Lab 8: Object Oriented Design!

Let's get started!

Today we'd like you to:

1. **Open Eclipse**. Remember to keep all your projects in the same workspace. Think of a workspace as a shelf, and each project is a binder of stuff on the shelf. If you need some help organizing your workspace, we'd be happy to help!
2. **Create a new Java project**. Call it something that will help keep your code organized, i.e., COMP 1510 Lab 8.
3. **Complete the following tasks**. Remember you can right-click the src file in your lab 8 project to quickly create a new Java class.
4. When you have completed the exercises, **show them to your lab instructor**. Be prepared to answer some questions about your code and the choices you made.
5. **Remember that for full marks your code must be properly indented, fully commented, and free of Checkstyle complaints**. Remember to activate Checkstyle by right-clicking your project in the Package Explorer pane and selecting Checkstyle > Activate Checkstyle.

What will you DO in this lab?

In this lab, you will:

1. Enhance your PasswordValidator by generating some helper methods to ensure a password meets stringent content requirements
2. Use a Scanner to open a file and copy text from it into an ArrayList
3. Measure the maximum number of heads in a row we can generate with a simple Coin class
4. Create a simple program that uses multiple classes including the ArrayList and the ArrayList's Iterator to manage a cat hotel (meow!).

Table of Contents

1. More Password (and GUI and String) Fun!

Let's modify our PasswordValidator JavaFX program from last week. So far, a password is valid if we correctly enter the same password in two fields. Let's make the requirements a bit more stringent:

1. Copy your PasswordValidator class from lab 7 to lab 8. Find it in your lab 7 project. Right click the class in the Package Manager, and choose copy. Right click the package ca.bcit.comp1510.lab8 and select paste.
2. Did you notice that when you copied the class to the new project, the package declaration in the file was automatically updated? Neat!
3. The PasswordValidator's event listener responds to the button press by checking if the two Strings are equal, i.e., they contains the same sequence of characters. We want to modify the event listener so that it uses some private helper methods to ensure the password is tougher to crack. Create a separate private helper method for each of the following cases. Each private helper method must accept a String as a parameter and return true or false depending on whether the String parameter:
    i. contains at least one capital (upper case) letter (A, B, C, …, X, Y, Z)
    ii. contains at least one lower case letter (a, b, c, …, x, y, z)
    iii. contains at least one number (0, 1, 2, 3, …, 8, 9)
    iv. contains at least one special character (one of ! " # $ % & ' ( ) or *)
    v. is at least 8 characters long and no more than 10 characters long.
4. Finally, there are usually constraints about whether passwords can contain certain words and sequences like 'password' or '123'. Let's create a separate class for our PasswordValidator to use.
5. Note: In Chapter 5 there is a program called URLDissector. URLDissector uses a Scanner to open a file and "read" the words stored in it. We're going to do this, so it is a good idea to refer to that code sample from the text.
6. Create a text file in your ca.bcit.comp1510.lab8 package called wordlist.txt. Add a few words to it, one per line, like password, 123, etc.
7. Create a new class called ForbiddenWords. ForbiddenWords contains two instance variables, an ArrayList of String, and a Scanner.
8. ForbiddenWords contains a single constructor which accepts a single String as a parameter. The String is the name of the file that contains the list of forbidden words. The constructor should:
    i. Instantiate the ArrayList of String
    ii. Instantiate the Scanner. Instead of reading from System.in, we want to read from a new File whose name is the String passed to the constructor. Your constructor should contain a line of code that looks like this:
    ```
    fileScanner = new Scanner(new File(fileName));
    ```
    iii. Use a while loop and the Scanner's hasNext method to loop through the list of words and add each one to the ArrayList of String.
9. Forbidden words contains a single public method called containsWord. It accepts a String, and returns true if the String is in the list of forbidden words, else it returns false.

10. Modify PasswordValidator so that it has a ForbiddenWords instance variable. Initialize the ForbiddenWords object inside the PasswordValidator start method. Pass in the name of your word list. For example, mine looks like this:

```
wordlist = new ForbiddenWords("src/ca/bcit/comp1510/lab8/words.txt");
```

11. Modify your PasswordValidator's event handler so that it uses the ForbiddenWords object to ensure the potential password does not contain any of the forbidden Strings.

2. Coins!

There is a very simple Coin class in chapter 5 of the text. You can find it in the COMP 1510 Examples project in the chapter 5 package. Copy it to your lab 8 project.

1. Let's write a program called CoinRunner. CoinRunner only contains a main method.
2. Inside the main method, you will need to declare and instantiate a Coin.
3. Using a for loop, flip the coin 100 times. Determine the longest run of heads in 100 flips of the coin:
    i. Suppose we flip the coin 10 times: H, H, H, H, T, T, T, H, H, T. The longest run of heads in this case is 4.
    ii. Suppose we flip the coin 5 times: H, T, H, H, T. The longest run of heads in this case is 2.
4. Your program should print out the length of the longest run of heads. For example, we flipped a coin 10 times and the output looked like this:

```
Tails
Tails
Heads
Heads
Heads
Heads
Heads
Heads
Tails
Heads
The longest run of heads is 6
```

3. Cats

It's time for some fun. Let's create a small program that manages a Cat Hotel:

1. Create a new class called Cat:
    i. A Cat has a name (String) and an age in years (int).
    ii. The constructor should accept a parameter for each instance variable. If the user tries to create a Cat with a negative age, give the new Cat an age of zero. If the user tries to create a Cat with a blank name or no name, call the cat "Cleo".

iii. A Cat needs an accessor for each instance variable.
iv. A Cat's name will not change. Make the instance variable that stores the name final, and don't create a mutator for it. You should create a mutator for the Cat's age.
v. Create a toString for the Cat.

2. Create a new class called CatHotel:
   i. A CatHotel has an ArrayList of Cat to store the guests, and a String for the hotel's name.
   ii. The Constructor should accept a String for the name of the hotel and assign it to the instance variable. The constructor should instantiate the ArrayList of Cat as well.
   iii. The CatHotel needs a public method called addCat. The method accepts a Cat as its parameter, and adds the Cat to the ArrayList of guests. There is no return vaue.
   iv. The CatHotel needs a public method called removeAllGuests(). It should remove (clear) all the guests from the hotel. We can do this by using one of the helpful ArrayList methods (check out the ArrayList Java doc!).
   v. The CatHotel needs a public method called guestCount() which should return the number of guests currently in the CatHotel.
   vi. The CatHotel needs a public method called removeOldGuests. This method should accept an integer. Use an iterator to iterate through the guests ArrayList and check each guest's age. If the guest is older than the integer passed to the method, remove it from the CatHotel. This method should return the number of Cats that were evicted from the CatHotel. To create an iterator that iterates over an ArrayList, you can use the Iterator return from the collection's iterator method like this:

```
Iterator<Cat> catIterator = catList.iterator();
while (catIterator.hasNext()) {
    Cat temp = catIterator.next();
    // Add your code here
}
```

   i. The CatHotel needs a public printGuestList method that should print the name of the CatHotel and then use a loop to print the toString method for each Cat currently in the CatHotel. Hint: use a for-each loop!

3. Create a Driver file that contains a main method. Inside the main method, create an instance of the CatHotel class. Use a Random object to add some cats with random ages to your CatHotel. Test the methods you implemented to make sure they work.

4. You're done! Show your lab instructor your work.