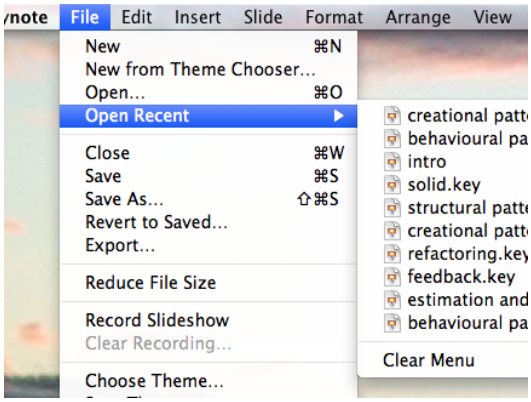
575 : Exercise 3 : Test Driven Development - Work in Pairs

Using Test-Driven Development (TDD), write some Java code to implement a RecentlyUsedList. Use JUnit in your IDE to write and run your unit tests.

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

The Recently Used List

This is a list of items such as recently dialled numbers on your phone, or recently opened files in an application on your computer.



- The list should be empty when initialised.
- We should be able to add things to the list.
- We should be able to retrieve items from the list.
- The most recent item should be first in the list.
- Items in the list are unique, so duplicate insertions should be moved rather than added.

Are there any error cases to consider? What types of object can your list hold?

Rules of Test-Driven Development

- Start by writing a test (1 test).
- Fill in enough code just to make it compile.
- Run all the tests, watch your test fail (red bar).
- Make the simplest possible change to your code that will make the test pass.
- Run all the tests and watch them pass (green bar).
- Consider refactoring.
- Commit your changes to version control.
- Repeat.

Only *add* to the implementation while you have a red bar (failing test). Refactor on green.

Questions to Consider

- Did you stick to the TDD discipline? Or did you break the rules?
- Is your code well tested?
- Did writing the tests help you to design the API for your class?
- How do you think the implementation you came up with compares with what you would have done if you weren't using TDD?
- Does the RecentlyUsedList class you've implemented implement List? Could it? Should it?

Getting The Skeleton Code

Get the outline from GitLab:

```
git clone https://gitlab.doc.ic.ac.uk/lab1819_spring/575_exercise3_login.git
```

You should work in a pair, but only one person needs to clone the repository. The best way to work is two people sitting at one computer. Work together on the code and make commits back to this repository. All the commits will be in the name of whoever is logged in, but that is ok. A good convention is to add both your names (or initials) to each commit comment.

Project Structure

When you clone from GitLab, you should get something with this structure — slightly different from previous weeks, as we use a structure that works with *Gradle* (see Running the Build below):

```
├── build.gradle
├── build.sh
├── config
├── gradle
├── gradlew
├── gradlew.bat
├── settings.gradle
└── src
    ├── main
    │   └── java
    │       └── ic
    │           └── doc
    │               └── Example.java
    └── test
        └── java
            └── ic
                └── doc
                    └── ExampleTest.java
```

There are two source folders, **src/main/java** and **src/test/java**. Inside each of these folders, is the **package ic.doc**. Write your code in this package, with tests under src/test/java, and application code under src/main/java. ExampleTest.java and Example.java are a useful reference, but you should delete these when you've finished your exercise. Do not change the directory structure.

Running the Build

We are using Gradle (https://gradle.org) to configure the build and test process. The build.gradle file defines the build process. You can look into this if you want, but you don't have to worry about the details. You have a handy script **build.sh** to run the compilation, tests and checks. You can just type:

```
./build.sh
```

The first time you do this Gradle may download a lot of things, which may take a little while. This shouldn't happen from the second time you run the build. Make sure you run `./build.sh` before you push, as this runs equivalent checks to those that the autotest will run in LabTS, and gives faster feedback.

Warnings You may see a "WARNING: An illegal reflective access operation has occurred" when you run build.sh . This (probably) isn't caused by your code, it's a problem with Groovy/Gradle, so don't worry about it. You can still get Build Successful even if you see this warning.

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.

For **IntelliJ IDEA**: you can *Import Project*, browse to cloned directory, *Import from external model,* choose Gradle, then keep clicking *Next* until finished. If no JDK is set, choose **JDK 10** (you can use an older JDK if you like, but you'll need to edit build.gradle. LabTS will run with JDK 10).

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 morning. However it is possible to submit up until **9am (in the morning) Mon 4th 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 and 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
- Evidence of following the TDD method
- Tests are well formed and expressed clearly
- Code is largely free from duplication
- Bonus marks for particularly good code or design (at the marker's discretion)