Chapter 8 Lab
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.

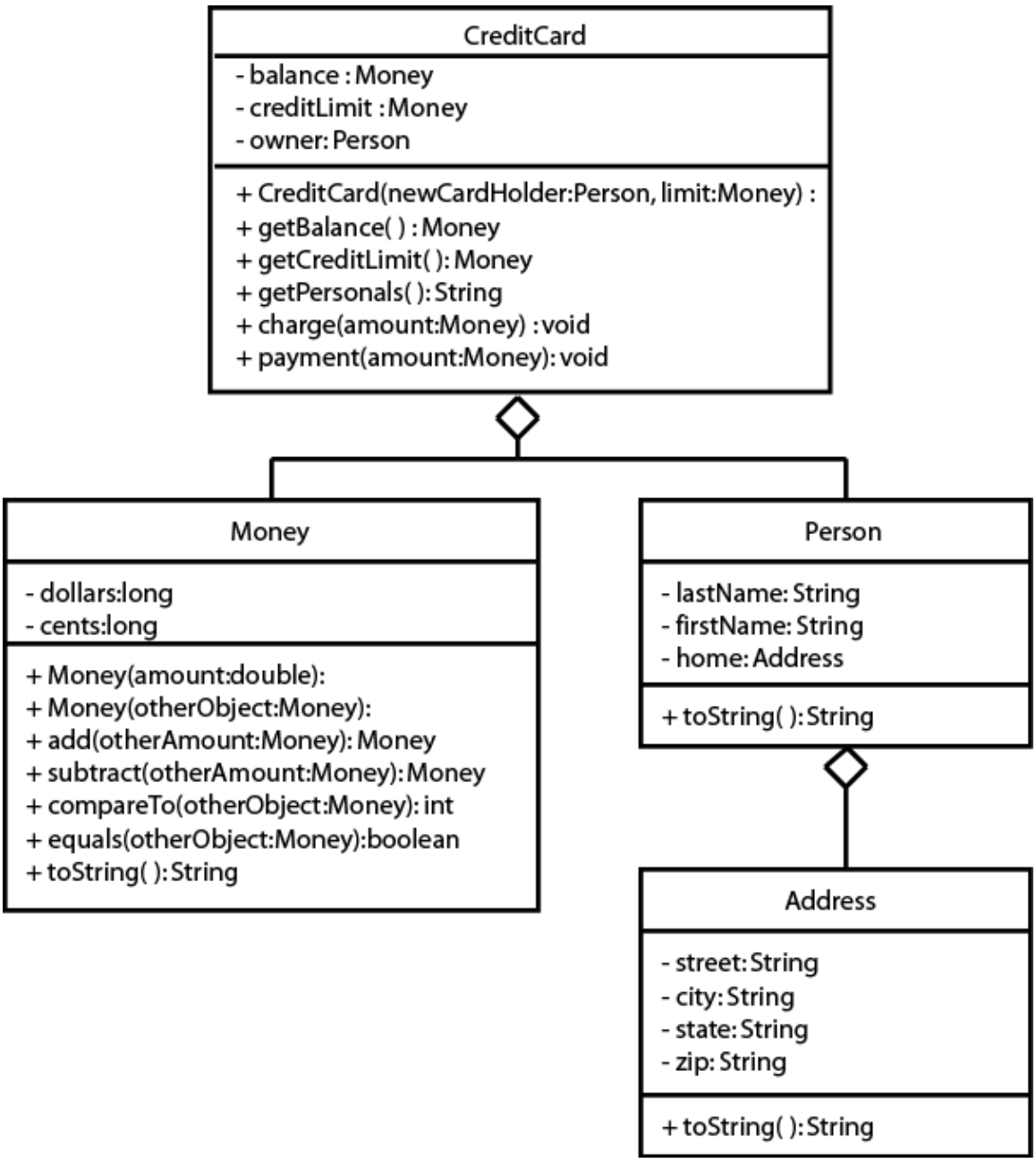
Attributes of the credit card include information about the owner, as well as a balance and credit limit. These things would be our instance fields. A credit card allows you to make payments and charges. These would be methods. As we have seen before, there would also be other methods associated with 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 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.



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), *MoneyDemo.java* (Code Listing 8.4), and *CreditCardDemo.java* (Code Listing 8.5) from the Student Files 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.

Task #2 Writing the `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 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
```

Code Listing 8.1 (`Address.java`)

```
/**
   This class defines an address using a street,
   city, state, and zipcode.
*/

public class Address
{
   // The street number and name
   private String street;

   // The city in which the address is located
   private String city;

   // The state in which the address is located
   private String state;

   // The zip code associated with the city and street
   private String zip;

   /**
      Constructor
      @param road Describes the street number and name.
      @param town Describes the city.
      @param st Describes the state.
      @param zipCode Describes the zip code.
   */

   public Address(String road, String town, String st,
                  String zipCode)
   {
      street = road;
      city = town;
      state = st;
      zip = zipCode;
   }
```

```
   /**
      The toString method
      @return Information about the address.
   */

   public String toString()
   {
      return (street + ", " + city +
              ", " + state + " " + zip);
   }
}
```

Code Listing 8.2 (`Person.java`)

```
/**
   This class defines a person by name and address.
*/

public class Person
{
   // The person's last name
   private String lastName;

   // The person's first name
   private String firstName;

   // The person's address
   private Address home;

   /**
      Constructor
      @param last The person's last name.
      @param first The person's first name.
      @param residence The person's address.
   */

   public Person(String last, String first,
                 Address residence)
   {
      lastName = last;
      firstName = first;
      home = residence;
   }
```

```
   /**
      The toString method
      @return Information about the person.
   */

   public String toString()
   {
      return(firstName + " " + lastName +
             ", " + home.toString());
   }
}
```

Code Listing 8.3 (`Money.java`)

```
/**
   This class represents nonnegative amounts of money.
*/

public class Money
{
   // The number of dollars
   private long dollars;

   // The number of cents
   private long cents;

   /**
      Constructor
      @param amount The amount in decimal format.
   */

   public Money(double amount)
   {
      if (amount < 0)
      {
         System.out.println("Error: Negative amounts " +
                            "of money are not allowed.");
         System.exit(0);
      }
      else
      {
         long allCents = Math.round(amount * 100);
         dollars = allCents / 100;
         cents = allCents % 100;
      }
   }
```

```
// ADD LINES FOR TASK #1 HERE
// Document and write a copy constructor

/**
   The add method
   @param otherAmount The amount of money to add.
   @return The sum of the calling Money object
           and the parameter Money object.
*/

public Money add(Money otherAmount)
{
   Money sum = new Money(0);

   sum.cents = this.cents + otherAmount.cents;

   long carryDollars = sum.cents / 100;

   sum.cents = sum.cents % 100;

   sum.dollars = this.dollars +
                 otherAmount.dollars +
                 carryDollars;

   return sum;
}

/**
   The subtract method
   @param amount The amount of money to subtract.
   @return The difference between the calling Money
           object and the parameter Money object.
*/

public Money subtract (Money amount)
{
   Money difference = new Money(0);

   if (this.cents < amount.cents)
   {
      this.dollars = this.dollars - 1;
      this.cents = this.cents + 100;
   }
```

```
      difference.dollars = this.dollars - amount.dollars;
      difference.cents = this.cents - amount.cents;

      return difference;
   }

   /**
      The compareTo method
      @param amount The amount of money to compare against.
      @return -1 if the dollars and the cents of the
              calling object are less than the dollars and
              the cents of the parameter object.
              0 if the dollars and the cents of the calling
              object are equal to the dollars and cents of
              the parameter object.
              1 if the dollars and the cents of the calling
              object are more than the dollars and the
              cents of the parameter object.
   */

   public int compareTo(Money amount)
   {
      int value;

      if(this.dollars < amount.dollars)
         value = -1;
      else if (this.dollars > amount.dollars)
         value = 1;
      else if (this.cents < amount.cents)
         value = -1;
      else if (this.cents > amount.cents)
         value = 1;
      else
         value = 0;

      return value;
   }

   // ADD LINES FOR TASK #2 HERE
   // Document and write an equals method
   // Document and write a toString method
}
```

Code Listing 8.4 (`MoneyDemo.java`)

```
/**
   This program demonstrates the Money class.
*/

public class MoneyDemo
{
   public static void main(String[] args)
   {
      // Named constants
      final int BEGINNING = 500; // Beginning balance
      final Money FIRST_AMOUNT  = new Money(10.02);
      final Money SECOND_AMOUNT = new Money(10.02);
      final Money THIRD_AMOUNT  = new Money(10.88);

      // Create an instance of the Money class with
      // the beginning balance.
      Money balance = new Money(BEGINNING);

      // Display the current balance.
      System.out.println("The current amount is " +
                         balance.toString());

      // Add the second amount to the balance
      // and display the results.
      balance = balance.add(SECOND_AMOUNT);
      System.out.println("Adding " + SECOND_AMOUNT +
                         " gives " + balance.toString());

      // Subtract the third amount from the balance
      // and display the results.
      balance = balance.subtract(THIRD_AMOUNT);
      System.out.println("Subtracting " + THIRD_AMOUNT +
                         " gives " + balance.toString());

      // Determine if the second amount equals
      // the first amount and store the result.
      boolean equal = SECOND_AMOUNT.equals(FIRST_AMOUNT);
```

```
      // Display the result.
      if(equal)
      {
         // The first and second amounts are equal.
         System.out.println(SECOND_AMOUNT + " equals " +
                            FIRST_AMOUNT);
      }
      else
      {
         // The first and second amounts are not equal.
         System.out.println(SECOND_AMOUNT +
                            " does not equal " +
                            FIRST_AMOUNT);
      }

      // Determine if the third amount equals
      // the first amount and store the result.
      equal = THIRD_AMOUNT.equals(FIRST_AMOUNT);

      // Display the result.
      if(equal)
      {
         // The third and first amounts are equal.
         System.out.println(THIRD_AMOUNT + " equals " +
                            FIRST_AMOUNT);
      }
      else
      {
         // The third and first amounts are not equal.
         System.out.println(THIRD_AMOUNT +
                            " does not equal " +
                            FIRST_AMOUNT);
      }
   }
}
```

Code Listing 8.5 (`CreditCardDemo.java`)

```
/**
   This program demonstrates the CreditCard class.
*/
```

```
public class CreditCardDemo
{
   public static void main(String[] args)
   {
      // Named constants
      final Money CREDIT_LIMIT  = new Money(1000);
      final Money FIRST_AMOUNT  = new Money(200);
      final Money SECOND_AMOUNT = new Money(10.02);
      final Money THIRD_AMOUNT  = new Money(25);
      final Money FOURTH_AMOUNT = new Money(990);

      // Create an instance of the Person class.
      Person owner = new Person("Christie", "Diane",
                     new Address("237J Harvey Hall",
                     "Menomonie", "WI", "54751"));

      // Create an instance of the CreditCard class.
      CreditCard visa = new CreditCard(owner,
                        CREDIT_LIMIT);

      // Display the credit card information.
      System.out.println(visa.getPersonals());
      System.out.println("Balance: " + visa.getBalance());
      System.out.println("Credit Limit: " +
                         visa.getCreditLimit());

      System.out.println();   // To print a new line

      // Attempt to charge the first amount and
      // display the results.
      System.out.println("Attempting to charge " +
                         FIRST_AMOUNT);
      visa.charge(FIRST_AMOUNT);
      System.out.println("Balance: " + visa.getBalance());

      System.out.println();   // To print a new line

      // Attempt to charge the second amount and
      // display the results.
      System.out.println("Attempting to charge " +
                         SECOND_AMOUNT);
      visa.charge(SECOND_AMOUNT);
      System.out.println("Balance: " + visa.getBalance());

      System.out.println();   // To print a new line
```

```
        // Attempt to pay using the third amount and
        // display the results.
        System.out.println("Attempting to pay " +
                           THIRD_AMOUNT);
        visa.payment(THIRD_AMOUNT);
        System.out.println("Balance: " + visa.getBalance());

        System.out.println();   // To print a new line

        // Attempt to charge using the fourth amount and
        // display the results.
        System.out.println("Attempting to charge " +
                           FOURTH_AMOUNT);
        visa.charge(FOURTH_AMOUNT);
        System.out.println("Balance: " + visa.getBalance());
    }
}
```