Automating *the* *right* tests ...

Java / JUnit

Objectives

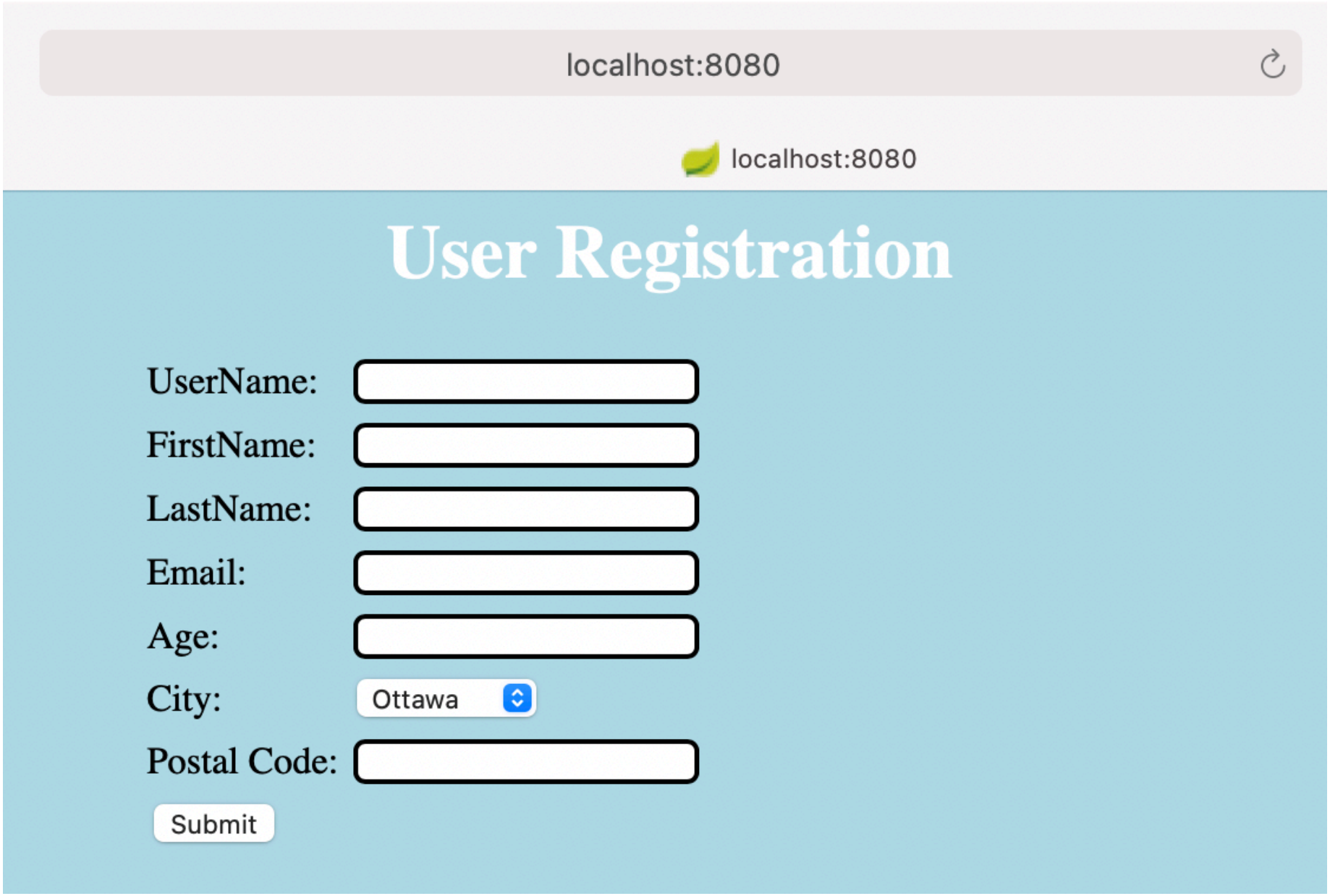
- Manually testing based on equivalence classes
- Automated test case execution into JUnit tests
  - Manually written tests (at your discretion)
  - Parameterized tests (tests based on data)
- Continue practice of Git/GitHub

Using seg3103_playground

- Move lab01 code into **/lab01** directory
- Create **/lab02** directory
  - Extract **registration.zip** and **ecs.zip**
  - Make sure you can run user-registration-app
  - Make sure you can compile / run tests for ecs code
  - Commit all code BEFORE you start making any changes

EQUIVALENCE PARTITION TESTS EXECUTION

- The jar file **user-registration-app-0.1.0.jar** bundles a prototype implementation of the registration page with a tomcat embedded web server. You can run the application from the command line:
  - **java -jar user-registration-app-0.1.0.jar**
- Then visit http://localhost:8080/


localhost:8080
localhost:8080
User Registration
UserName:
FirstName:
LastName:
Email:
Age:
City:
Ottawa
Postal Code:
Submit

Exercise 1

- Run the sample test cases created from the tutorial, and report the results in the table below with the following header:

Test Case	Expected Results	Actual Results	Verdict (Pass, Fail, Inconclusive)

- For each test case, specify the Expected Results (from the test case), the Actual Results (from the execution) and a Verdict (Pass if Actual Result and Expected Results match, Fail if Actual Result and Expected Results do not match and Inconclusive for cases where a determination can not be made).

WHAT? By Hand?

- The test cases in the previous example are run manually.
- This is evidently time consuming, error prone and inefficient (unfortunately this is also how they do in some organizations).
- Fortunately, there are tools available to automated these types of tests, some of which will be introduced in this course (and others you can discover on your own)

JUNIT PARAMETERIZED RUNNER

- The **ecs** project contains a class **Bit.java** and two test classes.
  - BitTest.java - explicit JUnit tests
  - BitAndTest.java - parameterized **Bit.and** tests
- Note that the BitTest.java runs on JUnit 5 and BitAndTest.java runs on JUnit 4
- Make sure you can compile and run the tests

```
Thanks for using JUnit! Support its development at https://junit.org/sponsoring

╷
├─ JUnit Jupiter ✔
│  ├─ DateTest ✔
│  │  └─ nextDate_sample() ✔
│  └─ BitTest ✔
│     ├─ constructor_int_ok() ✔
│     ├─ constructor_int_tooLarge() ✔
│     ├─ constructor_int_tooSmall() ✔
│     ├─ constructor_Bit() ✔
│     ├─ hashCode_values() ✔
│     ├─ getIntValue() ✔
│     ├─ equals() ✔
│     ├─ toString_values() ✔
│     ├─ or() ✔
│     ├─ and() ✔
│     ├─ not() ✔
│     ├─ xor() ✔
│     ├─ setValue() ✔
│     └─ constructor_default_0() ✔
└─ JUnit Vintage ✔
   └─ BitAndTest ✔
      ├─ [0] ✔
      │  └─ testAnd[0] ✔
      ├─ [1] ✔
      │  └─ testAnd[1] ✔
      ├─ [2] ✔
      │  └─ testAnd[2] ✔
      └─ [3] ✔
         └─ testAnd[3] ✔
```

DATE NEXTDATE METHOD

- The class Date (Data.java) provides a basic implementation of a data structure for dates. Method nextDate returns an instance of Date corresponding to the date of the day after the executing instance
- The implementation must ensure different constraints for dates are respected such as (see next slide)

A VALID DATE ...

- A valid date is a triple (year, month, day) with
  - year>=0
  - 1<= month <= 12
  - 1 <= day <= 31
- The month of February counts 28 days on a non-leap year and 29 days on a leap year.
  - A year is a leap year if it is divisible by 4, unless it is a century year.
  - Century years are leap years only if they are multiples of 400. So 2000 is a leap year while the year 1900 is not a leap year.

TC	Input (y m d)	Expected Output (y m d)
1	1700 06 20	1700 06 21
2	2005 04 15	2005 04 16
3	1901 07 20	1901 07 21
4	3456 03 27	3456 03 28
5	1500 02 17	1500 02 18
6	1700 06 29	1700 06 30
7	1800 11 29	1800 11 30
8	3453 01 29	3453 01 30
9	444 02 29	444 03 01
10	2005 04 30	2005 05 01
11	3453 01 30	3453 01 31
12	3456 03 30	3456 03 31
13	1901 07 31	1901 08 01
14	3453 01 31	3453 02 01
15	3456 12 31	3457 01 01
16	1500 02 31	IllegalArgumentException
17	1500 02 29	IllegalArgumentException
18	-1 10 20	IllegalArgumentException
19	1458 15 12	IllegalArgumentException
20	1975 6 -50	IllegalArgumentException

Test cases for nextDate

The following table lists a test suite consisting of a set of test cases defined for method nextDate

EXERCISE 2

- Implement ***explicit*** tests using JUnit 5
  - **DateTest.java**
- Implement ***Parameterized*** tests using JUnit 4 or 5. You will need to create two Parameterized test suites:
  - ***DateNextDateOkTest.java*** for test cases that run OK and return a date
  - **DateNextDateExceptionTest.java** for test cases that DO result in an exception

Submission

- All work should written under
  - **seg3103_playground/lab02**
- Create **README.md** to summarize your work
  - Summarize your solution here (make it *easy* to grade)
  - Use markdown tables for tabular data, for example in Exercise 1
  - Embed screenshots where appropriate, for example ![description](assets/tc01.png)
- Write Java / JUnit code for Exercise 2
  - DateTest.java
  - DateNextDateExceptionTest.java
  - DateNextDateOkTest.java
- Share your repository with the teacher and TA (s)
  - Submissions to BrightSpace should clearly reference your GitHub repository