**Lab Session #3**

**Introduction**

A general task you'll have to do every week is to experiment with the source code shown in the lecture. Additionally, if you did not finish any of the tasks from last week, you have to make sure you caught up with any remaining tasks.

**Task #1: Review Java Anonymous Classes**

Make sure you understand how anonymous classes work in Java. The source code for Chapter 2 contains this little puzzler, MeaningOfThis.java:

package lambdasinaction.chap2;  
  
public class MeaningOfThis  
{  
 public final int value = 4;  
 public void doIt()  
 {  
 int value = 6;  
 Runnable r = new Runnable(){  
 public final int value = 5;  
 public void run(){  
 int value = 10;  
 System.out.println(this.value);  
 }  
 };  
 r.run();   
 }  
 public static void main(String...args)  
 {   
 MeaningOfThis m = new MeaningOfThis();  
 m.doIt(); // ???   
 }  
}

What will the output be when this code is executed: 4, 5, 6, or 42? Explain why!

**Task #2: Predicates**

The source code for Chapter 2 contains a revised version of FilteringApples.java, which works with predicates. Experiment with the source code until you are comfortable working with predicates. Add JUnit tests for this program and compute the test coverage with JaCoCo.

**Task #3: Write a flexible prettyPrintApple method**

Write a prettyPrintApple method that takes a List of Apples and that can be parametrized with multiple ways to generate a String output from an apple (a bit like multiple customized toString methods). For example, you could tell your prettyPrintApple method to print only the weight of each apple. In addition, you could tell your prettyPrintApple method to print each apple individually and mention whether it’s heavy or light. The solution is similar to the filtering examples we’ve explored so far. To help you get started, here's a rough skeleton of the prettyPrintApple method:

public static void prettyPrintApple(List<Apple> inventory, ???){  
 for(Apple apple: inventory) {  
 String output = ???.???(apple);  
 System.out.println(output);  
 }  
}

(*Hint:* create an AppleFormatter interface with different implementations, similar to the ApplePredicate you've seen before.)

**Task #4: Java Lambdas**

Work with the example programs for Chapter 3, Lambdas.java and Sorting.java. Make sure you understand how to construct and execute lambda functions in your code. Experiment with writing different lambdas for filtering and sorting apples.

Java has a class called ThreadLocal that acts as a container for a value that’s local to your current thread. In Java 8 there is a new factory method for ThreadLocal that takes a lambda expression, letting you create a new ThreadLocal without the syntactic burden of subclassing. Find this method in Javadoc or using your IDE.  Now write a new lambda using the Java DateFormatter class, which isn’t thread-safe. Use the constructor to create a thread-safe DateFormatter instance that prints dates like this: "01-Jan-1970".

**Task #5: Execute Around Pattern**

We covered the ‘Execute Around Pattern’ for data processing in the lecture. The source code for Chapter 3 has an example implementation in ExecuteAround.java. Run the program and make sure you understand how lambdas are passed for behavior parametrization. Now add a new processing lambda that computes the number of lines for a file. It should be called like this:

String numLines = processFile( <your\_new\_lambda> );  
System.out.println(numLines);

That's all for this lab!