Lab 11

1. A Java class called `RemoveDupsClass` has been provided to you in the lab folder.
   A static method -

   ```
   public static String[] removeDups(String[] array)
   ```

   inside `RemoveDupsClass` accepts an array of Strings and produces a new array in which all duplicate Strings in the original input array have been removed.
   Create a JUnit Test Case `TestRemoveDups` to test this method. In `TestRemoveDups`, create a method called `testRemoveDups`, with `void` return type and no arguments. This method should pass in some array (hard-coded array is fine) to the `removeDups` method.

   After the call, the variable result should contain an array of `String`s without duplicates, and each of these `String`s should be an element of the original input array.

   Verify that both these things are true. You can use the JUnit function `assertTrue` to do tests.

5. In the package lesson11.labs.prob5, there is a class `FixThis` in which a stream `map` is called which accesses another method that throws an `Exception`. The code will not compile as it is written. Use one of the Java 8 exception-handling strategies to get the code to compile and run – create a new class `FixThisSoln` for this purpose. A `main` (commented) method is provided. Expected output for the first call to `processList` is

   ```
   [not, too, big, yet]
   ```

   However, the second call should throw a `RuntimeException`.

7. In the `package lesson11.labs.prob7`, there are classes `Main` and `Employee`. The `main` method in `Main` loads a list of `Employee` s and then attempts to print, in sorted order, the full names of those `Employee`s whose salary is greater than 100,000 and whose last name begins with any letter that comes after 'M' in the alphabet. This exercise asks you to refactor this processing step in the `main` method so that it can be unit tested, using the techniques mentioned in the Lesson. Do the following:
   a. It is difficult to test an expression that simply prints to console. Move this processing step into two methods, `asString(List)`, which does the same processing, but returns a `String` rather than printing to the console, and `printEmps(List)`, which calls `asString` and then prints the string to the console. Replace the processing step in the `main` method with a call to `printEmps`.
   b. Create two packages, `soln1, soln2`, where you will put the two different types of solutions you will develop for testing this code.
   c. In `soln1`, create a JUnit `Test` class that tests the `asString` method. Make sure you test with a few `Employee` instances so that at least one `Employee` is excluded from the

list and at least one is included in the list. This is an example of the Simple approach mentioned in the slides.

   d. In `soln2`, refactor the `asString` method so that method references are used to call auxiliary methods, as in the Complex case described in the lecture. Create auxiliary methods `salaryGreaterThan100000(Employee e)` and `lastNameAfterMEmployee e)` for this purpose. Then create a `Test` class in `soln2` that tests these auxiliary methods, along with the `fullName(Employee e)` method. Does this approach provide a good test for the `asString` method?

8. In the package there is a class Queue. Do the following:
   a. Show that Queue is not threadsafe by setting up a multithreaded environment in which you create a race condition.
   b. Modify Queue so that it is threadsafe, and verify in your test environment that you have been successful.