COSC 24L3: Lab Assignment #8
Computer Science Department @ Dallas Baptist University
Fall 2023

Objectives

- Be able to write a copy constructor
- Be able to write `equals` and `toString` methods
- Be able to use objects made up of other objects (aggregation)
- Be able to write methods that pass and return objects

Introduction

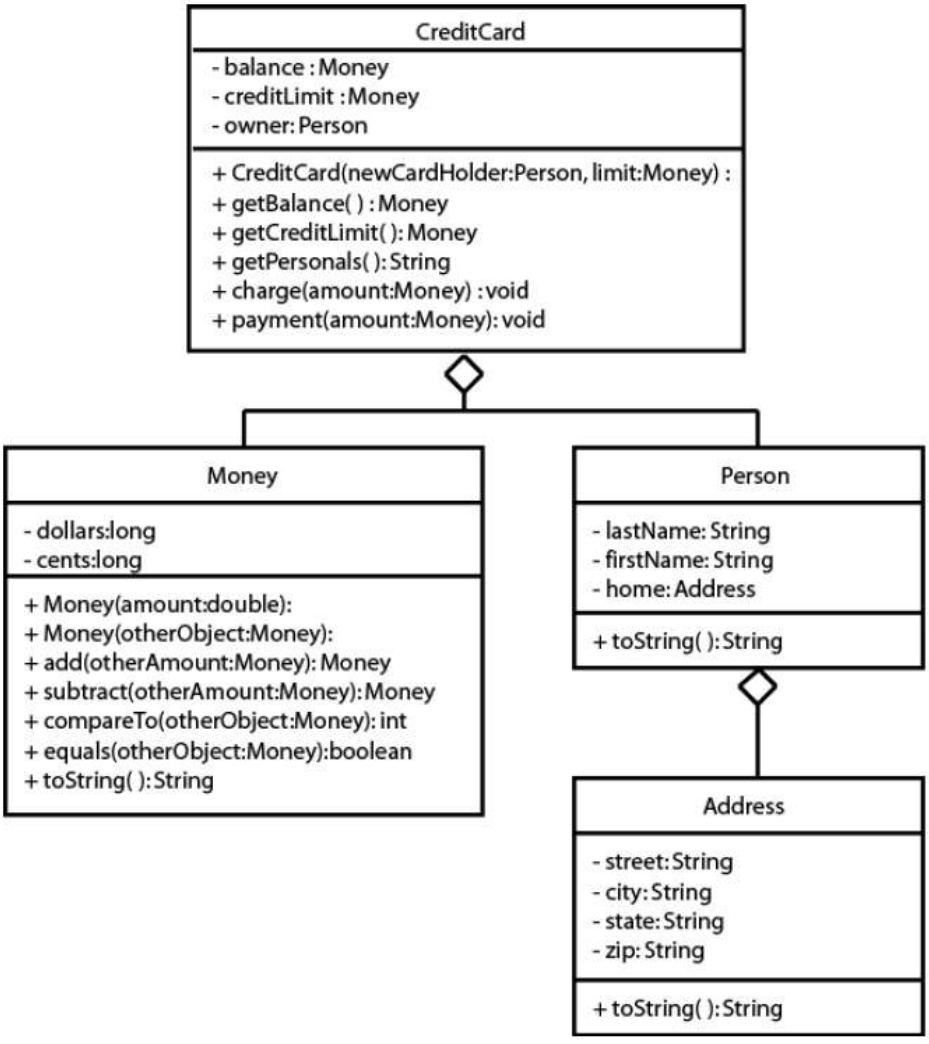
A credit card includes information about the owner, as well as a balance and credit limit. These things would be the instance fields. A credit card allows you to make payments and charges. These would be methods. As we have seen before, there would also be other methods associated with this object in order to construct the object and access its fields.

Examine the UML diagram that follows. Notice that the instance fields in the **CreditCard** class are other types of objects: a **Person** object and a **Money** object. We can say that the CreditCard object *"has a"* Person object, which means aggregation, and the Person object *"has a"* Address object as one of its instance fields. This aggregation structure can create a very complicated object. We will try to keep this lab reasonably simple.

To start with, we will be editing a partially written class, `Money`. The constructor that you will be writing is a copy constructor. This means it should create a new object, but with the same values in the instance variables as the object that is being copied.

Next, we will write the **equals** and **toString** methods. These are very common methods that are needed when you write a class to model an object. You will also see a **compareTo** method that is also a common method for objects.

After we have finished the Money class, we will write a CreditCard class. This class contains Money objects, so you will use the methods that you have written to complete the Money class. The CreditCard class will explore passing objects and the possible security problems associated with it. We will use the copy constructor we wrote for the Money class to create new objects with the same information to return to the user through the accessor methods.


CreditCard
- balance : Money
- creditLimit : Money
- owner: Person
+ CreditCard(newCardHolder:Person, limit:Money) :
+ getBalance( ) : Money
+ getCreditLimit( ): Money
+ getPersonals( ): String
+ charge(amount:Money) : void
+ payment(amount:Money): void
Money
- dollars:long
- cents:long
+ Money(amount:double):
+ Money(otherObject:Money):
+ add(otherAmount:Money): Money
+ subtract(otherAmount:Money): Money
+ compareTo(otherObject:Money): int
+ equals(otherObject:Money):boolean
+ toString( ): String
Person
- lastName: String
- firstName: String
- home: Address
+ toString( ): String
Address
- street: String
- city: String
- state: String
- zip: String
+ toString( ): String

Task #1 Writing a Copy Constructor

1. Copy the files *Address.java, Person.java, Money.java*, *MoneyDemo.java,* and *CreditCardDemo.java.* `Address.java`, `Person.java`, `MoneyDemo.java`, and `CreditCardDemo.java` are complete and will not need to be modified. We will start by modifying `Money.java`.
2. Overload the constructor. The constructor that you will write will be a copy constructor. It should use the parameter `Money` object to make a duplicate `Money` object, by copying the value of each instance variable from the parameter object to the instance variable of the new object.

Task #2 Writing the `equals` and `toString` methods

1. Write an `equals` method. The method compares the instance variables of the calling object with instance variables of the parameter object for equality and returns `true` if the `dollars` and the `cents` of the calling object are the same as the `dollars` and the `cents` of the parameter object. Otherwise, it returns `false`.
2. Write a `toString` method. This method will return a `String` that looks like currency, including the dollar sign. Remember that if you have less than 10 cents, you will need to put a 0 before printing the `cents` so that it appears correctly with 2 decimal places.
3. Compile, debug, and test by running the `MoneyDemo` program. You should get the following output:

```
The current amount is $500.00
Adding $10.02 gives $510.02
Subtracting $10.88 gives $499.14
$10.02 equals $10.02
$10.88 does not equal $10.02
```

Task #3 Passing and Returning Objects

1. Create the `CreditCard` class according to the UML diagram. It should have data fields that include an `owner` of type `Person`, a `balance` of type `Money`, and a `creditLimit` of type `Money`.
2. It should have a constructor that has two parameters, a reference to a `Person` object to initialize the `owner` and a reference to a `Money` object to initialize the `creditLimit`. The `balance` can be initialized to a `Money` object with a value of zero. Remember you are passing in objects (passed by reference), so you are passing the memory address of an object. If you want your `CreditCard` to have its own `creditLimit` and `balance`, you should create a new object of each using the copy constructor in the `Money` class.

3. It should have accessor methods to get the `balance` and the `creditLimit`. Since these are `Money` objects (passed by reference), we don't want to create a security issue by passing out addresses to components in our `CreditCard` class, so we must return a new object with the same values. Again, use the copy constructor to create a new object of type `Money` that can be returned.
4. It should have an accessor method to get the information about the `owner`, but in the form of a `String` that can be printed out. This can be done by calling the `toString` method for the `owner` (an instance of the `Person` class).
5. It should have a method that will charge to the `CreditCard` by adding the `amount` passed in the parameter to the `balance`, but only if it will not exceed the `creditLimit`. If the `creditLimit` will be exceeded, the `amount` should not be added, and an error message can be printed to the console.
6. It should have a method that will make a payment on the `CreditCard` by subtracting the `amount` passed in the parameter from the `balance`.
7. Compile, debug, and test it out completely by running the `CreditCardDemo` program.
8. You should get the output:

```
Diane Christie, 237J Harvey Hall, Menomonie, WI 54751
Balance: $0.00
Credit Limit: $1000.00
Attempting to charge $200.00
Charge: $200.00
Balance: $200.00
Attempting to charge $10.02
Charge: $10.02
Balance: $210.02
Attempting to pay $25.00
Payment: $25.00
Balance: $185.02
Attempting to charge $990.00
Exceeds credit limit
Balance: $185.02
```