2023

CP6200 JavaProgramming2 OER - OOP Course Project

Shoshana Marcus
CUNY Kingsborough Community College

Kingsborough Computer College, CUNY
Department of Mathematics and Computer Science
Dr. Shoshana Marcus

CP 6200: Java Programming II
Introduction to Object Oriented Programming

Programming Assignment

Objective: In this assignment you will get comfortable creating your own set of classes in Java that use inheritance and aggregation. In the first part, you will implement a credit card class. In the second phase, you will use inheritance to create several different kinds of classes for charge cards (among them are credit cards and debit cards). In the third phase, you will create a wallet class to manage a wallet, containing different kinds of charge cards as well as cash, to make purchases accordingly.

Phase I: Credit Card Class

Write a class named CreditCard that has (at least) the following member variables:

- **name.** A String that holds the card holder's name.
- **cardNumber.** A field that holds the credit card number.
- **balance.** A double that stores the current credit card balance.
- **spendingLimit.** A `double` that stores the spending limit of the card holder.
- *additional fields that you can think of.*

In addition, the class should have (at least) the following member methods:

- **Constructor.** The constructor should accept the card holder's name and card number and assign these values to the object's corresponding member variables. The constructor should initialize the spending limit to $3,000 and the balance to $0.
- **Accessors.** Appropriate accessor methods should be created to allow values to be retrieved from an object's member variables.
- **purchase.** This method should add the amount specified as a parameter to the balance member variable each time it is called, assuming the purchase does not cause the balance to exceed the customer's spending limit.
- **increaseSpendingLimit.** This method should increase the spending limit by $100 each time it is called.
- **payBill.** This method should reset the balance to 0.
- **toString.** to render the state of a CreditCard as a String.

Input validation: Whenever a credit card number is modified, verify that it is of reasonable length.

Demonstrate the class in a program that creates a CreditCard object and allows the user to change and view the state of the credit card with a menu driven program.

1. View Card Information.
2. Purchase an Item: ask the user the purchase amount and increase the card balance accordingly.
3. Pay Bill: call payBill method to set the balance to 0.
4. Increase Spending Limit: ask the user how much the spending limit should be, and call the increaseSpendingLimit method the appropriate number of times.

Keep in mind: input and output should not be performed directly in the CreditCard class. All input and output to the user should be done in the main method (or other methods that are in a separate class) to increase flexibility of the classes we will design later on in this project.

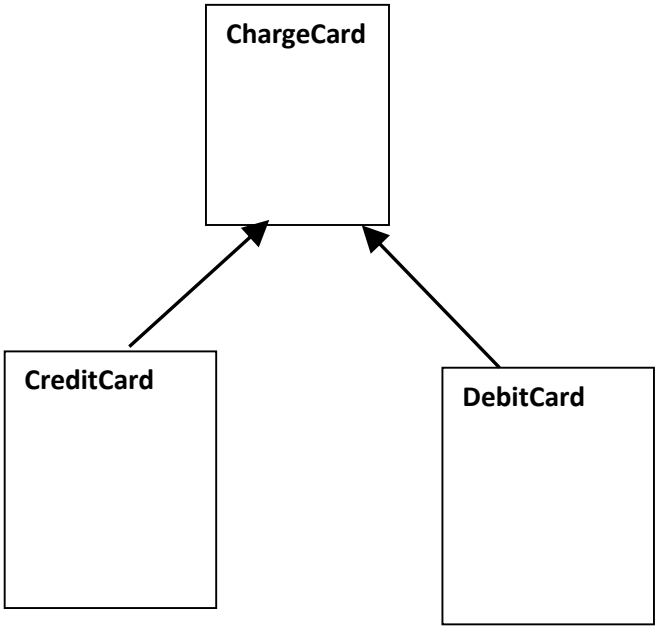
Phase II: Credit and Debit cards with Inheritance

We would like to design a class that stores information about a debit card. Since debit cards and credit cards have a lot of similarity, the commonality can be extracted to a superclass called ChargeCard. Then DebitCard and CreditCard are both subclasses of ChargeCard.

A fundamental difference between credit cards and debit cards is in how they make purchases, but realize that they both have the capacity to make purchases. A purchase *increases* the balance of a credit card while it *decreases* the balance of a debit card.

Some distinctions between a credit card and a debit card lie in how the user makes payments towards them. The bill can be paid on a CreditCard to decrease its balance while a deposit or withdrawal can affect the balance of a DebitCard. A credit card has a spending limit to limit the balance on the card. Purchases can be made on a debit card as long as there are funds in the account. The balance on a debit card can become negative, so long as the amount does not exceed the overdraft limit, in which case an overdraft fee is applied.

In both a credit card and debit card, it should be possible to the maximum that can be spent.



Do you think ChargeCard should be an abstract or a concrete class? Write a comment before the class begins in which you explain your design decision.

Make a parameterized constructor in the superclass that can be called from all of its subclasses.

Keep in mind: input and output should not be performed in the CreditCard or DebitCard or ChargeCard classes. All input and output to the user should be performed in the main method method (or other methods that are in a separate class) to increase flexibility of the classes we will design later on.

Debit Card Class

The fields in the DebitCard class should include the following. Realize that the data members that are shared with the CreditCard class should be implemented in the ChargeCard class to eliminate redundancy.

- **name.** A String that holds the card holder's name.
- **cardNumber.** A field that holds the credit card number.
- **balance.** A `double` that stores the current credit card balance.
- **overdraftLimit.** A `double` that stores the overdraft limit of the account.
- **overdraftFee**. A `double` that stores the fee to apply whenever expenses exceed the balance.
- **feesIncurred**. A `double` that keeps track of fees that the account holder owes.
- *additional fields that you can think of.*

The DebitCard class should support (at least) the following member methods:

- **Constructor.** The constructor should accept the card holder's name and card number and assign these values to the object's corresponding member variables. The constructor should initialize the overdraft fee to $50, the overdraft limit to $100, and the balance to $0.
- **Accessors.** Appropriate accessor methods should be created to allow values to be retrieved from an object's member variables.
- **purchase.** This method should subtract the amount specified as a parameter from the balance member variable each time it is called, and tap into the overdraft limit when necessary (and apply the overdraft fee whenever this happens).
- **increaseOverdraftLimit.** This method should add 100 to the overdraftLimit member variable each time it is called.
- **depositFunds.** This method should increase the balance as specified in a parameter.
- **withdrawFunds.** This method should decrease the balance as specified in a parameter.
- **payFees.** This method should reset the feesIncurred member variable to 0.
- **toString.** Render the state of a DebitCard as a String.

Demonstrate the class in a program that creates a DebitCard object and allows the user to change and view the state of the debit card with a menu driven program.

1. View Card Information.
2. Purchase an Item: ask the user the purchase amount and decrease the card balance accordingly.
3. Make Deposit: ask the user how much money to deposit and call makeDeposit method to increase the balance accordingly.
4. Increase Overdraft Limit: ask the user how much the spending limit should be, and call the increaseSpendingLimit method the appropriate number of times.
5. Pay Fees: call the payFees member method to reset the fees incurred to 0.

Make sure that the menu-driven program you have already written for the CreditCard class still works as you expect it to!

Phase III - Wallet class

Write a class called **Wallet** to store information about the current state of a wallet. A wallet has different compartments for the dollar bills, loose change, credit / debit cards and ID cards.

Data Members should include:

- Amount of cash - keep track of value of dollar bills and value of change separately.
- Array of ChargeCards – it can contain both CreditCards and DebitCards.
- Array of ID cards – this can be simply an array of Strings or it can use a specialized class with additional fields. *It's up to you how to store this information.*

Methods should include:

- at least two constructors
- accessor, mutator methods
- TotalCashOnHand - total value of dollar bills and change
- TotalCanSpend - includes both cash and amount that can be spent on credit / debit cards.
- a method that tells you how many ID Cards are in the wallet
- a method that tells you how many credit / debit cards are in the wallet
- a method that lets you add a credit / debit card to the set
- a method to append an ID card to the set
- toString method - should delegate to CreditCard class to print information about each credit card and DebitCard class to print information about each debit card.

Remember to keep the design paradigms of data hiding and encapsulation in mind as much as possible. Delegate tasks to the component classes (of ChargeCard, DebitCard, CreditCard) whenever you can.

Perform input validation as appropriate.

Driver Program

Write a menu-driven program that allows the user to interact with a single Wallet object.

1. Print all information about Wallet. [This should call the toString method of the Wallet class.]
2. Add a Credit Card. [Prompt the user for information about the Credit Card]
3. Add a Debit Card. [Prompt the user for information about the Debit Card.]
4. Add an ID Card. [Prompt the user for information about the ID card.]
5. Add Cash. [Prompt the user for the value of additional coins or dollar bills.]
6. Make a purchase. [Should ask for purchase amount. Then let the user choose whether to pay with cash or charge card. If cash, make sure there is sufficient cash in the wallet to cover the purchase, and then deduct accordingly. If charge card, there should be a submenu in which the user chooses a card from the list to use. Then the system should check that the card can be used for that purchase, i.e., that there are sufficient funds available on that card, and actually process the purchase on the selected ChargeCard.]

Bonus: Gift Card class

Design a GiftCard class that also extends ChargeCard and integrate it in the Wallet class. Gift cards come preloaded with a specific amount of money and the user can never spend more than that predetermined amount.