P03 Elastic Piggy Bank

Programming II (CS 300) Spring 2020
Department of Computer Sciences
University of Wisconsin–Madison

Due: 9:59PM on February 12, 2020
Pair Programming **is Allowed.**
Register by Sunday, Feb 9 @ 9:59PM

Overview and Learning Objectives:

Frequently the provided objects and data types in Java are insufficient for the specific needs of a program, and it can simplify your life significantly to create a custom data type tailored exactly to your needs. This write-up may be more succinct than usual, as you have already thought about most of this problem already when you solved it with arrays.

This time, you'll be coding up an expandable Piggy Bank (though not *infinitely* expandable) with much of the same functionality as your first programming assignment.

Grading Rubric:

5 points	**Pre-Assignment Quiz**: You should not have access to this write-up until after you have completed the corresponding pre-assignment quiz through Canvas.
20 points	**Immediate Automated Tests**: Upon submission of your assignment to Gradescope, you will receive feedback from automated grading tests about whether specific parts of your submission conform to this write-up specification. If these tests detect problems in your code, they will attempt to give you some feedback about the kind of defect that they noticed. Note that passing all of these tests does NOT mean your program is correct. To become more confident in this, you must write and run additional tests of your own.
15 points	**Supplemental Automated Tests**: When your final grade feedback appears on Gradescope, it will include the feedback from these additional automated grading tests. These tests are similar to the Immediate Automated Tests, but check your code against different requirements within this specification.
10 points	**Manual Grading Feedback**: We will focus on looking at your algorithms, use of programming constructs, and the style and readability of your code. This grading usually takes about a week, after which you will find feedback on Gradescope.

Additional Assignment Requirements:

ArrayList and **Arrays** are once again off-limits. You may only import `java.util.Random`.

You may explicitly define a public **toString()** method for your instantiable classes, if you like.

Reminders:

Everyone must complete the pair programming registration form by Sunday, February 9 @ 9:59PM.

Ensure that your code is styled in conformance with the Course Style Guide.

0. Project Files and Setup

Create a new Java Project in Eclipse called P03 Elastic Piggy Bank. Then create three Java classes within that project's src folder:

- ElasticBank
- Coin
- ElasticTester

The **only** class which should have a main method is the tester class. ElasticBank and Coin will be *instantiable* classes.

You may add a fourth driver class to the project, loosely following the specification from P01, but we will not be providing further description of this class and it is not required for this assignment.

1. Creating First Instantiable Class

To keep things simple, begin with the Coin class. Coins have two identifying features: a string representing their name and an integer value representing their worth. Your Coin class should look like this:

- name — String, private
- value — int, private
- **public** Coin(String name, **int** value)
- **public** String getName()
- **public int** getValue()

And that's it! Once a Coin has been created, its name and value will not change, so we do not need to define mutators, only accessors. Don't make these fields **final**, just **private**.

The constructor should expect two arguments: a String representing the name of the coin, and a positive integer representing the value. You can assume that the arguments will be valid, and just create the Coin without further checks ~~and toss it to your Witcher~~.

2. Testing First Instantiable Class

In your ElasticTester, write a test function that creates two (2) coins of different names and values (for example, a penny and a quarter). Test your accessor functions on both coins!

```
public static boolean testCoinInstantiableClass () {
	Coin penny = new Coin("PENNY", 1);
	Coin quarter = new Coin("QUARTER", 25);
	if (!penny.getName().equals("PENNY")) return false;
	if (penny.getValue() != 1) return false;
	if (!quarter.getName().equals("QUARTER")) return false;
	if (quarter.getValue() != 25) return false;
	return true;
}
```

3. Creating Second Instantiable Class

The ElasticBank is similar to your Piggy Bank from P01, but with an important twist: its capacity can expand TWICE before it is "full".

ElasticBanks have the following **private** members:

- coins — array of Coins, private
- size — int, private
- expansionsLeft — int, private
- rand — Random, private, **static** (for use in the removeCoin() method)

Initialize the Random object right away, with whatever seed value you like.

ElasticBank will have two constructors:

- **public** ElasticBank()
- **public** ElasticBank(**int** initialCapacity)

If no argument is provided, the default initial capacity should be set to **10** Coins; otherwise set the initial capacity of the coins array to the value of the argument. All ElasticBanks may expand **2** times beyond their initial capacity.

The accessor methods for ElasticBank are:

- **public int** capacity() — **returns** the current capacity of the coins array (that is, how many total coins COULD fit in it right now, not including any future expansions)
- **public int** getExpansions() — **returns** the number of expansions left
- **public int** getSize() — **returns** the current number of Coins in the ElasticBank
- **public int** getBalance() — **returns** the current total value of coins in the ElasticBank
- **public** String getCoins() — **returns** a String representation of the Coins in the bank (see below):

Because arrays can be modified if a reference is returned, your accessor method for coins should create and return a String representation of the contents of the bank. Each Coin should be represented as "(name, value)", and the string representation should contain all of these pairs in one space-separated line. For example:

```
"(PENNY, 1) (QUARTER, 25) (PENNY, 1) (DIME, 10) (NICKEL, 5)"
```

The mutator methods for ElasticBank are:

- **public** Coin removeCoin() — removes a Coin from coins at random and returns it, replacing it with a **null** reference in the coins array.
- **public void** empty() — empties the ElasticBank entirely, replacing all Coins in coins with **null**.
- **public void** addCoin(Coin c) — adds a Coin to the bank and adjusts the capacity of coins if necessary and possible

The removeCoin() and empty() methods should work as they did in P01.

When the ElasticBank's coins array still has empty spaces, addCoin() works as in P01. When it is full, however, instead of failing as before, create a new array with **10 additional** spaces, copy the contents of the previous array over, and add the new Coin. Be sure to decrement expansionsLeft; this operation should only be allowed to happen **2** times during the life of the ElasticBank.

If you try it a third time (that is, when expansionsLeft is 0), the ElasticBank will burst! Use your empty() method to simulate the coins spilling everywhere. After this occurs, go ahead and add that Coin to the now-empty ElasticBank. To be clear:

- If you addCoin() and there is room in the ElasticBank, add the Coin.
- If you addCoin() and the ElasticBank is full:
    - If there are expansions left, expand the bank by 10 slots and then add the Coin (making sure you decrease the number of expansions left)
    - If there are **no** expansions left, empty the bank and then add the Coin

(You can add Coins to the empty ElasticBank again afterward, but it will not expand again and will keep spilling if you fill it up. It's not a great bank.)

4. Testing Second Instantiable Class

As this second class isn't *much* different from P01, we're not going to provide specific tests this time. Refer back to P0 for ideas on testing, and make sure you add additional tests to check the expanding part of addCoin().

Be careful: as when you tested Coin, you will want to make at least 2 different ElasticBank objects, and verify that their values are *different*. An instantiable class is merely a blueprint, and the objects you create from it should be distinct.

```
public static boolean testBalanceAccessors () {
      ElasticBank one = new ElasticBank(5);
      ElasticBank two = new ElasticBank(7);
      one.addCoin(new Coin("PENNY", 1));
      two.addCoin(new Coin("NICKEL", 5));
      if (one.getBalance() != 1) return false;
      if (two.getBalance() != 5) return false;
      return true;
}
```

5. Assignment Submission

You should submit Coin.java, ElasticBank.java, and ElasticTester.java through gradescope.com. Your score for the assignment is based on your submission marked "active" and made prior to the hard deadline. If you are working with a partner, be sure to review and follow the course Pair Programming Policy for linking your final submission to both your and your partner's account.