CSD 3464 – ASSIGNMENT 03 (Question 05)

Overview:

The following question is an extension to *Absolute Java* (6th Ed.)'s Chapter 05 Programming Project Q5 (pg. 341) – Part One and Two. Please follow the instructions included in **this** document and implement the following Java files:

⇒ MoneyDemo.java *(Contains a `main()` method)*
⇒ Money.java

The above classes/files should both be inside a package called *q5*.

Instructions:

Create a `Money` class with `private` fields that hold a positive `int dollars` and a positive `int cents`. In addition, create accessors and mutators for each field; with respect to the mutators, ensure `dollars` and `cents` are greater than or equal to 0 and `cents` (illegal values should be set to 0). If the `cents` provided in any constructor or setter is greater than or equal to 100 add the appropriate value to `dollars`.

The `Money` class should have three constructors: the first should be a no-parameter constructor which sets the fields `dollars` and `cents` to 0 while the second should take one parameter `int dollars` and sets the field `dollars` to the value provided (assuming it is a valid value). Lastly, the class will have a two-parameter constructor that takes `int dollars` and `int cents` and sets the field `dollars` to the provided value **plus** any whole dollars contained within the parameter `cents` (100 cents = 1 dollar) , while the field `cents` should be set to the provided value **minus** any whole dollars contained in the parameter `cents` (forces the `cents` field to be between 0 and 99, inclusive).

For example, if a value of 100 dollars and 123 cents is provided to the two-argument constructor, `dollars` should be set to `101` and `cents` should be set to `23`. Similarly, if a value of `-3` dollars and `-1` cent is provided both the `dollars` and `cents` fields should both be set to `0`.

In terms of methods, the `Money` class should have a `public` method `add` which takes a single parameter `Money money` and modifies the "this" object so its `dollars` and `cents` fields are set to the result of the **adding** the current object's `dollars` and `cents` fields to the parameter `money`'s `dollars` and `cents` fields. Remember, you need to account for handling a situation where adding the two `cents` fields exceeds 100 as this represents a whole `dollar`.

Similarly, the `Money` class should have a `public` method `minus` which takes a single parameter `Money money` and modifies the "this" object so its `dollars` and `cents` fields are set to the result of the **subtracting** the parameter `money`'s `dollars` and `cents` fields from the current object's `dollars` and `cents` fields. If the value resulting from subtraction leads to a negative result return a `new Money()` object with its fields `dollars` and `cents` fields set to `0`.

The class should also contain two `static` implementations of `add` and `minus` which both take two parameters `Money money1` and `Money money2` and return a `new Money()` object. The `static add` method should **add** the two `Money` objects, while the `static minus` method should **subtract** `money2` from `money1`. The add and subtract logic should be the same as that from the previous paragraph.

Your class must also include the method `public equals()` which compares two `Money` objects and returns the `boolean` value `true` if and only if both the `dollars` and `cents` of the two `Money` objects are the same.

Furthermore, the class `Money` must also include the method `public toString()` which returns a `String` object in the format "$d.cc" where d is the `dollar` value and cc represents the `cents` value. Any `cents` value of `0-9` should be proceeded by an additional `0`; for example, the `cents` value `8` should be formatted as "$d.08".

- The simplest way to implement the above `toString()` is to represent your `dollars` and `cents` as a single `double` variable and then use the `NumberFormat` class to format the `String` properly as a currency (*see chapter 02*).

You are also required to write a `static` class called `MoneyDemo` in another file that contains a `main()`. The purpose of this class is to test the functionality of your `Money` class. It is up to you to decided what to include in the `main()` to test the class; however, all `public` facing methods and constructors need to be tested for correctness. Additionally, you should test to ensure invalid entries (`dollars` and `cents`) passed to the constructors and mutator methods are correctly handled.

The basic structure of your `Money` class is required to look like the following:

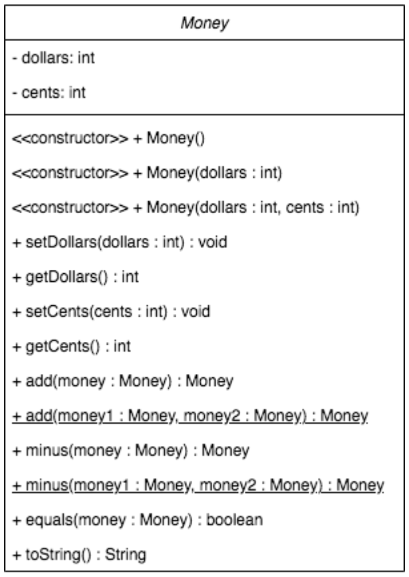
```
public class Money
{

     private int dollars;
     private int cents;

     // No Parameter Constructor
     public Money()
     {
        // insert code here
     }

     // One Parameter Constructor
     public Money(int dollars)
```

```
{
   // insert code here
}


// Two Parameter Constructor
public Money(int dollars, int cents)
{
    // insert code here
}

// Mutator - Dollars
public void setDollars(int dollars)
{
    // insert code here
}

// Accessor - Dollars
public int getDollars()
{
    // insert code here
}


// Mutator - Cents
public void setCents(int cents)
{
   // insert code here
}


// Accessor - Cents
public int getCents()
{
   // insert code here
}

// add method
public void add(Money money)
{
   // insert code here
}

// minus method
public void minus(Money money)
{
   // insert code here
}

// STATIC add method
public static Money add(Money money1, Money money2)
{
```

```
            // insert code here
        }

        // STATIC minus method
        public static Money minus(Money money1, Money money2)
        {
            // insert code here
        }


        // toString method
        @Override
        public String toString()
        {
            // insert code here
        }


        public boolean equals(Money money)
        {
            // insert code here
        }

}
```

UML Diagram:

The below UML diagram is provided for your convivence. Ensure all shown fields, constructors, and methods are included in your `Money` class prior to your final submission. Note: underlined values represent `static` methods.



Money

- dollars: int
- cents: int

<<constructor>> + Money()
<<constructor>> + Money(dollars : int)
<<constructor>> + Money(dollars : int, cents : int)
+ setDollars(dollars : int) : void
+ getDollars() : int
+ setCents(cents : int) : void
+ getCents() : int
+ add(money : Money) : Money
+ add(money1 : Money, money2 : Money) : Money
+ minus(money : Money) : Money
+ minus(money1 : Money, money2 : Money) : Money
+ equals(money : Money) : boolean
+ toString() : String