Homework 9

Vegas Baby!

1 Introduction

Do you like to gamble? This HW will give you a chance to create a game of chance. You'll be creating a slot machine (one-armed bandit) simulation. A slot machine is a gambling machine commonly found in Las Vegas, but here we will just be building one for fun. A slot machine consists of three spinning circular drums which each have many different symbols around them. The user pulls down a handle and the drums start spinning. They each then stop randomly on a symbol and if certain combinations of symbols come up, then the player wins. Just think of it like 3 random number generators.

2 Problem Description

In our game, each of the three positions have four different symbols: bell, grape, cherry and a dash or line. Each drum has one of the bell, grape, and cherry, and five of the lines. Thus, each of the eight different possibilities (bell, grape, cherry, line, line, line, line, line) should come up with equal probability on each spin. The user can bet round dollar amounts of one dollar or more. The machine pays out combinations as shown below (multiples of the bet amount).

	Combo		**Payoff**
Bell	Bell	Bell	10
Grape	Grape	Grape	7
Cherry	Cherry	Cherry	5
Cherry	Cherry	******	3
Cherry	******	Cherry	3
******	Cherry	Cherry	3
Cherry	******	******	1
******	Cherry	******	1
******	******	Cherry	1

The ****** can be any character including the lines.

3 Solution Description

Your objective on this HW is to implement a GUI application using JavaFX that has a cell for each of the 3 random symbols coming up and a button which starts a “spin”. The interface should display the user’s current money holdings (starts at $100). It also should have a text field where the user can enter his or her bet with a minimum bet of $1 (default) and a max up to the person’s present holding.

Your application should also have a set of two radio buttons: Regular and Test. When the slot machine is in regular mode, the game should play as described above. When it is in Test mode, no Line symbols should ever come up on any of the drums. (The Bell, Grape, and Cherry should have equal likelihood of coming up.) This will allow us to test your game with more chances of winning combinations coming up.

The normal flow is for the user to enter a bet by typing it in (or just using the default $1) and then pressing the spin button. The three windows on the drums then should show the random symbols that come up. From there, the program calculates the person’s winnings (if any) and updates the current money holdings. For example, suppose the user wagers $20 and then two lines and one cherry come up. In that case, the person wins another $20 which should be added to their money total that was present before the wager.

You should try to make your user interface be as functional and look as good as possible. Do a nice layout of the individual components. Feel free to embellish. For instance, go out and find an image file for the different symbols to use in your interface (remember that labels can take images). At a minimum, you are required to have the three symbol cells, the spin button, the wager text entry field, a text label showing the current money amount, and the radio buttons for regular and test mode. Beyond that, it’s up to you.

Here’s some advice for this HW. Work incrementally. Get a basic UI up and running. Add the buttons, cells for display, etc., piece by piece. Think of which type of layout manager(s) you want to use.

You should put your slot machine software in a class named `Slot`. Additionally, you should have a class named `Vegas` that simply holds `main` and creates an instance of the `Slot` class.

4 Extra Credit

We will make available 10 points of extra credit on this HW for applications that are particularly creative, engaging, beautiful, etc. There is no set criteria for this. We will make a subjective judgment.

5 Javadocs

For this assignment you will be commenting your code with Javadocs. Javadocs are a clean and useful way to document your code’s functionality. For more information on what Javadocs are and why they are awesome, the online documentation for them is very detailed and helpful.

You can generate the javadocs for your code using the command below, which will put all the files into a folder called javadoc:

```
$ javadoc *.java -d javadoc
```

The relevant tags that you need to have are `@author`, `@version`, `@param`, and `@return`. Here is an example of a properly Javadoc'd class:

```java
import java.util.Scanner;

/**
 * This class represents a Dog object.
 * @author George P. Burdell
 * @version 13.31
 */
public class Dog {

    /**
     * Creates an awesome dog (NOT a dawg!)
     */
    public Dog(){
        ...
    }

    /**
     * This method takes in two ints and returns their sum
     * @param a first number
     * @param b second number
     * @return sum of a and b
     */
    public int add(int a, int b){
        ...
    }
}
```

Take note of a few things:

1. Javadocs are begun with `/**` and ended with `*/`.
2. Every class you write must be Javadoc'd and the `@author` and `@version` tag included. The comments for a class start with a brief description of the role of the class in your program.
3. Every non-private method you write must be Javadoc'd and the `@param` tag included for every method parameter. The format for an `@param` tag is `@param <name of parameter as written in method header> <description of parameter>`. If the method has a non-void return type, include the `@return` tag which should have a simple description of what the method returns, semantically.

5.1 Javadoc and Checkstyle

You can use the Checkstyle jar mentioned in the following section to test your javadocs for completeness. Simply add -j to the checkstyle command, like this:

```
$ java -jar checkstyle-6.2.1.jar -j *.java
Audit done. Errors (potential points off):
0
```

6 Checkstyle

You must run checkstyle on your submission. The checkstyle cap for this assignment is **20** points.
Review the Style Guide and download the Checkstyle jar. Run Checkstyle on your code like so:

```
$ java -jar checkstyle-6.2.2.jar -a *.java
Audit done. Errors (potential points off):
0
```

The message above means there were no Checkstyle errors. If you had any errors, they would show up above this message, and the number at the end would be the points we would take off.

The Java source files we provide contain no Checkstyle errors. For this assignment, there will be a maximum of **20** points lost due to Checkstyle errors (1 point per error). In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

7 Turn-in Procedure

Non-compiling or missing submissions will receive a zero. NO exceptions

Submit all of the Java source files you modified and resources your program requires to run to T-Square. **Do not submit** any compiled bytecode (`.class` files) or the Checkstyle jar file. When you're ready, double-check that you have submitted and not just saved a draft.

Please remember to run your code through Checkstyle!

7.1 Verify the Success of Your Submission to T-Square

Practice safe submission! Verify that your HW files were truly submitted correctly, the upload was successful, and that the files compile and run. It is solely your responsibility to turn in your homework and practice this safe submission safeguard.

1. After uploading the files to T-Square you should receive an email from T-Square listing the names of the files that were uploaded and received. If you do not get the confirmation email almost immediately, something is wrong with your HW submission and/or your email. Even receiving the email does not guarantee that you turned in exactly what you intended.
2. After submitting the files to T-Square, return to the Assignment menu option and this homework. It should show the submitted files.
3. Download copies of your submitted files from the T-Square Assignment page placing them in a new folder.
4. Recompile and test those exact files.
5. This helps guard against a few things.
   (a) It helps insure that you turn in the correct files.
   (b) It helps you realize if you omit a file or files. [1] (If you do discover that you omitted a file, submit all of your files again, not just the missing one.)
   (c) Helps find last minute causes of files not compiling and/or running.

[1] Missing files will not be given any credit, and non-compiling homework solutions will receive few to zero points. Also recall that late homework will not be accepted regardless of excuse. Treat the due date with respect. The real due date is midnight. Do not wait until the last minute!