575 : Exercise 5 : Re-use and Extensibility - Work in Pairs

In lectures we developed some code for a Fibonacci Sequence object using TDD. We'll use that code as the starting point for this exercise.

Look at pages 2 onwards of this spec for details of how to get the skeleton code, test it, and submit.

The spec for the Fibonacci Sequence was:

```
FibonacciSequence
- defines the first two terms to be one
- has each term equal to the sum of the previous two
- is not defined for negative terms
- can be iterated through
```

Now we're going to create a Triangular Numbers Sequence. We'll re-use the code from the Fibonacci example. Copy and paste (literally) the code and tests from the Fibonacci example into new files and edit it to match this spec:

```
TriangleNumbersSequence
- defines the first term to be one
- has each term equal to (n+1)(n+2)/2
- is not defined for negative terms
- can be iterated through
```

You should now have two separate sequences working, but you'll notice that there's quite a lot of duplication between the two classes.

1) Refactor the code in the to reduce the duplication by using a Template Method pattern.

2) Refactor the code to reduce duplication a different way, using a Strategy pattern.

The skeleton code contains two copies of the original Fibonacci code, in two different packages. Work on each of your two solutions in the relevant package so that you can compare them and submit both.

Your tests should all still work without changing the assertions, though you will probably need to change the way that your objects are constructed in the tests (at least when using the Strategy).

Questions to Consider

To what extent did you use automated refactorings to complete the task?

What are the pros and cons of each approach?

Is there any duplication in the tests (within each package)? Could you do anything about this?

Can you draw a UML class diagram of each of your solutions?

Can you calculate the amount of coupling in each case?

Getting The Skeleton Code

Get the outline from GitLab:

```
git clone https://gitlab.doc.ic.ac.uk/lab1920_spring/575_tut5_login.git
```

You should work in a pair, but only one person needs to clone the repository.

Project Structure

When you clone from GitLab, you should find a similar structure to what we had in the previous exercises.

```
├── build.gradle
├── build.sh
├── config
├── gradle
├── gradlew
├── gradlew.bat
├── settings.gradle
└── src
    ├── main
    │   └── java
    │       └── ic
    │           └── doc
    │               ├── strategy
    │               │   └── FibonacciSequence.java
    │               └── templatemethod
    │                   └── FibonacciSequence.java
    └── test
        └── java
            └── ic
                └── doc
                    ├── matchers
                    ├── strategy
                    │   └── FibonacciSequenceTest.java
                    └── templatemethod
                        └── FibonacciSequenceTest.java
```

There are two packages ic.doc.templatemethod, and ic.doc.strategy. Initially both of these contain a copy of the same code/test. Rework these files as per the instructions, putting your template method pattern implementation and strategy pattern implementation under the relevant packages. Be careful with multiple classes of the same name - check which one is being imported and used in each file.

Running the Build

Again we are using Gradle to configure the build and test process. The build.gradle file defines the build process, but you are given a handy script build.sh to run everything. So you can just type:

```
./build.sh
```

Make sure you run `./build.sh` before you push, as this runs equivalent checks to those that the autotest will run in LabTS.

Importing the code into an IDE…

As we have a Gradle build file (build.gradle), your IDE should be able to recognise the structure. But you can set up your IDE however you like.

Build Results

The build has a number of stages and checks a number of qualities of your code. All of these have to pass for the whole build to pass.

Compilation: compiles your code - obviously your code has to compile correctly.

Test Results: runs all of your tests - if any of them fail, it fails the build.

After building, you can see your test results report by opening this file in a browser:

```
<your-clone>/build/reports/tests/test/index.html
```

Test Coverage: checks how much of your application code is covered by your tests. We have set a threshold of 80% for this exercise. If your coverage is below 80%, the build will fail.

The test coverage report will be at

```
<your-clone>/build/reports/coverage/index.html
```

Checkstyle: checks that the style and formatting of your code follows the Google Java style guide. If you set up the Google style guide in your IntelliJ configuration then you can easily auto-format your code to pass these checks.

After building, you will see any checkstyle errors on the console, or can view a report by opening these files in a browser:

```
<your-clone>/build/reports/checkstyle/{main.html,test.html}
```

If Everything Passes…

You should see something at the end of the build log like:

```
BUILD SUCCESSFUL in 12s
8 actionable tasks: 8 executed
```

If you have completed the required functionality, the build passes, and you are happy with your code then you are ready to submit.

Submission

When you have finished, make sure you have pushed all your changes to GitLab (source code only, not generated files under *build*). You can use LabTS (https://teaching.doc.ic.ac.uk/labts) to test the version of your work that is on GitLab. Then you need to submit a revision to CATe by clicking the “Submit to CATe” button on LabTS.

Whoever cloned the repository originally should submit to CATe, and then add your partner as a group member. The other person should sign the submission to confirm.

Deadline

You are strongly encouraged to do the exercise during the lab session on Thursday afternoon. However it is possible to submit up until **9am (in the morning) Mon 17th Feb**. This is not intended to be a large exercise, so it should not take a lot of time.

Assessment

The markers expect that your submission passes the automated tests and checks:

- Code compiles
- Tests pass
- Test coverage check passes
- Style check passes

Make sure you have 3/3 on LabTS.

If you pass these automated checks then the markers will review the design of your code and award marks based on:

- Code meets requirements
- Effective implementation of the Template Method Pattern
- Effective implementation of the Strategy Pattern
- General code quality
- Code is largely free from duplication (within a given package)
- Bonus marks for particularly good code or design (at the marker's discretion)