成绩	

重庆邮电大学

实验报告

2020-2021 学年第 2 学期

计算机科学导论

（第 6 次试验）

班级：34082003

姓名：黄凯升

学号：2020215138

指导老师：许汀汀

课程名称：计算机科学导论

实验时间：2021 年 5 月 13 日

实验地点：综合实验大楼 A511/A512

1 实验名称

More Classes and Objects

2 实验目的

- 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

3 实验内容

Task#1 Writing a Copy Constructor

Make an overloaded constructor to make duplication of objects for class Money.

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

Write and document an `equals` method and a `toString` method. The former one compares the objects to determine if they equal to each other. The latter one generates a string that has "$" sign and the amount of money in 2 decimal places.

Task #3 Passing and Returning Objects

Create a CreditCard class according to the UML diagram. Make balance initialized to zero. While constructing, the credit limit should be duplicated inside the class. Make some accessor methods, which should return duplications of data fields. Write charge method and payment method. Once succeeded, it should print a log to standard output. And if the amount to be charged is greater than credit limit, print an error message and don't increase the balance.

4 实验方法(原理、流程图)

The development environment is:

- OS: Ubuntu 20.04.2 LTS on Windows 10 (WSL1, Kernel build 19041)
- IDE/Editor: Visual Studio Code
- Java Runtime: OpenJDK 14.0.2 (build 14.0.2+12-Ubuntu-120.04)

For Task #1, we should write a copy constructor for class Money by overloading the constructor. The overloading of method is simple, just to write another method with the same name but different parameter list. The copy constructor just copies all data fields from source object to this object.

For Task #2, we should create a Boolean returning method `equals` with a Money type parameter. Notice that there is a method implemented called `compareTo`. It returns 0 when another Money object is equals to this. So, we can use it simply, just to determine if it returns 0. The method `toString` is also easy to implement. We should generate a string starts with a "$" symbol, dollars and cents in 2 decimal places separated by dot. Without doubt, the method `String.format` can perfectly meet our requirements. Just use "`$%d.%02d`" as the format string, and pass the dollars and cents to the method.

For Task #3, we should create a class called CreditCard, which is really easy to write data fields and methods' declarations according to the UML diagram. In the constructor, we can first initialize the balance to zero. And for safety, some fields like `creditLimit` should be copied from source objects. Also, accessor methods of some fields like `getBalance` and `getCreditLimit` should also return a copy of corresponding object. In method charge, we should first compare the amount to the credit limit. If not exceeds, the balance can be added. Otherwise an error message should be printed. Because the class Money is immutable like String, method `add` and `subtract` of Money don't change the original object, instead they return a new object. So, we should let the field points to new object.

5 实验结论

The lab has finished successfully. The programs can completely achieve all goals. Here is the screenshot.

6 实验体会和收获

In this lab we practiced a lot of object-oriented programming. We can write copy constructor by overloading, which is a really useful feature in engineering. Also, we have aggregated objects and classes. And we have experienced passing by reference. But I think the most important one is that we were introduced to the idea of immutable, which is commonly used in functional programming like Haskell.

7 程序代码

Money.java:

```java
/** Objects represent nonnegative amounts of money */
public class Money {
    /** A number of dollars */
    private long dollars;
    /** A number of cents */
    private long cents;

    /**
     * Constructor creates a Money object using the amount of money in dollars and
     * cents represented with a decimal number
     *
     * @param amount the amount of money in the conventional 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;
```

```
        }
    }

    /**
     * Constructor duplicates a Money object with an existing source object.
     * @param src the source object used to duplicate
     */
    public Money(Money src) {
        this.dollars = src.dollars;
        this.cents = src.cents;
    }

    /**
     * Adds the calling Money object to the parameter Money object.
     *
     * @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;
    }

    /**
     * Subtracts the parameter Money object from the calling Money object and
     * returns the difference.
     *
     * @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;
    }

    /**
     * Compares instance variable of the calling object with the parameter object.
     * It returns -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, and 1 if the dollars and the cents of the calling object
     * are more than the dollars and the cents of the parameter object.
     *
     * @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, and 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;
    }
```

```
    /**
     * Determine if an amount of money equals to this
     * @param amount the amount of money to compare against
     * @return if they're equal
     */
    public boolean equals(Money amount) {
        return this.compareTo(amount) == 0;
    }

    /**
     * Generate human-friendly string to represent the money
     */
    public String toString() {
        return String.format("$%d.%02d", this.dollars, this.cents);
    }
}
```

CreditCard.java:

```
public class CreditCard {
    private Money balance;
    private Money creditLimit;
    private Person owner;

    /**
     * Constructor to make a new credit card
     * @param newCardHolder the person who holds the card
     * @param limit         the credit limit
     */
    public CreditCard(Person newCardHolder, Money limit) {
        this.owner = newCardHolder;
        this.creditLimit = new Money(limit);
        this.balance = new Money(0.0);
    }

    /**
     * get the balance of credit card
     * @return balance
     */
    public Money getBalance() {
        return new Money(this.balance);
    }

    /**
     * get the credit limit of credit card
     * @return credit limit
     */
    public Money getCreditLimit() {
        return new Money(this.creditLimit);
    }

    /**
     * get the personal information of the owner
     * @return owner's information
     */
    public String getPersonals() {
        return this.owner.toString();
    }

    /**
     * add fund to the card
     * @param amount the amount to be charged
     */
    public void charge(Money amount) {
        Money newBalance = balance.add(amount);
        if (newBalance.compareTo(this.creditLimit) == 1) {
            System.out.println("Exceeds credit limit");
            return;
        }
        System.out.println("Charge: " + amount);
        this.balance = newBalance;
    }

    /**
     * pay using the card
     * @param amount the amount to be paid
```

```
     */
    public void payment(Money amount) {
        System.out.println("Payment: " + amount);
        this.balance = this.balance.subtract(amount);
    }
}
```