# Homework 3: Slippery Slot Machine

COSC150

**Assigned:** 9/28/2018
**Due:** 10/15/2018 at 11:59PM ET

This assignment is worth 180 points.

# Description:

You are a slot machine repair person. The Slippery Slot Machine Company makes an online slot machine, and at the core of it is a program that computes the user's winnings. You are given a copy of the bytecode of the program (available on Autolab for download) and the rules listed below for how it should work. Your job is to write JUnit tests that verify the implementation's correctness or find any errors. And then you can go ahead and fix the code (and re-test to verify).

## Part One, in which you identify flaws in our buggy code

The slot machine is supposed to work like this: for a single $2 play, five numbers are selected independently and uniformly at random, each between 1 and 50 inclusive. Payoff is according to the following rules, and if more than one rule applies, add the payment values.

1. If all 5 numbers are the same, pay $1,000,000.
2. If 4 are the same and 1 is different, pay $10,000.
3. If 3 numbers are the same as each other and the other 2 are also the same as each other (but not the same as the 3, i.e., a "full house"), then pay $5000.
4. If 3 numbers are the same and the other 2 are not the same as the 3 or each other, pay $100.

5. If 2 numbers are the same (and otherwise no matches) the payoff is $10.
6. If one or more of the numbers is a perfect (integer) square, add $7 (awarded at most once per spin)
7. If one or more of the numbers is 42 add $2 (awarded at most once per spin).
8. If one or more of the numbers is a power of 2, add $3 (awarded at most once per spin).

*Clarification:* If the same rule applies more than once (for example, with rule 5, a spin has two sets of two pairs), then the award should only be given once.

The program SlipperySlot class has two methods:

`int[] pullTheLever()`

generates a random 5 digit number, delivered as an int array, and

`int payOff( int[] m )`

computes the payment (in dollars) due the user, where m is an array of five integers that represents the spin of the slot machine. If none of the rules (1 through 8) above apply, the function should return 0.

**You will write a JUnit fixture named TestSlipperySlot.java that contains JUnit tests for the `payOff` method of SlipperySlot.class.** Your program should not include a main().

Your unit tests should be sufficiently comprehensive to catch major errors in the payOff logic described above. For your sanity's sake, you can assume that `payOff` does not have any malicious code and does not exhibit misbehavior if it observes a specific random number. (For example, it is not the case that `payOff` works correctly unless the number is 23-19-19-10-31. We are not that evil[1].) In other words, your goal is to write JUnit tests that catch "unintentional" programming errors[2].

**Important:** We reserve the right to use several different broken SlipperySlot implementations for evaluating your unit tests. That is, while we'll provide one for development of your unit tests, for the final grade, we may adjust the autograder to use different buggy implementations. Your JUnit tests thus need to be fairly comprehensive.

[1] Says we.

[2] OK, so clearly we introduced intentional programming errors into SlipperySlot.class. But our intentional errors are designed to mimic common, unintentional programming errors. Plus, it's certainly possible that we unintentionally screwed something else up, which your JUnit tests should catch.

Note that we are giving you SlipperySlot as a .jar file. The jar file contains several compiled Java classes. The only one you need to know about is the SlipperySlot class.

You'll want to add the SlipperySlot.jar file to your Eclipse project. To do this, once you've created a new Java Project in Eclipse:

1. Go to Project → Properties
2. Select "Java build path"
3. Click on Libraries tab
4. Click on "Add JARS…"
5. Select SlipperySlot.jar

This part of the assignment is worth 100 points. Roughly, this corresponds to 10 points for catching the flaw in eight buggy versions of our SlipperySlot implementations and 20 points for not finding flaws in two correct implementations.

---

## Part Two, in which you write better code than we do

Next, you'll write your own implementation, called BetterSlot, that correctly implements the `pullTheLever()` and `payOff()` methods.

This part of the assignment is worth 80 points. A rough grading rubric for this part is as follows:

| **Amazing Feat** | **Points Awarded** |
|---|---|
| You submitted something. Anything. | 10 |
| It compiles! | 10 |
| Rule 1 correct | 5 |
| Rule 2 correct | 5 |
| Rule 3 correct | 5 |
| Rule 4 correct | 5 |

| | |
|---|---|
| Rule 5 correct | 5 |
| Rule 6 correct | 5 |
| Rule 7 correct | 5 |
| Rule 8 correct | 5 |
| Rules work correctly together | 20 |

You should strongly consider running your JUnit tests against BetterSlot. The tests should all succeed.

However, when you submit your homework, please make sure that your JUnit tests are aimed at the SlipperySlot class, **not** your improved implementation.

# OK, I've just finished my amazing JUnit tests and BetterSlot. How do I turn these things in?

**What to turn in:**

You will turn in a single zip file that contains both **TestSlipperySlot.java** and **BetterSlot.java**, along with any other supporting .java files (if you have multiple classes, for example). Your .zip file should not have any directories; that is, everything should be in the "root" of the .zip file. You do not need to include SlipperySlot.jar (or any of the files contained therein) in your .zip file submission.

If you have trouble creating .zip files, please post a note on Piazza.

Once you have created a .zip file that contains your code (again, in the top (and only) directory of the zip file), upload it to Autolab at https://autolab.georgetown.edu/. In Autolab, look for the "SlipperySlot" assignment.

**Where to go for help:**

For questions about this assignment, please post to Piazza at https://piazza.com/georgetown/fall2018/cosc150/home.