ASSIGNMENT-7

- Sk.sumiya
192372090

Java Foundations

Practices - Section 7:
Date Night at the Arcade

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 private. 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 possible to perform calculations with properties that belong to all 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.

Perform whatever other actions might be necessary to test your program.

Solution:

Card Class

The Card class will represent a game card with properties for credit balance, ticket balance, and a unique card number.

```
public class Card {
    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 int getTicketBalance() {
    return ticketBalance;
}

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

public void useCredits(int credits) {
    if (credits <= creditBalance) {
        this.creditBalance -= credits;
    } else {
        System.out.println("Insufficient credits.");
    }
}

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

```
    public void transferTo(Card otherCard, int credits, int tickets) {
        if (credits <= this.creditBalance) {
            this.creditBalance -= credits;
            otherCard.addCredits(credits);
        }

        if (tickets <= this.ticketBalance) {
            this.ticketBalance -= tickets;
            otherCard.addTickets(tickets);
        }
    }
}
```

Game Class

The Game class will represent an arcade game. It requires a certain number of credits to play and awards tickets.

```
import java.util.Random;

public class Game {
    private int creditsRequired;

    public Game(int creditsRequired) {
        this.creditsRequired = creditsRequired;
    }

    public void play(Card card) {
        if (card.getCreditBalance() >= creditsRequired) {
```

```
            card.useCredits(creditsRequired);
            int ticketsWon = new Random().nextInt(10); // Random tickets won
            card.addTickets(ticketsWon);
            System.out.println("Card " + card.getCardNumber() + " won " + ticketsWon + "
tickets. Total tickets: " + card.getTicketBalance());
        } else {
            System.out.println("Insufficient credits to play the game.");
        }
    }
}
```

PrizeCategory Class

The PrizeCategory class represents a type of prize that can be redeemed with tickets.

```
public class PrizeCategory {
    private String name;
    private int ticketsRequired;
    private int count;

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

    public String getName() {
        return name;
    }
```

```
    public int getTicketsRequired() {
        return ticketsRequired;
    }

    public int getCount() {
        return count;
    }

    public boolean redeemPrize(Card card) {
        if (card.getTicketBalance() >= ticketsRequired && count > 0) {
            card.addTickets(-ticketsRequired); // Reduce tickets from the card
            count--;
            System.out.println("Prize " + name + " awarded. Remaining: " + count);
            return true;
        } else if (count <= 0) {
            System.out.println("Prize " + name + " is out of stock.");
        } else {
            System.out.println("Insufficient tickets for prize " + name + ".");
        }
        return false;
    }
}
```

Terminal Class

The Terminal class handles operations like adding credits, checking balances, transferring credits and tickets, and redeeming prizes.

```
public class Terminal {
    private PrizeCategory[] prizes;

    public Terminal(PrizeCategory[] prizes) {
        this.prizes = prizes;
    }

    public void addCredits(Card card, int money) {
        int credits = money * 2; // 2 credits per dollar
        card.addCredits(credits);
        System.out.println("Added " + credits + " credits to Card " + card.getCardNumber() +
". Total credits: " + card.getCreditBalance());
    }

    public void checkBalances(Card card) {
        System.out.println("Card " + card.getCardNumber() + " has " +
card.getCreditBalance() + " credits and " + card.getTicketBalance() + " tickets.");
    }

    public void transfer(Card fromCard, Card toCard, int credits, int tickets) {
        fromCard.transferTo(toCard, credits, tickets);
        System.out.println("Transferred " + credits + " credits and " + tickets + " tickets from
Card " + fromCard.getCardNumber() + " to Card " + toCard.getCardNumber() + ".");
        checkBalances(fromCard);
        checkBalances(toCard);
    }
```

```
    public void redeemPrize(Card card, String prizeName) {
        for (PrizeCategory prize : prizes) {
            if (prize.getName().equals(prizeName)) {
                prize.redeemPrize(card);
                return;
            }
        }
        System.out.println("Prize " + prizeName + " not found.");
    }
}
```

Main Class

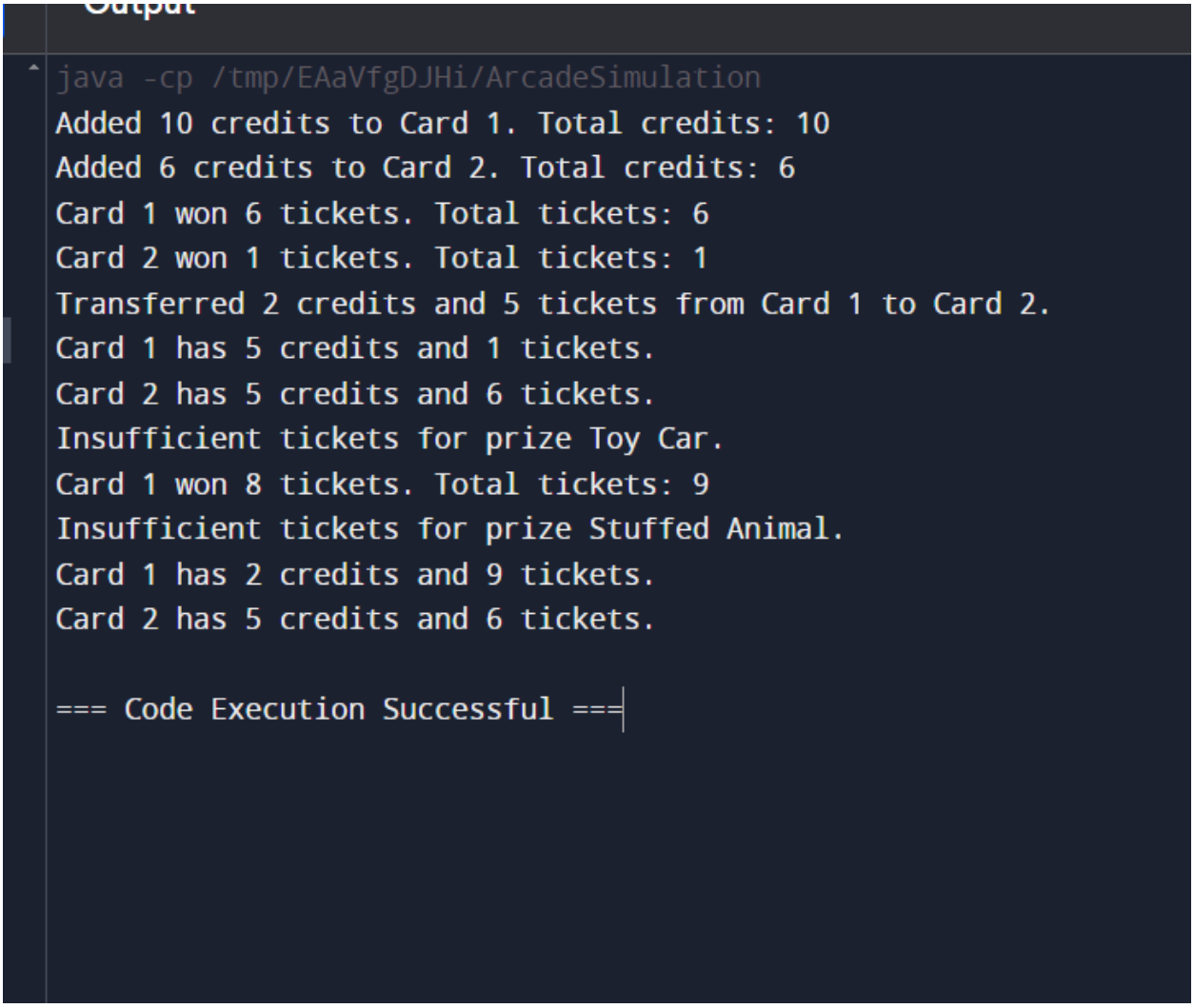
The main method will instantiate objects and simulate interactions in the arcade.

```
public class ArcadeSimulation {
    public static void main(String[] args) {

        Card card1 = new Card(1);
        Card card2 = new Card(2);
        PrizeCategory[] prizes = {
            new PrizeCategory("Toy Car", 10, 5),
            new PrizeCategory("Stuffed Animal", 20, 2),
            new PrizeCategory("Action Figure", 30, 1)
        };
        Terminal terminal = new Terminal(prizes);
        terminal.addCredits(card1, 5);
        terminal.addCredits(card2, 3);
        Game game = new Game(3);
```

```
        game.play(card1);
        game.play(card2);
        terminal.transfer(card1, card2, 2, 5);
        terminal.redeemPrize(card2, "Toy Car");
        game.play(card1);
        terminal.redeemPrize(card1, "Stuffed Animal");
        terminal.checkBalances(card1);
        terminal.checkBalances(card2);
    }
}
```

Output:


Output
java -cp /tmp/EAaVfgDJHi/ArcadeSimulation
Added 10 credits to Card 1. Total credits: 10
Added 6 credits to Card 2. Total credits: 6
Card 1 won 6 tickets. Total tickets: 6
Card 2 won 1 tickets. Total tickets: 1
Transferred 2 credits and 5 tickets from Card 1 to Card 2.
Card 1 has 5 credits and 1 tickets.
Card 2 has 5 credits and 6 tickets.
Insufficient tickets for prize Toy Car.
Card 1 won 8 tickets. Total tickets: 9
Insufficient tickets for prize Stuffed Animal.
Card 1 has 2 credits and 9 tickets.
Card 2 has 5 credits and 6 tickets.
=== Code Execution Successful ===