe, Roshan D.Yapa created Cricket Match Outcome Prediction Using using R-Studio by accessing Twitter API network analysis. In the paper, According to the logistic models that have created, combination of degree and betweenness centrality measurements have the highest accuracy (92%).

5. Jacob Perricone has predicted Results for Professional Basketball Using NBA API Data using K-nearest neighbours with 10 neighbours and 30 leafs ,Neural Networks, Logistic Regression , Support Vector Machine with the linear, rbf, and sigmoid kernels.