Automatiser *les bons tests* ...

Java / JUnit

Objectifs

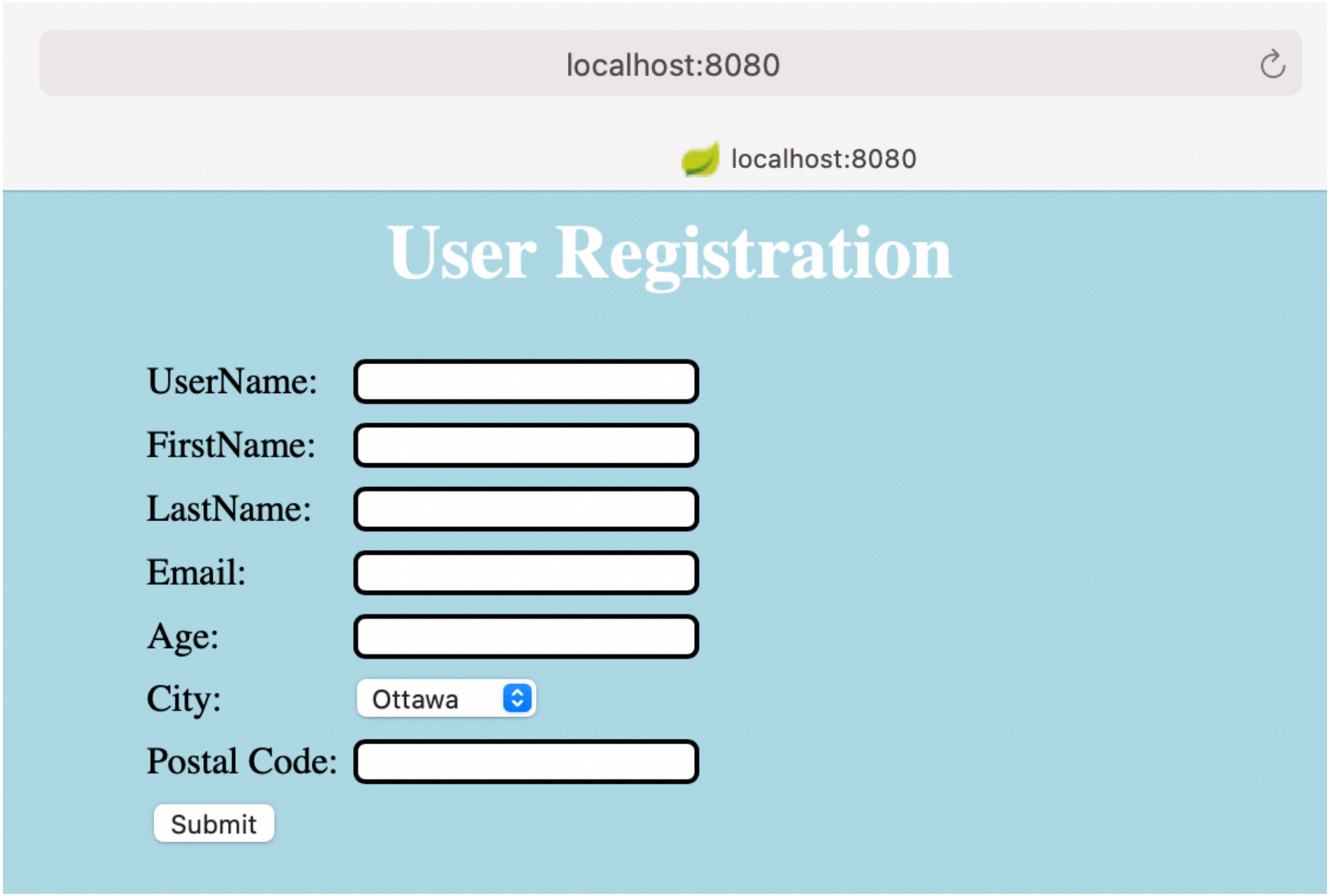
- Test manuel basé sur ;es classes d'équivalence
- Exécution automatisée des cas de test dans les tests JUnit
  - Tests écrits *manuellement* (à votre discrétion)
  - Tests paramétrer (tests basés sur des données)
- Continuer la pratique de Git / GitHub

Avec seg3103_playground

- Déplacez le code lab01 dans le répertoire **/lab01**
- Créer le répertoire **/lab02**
  - Unzip **registration.zip** et **ecs.zip**
  - Exécuter l'app user-registration-app
  - Compiler / exécuter des tests pour le code ecs
- Validez votre environment et commit le code AVANT de appliquer vos modifications

Classes d'équivalences exécution de tests

- Le fichier jar **user-registration-app-0.1.0.jar** regroupe un prototype d'implémentation de la page d'enregistrement avec un serveur web embarqué Tomcat. Vous pouvez exécuter l'application à partir de la ligne de commande:
  - **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

Exercice 1

- Exécutez les exemples de tests créés dans la session du tutoriel et rapportez les résultats dans une table ayant l'en-tête suivante:

Cas de Test	Résultats Escomptés	Résultats Actuels	Verdict (Succès, Échec, Non-concluant)

- Pour chaque cas de test, spécifiez les résultats attendus (de la spécification du cas de test), les résultats réels (de l'exécution) et un verdict (Succès si le résultat réel et les résultats attendus correspondent, Échec si le résultat réel et les résultats attendus ne concordent pas et Non-concluant au cas où une détermination ne peut être faite).

JUnit Parameterized Runner

- Le projet **ecs** contient une classe **Bit.java** et deux suite de tests
  - BitTest.java - tests JUnit explicites
  - BitAndTest.java - suite de tests paramétrer pour la méthode Bit and
- Notez que BitTest.java s'exécute sur JUnit 5 et BitAndTest.java s'exécute sur JUnit 4.
- Assurez-vous que vous pouvez compiler et exécuter les 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] ✔
```

Méthode Date nextDate

- La classe Date (Date.java) fournit une implémentation de base d'une structure de données pour des dates. La méthode nextDate retourne une instance de Date correspondant à la date du lendemain de l'instance exécutante.
- L'implémentation doit veiller à ce que différentes contraintes pour les dates, telles que les suivantes, soient respectées (voir la diapositive suivante)

Une date valide ...

- Une date valide est un triplé (year, month, day) avec
  - year>=0
  - 1<= month <= 12
  - 1 <= day <= 31
- Le mois de Février compte 28 jours pour une année non bissextile et 29 jours pour une année bissextile.
  - Une année est une année bissextile si elle est divisible par 4, sauf s'il s'agit d'une année siècle.
  - Les années correspondant à des changement de siècle sont des années bissextiles que si elles sont multiples de 400. Donc, 2000 est une année bissextile tandis que l'année 1900 n'est pas une année bissextile.

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

Cas de test pour nextDate

Le tableau suivant énumère une suite de tests composée d'un ensemble de cas de test définis pour la méthode nextDate.

Exercice 2

- Donnez une implémentation explicite en Junit 5 de ces tests sans le runner Parameterized (**DateTest.java**)
- Fournissez une implémentation Junit de ces tests avec le runner Parameterized, vous devrez créer deux suites de tests paramétrer:
  - **DateNextDateOkTest.java** pour les cas de test ne générant pas d'exception
  - **DateNextDateExceptionTest.java** pour les cas de test générant une exception

Soumission

- Tous les travaux doivent être dans
  - **seg3103_playground/lab02**
- Créez **README.md** pour résumer votre travail
  - Résumez votre solution (faciliter le marquage
  - Utilisez des tableaux markdown, par exemple dans l'exercice 1
  - Intégrez des screenshots, par exemple! [Description] (assets / tc01.png)
- Écrire du code Java / JUnit pour l'exercice 2
  - DateTest.java
  - DateNextDateExceptionTest.java
  - DateNextDateOkTest.java
- Partagez votre répo avec l'enseignant et les AEs
  - Les soumissions à BrightSpace doivent clairement référencer votre référentiel GitHub