Tests

Programmation Orientée Objet

Licence mention Informatique
Université Lille – FST - Informatique

FACULTÉ DES SCIENCES ET TECHNOLOGIES
Département Informatique

Université de Lille

Tests

Règle

Un code non testé n'a aucune valeur.

Corollaire

Tout code doit être testé

- **test unitaire**
  Tester les différentes parties d'un programme indépendamment les unes des autres.
- **test de non régression**
  Vérifier que le nouveau code ajouté ne corrompt pas les codes précédents : les tests précédemment réussis doivent encore l'être.

Mise en œuvre

- utilisation du framework `JUnit`.
- s'appuie sur des **assertions**
- voir documents du TP 5 + sur portail onglet « Documents ».

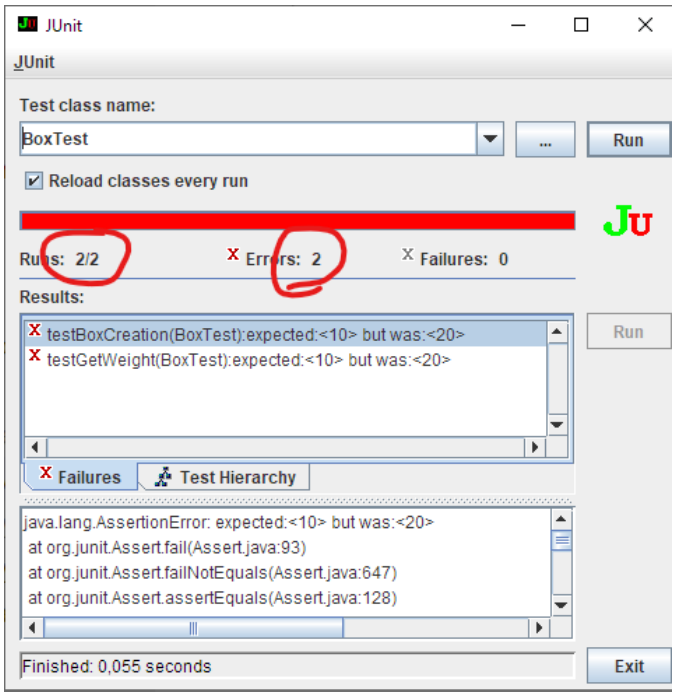
```
package robot;
public class Box {
  /** ...  */
  public Box(int weight) {
    this.weight = weight;
  }
  /** weight of the box */
  private int weight;
  /** @return this box's weight */
  public int getWeight() {
    return this.weight;
  }
}
```

```
package robot;
public class BoxTest {

  @Test
  public void testCreationIsOk() {
    Box someBox = new Box(10);
    assertEquals(10,someBox.getWeight());
  }
  ...
}
```

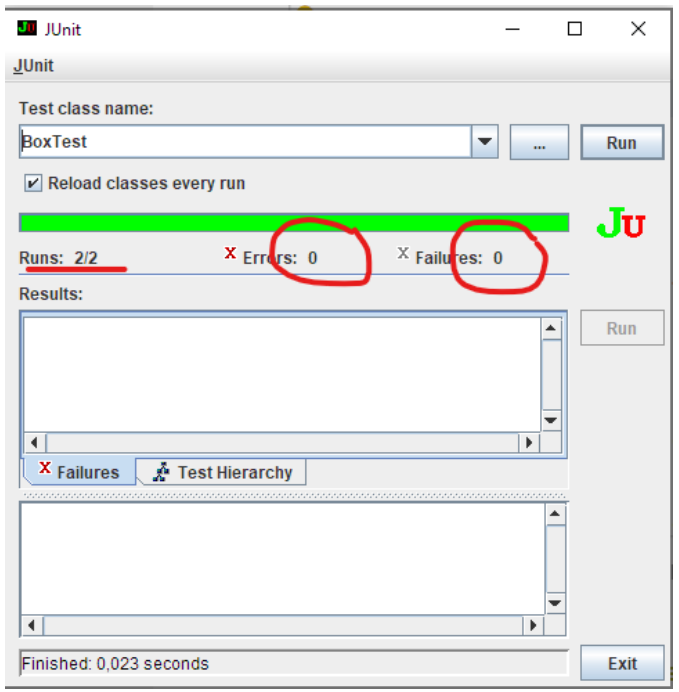
« get the green bar »

```
javac -classpath test4poo.jar test/robot/BoxTest.java

java -jar test4poo.jar robot.BoxTest
```

tests en échec



tests passés avec succès



`test4poo.jar` outil spécifique cours POO

méthodes de test

- préfixée de l'annotation `@Test`
- signature de la forme « `public void testMethod()` »
- le corps de la méthode contient des **assertions**

  `assertTrue`, `assertEquals`, etc.

  le test est réussi si toutes les assertions sont vérifiées
- plusieurs méthodes de tests peuvent être nécessaires pour tester la correction d'une méthode

- principe
  1. créer la situation initiale et vérifier les « préconditions »
  2. appeler la méthode testée
  3. à l'aide d'assertions, vérifier les « postconditions » = situation attendue après l'exécution de la méthode

Un robot peut porter une caisse d'un poids maximal défini à la construction du robot. Initialement un robot ne porte pas de caisse. S'il porte déjà une caisse il ne peut en prendre une autre.

```
import ...;
public class RobotTest {
  @Test
  public void NotCarryingABoxWhenCreated() {
     Robot robbie = new Robot(15);
     // aucune caisse portée ?
     assertFalse(robbie.isCarryingABox());
  }
}
```

Robot
...
+ Robot(int) + isCarryingABox() : boolean + takeBox(b : Box) + getCarriedBox() : Box

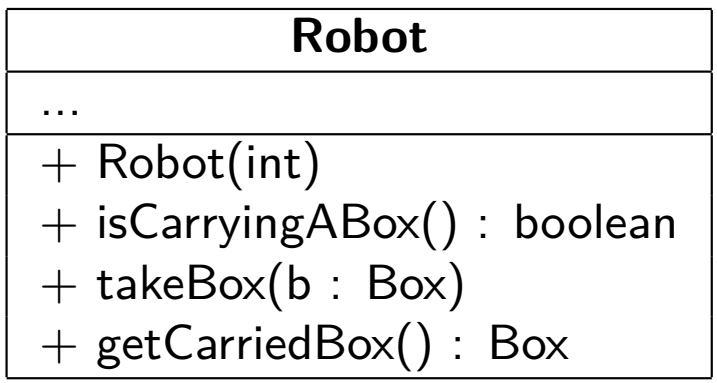
Un robot peut porter une caisse d'un poids maximal défini à la construction du robot. Initialement un robot ne porte pas de caisse. S'il porte déjà une caisse il ne peut en prendre une autre.

```
import ...;
public class RobotTest {
  ...
  @Test
  public void robotCanTakeLightBox() {
     // situation initiale : un robot et une caisse
     Robot robbie = new Robot(15);
     Box b = new Box(10);
     // précondition : robot ne porte rien
     assertFalse(robbie.isCarryingABox());
     // exécution de la méthode testée
     robbie.takeBox(b);
     // postcondition : la caisse portée est bien b
     assertSame(b, robbie.getCarriedBox());
  }
 }
```

Robot
...
+ Robot(int) + isCarryingABox() : boolean + takeBox(b : Box) + getCarriedBox() : Box

Un robot peut porter une caisse d'un poids maximal défini à la construction du robot. Initialement un robot ne porte pas de caisse. S'il porte déjà une caisse il ne peut en prendre une autre.

```
import ...;
public class RobotTest {
  ...
  @Test
  public void robotCannotTakeTooHeavyBox() {
     Robot robbie = new Robot(15);
     Box b = new Box(20);
     assertFalse(robbie.isCarryingABox());
     // exécution de la méthode testée
     robbie.takeBox(b);
     // toujours aucune caisse portée
     assertFalse(robbie.isCarryingABox());
  }
 }
```



Un robot peut porter une caisse d'un poids maximal défini à la construction du robot. Initialement un robot ne porte pas de caisse. S'il porte déjà une caisse il ne peut en prendre une autre.

```
import ...;
public class RobotTest {
  ...
  @Test
  public void robotCanTakeOnlyOneBox() {
    // situation initiale : robot portant une caisse
    Robot robbie = new Robot(15);
    Box b1 = new Box(10);
    robbie.takeBox(b1);
    // précondition : b1 est bien la caisse portée
    assertSame(b1, robbie.getCarriedBox());
    Box b2 = new Box(5);
    // exécution de la méthode testée
    robbie.takeBox(b2);
    // postcondition : la caisse portée est toujours b1
    assertSame(b1, robbie.getCarriedBox());
  }
}
```

Robot
...
+ Robot(int) + isCarryingABox() : boolean + takeBox(b : Box) + getCarriedBox() : Box

Méthodologie

Travailler une méthode à la fois :

1. définir la signature de la méthode,
2. écrire la javadoc de la méthode,
3. écrire les tests qui permettront de contrôler que le code écrit pour la méthode est correct = répond aucahire des charges
4. coder la méthode,
5. exécuter les tests définis à l'étape 3, en vérifiant la non régression,
6. si les tests sont réussis passer à la méthode suivante (étape 1) sinon recommencer à l'étape 4.

Il ne s'agit pas de travailler plus, mais d'être plus efficace.