Lab 10

1. *TDD Practice.* Try developing code in a simple case using the TDD approach. For this problem, create a class TDDPractice and a method changeLastCharToUpper(List<String> words). The method should change the last letter of each String in the input list to upper case. At first, write a method that just returns an empty list. Then create a test class TestTDDPractice that has a method test (remember to annotate with @Test); the test method should perform a test to validate that your method works. When you run the test, it should fail initially (since you have not coded changeLastCharToUpper yet). Then write the code for changeLastCharToUpper, and test again.

   Write a few comments about your experience doing this exercise. Does this approach seem useful?

2. *Custom Annotations.* In this problem, you will use an expanded version of the custom annotation `@BugReport` discussed in the slides to create a small bug-reporting tool. The `@BugReport` annotation has been expanded for you to include two new elements:

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface BugReport {
    String assignedTo() default "<unassigned>";
    int severity() default 0;
    String description() default "";
    String reportedBy() default "<unnamed>";

}
```

   This annotation, together with start-up code for the reporting tool `BugReportGenerator` and a `Main` class, can be found in the package `lesson10.labs.prob2.bugreporter`. You will need to complete the code in the `BugReportGenerator`, according to the specifications below.

   Instances of the annotation have been placed at the class level in each of the classes in the package `lesson10.labs.prob2.javapackage`, in order to indicate problems that need to be fixed in each of these classes, together with names of the individuals assigned to make the bugfixes. For instance:

```java
@BugReport(assignedTo="Tom Jones", reportedBy="Corazza", description="computePerimeter incorrect")
public class Circle implements ClosedCurve {
    private double radius;
    public Circle(double radius) {
        this.radius = radius;
    }
    public double getRadius() {
        return radius;
    }
    public void setRadius(double radius) {
        this.radius = radius;
    }

    @Override
    public double computePerimeter() {
        return Math.PI * radius * radius;
    }
}
```

The method `reportGenerator` in the `BugReportGenerator` class should do the following:

(1) Form a list of all classes in the package `lesson10.labs.prob2.javapackage`
(2) For each class in the package, extract the bug report information supplied by the elements of the `@BugReport` annotation
(3) Create a report that indicates the list of bugs (with detailed information) that is assigned to each bugfixer (format shown below)
(4) Output the report to a file `bug_report.txt`.

For (1), a method `ClassFinder.find(`**`PACKAGE_TO_SCAN`**`)` has been provided for you already; it extracts a list of `Class`es from a given package; the source for this method can be found in `lesson10.labs.prob2.classfinder`; it does not need to (and should not be) modified.

When your code is complete and the `main` method of `Main` is run, the output file `bug_report.txt` should look like this:

```
Tom Jones
  reportedBy: Corazza
  classname: lesson10.labsolns.prob2.javapackage.Circle
  description: computePerimeter incorrect
  severity: 0

  reportedBy: Corazza
  classname: lesson10.labsolns.prob2.javapackage.ClosedCurve
  description:
  severity: 1

Joe Smith
  reportedBy: Corazza
  classname: lesson10.labsolns.prob2.javapackage.DataMiner
  description: Should use Logger
  severity: 1

  reportedBy: Corazza
  classname: lesson10.labsolns.prob2.javapackage.Rectangle
  description: computePerimeter incorrect
  severity: 2
```

In bugreporter.BugReportGenerator.java class, you need to implement reportGenerator(). You can forloop the class List, then from each class to check if annotation present, then get annotations. Once you get Annotation, you can get attributes of the annotation. Then print it out.

3. In the package lesson10.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 (commented) `main` method is provided. Expected output for the first call to `processList` is

   `[not, too, big, yet]`

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

In FixThis, uncomment processList(), then you'll get the compiler error. This is caused by doingNothingIfShort(). There are several ways in the class to solve this problem. You can choose the easiest one or try different ways to solve it.

4. In the package `lesson10.labs.prob6`, there is a class `GuestListPreJava8` which includes a method for extracting (in sorted order) from a list of invited guests (for a particular event) all those guests who have said they will attend the event, who are female, and who are not "illegal." The implementation has been done using pre-Java 8 techniques. Your job in this exercise is to rewrite the primary method `printListOfExpectedFemaleGuests` by creating a `Stream` pipleline and using filters and maps, as necessary. Checking whether a guest is "illegal" involves a checked exception. You will need to use techniques discussed in the lecture to handle this. All the code you need has been provided for you; you only need to write code for the method `printListOfExpectedFemaleGuests`.

   1) **Use lambda expression to rewrite** `printListOfExpectedFemaleGuests() in GuestListJava8.java class.`
   2) **When you do 1) step, you will face compiler error. You can fix this issue using the same way you did in Question 3.**

5. In the `package lesson10.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?

I think it's clear for this questions. Just follow the instruction to do what it said.

6. 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.

I think it's clear for this questions. Just follow the instruction to do what it said.

a. You can initial multiple threads to do add, remove on a queue. Then print the result see if it's as expected.

b. There are two ways to do that. Explicitly create a lock or use intrinsic lock.