Tutorial 4

Objectives

Use **javadoc** commenting, basic class construction, overloading methods/constructors, and method overriding.

Attendance Quiz

Please log on to cuLearn using one of the computers in the tutorial room and complete the attendance quiz. You can only access the quiz if you log in using one of the computers in the lab. You cannot use a laptop for this. This is a time limited quiz. Be sure to do this as soon as you arrive.

At the end of the tutorial a TA will assign you a grade, call it G, which is 0, 1 or 2, depending on the progress you make during the tutorial. If you spend your time reading mail and chatting with your friends you will receive 0. If you have completed the attendance quiz on time then G will be your tutorial grade. If you have not completed the attendance quiz on time, then your tutorial grade will be max(0, G - 1/2). Each tutorial grade will therefore be one of 0, 0.5, 1.5 or 2.

In order to receive full marks for this tutorial, you must fully complete parts 1 and 2, and make good progress into part 3.

---

0) Make a directory called **comp1406t4**. Download all the tutorial 3 files to this directly.

1) In this problem, you will use **Javadoc**.

Look at the code in the **Find** class. You'll notice that the comments might look slightly different than normal Java comments. The commenting used is in the Javadoc format. What is Javadoc? From wiki, we see

Javadoc is a documentation generator from Oracle Corporation for generating API documentation in HTML format from Java source code. The HTML format is used to add the convenience of being able to hyper-link related documents together.

An **API** is the application programming interface. This is the interface between the writers of the code (classes) and the users of the code.

In order to generate the *html* code for the API, we will use the **javadoc** program from the command line (using cmd or terminal). In the directory where you compile and run your code for this tutorial, type

Windows Users

```
javadoc -d comp1406t4\docs comp1406t4\Find.java -author -version
```

OS X or Linux Users

```
javadoc -d comp1406t4/docs comp1406t4/Find.java -author -version
```

This will create a new directory called **docs** in your **comp1406t4** directory. Inside this new directory, is the *html* code for the API for the Find class. In the windows file viewer, click on the **index.html** file. This will open the file in a web browser. Click on the **Find** class (left pane) and see the API for the Find class.

What happened?

- *-d comp1406t4\docs* specifies where to put the html files
- *comp1406t4\Find.java* specified which file to generate javadocs for (use *.java for all java files)
- *-author -version* specify to list the author and version (if specified in the java files).

How do you write javadoc comments? In your code, you can add a special comment block just **before** a class, attribute, method or constructor declaration. This block of comments will be used in the generated API to describe whatever it is that the comment blocks comes before. There are some special tags that the Javadoc tool will read and use in this comment block. Here some simple rules

- The comment block must start with `/**` (two stars instead of 1) and end with `*/`
- For methods, each input argument should have an associated **@param** tag describing that input. (We can add pre-conditions on the argument here.)
- For non-void methods, the **@return** tag is used to describe the output (and any post-conditions).
- You can use basic HTML tags to help format the text. For example, **`main`** will format *main* in code format. Use `<p>` to start a new paragraph (with a blank line).
- The **@author** tag will list the author of the class or method

Now, go and add **javadoc** comments to the **MoneyDocs** class. This should include comments for the entire class, each constructor and each method in the class. Run the **javadoc** program to check that the html files created are as they should be.

More Reading

http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html
https://www.codeproject.com/Articles/658382/Basic-Javadoc-guide

2) Modify the **Money** class that is provided. Note that this class is **very** similar to the one from Tutorial 3. If you have completed Tutorial 3, feel free to copy code from there to here to help with this problem. Look at the provided Money class here to see what is different from the previous tutorial.

The **Money** class is a simple class that stores money as dollars and cents. For example, $12.73 will be stored as 12 dollars and 73 cents. The cents value stored should never be greater than 99, so 3 dollars and 164 cents should actually be stored as 4 dollars and 64 cents.

The class has only one method, **toString()**, which returns a String representation of the money object. Here we have

overridden the toString() method inherited from Object. Your first task is to create four constructors for the class as follows:

```
public Money(String name ){...}
  // create an object with zero dollars and cents.
  // sets the name of the object

public Money(String name, int c){...}
  // create an object with c cents
  // (adjusting dollars and cents so that 0<=cents<=99)
  // sets the name of the object

public Money(String name, int d, int c){...}
  // create an object with d dollars and c cents
  // (adjusting dollars and cents so that 0<=cents<=99)
  // sets the name of the object

pubic Money(String name, int[] coins){...}
  // input array has 6 elements and corresponds to
  // {#toonies, #loonies, #quarters, #dimes, #nickels, #pennies}
  // {$2, $1, $0.25, $0.10, $0.05, $0.01}
  // create an object with total money passed in array
  // (adjusting internal dollars and cents so that 0<=cents<=99)
  // sets the name of the object
```

In all the constructors, be sure that the internal state (dollars and cents) represents the total money and that cents is not greater than 99.

The **Money** class **overrides** the **toString()** from the Object class.You can use it to help test/debug your code. It returns a String representation of the money.

Use the testing program **TestMoney.java** to help test your constructors.

Next, add the following instance methods to your **Money** class:

```
public void add(int c){...}
 // adds c cents to the current value
 // Again, be sure the internal states
 // does not have cents greater than 99

public void add(int d, int c){...}
 // adds d dollars and c cents to the current value
 // Again, be sure the internal states
 // does not have cents greater than 99

public int remove(int c){...}
 // removes c cents from the current amount of money,
 // if there is enough money to remove c cents.
 // Otherwise, removes as much as it can.
 // Returns the actual amount removed.
 // Note: the input will satisfy c >= 0 (and it may be > 100).
```

Be sure to test your methods. Pay special attention to the **remove** method. As with the constructors, the intention is that your internal representation of the money will satisfy the condition **0 <= cents <= 99** at all times. Adjust your dollars and cents so that this is always maintained.

In the **TestMoney** class, add three new tests (one for add(int), one for add(int, int) and one for remove(int)).

More Reading

https://docs.oracle.com/javase/tutorial/java/javaOO/constructors.html

3) Create a class, in the **comp1406t4** package, called **MyBank**. The class will keep track of all the money you have (bank accounts, piggy banks, wallets, etc).

Your bank will store at most 100 money objects in an array of fixed size (100). This is a **partially filled array**. The order you store them doesn't matter, but you should store all objects at the *front* of the array (low index values). Adding a new money is easy, since you just add one to the back (the next free space in the array). When you remove one you need to *close* the gap it creates though. This is not an issue though, since you can simply move the last money object into the gap.

Complete all methods in the class. The **filter** method will print the name and value (using the **toString()** method) of all money objects with value in a certain range.

Write a **main** method to test your class.

–