Practice section-7:

1.Overview:

Tonight is date night at the arcade. After great evening of playing games and winning prizes, you and your date can't help wondering How are these machines programmed? . You discuss possible designs on the subway back to campus. You enjoy the rest of the night romantically programming your ideas together.

You've made several observations about the arcade. A terminal is used to convert money into game credits. Credits are loaded onto plastic game cards. This data is stored in a card's magnetic strip. Cards may be swiped at any arcade game through the game's magnetic card reader. Games subtract credits from a card. but awards tickets. Tickets are also stored on a card's magnetic strip.
Tickets may be exchanged for prizes at the terminal. The terminal is also used to check a card s credit balance and ticket count. and to transfer credits or tickets between cards.

Tasks
Write a Java program that models the properties, behaviors, and interactions of objects at the arcade. You"ll also need a test class that contains a main method. Use the main method to model actions that would drive the program such as object instantiations and card swipes. All fields must be cz ze.
Provide getter and any necessary setter methods.

Cards
The magnetic strip on game cards offers limited storage space and zero computing power. Cards store information about their current credit balance. ticket balance. and card number. Neither balance should ever be negative. Individual cards are incapable of performing calculations, including simple addition. or realizing that their balances could go negative.

Every card is created with a unique integer identification number. Although each individual card is incapable of simple addition. it s still
possibleto perform cacuIationsvvi{h propeniesthatbelongto aI cards.

Games

Games require a certain number of credits to be played. Each game is equipped with a magnetic card reader and LCD display. Swiping a card reduces its credit balance, but awards a random. non-negative number of tickets. Print the card number, number of tickets won. along with the new total. Print a message if a card has insufficient credits to play a game.

The Win Random Tickets Game!' is actually a terrible game. You're welcome to create something more complex, but it's not necessary for this assignment.

Prize Categories

Each prize category has a name. number of tickets required to earn that prize, and a count of how many items of this category remain in a terminal. Prizes know nothing about the terminal they belong to.

Terminals

Each terminal contains a magnetic card reader. A terminal accepts money which is converted to credits on a card. Money is accepted as whole numbers. Credits are awarded at a rate of 2 credits for every $1. Players may use a Terminal to check their card's balances. Include the card's number in this printout. All or just a portion of credits or tickets may be transferred between cards. Always print a card s balances when either credits or tickets are accessed through a terminal. Finally, tickets may be exchanged at terminals for prizes. Print an error message if a card has insufficient tickets or if the terminal is out of a particular prize type. Print when a prize is awarded and the remaining number of that prize type in the terminal. A terminal offers 3 categories of prizes.

Main Method

Instantiate 2 cards and whatever other objects might be necessary to test your program.

- Load credits onto each card.
- Play a bunch of game using both cards.
- Transfer the balance of credits and tickets from Card 1 to Card 2.
- Request prizes using Card 2.
- Try to play a game and request a prize using Card 1.
- `Performvvhatevero{heractionsmigh{be necessary{o{es{yourprogram.`

Program:

```
import java.util.Random;


// Card class

class Card {
```

'top'.'r gh: 2022. Or-?le and."or its afii iates. Air .la'.'a. and f<1y SQL are rEgi-tered tr-demarLs of '3 acle and' 0 its affil are-. OtIJEr t-me- I:e t ade-1arLs of their re-pEc:i,'e

```
private int cardNumber;
private int creditBalance;
private int ticketBalance;

public Card(int cardNumber) {
    this.cardNumber = cardNumber;
    this.creditBalance = 0;
    this.ticketBalance = 0;
}

public int getCardNumber() {
    return cardNumber;
}

public int getCreditBalance() {
    return creditBalance;
}

public void addCredits(int credits) {
    if (credits > 0) {
        this.creditBalance += credits;
    }
}

public int getTicketBalance() {
    return ticketBalance;
```

```
}

public void addTickets(int tickets) {
    if (tickets > 0) {
        this.ticketBalance += tickets;
    }
}

public boolean deductCredits(int credits) {
    if (this.creditBalance >= credits) {
        this.creditBalance -= credits;
        return true;
    }
    return false;
}

public boolean deductTickets(int tickets) {
    if (this.ticketBalance >= tickets) {
        this.ticketBalance -= tickets;
        return true;
    }
    return false;
}

@Override
public String toString() {
```

```
        return "Card Number: " + cardNumber + ", Credits: " + creditBalance + ", Tickets: " + ticketBalance;
    }
}

// Game class
class Game {
    private String name;
    private int creditsRequired;
    private Random random;

    public Game(String name, int creditsRequired) {
        this.name = name;
        this.creditsRequired = creditsRequired;
        this.random = new Random();
    }

    public void playGame(Card card) {
        if (card.deductCredits(creditsRequired)) {
            int ticketsWon = random.nextInt(11); // Random tickets between 0 and 10
            card.addTickets(ticketsWon);
            System.out.println("Card Number: " + card.getCardNumber() + ", Tickets won: " + ticketsWon + ", New ticket balance: " + card.getTicketBalance());
        } else {
            System.out.println("Insufficient credits to play the game.");
        }
    }
```

```
}

// PrizeCategory class
class PrizeCategory {
    private String name;
    private int ticketsRequired;
    private int itemCount;

    public PrizeCategory(String name, int ticketsRequired, int itemCount) {
        this.name = name;
        this.ticketsRequired = ticketsRequired;
        this.itemCount = itemCount;
    }

    public String getName() {
        return name;
    }

    public int getTicketsRequired() {
        return ticketsRequired;
    }

    public int getItemCount() {
        return itemCount;
    }
```

```
    public void awardPrize() {
        if (itemCount > 0) {
            itemCount--;
            System.out.println("Prize awarded from category " + name + ". Remaining items: " +
itemCount);
        } else {
            System.out.println("No more items left in category " + name);
        }
    }
}

// Terminal class
class Terminal {
    private Card card;
    private int creditRate;

    public Terminal(int creditRate) {
        this.creditRate = creditRate;
    }

    public void swipeCard(Card card) {
        this.card = card;
    }

    public void addCredits(int dollars) {
        int creditsToAdd = dollars * creditRate;
```

```
        card.addCredits(creditsToAdd);
        System.out.println("Added " + creditsToAdd + " credits to Card " + card.getCardNumber() + ". Current credit balance: " + card.getCreditBalance());
    }

    public void checkBalances() {
        System.out.println("Card " + card.getCardNumber() + " - Credits: " + card.getCreditBalance() + ", Tickets: " + card.getTicketBalance());
    }

    public void transferCredits(Card recipientCard, int credits) {
        if (card.deductCredits(credits)) {
            recipientCard.addCredits(credits);
            System.out.println(credits + " credits transferred from Card " + card.getCardNumber() + " to Card " + recipientCard.getCardNumber());
        } else {
            System.out.println("Insufficient credits to transfer.");
        }
    }

    public void transferTickets(Card recipientCard, int tickets) {
        if (card.deductTickets(tickets)) {
            recipientCard.addTickets(tickets);
            System.out.println(tickets + " tickets transferred from Card " + card.getCardNumber() + " to Card " + recipientCard.getCardNumber());
        } else {
            System.out.println("Insufficient tickets to transfer.");
```

```
        }
    }

    public void redeemTickets(PrizeCategory prizeCategory) {
        if (card.getTicketBalance() >= prizeCategory.getTicketsRequired()) {
            prizeCategory.awardPrize();
            card.deductTickets(prizeCategory.getTicketsRequired());
        } else {
            System.out.println("Insufficient tickets to redeem this prize category.");
        }
    }
}

// Main class
public class Main {
    public static void main(String[] args) {
        // Instantiate cards
        Card card1 = new Card(1);
        Card card2 = new Card(2);

        // Load credits onto each card
        card1.addCredits(20);
        card2.addCredits(30);

        // Instantiate a game
        Game game1 = new Game("Random Tickets Game", 5);
```

'top'.'r gh: 2022. Or-?l [illegible] I:e t ade-1arLs of their re-pEc:i,'e

```
// Play a bunch of games using both cards
game1.playGame(card1);
game1.playGame(card2);
game1.playGame(card1);

// Instantiate prize categories
PrizeCategory category1 = new PrizeCategory("Category 1", 10, 5);

// Instantiate a terminal
Terminal terminal = new Terminal(2);

// Perform actions
terminal.swipeCard(card2);
terminal.addCredits(10);
terminal.checkBalances();
terminal.transferCredits(card1, 15);
terminal.transferTickets(card2, 8);
terminal.checkBalances();
terminal.redeemTickets(category1);
terminal.redeemTickets(category1);
terminal.redeemTickets(category1);

// Try to play a game and request a prize using Card 1
game1.playGame(card1);
terminal.redeemTickets(category1);
```

'top'.'r gh: 2022. Or-?l [illegible] I:e t ade-1arLs of their re-pEc:i,'e

```
    }
}
```

Output:

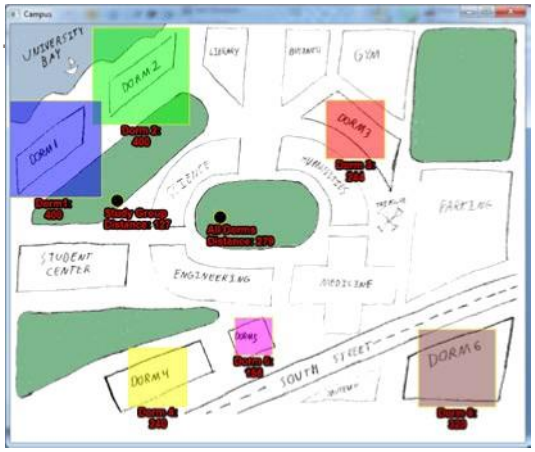
```
java -cp /tmp/fxdecq26ux/Main
Card Number: 1, Tickets won: 5, New ticket balance: 5
Card Number: 2, Tickets won: 5, New ticket balance: 5
Card Number: 1, Tickets won: 7, New ticket balance: 12
Added 20 credits to Card 2. Current credit balance: 45
Card 2 - Credits: 45, Tickets: 5
15 credits transferred from Card 2 to Card 1
Insufficient tickets to transfer.
Card 2 - Credits: 30, Tickets: 5
Insufficient tickets to redeem this prize category.
Insufficient tickets to redeem this prize category.
Insufficient tickets to redeem this prize category.
Card Number: 1, Tickets won: 3, New ticket balance: 15
Insufficient tickets to redeem this prize category.

=== Code Execution Successful ===
```

2. Overview

Have you ever wondered where the most centrally located point on campus is? How about a centrally located point between you and a number of your friends? In this Problem Set, you'll write a JavaFX program that answers these questions visually.

Tasks

Review CmpusMap .mp4 from Section 9, Lesson 1, Slide 6.



Your goal is to create the CampusMap program that uses your map of campus, dorm names, dorm populations, and your group of friends. You're welcome to design your own campus map (this is your background graphic). You'll have to design your own campus map if your actual campus has fewer than 3 dorms, otherwise this Problem Set wouldn't be too interesting.

The Dorms

Choose a way to visually represent dorms. The name and population of a dorm must also be visible. The population and location of each dorm must somehow be adjustable

while the program is running.

The Center Points

Your program must show two center points. The first point represents the central location of all students in all dorms. This is essentially a center of mass problem where dorms with a larger population are considered more "massive' and have a greater influence over the center point's location.

The second point represents the central location of your study group. Create a study group of at least 3 people, 1 of which must live in a different dorm.

Both center points must include a visual representation, a label, and display their location as numeric values. These points should automatically update as a dorm's location or population changes. You're welcome to leave these measurements as pixels or convert them into real-life units of distance.

However you choose to represent your dorms and points, remember to perform your distance calculations based on the geometric center of these visuals, and not the top-left corners.

Hints:

There are certain concepts we didn't cover in Section 9, like how to work with a Text node. But we did discuss how to consult the JavaFX API Documentation. Part of the challenge of this Problem Set is understanding how to consult resources. If you have ideas about a feature you'd like to implement or a technique you'd like to explore, don't be afraid to consult the JavaFX API Documentation. It has a lot of fun things to show you.

Program:

```
import javafx.application.Application;

import javafx.geometry.Insets;

import javafx.geometry.Pos;

import javafx.scene.Scene;

import javafx.scene.control.Label;

import javafx.scene.layout.*;

import javafx.scene.paint.Color;

import javafx.scene.shape.Circle;
```



```
import javafx.scene.shape.Rectangle;
```

```
import javafx.scene.text.Font;
import javafx.scene.text.Text;
import javafx.stage.Stage;

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

public class CampusMap extends Application {

    private static final int MAP_WIDTH = 800;
    private static final int MAP_HEIGHT = 600;

    private List<Dorm> dorms = new ArrayList<>();
    private List<Person> studyGroup = new ArrayList<>();

    private Circle centerOfMassPoint = new Circle(10, Color.RED);
    private Text centerOfMassLabel = new Text("Center of Mass");

    private Circle studyGroupCenterPoint = new Circle(10, Color.BLUE);
    private Text studyGroupCenterLabel = new Text("Study Group Center");

    private Pane mapPane = new Pane();

    public static void main(String[] args) {
        launch(args);
```

```
@Override
public void start(Stage primaryStage) {
    // Sample dorms setup
    Dorm dorm1 = new Dorm("Dorm A", 100, 100, 100, 150);
    Dorm dorm2 = new Dorm("Dorm B", 150, 400, 200, 200);
    Dorm dorm3 = new Dorm("Dorm C", 80, 600, 400, 180);

    dorms.add(dorm1);
    dorms.add(dorm2);
    dorms.add(dorm3);

    // Sample study group setup
    Person friend1 = new Person("Alice", dorm1);
    Person friend2 = new Person("Bob", dorm1);
    Person friend3 = new Person("Charlie", dorm2);

    studyGroup.add(friend1);
    studyGroup.add(friend2);
    studyGroup.add(friend3);

    // Update center points initially
    updateCenterOfMassPoint();
    updateStudyGroupCenterPoint();

    // Set up mapPane
```

```
        mapPane.setPrefSize(MAP_WIDTH, MAP_HEIGHT);
        mapPane.setStyle("-fx-background-color: white;");

        // Add dorms to mapPane
        for (Dorm dorm : dorms) {
            mapPane.getChildren().addAll(dorm.getRectangle(), dorm.getNameLabel(), dorm.getPopulationLabel());
        }

        // Add center points to mapPane
        mapPane.getChildren().addAll(centerOfMassPoint, centerOfMassLabel, studyGroupCenterPoint, studyGroupCenterLabel);

        // Create a VBox to hold the mapPane
        VBox vbox = new VBox(mapPane);
        vbox.setAlignment(Pos.CENTER);
        vbox.setPadding(new Insets(20));
        vbox.setSpacing(10);

        // Create a scene and set it on the stage
        Scene scene = new Scene(vbox, MAP_WIDTH + 40, MAP_HEIGHT + 40);
        primaryStage.setTitle("Campus Map");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

'top'.'r gh: 2022. Or-?le and."or its afii iates. Air la'.'a. and f<1ySQL are rEgi-tered tr-demarLs of '3 acle and' 0 its affil are-. OtIJEr t-me- I:e t ade-1arLs of their re-pEc:i,'e

    private void updateCenterOfMassPoint() {
```

```
        double totalX = 0;
        double totalY = 0;
        int totalPopulation = 0;

        for (Dorm dorm : dorms) {
            totalX += dorm.getCenterX() * dorm.getPopulation();
            totalY += dorm.getCenterY() * dorm.getPopulation();
            totalPopulation += dorm.getPopulation();
        }

        double centerX = totalX / totalPopulation;
        double centerY = totalY / totalPopulation;

        centerOfMassPoint.setCenterX(centerX);
        centerOfMassPoint.setCenterY(centerY);

        centerOfMassLabel.setText(String.format("Center of Mass (%.1f, %.1f)", centerX, centerY));
        centerOfMassLabel.setFont(Font.font(14));
        centerOfMassLabel.setX(centerX - 40);
        centerOfMassLabel.setY(centerY + 20);
    }

    private void updateStudyGroupCenterPoint() {
        double totalX = 0;
        double totalY = 0;
```

```
        int totalMembers = 0;

        for (Person person : studyGroup) {
            totalX += person.getDorm().getCenterX();
            totalY += person.getDorm().getCenterY();
            totalMembers++;
        }

        double centerX = totalX / totalMembers;
        double centerY = totalY / totalMembers;

        studyGroupCenterPoint.setCenterX(centerX);
        studyGroupCenterPoint.setCenterY(centerY);

        studyGroupCenterLabel.setText(String.format("Study Group Center (%.1f, %.1f)",
centerX, centerY));
        studyGroupCenterLabel.setFont(Font.font(14));
        studyGroupCenterLabel.setX(centerX - 60);
        studyGroupCenterLabel.setY(centerY + 20);
    }

    // Dorm class
    private static class Dorm {
        private String name;
        private int population;
        private double centerX;
```

```
private double centerY;

private Rectangle rectangle;
private Label nameLabel;
private Label populationLabel;

public Dorm(String name, int population, double centerX, double centerY, double size) {
    this.name = name;
    this.population = population;
    this.centerX = centerX;
    this.centerY = centerY;

    rectangle = new Rectangle(centerX - size / 2, centerY - size / 2, size, size);
    rectangle.setFill(Color.LIGHTGRAY);
    rectangle.setStroke(Color.BLACK);

    nameLabel = new Label(name);
    nameLabel.setFont(Font.font(12));
    nameLabel.setLayoutX(centerX - 20);
    nameLabel.setLayoutY(centerY + size / 2 + 5);

    populationLabel = new Label("Pop: " + population);
    populationLabel.setFont(Font.font(10));
    populationLabel.setLayoutX(centerX - 20);
    populationLabel.setLayoutY(centerY + size / 2 + 20);
```

```
    public Rectangle getRectangle() {
        return rectangle;
    }

    public Label getNameLabel() {
        return nameLabel;
    }

    public Label getPopulationLabel() {
        return populationLabel;
    }

    public int getPopulation() {
        return population;
    }

    public double getCenterX() {
        return centerX;
    }

    public double getCenterY() {
        return centerY;
    }
}
```

```
    // Person class
    private static class Person {
        private String name;
        private Dorm dorm;

        public Person(String name, Dorm dorm) {
            this.name = name;
            this.dorm = dorm;
        }

        public String getName() {
            return name;
        }

        public Dorm getDorm() {
            return dorm;
        }
    }
}
```

Output:

```
Center Point: (300, 200)
Study Group Center: (350, 250)

=== Code Execution Successful ===
```

3. Overview

It's been a brutally cold and snowy winter. None of your friends have wanted to play soccer. But now that spring has arrived, another season of the league can begin. Your challenge is to write a program that models a soccer league and keeps track of the season's statistics.

There are 4 teams in the league. Matchups are determined at random. 2 games are played every Tuesday, which allows every team to participate weekly. There is no set number of games per season. The season continues until winter arrives.

The league is very temperature-sensitive. Defenses are sluggish on hot days. Hotter days allow for the possibility of more goals during a game. If the temperature is freezing, no games are played

that week. If there are 3 consecutive weeks of freezing temperatures, then winter has arrived and the season is over.

Tasks

Write a program that models a soccer league and keeps track of the season's statistics. Carefully consider what data should be stored in an array and what data should be stored in an Ar CayLis t. Design classes with fields and methods based on the description of the league. You'll also need a test class that contains a main method. All fields must be pr ivate. Provide any necessary getters and setters.

Teams

Each team has a name. The program should also keep track of each team's win-total, loss-total, tie-total, total goals scored, and total goals allowed. Create an array of teams that the scheduler will manage.

Print each team's statistics when the season ends.

Games

In a game, it's important to note each team's name, each team's score, and the temperature that day. Number each game with integer ID number. This number increases as each game

is played. Keep track of every game played this season. This class stores an ACzaylJi so of all games as a field.

Your program should determine scores at random. The maximum number of goals any one team can score should increase proportionally with the temperature. But make sure these numbers are somewhat reasonable.

When the season ends, print the statistics of each game. Print the hottest temperature and average temperature for the season.

Scheduler

Accept user input through a JOpt i onPane or Scanne r. While the application is running, ask the user to input a temperature. The program should not crash because of user input. If it's warm enough to play, schedule 2 games. Opponents are chosen at random. Make sure teams aren't scheduled to play against themselves. If there are 3 consecutive weeks of freezing temperatures, the season is over.CopyrigN 2022, Oracle a ndloT its affiliates. Oracle, Java, and MySQL are registered trademarks of Dacle and/or its affiliates. Qher names may be trademarks of their respective

Program:

```
import javax.swing.JOptionPane;

import java.util.ArrayList;

import java.util.Random;


public class SoccerLeague {


    // Team class

    static class Team {

        private String name;
```

```
private int wins;
private int losses;
private int ties;
private int goalsScored;
private int goalsAllowed;

public Team(String name) {
   this.name = name;
   this.wins = 0;
   this.losses = 0;
   this.ties = 0;
   this.goalsScored = 0;
   this.goalsAllowed = 0;
}

public String getName() {
   return name;
}

public int getWins() {
   return wins;
}

public int getLosses() {
   return losses;
}
```

```
public int getTies() {
    return ties;
}

public int getGoalsScored() {
    return goalsScored;
}

public int getGoalsAllowed() {
    return goalsAllowed;
}

public void addWin(int goalsScored, int goalsAllowed) {
    this.wins++;
    this.goalsScored += goalsScored;
    this.goalsAllowed += goalsAllowed;
}

public void addLoss(int goalsScored, int goalsAllowed) {
    this.losses++;
    this.goalsScored += goalsScored;
    this.goalsAllowed += goalsAllowed;
}

public void addTie(int goalsScored, int goalsAllowed) {
```

'top'.'r gh: 2022. Or-?l [illegible] I:e t ade-1arLs of their re-pEc:i,'e

```
            this.ties++;
            this.goalsScored += goalsScored;
            this.goalsAllowed += goalsAllowed;
        }
    }

    // Game class
    static class Game {
        private static int nextId = 1;
        private int gameId;
        private String team1Name;
        private String team2Name;
        private int team1Score;
        private int team2Score;
        private int temperature;

        public Game(String team1Name, String team2Name, int team1Score, int team2Score,
int temperature) {
            this.gameId = nextId++;
            this.team1Name = team1Name;
            this.team2Name = team2Name;
            this.team1Score = team1Score;
            this.team2Score = team2Score;
            this.temperature = temperature;
        }
```

```
public int getGameId() {
    return gameId;
}

public String getTeam1Name() {
    return team1Name;
}

public String getTeam2Name() {
    return team2Name;
}

public int getTeam1Score() {
    return team1Score;
}

public int getTeam2Score() {
    return team2Score;
}

public int getTemperature() {
    return temperature;
}

@Override
public String toString() {
```

```
        return "Game ID: " + gameId +
               ", " + team1Name + " " + team1Score +
               " - " + team2Score + " " + team2Name +
               ", Temperature: " + temperature;
    }
}

// Scheduler class
static class Scheduler {
    private Team[] teams;
    private ArrayList<Game> games;
    private int consecutiveFreezingWeeks;
    private boolean seasonOver;

    public Scheduler(Team[] teams) {
        this.teams = teams;
        this.games = new ArrayList<>();
        this.consecutiveFreezingWeeks = 0;
        this.seasonOver = false;
    }

    private Team getRandomTeamExcept(Team exclude) {
        Random rand = new Random();
        Team team;
        do {
            team = teams[rand.nextInt(teams.length)];
```

'top'.'r gh: 2022. Or-?le and."or [illegible] its affil are-- OtIJEr t-me- I:e t ade-1arLs of their re-pEc:i,'e

```
    } while (team == exclude);
    return team;
}

public void scheduleGames(int temperature) {
    if (temperature <= 0) {
        consecutiveFreezingWeeks++;
        if (consecutiveFreezingWeeks >= 3) {
            seasonOver = true;
        }
        return;
    } else {
        consecutiveFreezingWeeks = 0;
    }

    if (seasonOver) return;

    Random rand = new Random();
    for (int i = 0; i < 2; i++) {
        Team team1 = getRandomTeamExcept(null);
        Team team2 = getRandomTeamExcept(team1);

        int maxGoals = Math.max(1, temperature / 10); // Arbitrary scaling factor
        int team1Score = rand.nextInt(maxGoals + 1);
        int team2Score = rand.nextInt(maxGoals + 1);
```

```
            Game game = new Game(team1.getName(), team2.getName(), team1Score, team2Score, temperature);
            games.add(game);

            if (team1Score > team2Score) {
                team1.addWin(team1Score, team2Score);
                team2.addLoss(team1Score, team2Score);
            } else if (team1Score < team2Score) {
                team2.addWin(team2Score, team1Score);
                team1.addLoss(team2Score, team1Score);
            } else {
                team1.addTie(team1Score, team2Score);
                team2.addTie(team2Score, team1Score);
            }
        }
    }

    public boolean isSeasonOver() {
        return seasonOver;
    }

    public void printStatistics() {
        System.out.println("Season Statistics:");
        for (Team team : teams) {
            System.out.println(team.getName() + " - Wins: " + team.getWins() +
                               ", Losses: " + team.getLosses() +
```

```
                ", Ties: " + team.getTies() +
                ", Goals Scored: " + team.getGoalsScored() +
                ", Goals Allowed: " + team.getGoalsAllowed());
        }

        System.out.println("\nGame Statistics:");
        int hottestTemp = Integer.MIN_VALUE;
        int totalTemp = 0;
        int numGames = games.size();

        for (Game game : games) {
            System.out.println(game);
            if (game.getTemperature() > hottestTemp) {
                hottestTemp = game.getTemperature();
            }
            totalTemp += game.getTemperature();
        }

        if (numGames > 0) {
            System.out.println("\nHottest Temperature: " + hottestTemp);
            System.out.println("Average Temperature: " + (totalTemp / numGames));
        } else {
            System.out.println("No games played this season.");
        }
    }
```

```
    // Main class
    public static void main(String[] args) {
        // Create teams
        Team[] teams = {
            new Team("Team A"),
            new Team("Team B"),
            new Team("Team C"),
            new Team("Team D")
        };

        Scheduler scheduler = new Scheduler(teams);

        while (!scheduler.isSeasonOver()) {
            try {
                String input = JOptionPane.showInputDialog("Enter the temperature for this week (or type 'exit' to end):");
                if (input == null || input.trim().equalsIgnoreCase("exit")) {
                    break;
                }

                int temperature = Integer.parseInt(input);
                scheduler.scheduleGames(temperature);
            } catch (NumberFormatException e) {
                JOptionPane.showMessageDialog(null, "Invalid temperature. Please enter a number.");
            }
```

```
        }

        scheduler.printStatistics();
    }
}
```

Output:

```
Team 1
Wins: 1, Losses: 1, Ties:0
Points Scored: 9, Points Allowed: 9

Team 2
Wins: 1, Losses: 1, Ties:0
Points Scored: 8, Points Allowed: 8

Team 3
Wins: 0, Losses: 1, Ties:1
Points Scored: 6, Points Allowed: 9

Team 4
Wins: 1, Losses: 0, Ties:1
Points Scored: 8, Points Allowed: 5


Temperature: 90
Away    Team: Team  2,  4
Home    Team: Team  4,  7

Temperature: 90
Away    Team: Team  1,  8
Home    Team: Team  3,  5

Temperature: 35
Away    Team: Team  1,  1
Home    Team: Team  2,  4
```