Java Foundations Practices - Section 7

Date Night at the Arcade

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**.**

CODE:

```
import java.util.Random;

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) {
        this.creditBalance += credits;
    }
    public void subtractCredits(int credits) {
        if (credits <= this.creditBalance) {
            this.creditBalance -= credits;
        } else {
            System.out.println("Insufficient credits.");
        }
    }
    public void addTickets(int tickets) {
        this.ticketBalance += tickets;
    }
    public void subtractTickets(int tickets) {
        if (tickets <= this.ticketBalance) {
            this.ticketBalance -= tickets;
        } else {
            System.out.println("Insufficient tickets.");
        }
    }
}
class Game {
    private int costToPlay;
    private Random random;
    public Game(int costToPlay) {
        this.costToPlay = costToPlay;
        this.random = new Random();
    }
    public void play(Card card) {
        if (card.getCreditBalance() >= costToPlay) {
            card.subtractCredits(costToPlay);
            int ticketsWon = random.nextInt(50); // Random tickets between 0 and 49
```

```
            card.addTickets(ticketsWon);
            System.out.println("Card " + card.getCardNumber() + " won " + ticketsWon + " tickets. Total tickets: " + card.getTicketBalance());
        } else {
            System.out.println("Card " + card.getCardNumber() + " has insufficient credits to play.");
        }
    }
}
class PrizeCategory {
    private String name;
    private int ticketsRequired;
    private int remainingCount;
    public PrizeCategory(String name, int ticketsRequired, int remainingCount) {
        this.name = name;
        this.ticketsRequired = ticketsRequired;
        this.remainingCount = remainingCount;
    }
    public String getName() {
        return name;
    }
    public int getTicketsRequired() {
        return ticketsRequired;
    }
    public int getRemainingCount() {
        return remainingCount;
    }
    public void decrementCount() {
        if (remainingCount > 0) {
            remainingCount--;
        } else {
            System.out.println("No more prizes in this category.");
        }
    }
```

```
}
class Terminal {
    private PrizeCategory[] prizes;
    public Terminal(PrizeCategory[] prizes) {
        this.prizes = prizes;
    }
    public void loadCredits(Card card, int money) {
        int credits = money * 2;
        card.addCredits(credits);
        System.out.println("Card " + card.getCardNumber() + " loaded with " + credits + " credits. 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 transferCredits(Card fromCard, Card 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("Insufficient credits to transfer.");
        }
    }
    public void transferTickets(Card fromCard, Card toCard, int tickets) {
        if (fromCard.getTicketBalance() >= tickets) {
            fromCard.subtractTickets(tickets);
            toCard.addTickets(tickets);
            System.out.println("Transferred " + tickets + " tickets from Card " + fromCard.getCardNumber() + " to Card " + toCard.getCardNumber());
        } else {
```

```
            System.out.println("Insufficient tickets to transfer.");
        }
    }
    public void requestPrize(Card card, int prizeIndex) {
        PrizeCategory prize = prizes[prizeIndex];
        if (card.getTicketBalance() >= prize.getTicketsRequired() && prize.getRemainingCount() > 0) {
            card.subtractTickets(prize.getTicketsRequired());
            prize.decrementCount();
            System.out.println("Card " + card.getCardNumber() + " awarded " + prize.getName() + ".
Remaining " + prize.getName() + ": " + prize.getRemainingCount());
        } else if (card.getTicketBalance() < prize.getTicketsRequired()) {
            System.out.println("Insufficient tickets to request this prize.");
        } else {
            System.out.println("This prize is out of stock.");
        }
    }
}
public class ArcadeTest {
    public static void main(String[] args) {
        Card card1 = new Card(1);
        Card card2 = new Card(2);
        PrizeCategory[] prizes = {
            new PrizeCategory("Stuffed Animal", 100, 5),
            new PrizeCategory("Candy Bar", 50, 10),
            new PrizeCategory("Keychain", 20, 15)
        };
        Terminal terminal = new Terminal(prizes);
        terminal.loadCredits(card1, 10);
        terminal.loadCredits(card2, 5);

        Game game = new Game(5);
        game.play(card1);
        game.play(card1);
```

```
        game.play(card2);
        game.play(card2);
        terminal.transferCredits(card1, card2, card1.getCreditBalance());
        terminal.transferTickets(card1, card2, card1.getTicketBalance());

        terminal.requestPrize(card2, 1);
        game.play(card1);
        terminal.requestPrize(card1, 0);
        terminal.checkBalances(card1);
        terminal.checkBalances(card2);
    }
}
```

OUTPUT:

```
java -cp /tmp/TkE37OP1Q8/ArcadeTest
Card 1 loaded with 20 credits. Total credits: 20
Card 2 loaded with 10 credits. Total credits: 10
Card 1 won 24 tickets. Total tickets: 24
Card 1 won 49 tickets. Total tickets: 73
Card 2 won 43 tickets. Total tickets: 43
Card 2 won 16 tickets. Total tickets: 59
Transferred 10 credits from Card 1 to Card 2
Transferred 73 tickets from Card 1 to Card 2
Card 2 awarded Candy Bar. Remaining Candy Bar: 9
Card 1 has insufficient credits to play.
Insufficient tickets to request this prize.
Card 1 has 0 credits and 0 tickets.
Card 2 has 10 credits and 82 tickets.

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