SED : Exercise 6 : Creation and Dependency - Work in Pairs

In this tutorial we think about some of the different patterns we have seen for creating objects and systems of objects. For today's example we look at searching through a very large book catalogue.

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

Start off by looking at `BookSearchQueryTest` - it should pass when you run it.

Introduce Builder

In the given code, constructing a `BookSearchQuery` is a bit ugly (see the examples in `BookSearchQueryTest` where different queries are constructed - there are lots of nulls). Try to introduce another class that has responsibility for creating queries, but has a nicer API for clients. Do not change `BookSearchQuery`, but feel free to change `BookSearchQueryTest.`

Introduce Singleton

Also in the given code we make a new instance of the (huge) `BritishLibraryCatalogue` for every query. Imagine that this object takes a long time and hundreds of megabytes of memory to construct (although obviously the version we have in the exercise is actually quite small).

Take the given code for `BritishLibraryCatalogue` and change it to use a Singleton pattern to make sure that there is only ever one instance created.

Dependency Inversion

Once you have done that, think about the coupling and dependencies that exist between your objects. Perhaps we would like to re-use `BookSearchQuery` but have it work with an `ImperialCollegeLibraryCatalogue` instead. Refactor the code following the principle of dependency inversion to allow this, so that the dependency on the concrete class is broken, and we have a more flexible, re-usable design.

Testing in Isolation

Can you now change the unit test for the `BookSearchQuery` so that it is tested effectively, but without having to use a large library catalogue object. Feel free to change the way that `BookSearchQueryTest` works in order to improve things.

You should not make any changes to `QueryParser`, `Book` or `BookTest`.

Getting The Skeleton Code

Get the outline from GitLab:

```
git clone https://gitlab.doc.ic.ac.uk/lab2122_spring/sed_ex6_login.git
```

Project Structure

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

```
├── build.sh
├── config
├── build.gradle
└── src
    ├── main
    │   └── java
    │       └── ic
    │           └── doc
    │               ├── Book.java
    │               ├── BookSearchQuery.java
    │               └── catalogues
    │                   ├── BritishLibraryCatalogue.java
    │                   └── QueryParser.java
    │
    └── test
        └── java
            └── ic
                └── doc
                    ├── BookSearchQueryTest.java
                    └── BookTest.java
```

Follow the existing structure of directories and packages for code and tests, as per previous weeks.

You should not make any changes to `QueryParser`, `Book` or `BookTest`.

Running the Build

This is the same as previous weeks, you can just type `./build.sh`

Make sure you run `./build.sh` before you submit, as this is what the autotest will run.

The coverage check ignores BritishLibraryCatalogue.java - you do not need to write a unit test to cover that class.

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.

Deadline

There is a lab session timetabled for Tuesday 14:00-16:00 UK time where you can talk to us about the exercise. This is not intended to be a large exercise, so it should not take a lot of time.

The deadline for submission to CATE is **7pm UK time on Thursday 24th February**.

Only one person (whoever cloned the repository originally) needs to submit from LabTS to CATE. Once this is done, add your partner as a group member on CATE. The other person should sign the submission on CATE to confirm.

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:

- Effective implementation of builder
- Effective application of singleton and use of dependency inversion principle
- Effective testing of BookSearchQuery in isolation
- General code quality, clarity, freedom from duplication etc
- Bonus marks for particularly good code or design (at the marker's discretion)

Marks may also be deducted for poor quality code, errors, or for not meeting the requirements.