IUT Lyon 1

l'excellence technologique

# Tests unitaires

LP IEM

# Tests unitaires

- Qu'est-ce qu'un test unitaire ?
  - Les tests fonctionnels, d'intégration ou de validation testent les fonctionnalités d'un logiciel d'un point de vue externe ➪ tests "clients"
  - Tests unitaires s'appliquent au niveau d'une classe
  - Tests unitaires ➪ "tests programmeur"
  - Test des fonctionnalités élémentaires
    - Logique d'un algorithme
    - Rejet des entrées hors domaine
    - ...

# Tests unitaires

- Pourquoi des frameworks de tests unitaires ?
  - Généralement la première version d'un code est testée en détail (pas à pas, sortie de chaîne de caractère sur la console, ...)
  - Les problèmes arrivent lors des modifications (ajout de fonctionnalités, corrections de bugs,...)
  - Nécessité d'écrire des tests automatiques
    - ce n'est pas le programmeur qui vérifie les sorties du programme
    - les tests sont construits en utilisant les entrées, un contexte et le résultat attendu
    - le test vérifie que le résultat est conforme au résultat attendu

# Tests unitaires

- Pourquoi des frameworks de tests unitaires ?
  - Au cours du développement on construit des suites de test qui sont exécutées plusieurs dizaines de fois par jour
    ➪ vérifier que chaque modification significative ne casse pas le programme
  - Automatisation des tests
    - un clic pour lancer une suite de tests

# Tests unitaires

- JUnit
  - Framework de tests unitaire spécifique à Java
  - Intégré dans tous les IDE (Netbeans, Eclipse, ...)
  - Deux versions coexistent
    - Version 3 : utilise l'héritage pour définir les tests
    - Version 4 : s'appuie sur les annotations (introduites avec Java 5)

## JUnit 4 - exemple

```
import static org.junit.Assert.*;
import org.junit.*;

public class SlidingPuzzleTest {

    Connection c;

    @Before
    public void setUp() {
        c = Connection.newInstance();
    }

    @Test
    public void testConnectioOpening() {
        assertTrue(c.isClosed());
        c.open();
        assertFalse(c.isClosed());
    }
```

# Tests unitaires

- JUnit 4
  - Chaque méthode de test qui doit être invoquée automatiquement est marquée avec l'annotation @Test
  - Si on veut tester qu'une méthode lance bien une exception dans certaines conditions on annote la méthode avec @Test(expected = Exception.class)
    - On peut préciser la classe de l'exception attendue

# Tests unitaires

- JUnit 4
  - L'annotation @Before permet de dire à JUnit d'exécuter une méthode avant chaque test
    - Utile pour effectuer une initialisation commune à plusieurs méthodes de test
  - Les tests s'écrivent en utilisant les méthodes statiques de la classe Assert :
    - assertEquals(int arg1 , int arg2)
    - assertEquals(Object o1, Object o2)
      - définie pour tous les types primitifs et les objets
    - assertTrue(boolean condition)
    - assertFalse(boolean condition)

- JUnit 4
  - Méthodes statiques de la classe Assert (suite)
    - assertNull(Object object)
    - assertNotNull(Object object)
    - assertSame(Object expected, Object actual)
    - assertNotSame(Object unexpected, Object actual)
  - Toutes les méthodes statiques de la classe Assert existent aussi avec un argument supplémentaire de type String affiché en cas d'échec du test
    - assertTrue("Valeur négative", x < 0)
    - assertEquals("Nb de mesures different", m.length, l.size())

# Tests unitaires

- JUnit 4
  - Attention aux problèmes d'arrondis dans les tests d'égalité
    - Utiliser la méthode appropriée qui permet de spécifier une tolérance (delta) sur le test d'égalité

```
public static void assertEquals(String message,
                double expected,
                double actual,
                double delta)
```

  - Exemple :

```
assertEquals("Diagonale du carre", Math.sqrt(2), Carre.diagonale(2), 1e-6);
```

## JUnit 4 - Exercice

- Créer une classe immuable Point avec un constructeur prenant les coordonnées x et y en paramètre
  - Ajouter une méthode double distance(Point p)
  - Implémenter les méthodes hashcode() et equals(Object o)
  - Ecrire des tests unitaires pour valider ces méthodes

## JUnit 4 - Exercice

- Créer des classes Circle, Rectangle implémentant une interface Shape définissant :
  - une méthode boolean contains(Point p) qui renvoie vrai si les coordonnées du point (p.x,p.y) sont à l'intérieur de la forme
  - une méthode double area() retournant l'aire
- Constructeurs :
  - Circle (Point center, int radius)
  - Rectangle(Point topLeftCorner, int width, int height)
- Implémenter les méthodes hashcode() et equals(Object o)
- Ecrire des tests unitaires pour valider ces méthodes