# PSD3 Testing Lab – 21st January 2013

## *Aim:*

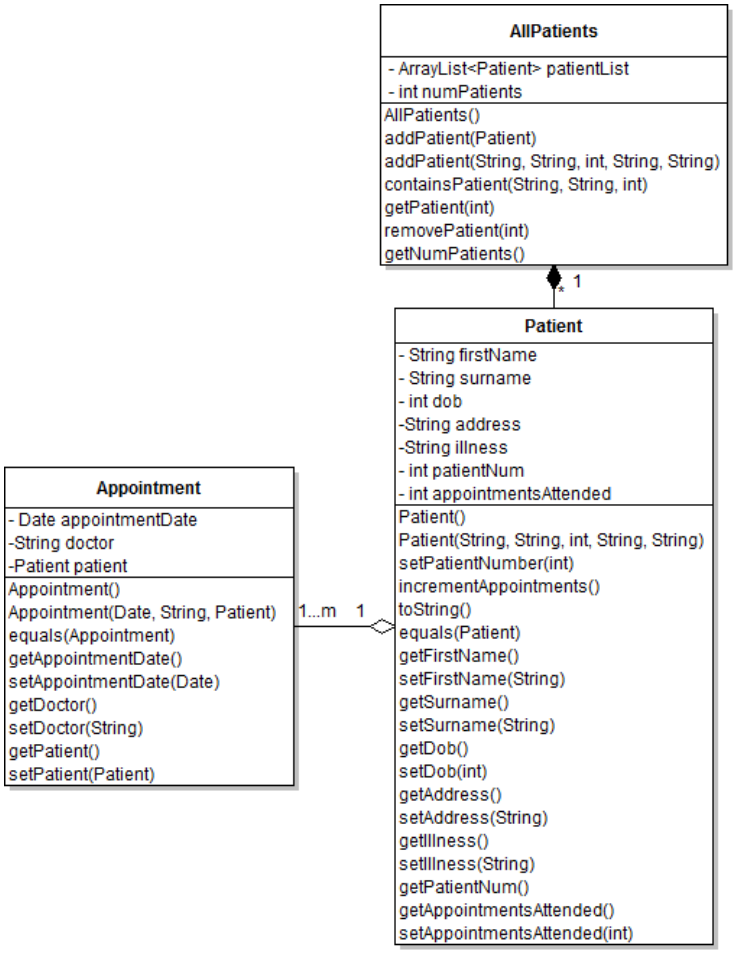
The aims of this lab are as follows:

1. To provide a simple example of constructing and executing an automated test suite in Java using the JUnit test harness framework in Eclipse.
2. To get you to construct an example of testing a non-functional requirement.
3. To use a use case description to extract a scenario and transform this into an acceptance test.

## *Scenario:*

During the course of this lab, the following scenario will be used:

You have been employed as part of a team to test a system which is being built by another team in your company which aims to manage appointments of patients in a small doctor's surgery. In the current iteration, the system needs to be able to add and remove appointments, and add and remove patients. The system built so far is represented in the class diagram below:



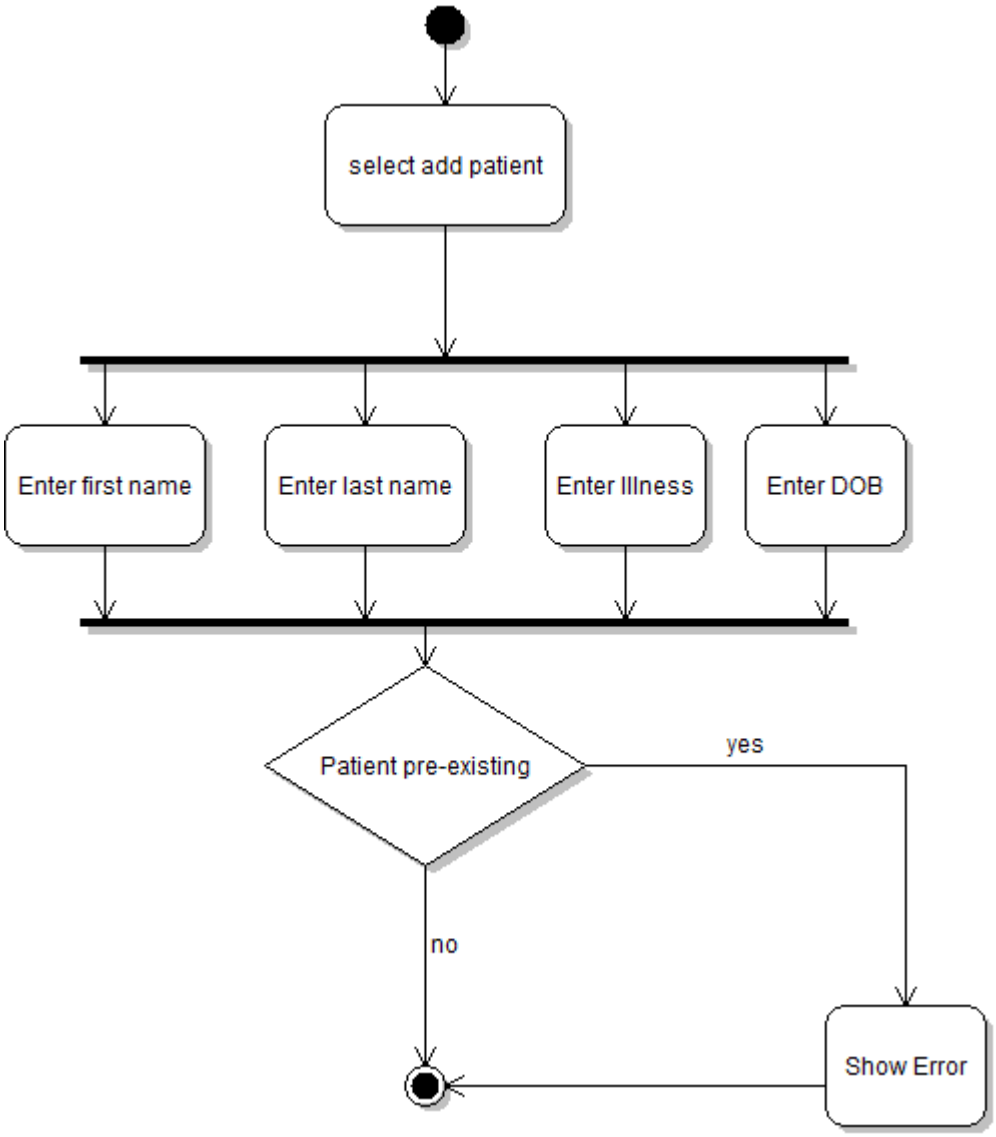
Notice that no user interface has been written yet, so any interaction with these classes is written directly by a developer. Some basic (non-extensive) unit tests have been written for the majority of the methods in these three classes. Your boss has asked you to write an **automated test suite** using JUnit for the test classes written so far, write a test for a non-functional requirement, and extend the existing test and code for the addPatient method in the AllPatients class. Further details of the steps required to complete these tasks are provided in the following section.

## *Tasks:*

1. First, download the java files from Moodle. The link is next to the lab session on the timetable and the file is called 'PatientManager.zip'. In this zip file there are 6 java files, three classes as shown in the class diagram, and corresponding test classes.
2. Set up a new Java project called "PatientManager" in your workspace in Eclipse, making sure to check 'Use Project folder as root for sources and class files' . Press Alt + F2 and search for Eclipse if you can't find it. Now Import a general file system, and import all the Java files you just downloaded. You may also have to add JUnit to the build path. This can be done as follows: Right click on your project PatientManager -> Properties, select Java Build Path, Libraries, click Add Library, click JUnit and select JUnit 4 from the drop down list.
3. Create a new class called "TestSuiteRunner.java"
4. In this class, add the following import statements before the class definition:
```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
```
5. Next, add the following code **also before the class definition**, which makes the class run as a test suite, and run all tests within the classes in the brackets after `@Suite.SuiteClasses`:
```
@RunWith(Suite.class)
@Suite.SuiteClasses(
{

})
```
There are three test classes, `PatientTest, AppointmentTest, and AllPatientsTest.` To add these to the test suite, place them in between the round brackets with .class after each, separated by commas. Now add your usual class definition i.e.
```
public class TestSuiteRunner
{

}
```
Now run the test suite. You should notice that all but one test passes. Look at the code for the test which fails.
6. The test which fails is `testTimeToAdd()` in `AllPatientsTest.java` . As the message indicates, this test fails as it hasn't yet been written. This test is an example of a non-functional test. The non-functional requirement is that addition of a patient to the patient list should take less than 1 millisecond. Write a test for this. Hint - `System.currentTimeMillis()` gives the current time in milliseconds as a long.
7. The `addPatient` method in `AllPatients.java` adds a patient to the patient list. Look at this method. Now examine the description of the 'Add Patient' use case provided as an activity diagram below.




Notice that the current `addPatient` method (called after receiving the information such as first name etc. from the UI not yet implemented) assumes the patient isn't already in the system and adds them. The corresponding test case `addPatientTest()` in the `AllPatientsTest` class also assumes that the patient doesn't exist in the system. Construct a scenario for the flow through the diagram where the patient already exists. Using your scenario, adapt the code in the `addPatientTest()` to make sure that if a patient exists then a duplicate is not added. Your test should fail initially. Adapt the code for `addPatient` in `AllPatients` to incorporate the decision. Continue running your updated `addPatientTest()` until the test passes. If the test passes, it can be used as part of your acceptance testing. Hint: there's a method `containsPatient()` in the `AllPatients` class which returns a -1 if a patient with the same first name, last name, and date of birth is in the list.