CS 1331: Introduction to Object Oriented Programming

Homework 4

Due: February 14, 2014 @ 8:00 PM

We're Going to Hawaii!

So now that the snow is gone, everyone's probably ready for some warmer weather, and what place has better weather than Hawaii? When going to Hawaii, it's always nice to come up with your own Hawaiian name, so for this homework you will be creating a program that will test out different names to see if they are Hawaiian.

Start by asking the user to input their possible Hawaiian name. You should then test that name to see if it contains only the letters in the Hawaiian alphabet: **(a, e, i, o, u, k, l, h, m, n)** (don't forget the user may type in capital letters). If it does, there are 3 options that could be printed out:

- There is a 50% chance it will print out: `Aloha, <insertNameHere>`
- A 30% chance it will print out: `Welcome to Hawaii, <insertNameHere>`
- And a 20% chance it will print out: `E komo mai, <insertNameHere>`

Once the response has been printed, your program should ask the user if they would like to try another name (i.e. print out `Would you like to try again? (y/n)`) and if the user enters y, your program should run again from the beginning. If they enter `n`, the program should end.

For example, if the user were to enter `Lilo`, your program could print out

```
Welcome to Hawaii, Lilo
```

Of course, if their name contains any letters *n*ot in the Hawaiian alphabet, we can't let them come with us to Hawaii. But that's no fun now is it? So instead, you'll write a method that goes through a person's name, character by character, and removes each letter that's not Hawaiian! After that you can print out one of the options listed above.

For example, if the user were to enter `John` here is what it would print out:

```
Sorry John, you aren't Hawaiian enough to come!
Let's make your name... ohn!
Aloha, ohn
```

Make sure you check the Code Structure section on the next page for our guidelines on how to layout your code!

Note: We don't really care about capitalization. It's a little tricky. Feel free to try though!

Code Structure

Since this is one of the first assignments that we are asking you to make multiple classes, we are going to layout how we'd like you to structure some of your code. You should split your code into two different classes.

- A class called `Hawaii` with three methods (you may name them however you like, as long as it's descriptive)
  - One to see if the String is Hawaiian
  - Another to choose the appropriate Welcome String to be printed
  - A third to fix a non-Hawaiian name
- A class called `HawaiiDriver` that will contain the main method to run the program and ask the user if they would like to continue testing names

Javadocs

Javadocing and commenting your code helps others, and even yourself, understand what your code is doing. You guys have been doing traditional comments all along, Javadocs are kind of like super comments that we typically use at the top of methods and classes. They are actually what is used to generate that beautiful API that you guys all know and love!

Javadocs are similar to comments but start off by using `/**` instead of `//` or `/*`. They also end in a *\.

Lets start off by javadocing a simple class named Dog. The first step is to javadoc the class header. We will place the javadoc statements above the class header, but below any import statements.

```
import java.util.Scanner;

/**
* This class represents a Dog object.
* @author Ethan Shernan
* @version 1.0
*/

public class Dog {
....
}
```

The `@author` and `@version` tags identify special characteristics for the javadoc. In each homework from now on, you will need to include a Javadoc at the top of each class you submit with these tags and a short description.

We can also Javadoc *methods* to help describe what they do, what they take in as parameters, and what they return. As an example, let's say we had an `add(int a, int b)` method that takes in two ints and returns another int.

```
/**
* This method takes in two ints and returns their sum
* @param a, b
* @return their sum
*/

public int add(int a, int b)){
...
}
```

Again, the `@param` and `@return` tags are helpful identifiers for specific information about the method (the parameters and returned value). In each homework from now on, you will need to Javadoc all of your methods with these tags and a short description.

Checkstyle

Just in case you forget how to do Checkstyle, here are the instructions again!

You have been provided with a Checkstyle Jar file and the CS 1331 Checkstyle configuration file (`cs1331-checkstyle.xml`). We do this to make sure you are following the proper style guidelines as posted in the Style Guide under T-Square resources. Please refer to these guidelines for any help as you are writing your programs.

We will run checkstyle on all the Java source files you submit. You can run Checkstyle with the command below (in the directory containing all your Java source files):

```
$ java -jar checkstyle-5.6-all.jar -c cs1331-checkstyle.xml *.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. You can easily count Checkstyle errors by using the command below and subtracting 2 from the number printed (which is how we will deduct points). For example:

```
$ java -jar checkstyle-5.6-all.jar -c cs1331-checkstyle.xml *.java | wc -l
2
```

Also means no errors.

If you have any questions about a Checkstyle error, please let a TA know! They should be pretty self-explanatory and you should be able to fix them easily. We will be taking points off for checkstyle on this assignment, so please make sure you run it!

Turn-in Procedure

Submit all of the Java source files you created to T-Square and your HTML page for your Applet. Do not submit any compiled bytecode (`.class` files). When you're ready, double-check that you have submitted and not just saved a draft.

Please remember to run your code with Checkstyle!

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 8PM Friday. Do not wait until the last minute!