# SED : Exercise 2 : Mock Objects - Work in Pairs

In this exercise you will write code for a component of a simple digital camera. The camera has two external controls, the power switch, and the shutter button.

The central component is the `Camera`, it is responsible for responding to user actions from the camera controls, and co-ordinating the transfer of data from the sensor to the memory card when a photograph is taken. You should develop the `Camera` class in a test-driven style, adding one test at a time to `CameraTest`. Use JUnit and jMock to write your tests.

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

**The Camera class**

A `Camera` is required to have the following behaviours:

1. switching the camera on powers up the sensor

2. switching the camera off powers down the sensor

3. pressing the shutter when the power is off does nothing

4. pressing the shutter with the power on copies data from the sensor to the memory card

Writing data to the memory card can take a few seconds

5. if data is currently being written, switching the camera off does not power down the sensor

6. once writing the data has completed, then the camera powers down the sensor

**Hints and Tips**

You are provided with interfaces to the `Sensor` and `MemoryCard` components (which would be developed by other teams). You shouldn't change these.

In order to be notified when writing data is complete, the `Camera` should implement the `WriteListener` interface. If the Camera implements this interface, then other components can call it to let it know the status of writing data.

**Questions to Consider**

What is the default power state of a new Camera?
Is there any duplication in your tests? Could it be factored out?
Are your tests focussed and to the point?

Could you draw a UML Sequence Diagram of the interactions that happen here.

**JMock Reference / Examples**

Refer to the slides for examples of how to use jMock, or for more detail, see http://www.jmock.org/cookbook.html

### Getting The Skeleton Code

Get the outline from GitLab:

```
git clone https://gitlab.doc.ic.ac.uk/lab2122_autumn/sed_ex2_login.git
```

You should work in a pair, but only one person needs to clone the repository. Work together on the code and make regular commits. 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 find a similar structure to what we had in the first exercise.

```
├── build.gradle
├── build.sh
├── config
├── gradle
├── gradlew
├── gradlew.bat
├── settings.gradle
└── src
    ├── main
    │   └── java
    │       └── ic
    │           └── doc
    │               └── camera
    │                   ├── Camera.java
    │                   ├── MemoryCard.java
    │                   ├── Sensor.java
    │                   └── WriteListener.java
    └── test
        └── java
            └── ic
                └── doc
                    └── camera
                        └── CameraTest.java
```

There are two source folders, **src/main/java** and **src/test/java**. Inside each of these folders, is the **package ic.doc.camera** Write your code in this package, with tests under src/test/java, and application code under src/main/java. You already have some classes under main, including the skeleton **Camera.java**, which we are aiming to develop more fully. Under test, you should find a skeleton **CameraTest.java**, where you should write your tests.

### 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.

For **IntelliJ IDEA**: when you start IntelliJ you can click ***Open or Import*** from the first dialog box, browse to cloned directory, and click *Open*. IntelliJ should recognise this as a gradle project and configure everything appropriately. If you don't have a JDK configured, we recommend installing OpenJDK 16.

**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.

## Deadline

There is a lab session timetabled for Tuesday 16:00-18:00 UK time where you can talk to use 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 21nd October**.

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:

- Code meets requirements
- Effective use of TDD with mock objects
- 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)