**Name:**

**CST 183 Lab Assignment #8 (Chapter 8)**

**More Classes and Objects**

**Lab 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**

We discussed objects in Chapter 6 and we modeled a television in the Chapter 6 lab. We want build on that lab, and work more with objects. This time, the object that we are choosing is more complicated. It is made up of other objects. This is called aggregation. A credit card is an object that is very common, but not as simple as a television. Attributes of the credit card include information about the owner, as well as a balance and credit limit. These things would be our 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 or a Money object. We can say that the CreditCard “has a” Person, 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 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.



**Task #1 Writing a Copy Constructor**

1. Copy the files Address.java (code listing 8.1), Person.java (code listing 8.2), Money.java (code listing 8.3), MoneyDriver.java (code listing 8.4), and CreditCardDemo.java (code listing 8.5) from the Student CD or as directed by your instructor. 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.

Here is the code for the overloaded constructor:

//Copy constructor, uses one Money object to make
//a duplicate Money object
public Money(Money otherObject)
{
 this.dollars = otherObject.dollars;
 this.cents = otherObject.cents;
}

**Task #2 Writing equals and toString methods**

1. Write and document 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 and document a toString method. This method will return a String that looks like money, 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 MoneyDriver.java driver program. You should get the 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

Here is a skeleton of the equals and toString methods:

//Compares instance variables of the calling object with the
//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.
public boolean equals(Money amount)
{
 // compare dollars and cents from money objects

return ????; // return the result of the comparison

}

//Describes the Money object
public String toString()
{
 // code to output money object

 return valueString;
}

**Task #3 Passing and Returning Objects**

1. Create a CreditCard class according to the UML Diagram on the back. 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 Person to initialize the owner and a Money value to initialize the creditLimit. The balance can be initialized to a Money value of zero. Remember you are passing in objects (pass by reference), so you have passed in the address to 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 available credit. Since these are objects (pass by reference), we don’t want to create an insecure credit card by passing out addresses to components in our credit card, 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.

1. 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 (who is a Person).
2. It should have a method that will charge to the credit card by adding the amount of Money in the parameter to the balance if it will not exceed the credit limit. If the credit limit will be exceeded, the amount should not be added, and an error message can be printed to the console.
3. It should have a method that will make a payment on the credit card by subtracting the amount of Money in the parameter from the balance.
4. Compile, debug, and test it out completely by running CreditCardDemo.java. You should get the output:

Diane Christie, 237J Harvey Hall, Menomonie, WI 54751 Balance: $0.00

Credit Limit: $1000.00 Attempt to charge $200.00 Charge: $200.00

Balance: $200.00 Attempt to charge $10.02 Charge: $10.02

Balance: $210.02 Attempt to pay $25.00 Payment: $25.00 Balance: $185.02

Attempt to charge $990.00 Exceeds credit limit Balance: $185.02

Here is a skeleton for the CreditCard.java

// To represent the a credit card

public class CreditCard
{
 private Money balance; //the current balance
 private Money creditLimit; //the approved credit limit
 private Person owner; //the owner of the credit card

 //Constructor creates a credit card for the Person parameter
 //with an approved credit limit of the Money parameter
 //and sets the balance to a Money amount of zero
 public CreditCard(Person newCardholder, Money limit)
 {

 }

 //Accessor method returns the balance
 public Money getBalance()
 {

 }

 //Accessor method returns the credit limit
 public Money getCreditLimit()
 {

 }

 //Accessor method returns information about the owner
 public String getPersonals()
 {

 }

 //Method to make a charge to the credit card, if
 //the credit limit will not be exceded
 public void charge(Money amount)
 {
 Money temp = new Money(balance.add(amount));
 if(temp.compareTo(creditLimit) > 0)
 {
 System.out.println("Exceeds credit limit");
 }
 else
 {
 balance = temp;
 System.out.println("Charge: " + amount);
 }
 }

 //Method to make a payment to the credit card
 public void payment(Money amount)
 {

 }
}

**Paste in the source code AND an output screenshot of your modified java program(s) here:**

Upload the completed assignment to the Lab 8 dropbox in D2L for grading.