**ASSIGNMENT 7**

1. import java.util.ArrayList;

import java.util.Random;

import java.util.Scanner;

public class SoccerLeague {

// Team class

public static class Team {

private String name;

private int wins;

private int losses;

private int ties;

private int totalGoalsScored;

private int totalGoalsAllowed;

public Team(String name) {

this.name = name;

this.wins = 0;

this.losses = 0;

this.ties = 0;

this.totalGoalsScored = 0;

this.totalGoalsAllowed = 0;

}

public String getName() {

return name;

}

public int getWins() {

return wins;

}

public int getLosses() {

return losses;

}

public int getTies() {

return ties;

}

public int getTotalGoalsScored() {

return totalGoalsScored;

}

public int getTotalGoalsAllowed() {

return totalGoalsAllowed;

}

public void addWin() {

this.wins++;

}

public void addLoss() {

this.losses++;

}

public void addTie() {

this.ties++;

}

public void addGoalsScored(int goals) {

this.totalGoalsScored += goals;

}

public void addGoalsAllowed(int goals) {

this.totalGoalsAllowed += goals;

}

}

// Game class

public static class Game {

private static int gameCount = 0;

private int gameId;

private Team awayTeam;

private Team homeTeam;

private int awayTeamScore;

private int homeTeamScore;

private int temperature;

public Game(Team awayTeam, Team homeTeam, int temperature) {

this.gameId = ++gameCount;

this.awayTeam = awayTeam;

this.homeTeam = homeTeam;

this.temperature = temperature;

this.awayTeamScore = new Random().nextInt(Math.max(1, temperature / 10));

this.homeTeamScore = new Random().nextInt(Math.max(1, temperature / 10));

awayTeam.addGoalsScored(awayTeamScore);

awayTeam.addGoalsAllowed(homeTeamScore);

homeTeam.addGoalsScored(homeTeamScore);

homeTeam.addGoalsAllowed(awayTeamScore);

if (awayTeamScore > homeTeamScore) {

awayTeam.addWin();

homeTeam.addLoss();

} else if (homeTeamScore > awayTeamScore) {

homeTeam.addWin();

awayTeam.addLoss();

} else {

awayTeam.addTie();

homeTeam.addTie();

}

}

public void printGameResult() {

System.out.println("Game #" + gameId);

System.out.println("Temperature: " + temperature);

System.out.println("Away Team: " + awayTeam.getName() + ", " + awayTeamScore);

System.out.println("Home Team: " + homeTeam.getName() + ", " + homeTeamScore);

}

}

// Scheduler class

public static class Scheduler {

private Team[] teams;

private ArrayList<Game> games;

private ArrayList<Integer> temperatures;

private int freezingWeeks;

public Scheduler(Team[] teams) {

this.teams = teams;

this.games = new ArrayList<>();

this.temperatures = new ArrayList<>();

this.freezingWeeks = 0;

}

public void startSeason() {

Scanner scanner = new Scanner(System.in);

Random random = new Random();

while (true) {

System.out.print("Enter this week's temperature: ");

int temperature = 0;

try {

temperature = Integer.parseInt(scanner.nextLine());

} catch (NumberFormatException e) {

System.out.println("Invalid input. Please enter a valid temperature.");

continue;

}

temperatures.add(temperature);

if (temperature <= 32) {

freezingWeeks++;

System.out.println("Too cold to play.");

if (freezingWeeks >= 3) {

System.out.println("Season is over");

printSeasonResults();

break;

}

continue;

} else {

freezingWeeks = 0;

}

ArrayList<Team> teamsList = new ArrayList<>();

for (Team team : teams) {

teamsList.add(team);

}

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

Team team1 = teamsList.remove(random.nextInt(teamsList.size()));

Team team2 = teamsList.remove(random.nextInt(teamsList.size()));

Game game = new Game(team1, team2, temperature);

games.add(game);

}

}

scanner.close();

}

public void printSeasonResults() {

System.out.println("RESULTS");

for (Team team : teams) {

System.out.println(team.getName());

System.out.println("Wins: " + team.getWins() + ", Losses: " + team.getLosses() + ", Ties: " + team.getTies());

System.out.println("Goals Scored: " + team.getTotalGoalsScored() + ", Goals Allowed: " + team.getTotalGoalsAllowed());

}

for (Game game : games) {

game.printGameResult();

}

int hottestTemp = temperatures.stream().mapToInt(v -> v).max().orElse(0);

double averageTemp = temperatures.stream().mapToInt(v -> v).average().orElse(0.0);

System.out.println("Hottest Temp: " + hottestTemp);

System.out.println("Average Temp: " + averageTemp);

}

}

// Main method

public static void main(String[] args) {

Team[] teams = {

new Team("Team 1"),

new Team("Team 2"),

new Team("Team 3"),

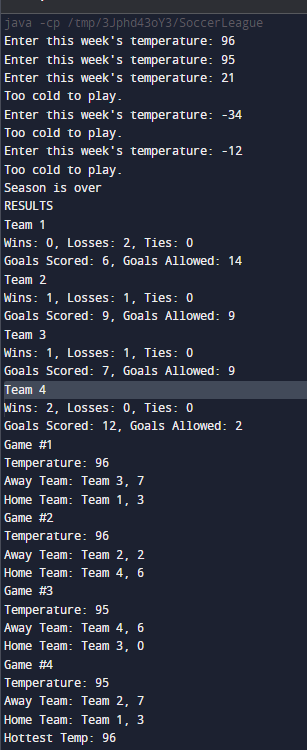
new Team("Team 4")

};

Scheduler scheduler = new Scheduler(teams);

scheduler.startSeason(); }

}



2.import java.util.Random;

class ArcadeCard {

private int cardNumber;

private int creditBalance;

private int ticketBalance;

public ArcadeCard(int cardNumber) {

this.cardNumber = cardNumber;

this.creditBalance = 0;

this.ticketBalance = 0;

}

public int getCardNumber() {

return cardNumber;

}

public int getCreditBalance() {

return creditBalance;

}

public int getTicketBalance() {

return ticketBalance;

}

public void addCredits(int credits) {

creditBalance += credits;

}

public void subtractCredits(int credits) {

if (creditBalance >= credits) {

creditBalance -= credits;

} else {

System.out.println("Insufficient credits.");

}

}

public void addTickets(int tickets) {

ticketBalance += tickets;

}

public void subtractTickets(int tickets) {

if (ticketBalance >= tickets) {

ticketBalance -= tickets;

} else {

System.out.println("Insufficient tickets.");

}

}

}

// Game class

class Game {

private String name;

private int creditsRequired;

private int ticketBalance;

public Game(String name, int creditsRequired) {

this.name = name;

this.creditsRequired = creditsRequired;

this.ticketBalance = 0;

}

public String getName() {

return name;

}

public int getCreditsRequired() {

return creditsRequired;

}

public int getTicketBalance() {

return ticketBalance;

}

public void play(ArcadeCard card) {

if (card.getCreditBalance() >= creditsRequired) {

card.subtractCredits(creditsRequired);

Random random = new Random();

int ticketsWon = random.nextInt(10);

card.addTickets(ticketsWon);

ticketBalance += ticketsWon;

System.out.println("Card " + card.getCardNumber() + " played " + name + " and won " + ticketsWon + " tickets.");

} else {

System.out.println("Card " + card.getCardNumber() + " does not have enough credits to play " + name + ".");

}

}

}

// 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 decreaseItemCount() {

if (itemCount > 0) {

itemCount--;

} else {

System.out.println("No more items left in category " + name);

}

}

}

// Terminal class

class Terminal {

private int creditRate;

private PrizeCategory[] prizeCategories;

public Terminal(int creditRate, PrizeCategory[] prizeCategories) {

this.creditRate = creditRate;

this.prizeCategories = prizeCategories;

}

public void insertMoney(int money, ArcadeCard card) {

int credits = money \* creditRate;

card.addCredits(credits);

System.out.println("Inserted $" + money + " into Card " + card.getCardNumber() + ". Added " + credits + " credits.");

}

public void checkCardBalance(ArcadeCard card) {

System.out.println("Card " + card.getCardNumber() + " has " + card.getCreditBalance() + " credits and " + card.getTicketBalance() + " tickets.");

}

public void transferCredits(ArcadeCard fromCard, ArcadeCard toCard, int credits) {

if (fromCard.getCreditBalance() >= credits) {

fromCard.subtractCredits(credits);

toCard.addCredits(credits);

System.out.println("Transferred " + credits + " credits from Card " + fromCard.getCardNumber() + " to Card " + toCard.getCardNumber() + ".");

} else {

System.out.println("Card " + fromCard.getCardNumber() + " does not have enough credits to transfer.");

}

}

public void requestPrize(ArcadeCard card, int categoryIndex) {

if (categoryIndex >= 0 && categoryIndex < prizeCategories.length) {

PrizeCategory category = prizeCategories[categoryIndex];

if (card.getTicketBalance() >= category.getTicketsRequired()) {

if (category.getItemCount() > 0) {

card.subtractTickets(category.getTicketsRequired());

category.decreaseItemCount();

System.out.println("Card " + card.getCardNumber() + " redeemed a prize from category " + category.getName() + ".");

System.out.println("Remaining " + category.getName() + " prizes: " + category.getItemCount());

} else {

System.out.println("No more prizes left in category " + category.getName() + ".");

}

} else {

System.out.println("Card " + card.getCardNumber() + " does not have enough tickets to redeem a prize from category " + category.getName() + ".");

}

} else {

System.out.println("Invalid prize category index.");

}

}

}

// Main class

public class ArcadeSimulation {

public static void main(String[] args) {

// Initialize cards

ArcadeCard card1 = new ArcadeCard(1);

ArcadeCard card2 = new ArcadeCard(2);

// Add initial credits

card1.addCredits(10);

card2.addCredits(20);

// Initialize games

Game game1 = new Game("Game 1", 5);

Game game2 = new Game("Game 2", 8);

// Play games

game1.play(card1);

game2.play(card2);

// Initialize prize categories

PrizeCategory[] prizeCategories = {

new PrizeCategory("Stuffed Animal", 50, 10),

new PrizeCategory("Action Figure", 100, 5),

new PrizeCategory("Puzzle", 150, 2)

};

// Initialize terminal

Terminal terminal = new Terminal(2, prizeCategories);

// Transfer credits

terminal.transferCredits(card1, card2, 5);

// Request prizes

terminal.requestPrize(card2, 0);

game1.play(card1);

terminal.requestPrize(card1, 1);

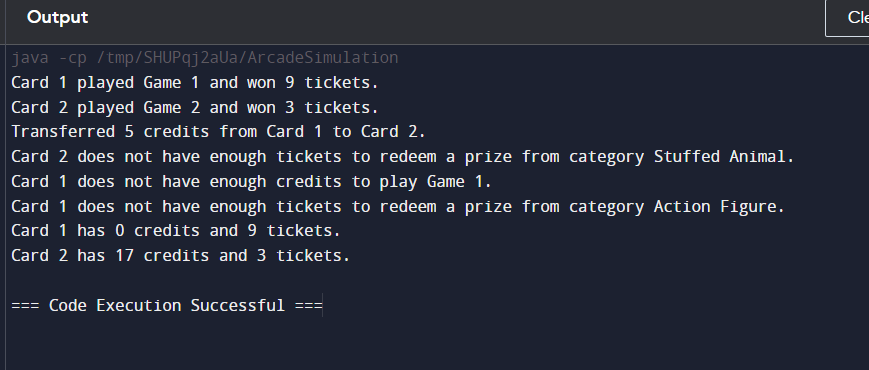
// Check balances

terminal.checkCardBalance(card1);

terminal.checkCardBalance(card2);

}

}



3.import java.util.ArrayList;

import java.util.Scanner;

class Dorm {

private String name;

private int population;

private double x, y;

public Dorm(String name, double x, double y, int population) {

this.name = name;

this.population = population;

this.x = x;

this.y = y;

}

public double getX() {

return x;

}

public double getY() {

return y;

}

public int getPopulation() {

return population;

}

public void setPopulation(int population) {

this.population = population;

}

public void setLocation(double x, double y) {

this.x = x;

this.y = y;

}

public String getName() {

return name;

}

}

class Student {

private Dorm dorm;

public Student(Dorm dorm) {

this.dorm = dorm;

}

public double getX() {

return dorm.getX();

}

public double getY() {

return dorm.getY();

}

}

public class CampusMap {

private static ArrayList<Dorm> dorms = new ArrayList<>();

private static ArrayList<Student> studyGroup = new ArrayList<>();

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Adding dorms

dorms.add(new Dorm("Dorm A", 100, 200, 100));

dorms.add(new Dorm("Dorm B", 500, 300, 150));

dorms.add(new Dorm("Dorm C", 300, 500, 200));

// Adding students to the study group

studyGroup.add(new Student(dorms.get(0)));

studyGroup.add(new Student(dorms.get(1)));

studyGroup.add(new Student(dorms.get(2)));

while (true) {

System.out.println("Current Dorm Populations:");

for (Dorm dorm : dorms) {

System.out.println(dorm.getName() + ": " + dorm.getPopulation());

}

System.out.println("Enter dorm name to update population (or 'exit' to finish):");

String dormName = scanner.nextLine();

if (dormName.equals("exit")) break;

System.out.println("Enter new population:");

int newPopulation = Integer.parseInt(scanner.nextLine());

for (Dorm dorm : dorms) {

if (dorm.getName().equals(dormName)) {

dorm.setPopulation(newPopulation);

}

}

updateCenters();

}

scanner.close();

}

private static void updateCenters() {

double allX = 0, allY = 0, totalPopulation = 0;

for (Dorm dorm : dorms) {

allX += dorm.getX() \* dorm.getPopulation();

allY += dorm.getY() \* dorm.getPopulation();

totalPopulation += dorm.getPopulation();

}

double centerX = allX / totalPopulation;

double centerY = allY / totalPopulation;

System.out.println(String.format("Center of All Students: (%.2f, %.2f)", centerX, centerY));

// Update the study group center

double studyX = 0, studyY = 0;

for (Student student : studyGroup) {

studyX += student.getX();

studyY += student.getY();

}

double studyCenterX = studyX / studyGroup.size();

double studyCenterY = studyY / studyGroup.size();

System.out.println(String.format("Center of Study Group: (%.2f, %.2f)", studyCenterX, studyCenterY)); }

}  
