Lab 9: 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 Comp1510Lab09*LastNameFirstInitial*
3. **Complete the following tasks**. Remember you can right-click the src file in your lab 9 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.

What will you DO in this lab?

In this lab, you will:

1. Test the provided Complex class
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. Some Complex Fun!

As an example of classes with methods that have objects of the same class as parameters, the Complex class was shown (if you want to know more about the mathematics underlying this class, see the Wikipedia page, https://en.wikipedia.org/wiki/Complex_number). This class had a test driver class but no JUnit tests. As we know JUnit tests are better than test drivers, lets improve the situation:

1. Copy the Complex and ComplexTester (from the Learning Hub) into your ca.bcit.comp1510.lab09 project for this lab.
2. Create a test class ComplexTest that tests the methods of the Complex class (put this into a test folder as in earlier labs).
3. Add code in the @Test methods to duplicate the tests in the test driver.
    i. the test driver class expects a human to look at the results. You will have to use assertions in place of a human.
    ii. you will also need to decide what tolerance to use for results which are not exact, i.e. involve fractions in their calculation or results.
    iii. Since the instance fields are double, more than 15 digits is not expected. Less if several arithmetic operations are needed in the calculations.
4. Check the test coverage to make sure it is green. This does not guarantee adequate testing but is a good start.
5. The Javadoc for Complex.exp() mentions Euler's number *e*. Can you find out where it is used?
6. The expression $e^{\pi i} + 1 = 0$ is called Euler's formula. It relates the five most important mathematical constants. How close does this work with the Complex class (hint: this calculation is in the ComplexTester class).
7. One testing strategy is consistency. Where methods are related, we can get additional tests confirming this
    i. getters and setters are opposites, such that if you set something you should be able to get the same thing back
    ii. exp() and log() are inverses, in that z = z.log().exp() = z.exp().log() (because $e^{2\pi i} = 1$, the logarithm is only defined up to a multiple of $2\pi i$. Log() is defined to have the imaginary part in the range $-\pi i$ to πi)
    iii. add() and sub() as well as multiply() and divide() are also opposites
    iv. ZERO and ONE are units of addition and multiplication, respectively
8. As testing is generally limited by the time available, feel free to test as much or as little as you want (minimally, test the cases given in ComplexTester).

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 9 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 CatHotelDriver.java 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! Submit your work.

If your instructor wants you to submit your work in the learning hub, export it into a Zip file in the following manner:

1. Right click the project in the Package Explorer window and select export…
2. In the export window that opens, under the General Folder, select Archive File and click Next
3. In the next window, your project should be selected. If not click it.
4. Click *Browse* after the "to archive file" box, enter in the name of a zip file (the same as your project name above with a zip extension, such as Comp1510Lab09BloggsF.zip *if* your name is Fred Bloggs) and select a folder to save it. Save should take you back to the archive file wizard with the full path to the save file filled in. Then click Finish to actually save it.
5. Submit the resulting export file *and* your answers document as the instructor tells you.