Data Structures (CSC 151)
Fall 2024
Project 1: We're in the Money **Due: Tuesday, 9/17/2024**

> Unless explicitly stated otherwise, you may not import anything into your projects. That means you may not use any built-in data structures like ArrayList or Vector. This rule applies to all projects in this course.

Learning Objectives:

1. Review the concept of *refactoring*
2. Review the difference between *behavior* and *implementation.* A change in the latter should not cause a change in the former.
3. Practice with *JUnit testing* (We will talk about it in class in week 2.)
4. Practice *modularity* using private helper methods
5. Practice writing *reusable* code by minimizing direct access to instance variables

1 Introduction

A roll-a-coin bank was a plastic toy/piggy bank from the 1980s. The user put a coin into it, and the coin rolled down a ramp and over a set of holes until it reached a hole into which it fit. Because US coins all have different diameters, the bank neatly sorted coins like pennies, nickels, dimes, and quarters. One could remove a stack of a particular coin from the bottom. This project is about a simulation of such a bank.

2 Setup

Go to Nexus and download the starter code for this project. Unzip or extract it. That will give you a folder called `project01_coinbank`. Open VS Code and open that folder (by using the "Open Folder" option from the "File" menu).

Once done, have a look at the file named Coinbank.java in the **original** package. This class models the roll-a-coin bank defined earlier. There are some named constants which avoid magic numbers, and the class has four instance variables to keep track of the number coins of each type. The public methods are explained in the Javadocs, and there are some private helper methods, too. Study the code and run the Client a few times to convince yourself that this code works. The Client simply fills the bank with random coins and then removes a stack of one of them. PITFALL ALERT: The code uses the Java switch statement. If you've never seen it before, Google it to make sure you understand it.

3 Your Mission

Your job is to refactor the Coinbank class. You won't change the original. Instead, you'll complete the version in the **proj1** package. In this version, the four instance variables have been replaced with a single int array of length 4 named holder. The pennies will be held in `holder[0]`, and as you can see in the code, I made four constants to get rid of the magic numbers so you can say `holder[PENNY]` to get to the pennies. You are not allowed to add or remove instance variables.

4 The Details

Even though the *implementation* of the Coinbank is changing dramatically (replacing instance variables is a big deal!) the *behavior* of the Coinbank will not. That is, the new one should work just like the old one did. Make this happen by completing the following methods. You are not allowed to alter method prototypes in any way.

1. Write the constructor. Note that I already made a constant for the size of the array. Use it.
2. Write the get and set methods. If you see logic being repeated in both methods, extract that logic out to a helper method.
3. Write the remove method in a reusable way. Remember that for better information hiding, we want to minimize the number of methods that access the instance variables directly. I wrote the insert method with this in mind so that *the exact same code works in both versions.* That's good design. But I wrote the remove method crappily. Look at the original code – see all the places in remove where I access the instance variables? That's what forces you to rewrite it now. Solve that problem by rewriting it in such a way that uses the behavior of other methods to get the job done instead of accessing the holder array directly. One way to make sure you did this correctly is to replace the remove method in the *original* Coinbank class with the code you write. It ought to work there too.

5 Testing

In CSC 120 you learned to do unit testing, in which you create tests that work on small pieces of code and that give you easy-to-read PASS/FAIL output. We're going to continue doing that in this course using the **JUnit framework**. We will talk about it in class in week 2. (Note: we are going to use **JUnit 5/Jupiter**. See the "Resources" tab on Nexus for information on how to use JUnit in VS Code.)

I've already written half a dozen JUnit tests for you in the CoinbankTest class. Study the code and review the example we did in class to understand what's going on. Notice how I used a private helper method to easily create a bank filled with coins that is ready to test. You should add your own tests to this class. How many? As many as you think necessary to cover boundary cases that I missed.

The other way you should test is by submitting your Coinbank.java file to Gradescope. (I'll make the Gradescope tests available by the beginning of week 2.) This web page is also JUnit based, but you don't get to see the code. I have tried to make the titles of my tests meaningful so you can tell what's going on (hint: it's the same six tests as in CoinbankTest plus a few more). I will be grading you on your Gradescope results so make sure to submit your code with enough time before the deadline that you can review Gradescope's responses and revise your code to fix issues that Gradescope flags.

6 Submit

Be sure to include the following honor code affirmation as a comment at the top of `Coinbank.java`: *I affirm that I have carried out the attached academic endeavors with full academic honesty, in accordance with the Union College Honor Code and the course syllabus.*
Be sure to also acknowledge any help you received in this comment.

Submit the following files to Gradescope:

- `Coinbank.java`
- `CoinbankTest.java`

Please upload **only those two files**.

Before you submit, check that your code satisfies the following requirements:

1. Are all of your files properly commented?
2. Are your files formatted neatly and consistently?
3. Did you clean up your code once you got it to work? It should not contain any code snippets that don't contribute to the purpose of the program, commented out code from earlier attempts, or unnecessary comments.
4. Do you use variable and method names that are informative?

5. Did you get rid of all magic numbers?

Finally, submit your files to Gradescope (*Project 1: Coinbank*).

7 Gentle Reminder

Programming assignments are *individual* projects. I encourage you to talk to others about the general nature of the project and ideas about how to pursue it. However, the technical work, the writing, and the inspiration behind these must be substantially your own. If any person besides you contributes in any way to the project, you must credit their work on your project. Similarly, if you include information that you have gleaned from other published sources, you must cite them as references. Looking at, and/or copying, other people's programs or written work is inappropriate, and will be considered cheating.

8 Grading guidelines

- Correctness: Your program does what the problem specifications require. (Based on Gradescope tests.)
- Testing: The evidence submitted shows that the program was tested thoroughly. The tests run every line of the code at least once. Different input scenarios, and especially border cases, were tested.
- Documentation: Every public method and class is documented in the appropriate format (Javadoc) and provides the necessary information. The information provided would enable a user to use the class/method effectively in their code.
- Programming techniques: Programming constructions have been used appropriately in such a way that make the code efficient and easy to read and maintain. technique or algorithm, this has been correctly implemented. For this project, I will particularly look for good information hiding and modularity (e.g. through private helper methods) and reusability of the written code (e.g. the design of the remove method and the appropriate use of constants).
- Style: The program is written and formatted to ensure readability. For example, naming conventions are obeyed, whitespace (blanks, line breaks, indentation) is used help structure the code, formatting is consistent and the code is well organized.