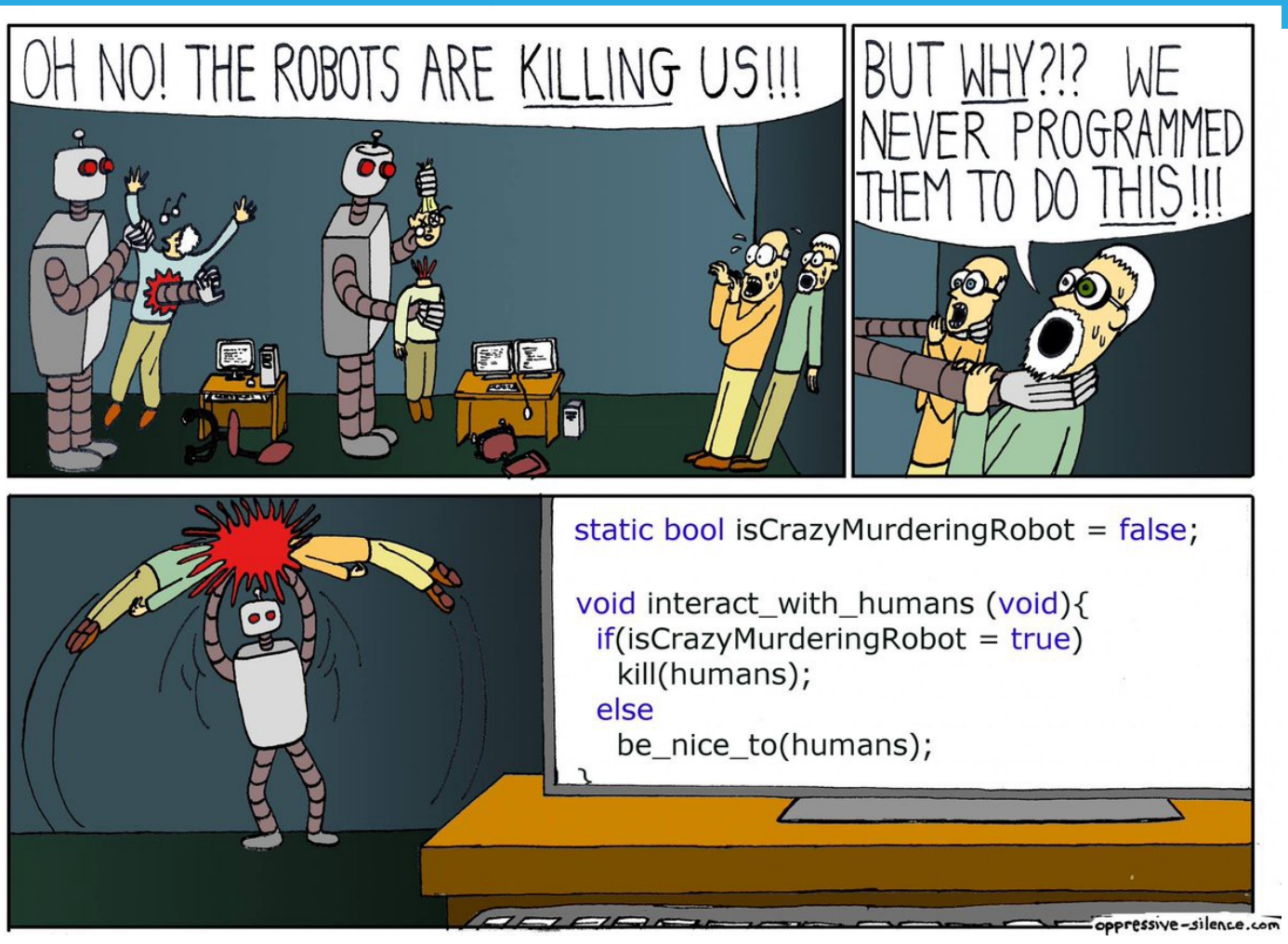
Cours de test et vérification

E.I.S.T.I. ING2 GSI/SIE

Introduction au test unitaire - JUnit


OH NO! THE ROBOTS ARE KILLING US!!!
BUT WHY?!? WE NEVER PROGRAMMED THEM TO DO THIS!!!
static bool isCrazyMurderingRobot = false;
void interact_with_humans (void){
if(isCrazyMurderingRobot = true)
kill(humans);
else
be_nice_to(humans);
}
oppressive-silence.com

Sommaire

1. Présentation
2. Implémenation
3. Assertions / Assomptions
4. Exceptions
5. Compilation / Exécution
6. Exemple
7. Fonctionnalités avancées

Test unitaires avec JUnit

Présentation de JUnit

- Framework d'automatisation des test unitaires;
- Langage JAVA;
- Inventé par Kent Beck et Erich Gamma;
- Version actuelle **JUnit5** (utilisation avec Java 8 et plus);
- Version **JUnit4** maintenue pour Java 5 et plus (avec les annotations), Junit5 : junit.vintage;
- Version JUnit 3.8 maintenue pour Java 4 et moins (pas d'annotations);
- Utilisable en ligne de commande via une archive jar;
- Intégrable dans les IDE (Eclipse, Intellij, NetBeans, VS Code...) via des plugins.

Test unitaires avec JUnit5

Test unitaires avec JUnit5

Méthodologie des test unitaires

1. Préparation (SetUp) de l'environnement des test (préconditions):
    - initialisations de valeurs, d'objets,
    - activation des logs,
    - ...
2. Execution des cas de tests (CT) prévus lors de l'écriture du programme;
3. Evaluation des résultats ou effets de bords engendrés par l'exécution des CT;
4. Nettoyage (TearDown) de l'environnement si besoin (post-conditions).

Test unitaires avec JUnit5

Implémentation des test unitaires avec JUnit5

L'implémentation se fait au sein d'une classe de test séparée du programme à tester. Les méthodes de cette classe sont décorées par des annotations Java:

- SetUp: annoter les méthodes avec `@BeforeAll`, `@BeforeEach`;
- Tests:
  - annoter les méthodes avec `@Test`,
  - utiliser à l'intérieur des méthodes les assertions,
  - `assertEquals(ValeurAttendue, ValeurCalculee, "texte si failure");`
- Tear Down: annoter les méthodes avec `@AfterAll`, `@AfterEach`.

Test unitaires avec JUnit5

Assertions

Elles permettent de définir les conditions de validité du test (égalité, différence, nullité, valeur booléenne, etc.).

Assomptions

Elles permettent de définir les critères de validité du contexte du test et empécher son exécution s'ils ne sont pas respectés.

Test unitaires avec JUnit5

Gestions des exceptions

- Assertion `assertThrows(Exception.class, fonctionTest);`
- Par exemple :

```
@Test
public void testPlante() {
System.out.println("ça va planter");
int res;
Assertions.assertThrows(java.lang.ArithmeticException.class,
                        () -> {    res = 5 / 0; });
}
```

Test unitaires avec JUnit5

Importation des classes

- Afin de compiler, votre classe de test doit importer les classes JUnit qu'elle utilise;
- Par exemple:

```
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.Test;
```

- Utilisation de la librairie `junit-platform-console-standalone`. Ne pas oublier d'inclure le `.jar` à votre classpath:

```
javac -d out MaClasse.java
javac -d out -cp junit-platform-console-standalone-x.x.x.jar:out MaClasseTest.java
```

Test unitaires avec JUnit5

Exécution des tests et récupération des résultats

- Utilisation de la librairie `junit-platform-console-standalone`
  - exécute l'ensemble des tests;
  - récolte les résultats.
- En ligne de commande:

```
java -jar junit-platform-console-standalone-x.x.x.jar  <options>
```

- Quelques Options marquantes
  - -cp : dossier des classes, jar externes,
  - --scan-classpath : cherche la classe de test dans le classpath,
  - -c : spécifier une classe de test,
  - -t/T : inclure/exclure un Tag,
  - ...
- Par défaut, les résultats sont affichés dans la console.

Test unitaires avec JUnit5

Exemple: division (implémentation)

- Fichier MyClass.java

```
public class MyClass {
    public int divide(int x, int y) {
        return x / y;
    }
}
```

- Fichier MyClassTest.java

```
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.DisplayName;

public class MyClassTest {
    @Test
    @DisplayName("Mon cas de test identité 10/1=10")
    public void testDivide() {
        MyClass tester = new MyClass();
        Assertions.assertEquals(10, tester.divide(10, 1), "10/1=1 FAILED");
    }
}
```

Test unitaires avec JUnit5

Exemple: division (exécution)

- Compilation:

```
javac -d out MyClass.java
javac -d out -cp junit-platform-console-standalone-x.x.x.jar:out MyClassTest.java
```

- Exécution:

```
java -jar junit-platform-console-standalone-x.x.x.jar -cp out --scan-classpath
```

- Résultat console:

```
|
├─ JUnit Jupiter ✔
│  └─ TestDiv ✔
│     └─ Mon cas de test identité 10/1=10 ✔
└─ JUnit Vintage ✔
...
```

Test unitaires avec JUnit5

Exemple: division (exécution erreur)

- Fichier MyClass.java

```
@Test
@DisplayName("Mon cas plante")
public void testBindon() {
    Assertions.assertEquals(2, testeur.divide(2, 5), "2/5=2 FAILED");
}
```

- Résultat d'exécution:

```
├─ JUnit Jupiter ✔
│  └─ TestDiv ✔
│     └─ Mon cas plante ✘ 2/5=2 FAILED ==> expected: <2> but was: <0>
└─ JUnit Vintage ✔

Failures (1):
  JUnit Jupiter:TestDiv:Mon cas plante
    MethodSource [className = 'TestDiv', methodName = 'testBindon', methodParamete
    => org.opentest4j.AssertionFailedError: 2/5=2 FAILED ==> expected: <2> but was
       [...]
```

Test unitaires avec JUnit5

Autres fonctionnalités simples

- Désactivation : `@Disabled`;
- Ordre : `@Order(nb)`(Annotation de méthode), `@TestMethodOrder(MethodOrderer.OrderAnnotation.class)` (Annotation de classe);
- Nommage : `@DisplayName("Nom")`;
- Assertions multiples : `assertAll`.

```
public void testPlusieurs() {
    Assertions.assertAll("Divisions classiques",
        () -> Assertions.assertTrue(testeur.divide(10, 5) == 2, "10/5=2 FAILED"),
        () -> Assertions.assertEquals(testeur.divide(10, 3), 3, "10/3=3 FAILED"));
}
```

À vous de jouer !