IUT Nantes
Pôle Sciences et technologie

Nantes Université

R2.03 - Qualité de développement 1
Automatisation des tests

CM3 – Test Unitaire

Jean-Marie Mottu (Lanoix, Le Traon, Baudry, Sunye)

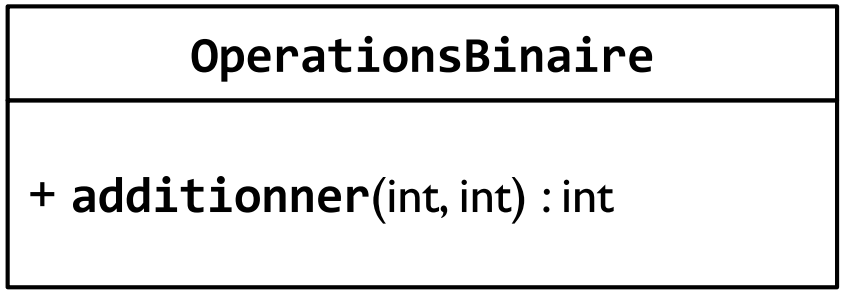
Cas des test

- Tester c'est Construire/Programmer/Exécuter un ensemble de
  - Cas de test
    - Une description : *"vérifier le passage à l'heure d'hiver"*
    - Une initialisation : *on doit instantier l'horloge du péage*
      - Par exemple créer un objet, le mettre dans un état précis
    - Une donnée de test : *+1 minute le* 31 oct. 2017 *à 02h59*
      - Par exemple certains paramètres de la méthode à tester

        (on détermine ces paramètres avec les techniques de prochains cours)
    - Un oracle : *il doit être 02h00 (et pas 03h00)*
      - Contrôler que l'exécution de la donnée de test respecte la specification

Test unitaire : Test intensif des unités de test

- La première étape de test après la programmation est le test unitaire
  - Tester une unité isolée du reste du système
- En Prog Objet, l'unité est la classe
  - Test unitaire = test de l'unité classe
  - Classe Sous Test
- On considère les classes indépendamment les unes des autres.




```
public class OperationsBinaire {//java
  /**
   * additionner deux entiers
   */
  public int additionner(int x, int y)
  {
                  return x + y;
  }
}
```

Test Unitaire != Mise au point

- Trop long de taper en console et de contrôler les résultats pour de multiples tests :

```
public class OperationsBinaire {//java

/**
 * additionner deux int
 */
public int additionner(int x, int y) {
        return x + y;
}

public static void main(String[] args) {
        OperationsBinaire op = new OperationsBinaire();
        int sum = op.additionner(Integer.valueOf(args[0]),
                                 Integer.valueOf(args[1]));
        System.out.println(args[0]+"+"+args[1]+" rend "+ sum);
}
}
```

Test Unitaire != Mise au point

- Un peu mieux :

```
public class OperationsBinaire {//java

/**
 * additionner deux int
 */
public int additionner(int x, int y) {
        return x + y;
}

public static void main(String[] args) {
        OperationsBinaire op = new OperationsBinaire();
        int sum = op.additionner(1, 2);
        System.out.println("1+2 censé rendre 3 rend "+ sum);
        sum = op.additionner(2, 3);
        System.out.println("2+3 censé rendre 5 rend "+ sum);

}
}
```

Test Unitaire != Mise au point

- On s'approche :

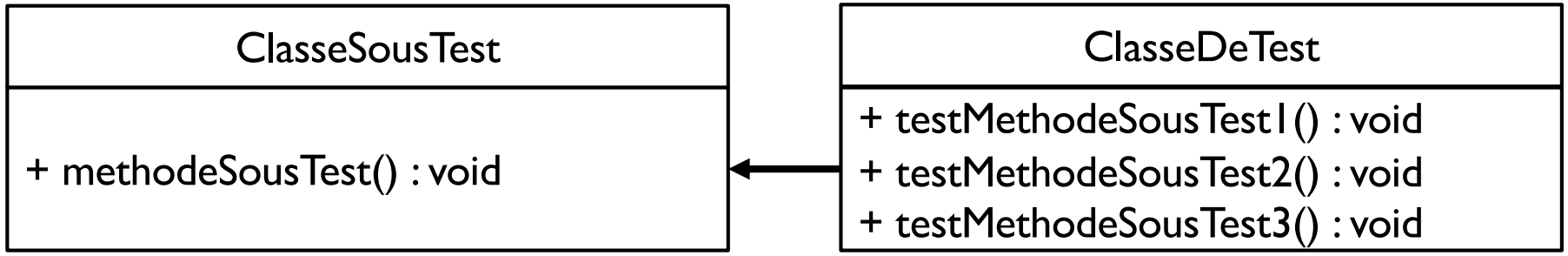
```
public class OperationsBinaire {//java

/**
 * additionner deux entiers
 */
public int additionner(int x, int y) {
        return x + y;
}

public static void main(String[] args) {
        OperationsBinaire op = new OperationsBinaire();
        int sum = op.additionner(1, 2);
        System.out.println("test de 1+2 donne "+ sum +
                                « ce qui est " + (sum == 3));
        sum = op.additionner(2, 3);
        System.out.println ("test de 2+3 est "+ sum +
                                « ce qui est " + (sum == 5));
```

Test Unitaire : Classes de test

- Consacrer des classes et des méthodes pour lancer des cas de test qui contrôlent le comportement du programme

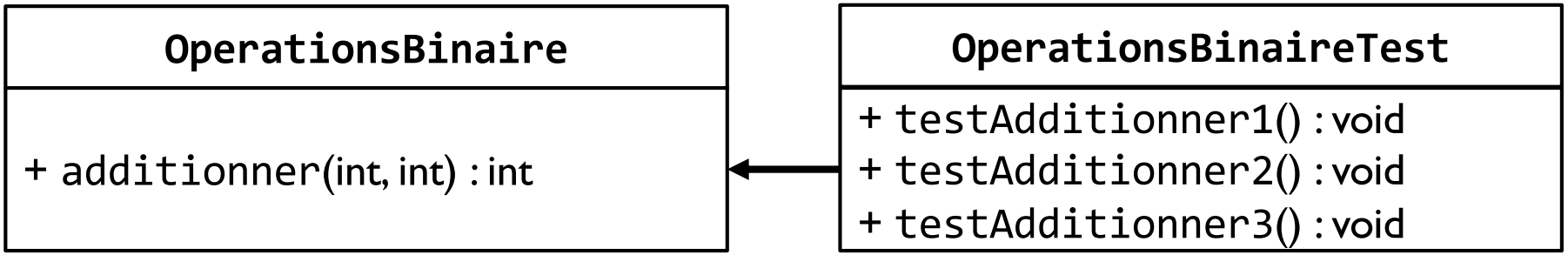



```
class OperationsBinaireTest {//java
	OperationsBinaire op = new OperationsBinaire();

	@Test
	void testAdditionner1() {
		int result = op.additionner(1, 2);
		assertEquals(3, result, "Somme de 1+2 devrait être 3");
}
```

Test Unitaire : Classes de test

- Par exemple




```java
public class OperationsBinaire {//java

  /**
   * additionner deux entiers
   */
  public int additionner(int x, int y)
  {
                return x + y;
  }
}
```

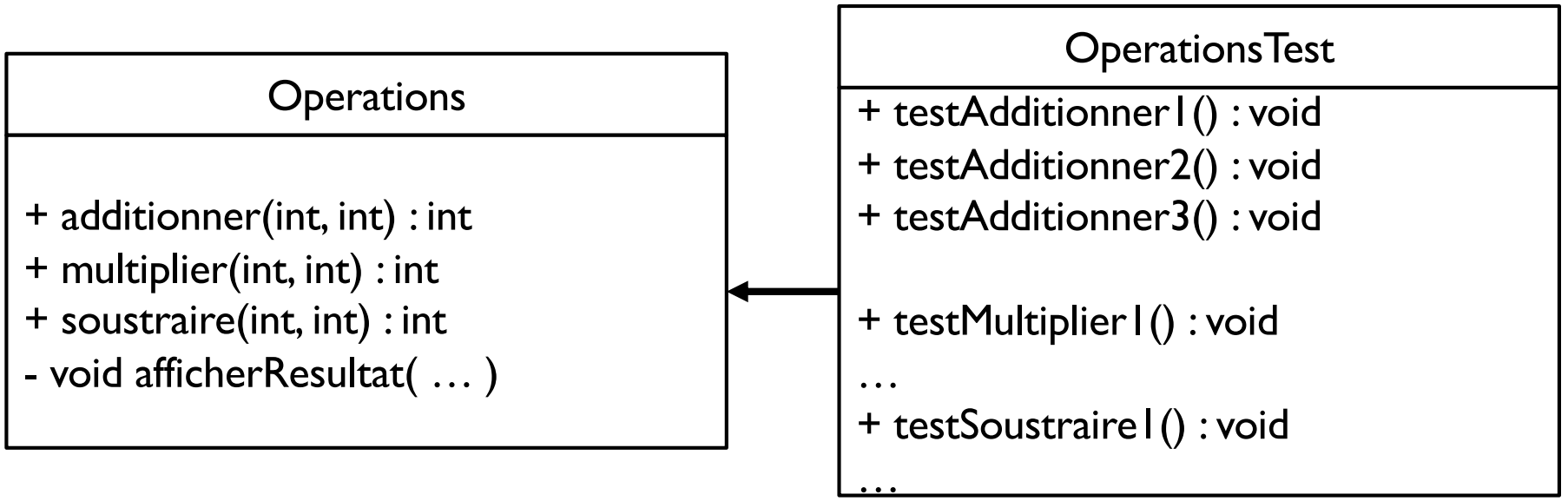
```java
class OperationsBinaireTest {//java
  OperationsBinaire op =
        new OperationsBinaire();

  @Test
  void testAdditionner1() {
    int result = op.additionner(1, 2);
    assertEquals(3, result,
       "Somme de 1+2 devrait être 3");
}
```

Méthodes de test

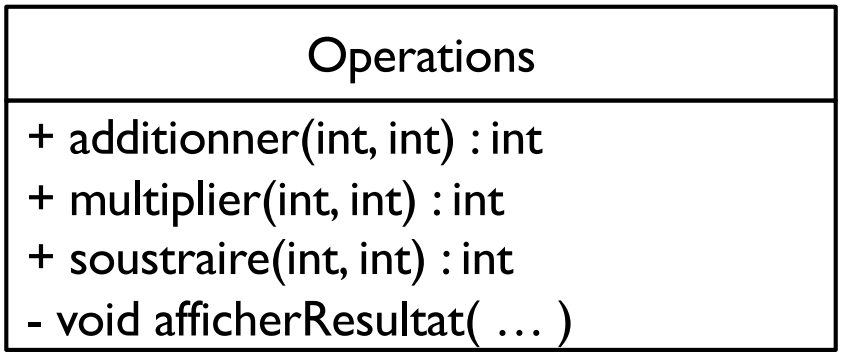
- Une méthode de test implémente un cas de test
  - Variante : une méthode de test peut éventuellement être paramétré(e) par des séries de données, implémentant une suite de cas de test (cf dernier partie de ce CM)
- Classe de test = Suite de tests indépendants
- Au moins une classe de test par classe testée
  - Regroupe les cas de test
  - Il peut y avoir plusieurs classes de test pour une classe testée

Exemple : test de la classe Operations



- Créer une classe de test qui manipule des instances de la classe Operations
- Au moins 3 cas de test (1 par méthode publique)
- Pas d'accès direct à la méthode afficherResultat, il faudra l'atteindre par l'intermédiaire d'autres méthodes

Exemple d'un cas de test : Addition de deux entiers



spécification du cas de test

```
//Premier test de l'addition de deux nombres
//L'addition de deux valeurs nulles donne nul
@Test
public void testAddition1(){//java
```

initialisation

```
    Operations op = new Operations();
```

appel avec donnée de test

```
    int res = op.additionner(0,0);
```

oracle

```
    assertEquals(0 , res, "Somme de nuls");
}
```

Test du point de vue client

- Le test d'une classe se fait à partir de classes extérieures du même package
- Les tests s'exécutent indépendamment les uns des autres
  - A la fin tout doit fonctionner
  - Néanmoins tant qu'il y a des fautes il est plus efficace d'ordonner la création des tests
    - quelles méthodes sont (inter)dépendantes ?
- Difficulté pour l'oracle
  - L'oracle prédit le résultat
  - Encapsulation : les attributs sont souvent privés
  - Difficile de récupérer l'état d'un objet
  - Penser à la testabilité au moment de la conception :
    - prévoir des accesseurs en lecture sur les attributs privés
    - des méthodes pour accéder à l'état de l'objet