

TESTS DES COMPOSANTS

Tests unitaires et tests d'intégration

2

Conception des tests

Tester une application

3

- Des erreurs répétitives, inexpliquées, récurrentes... peuvent provoquer la fin de l'utilisation d'une application.
- Les développeurs doivent s'assurer du bon fonctionnement de leur application

Conception d'une application

4

- Il faut prendre en compte le fonctionnement normal
- Il faut prévoir les cas d'erreurs et les gérer
 - message d'erreur pour l'utilisateur,
 - écriture dans un fichier de log...

Tester une application

5

- Pour vérifier que l'application se comporte comme cela a été prévu dans la phase de conception
- Ne se fait pas au hasard
- Optimiser les tests
 - ▣ Éviter des tests inutiles (perte de temps)
 - ▣ Pas d'oublis de tests

Plan de tests

6

- Définit la liste des tests à réaliser pour valider l'application

No test	Type scénario	Description	Résultat du test le
	Normal / Exception		
Résultats attendus			

Conception des tests

7

- En même temps que la conception des classes, méthodes...
- Test fonctionnement normal
- Test des exceptions
- Automatisation des tests : tests unitaires

Exemple : classe StockMed

8

- Conception des tests unitaires de la classe StockMed

```
//ajout d'une quantité de médicament au stock
public void ajoutStock(int qte)
{...3 lines }

//suppression de médicaments du stock
public int supprimeStock(int qte)
{...10 lines }

//calcul du cout du stock
public double prixStock()
{...3 lines }
```

Test de la méthode ajoutStock

9

No test	Type scénario	Description	Résultat du test
1	Normal	ajoute d'une quantité en stock de 100	
Résultats attendus			
La nouvelle quantité en stock est égale à l'ancienne quantité +100			

Test de la méthode prixStock

10

No test	Type scénario	Description	Résultat du test
2	Normal	Prix du médicament 5, quantité en stock : 100, la fonction retourne le prix du stock	
Résultats attendus			
Prix du stock : 500			

Exercice

11

- Définir le plan de test de la méthode `supprimeStock`

12

Mise en œuvre des tests unitaires

Tester une application

13

- De nombreux éléments à tester
- Tests au fur et à mesure du développement
- Tests unitaires (cf méthode Agile)
- Tests d'intégration

Tests unitaires

14

- Isolation complète : test une classe (et une méthode à la fois)
- Tests les plus simples possibles : éviter l'accès à des ressources
- Code qui exécute les méthodes, conventions de code doivent être respectées

Tests d'intégration

15

- Après les tests unitaires
- Méthode qui utilise une autre méthode (accès à BD, fichier...)
- Réalisés au fur et à mesure
- Vérification du fonctionnement de l'application

Test de recette

16

- Vérification des fonctionnalités de l'application
- Document de recette (validation de la livraison)

Tests unitaires



17

- Schéma AAA (Arrange, Act, Assert)
- Arranger : créer variables, objets nécessaires au test
- Agir : exécution de la méthode à tester
- Auditer : vérifier que le résultat est conforme aux attentes

Résultat du test

18

□ Binaire

-  Réussi : comportement du programme conforme à celui attendu
-  Echoué : comportement du programme différent de celui attendu

Automatiser les tests unitaires

19

- Evolution de l'application = évolution du code

- 3 règles
 - Tester le plus possible
 - Tester le plus tôt possible
 - Tester le plus souvent possible

Framework de test

20

- Framework permettant d'automatiser les tests
- Exécution des tests
- Contrôle du bon déroulement réalisé par le framework

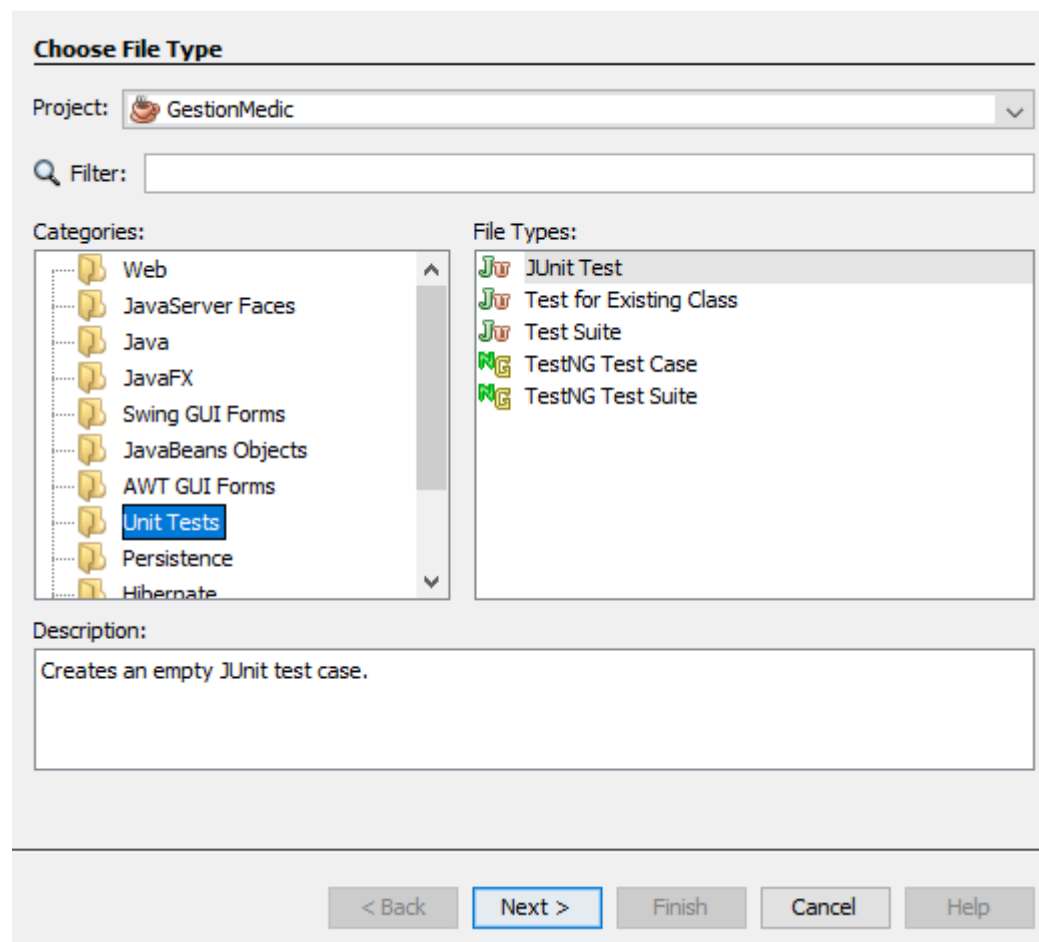
Framework JUnit

21

- Le plus utilisé en java
- Création de classes de test qui héritent de TestCase
- Ajout de méthodes de type assert (vérification)

Création d'une classe de test

22



Génération de la classe de test

23

@BeforeClass

```
public static void setUpClass() { }
```

@AfterClass

```
public static void tearDownClass() { }
```

@Before

```
public void setUp() {  
    //méthode exécutée avant chaque test
```

@After

```
public void tearDown() {  
    //méthode exécutée après chaque test }
```

Ajout d'une méthode de test

24

- Commence par le tag @Test
- Arrange
 - ▣ Préparer les objets nécessaires au test
- Act
 - ▣ Lancer la méthode à tester
- Assert
 - ▣ Vérifier le résultat de la méthode

Types de vérification

25

Méthode	Rôle
assertEquals	Vérifie que deux objets sont égaux
assertFalse	Vérifie que l'expression est fausse
assertNotNull	Vérifie que l'objet n'est pas nul
assertNotSame	Vérifie que deux références ne sont pas les mêmes
assertNull	Vérifie qu'un objet est nul
assertSame	Vérifie que deux références sont les mêmes
assertTrue	Vérifie que l'expression est vrai
fail	Provoque l'échec du test

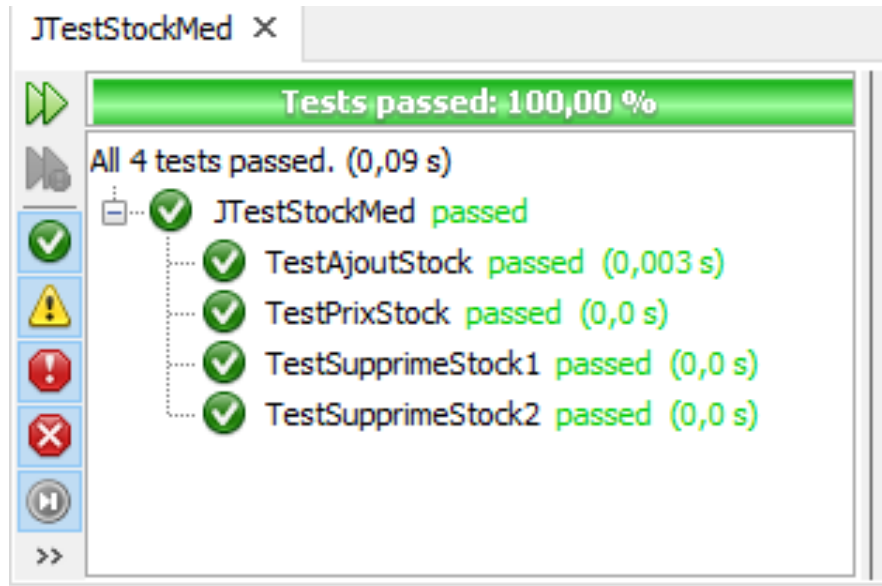
Exemple : test AjoutStock

26

- Arrange :
 - ▣ Créer un objet Medicament
 - ▣ Créer un objet StockMed (quantité en stock : 100)
- Act :
 - ▣ Appel de la méthode AjoutStock (quantité :100)
- Assert
 - ▣ Vérifie la modification de la quantité (nouvelle valeur : 200)

Exécution des tests

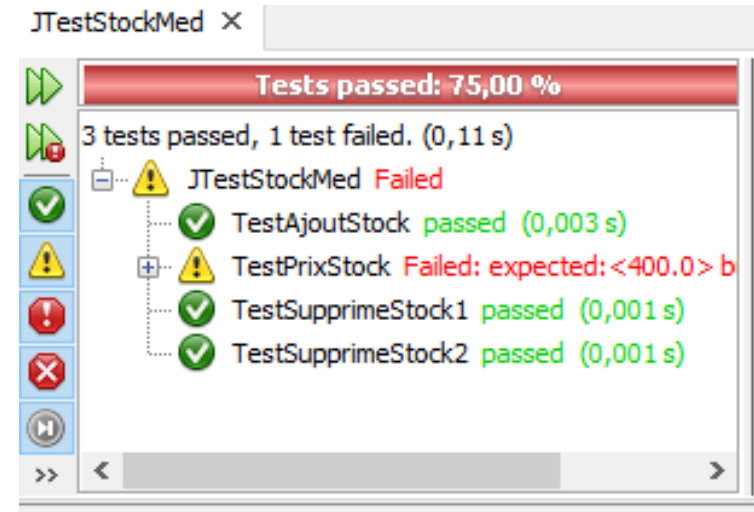
27



Erreurs détectées

28

- Lorsque la valeur testée avec une méthode assert n'a pas le résultat attendu



Option d'un test

29

Echec d'un test

- Possibilité de faire échouer un test avec la méthode
 - ▣ `fail()`
 - ▣ `fail(String message)`

Vérifier une exception

- Pour vérifier qu'une exception est levée
- Au niveau du tag `@Test`
(`expected=Classe de l'exception`)

Exercice

30

Méthode SupprimeStock

- Déclencher une exception `StockInsuffisantException` (classé à créer) lorsque la quantité à ôter est supérieure à la quantité restant en stock

Classe de test

- Ajouter une méthode de test pour vérifier la levée de l'exception