

# VALIDATION & VERIFICATION

## *DYNAMIC TESTING*

---

MASTER 1 ICE, 2017-2018

**BENOIT COMBEMALE**  
PROFESSOR, UNIV. TOULOUSE, FRANCE

[HTTP://COMBEMALE.FR](http://combemale.fr)  
[BENOIT.COMBEMALE@IRIT.FR](mailto:benoit.combemale@irit.fr)  
[@BCOMBEMALE](https://twitter.com/BCOMBEMALE)



# Plan

---

1. Définition
2. JUnit
3. Some examples from the field
4. Test-Driven Development

---

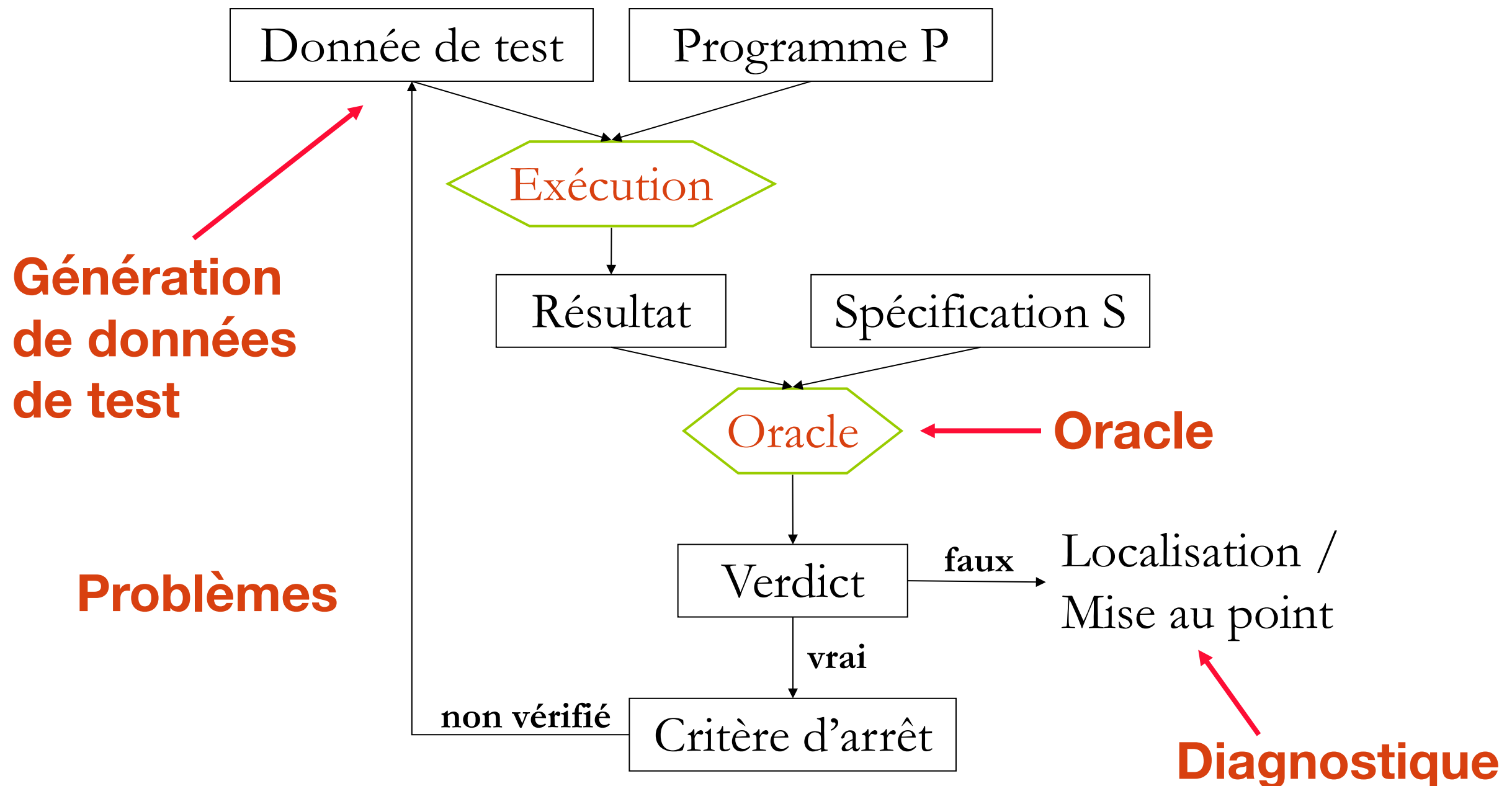
## **DYNAMIC TESTING: DEFINITION**

# Test dynamique

---

Exécuter le programme avec un ensemble de valeurs en entrée dans le but de révéler des erreurs dans l'implantation

# Le test dynamique : processus



# Le test dynamique

---

- Soit  $D$  le domaine d'entrée d'un programme  $P$  spécifié par  $S$ , on voudrait pouvoir dire
  - Soit  $D$  le domaine de  $P$ :  $\forall x \in D \ P(x) = S(x)$
- Test exhaustif impossible dans la plupart des cas
  - Domaine  $D$  trop grand, voire infini
  - Trop long et coûteux

# Le test dynamique

---

- On cherche alors un ensemble de données de test  $T$  tel que
  - $T \subset D$
  - si  $\forall x \in T P(x) = S(x)$  alors  $\forall x \in D P(x) = S(x)$
- Critère d'arrêt pour la génération de données de test
  - $\{\text{données de test}\} = T$

# Génération de test

---

- Génération déterministe
  - « à la main »
- Génération automatique aléatoire
- Génération automatique aléatoire contrainte
  - mutation
  - test statistique
- Génération automatique guidée par les contraintes
- Génération de test avancée (à partir des exigences, de modèles : *Model Based Testing*)



# Génération de test

---

- Reste à savoir quand on a suffisamment testé
  - critères de test structurels, fonctionnels
  - analyse de mutation
- Choisir le bon niveau pour le test

# La notion de couverture et de critère d'arrêt

---

- Critère d'arrêt = conditions objectives, mesurables, qui font qu'on peut considérer qu'une série de tests est suffisante
- Test fonctionnel (boîte noire)
  - Pour arrêter les tests, on se base sur des critères externes
    - Couverture des exigences
    - Couverture des fonctions des interfaces
    - Taux de défaillances par jour inférieurs à un seuil
- Test structurel (boîte blanche)
  - Pour arrêter les tests, on se base sur des critères internes
    - Couverture de la structure du système
  - Chaque composant, chaque dépendance entre composants
    - Couverture du code
- Le test structurel permet d'être plus précis dans l'élaboration du critère d'arrêt des tests

# Oracle

---

- Fonction qui évalue le résultat d'un cas de test
- Plus formellement
  - soit un programme  $P: \text{Dom}(P) \rightarrow \text{Ran}(P)$
  - une spécification  $S: \text{Dom}(P) \rightarrow \text{Ran}(P)$
  - une donnée de test  $X \in \text{Dom}(P)$
  - oracle  $O: \text{Dom}(P) \times \text{Ran}(P) \rightarrow \text{bool}$ 
$$O(X, P(X)) = \text{true} \text{ iff } P(X) = S(X)$$
- Problème : comment comparer  $P(X)$  et  $S(X)$ 
  - plus  $S$  est formalisé plus on peut automatiser

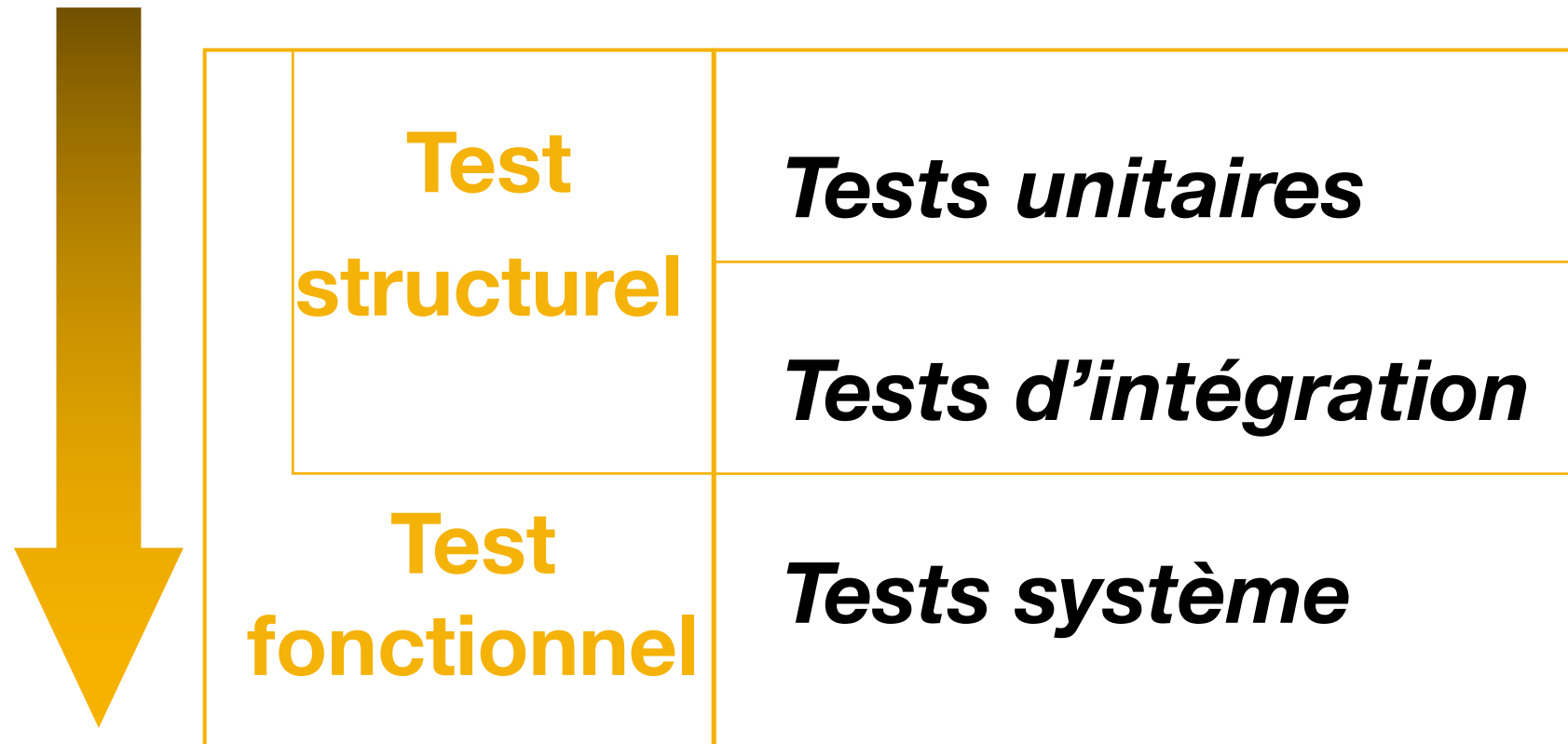
# Oracle

---

- Oracle manuel: on « regarde » le résultat et un humain évalue s'il est bon
- Construire le résultat exactement attendu
  - Comparer le résultat obtenu avec le résultat construit (diff)
- Assertions
  - dans le code (programmation défensive)
  - aux interfaces (design by contract)
    - set\_current\_node (cnode: Node)
    - pre** : cnode != null
    - post** : currentNode = cnode
  - dans les cas de test (par ex. JUnit)
- ...

# Etapes et hiérarchisation des tests

---



# Dynamic testing in brief

---

- Testers run the program to reveal the presence of errors
- 4 main activities
  - TEST CASE DESIGN
    - select data and scenario to run the program
  - TEST ORACLE DESIGN
    - express properties that must hold on these runs
  - TEST RUN
    - set the execution environment for test run
  - STOPPING CRITERION
    - ideally when there are no more errors
    - approximation, w.r.t coverage criteria

# Test automation

---

- Some tools exist to automate the following tasks
  - data/scenario generation
  - oracle checking
  - test case execution
  - coverage checking

**=> + details in the specification  
+ can be automated**

---

**JUNIT**



# Test unitaire OO

---

- Tester une unité isolée du reste du système
- L'unité est la classe
  - Test unitaire = test d'une classe
- Test du point de vue client
  - les cas de tests appellent les méthodes depuis l'extérieur
  - on ne peut tester que ce qui est public
  - Le test d'une classe se fait à partir d'une classe extérieure
- Au moins un cas de test par méthode publique
- Il faut choisir un ordre pour le test
  - quelles méthodes sont interdépendantes?

# Test unitaire OO

---

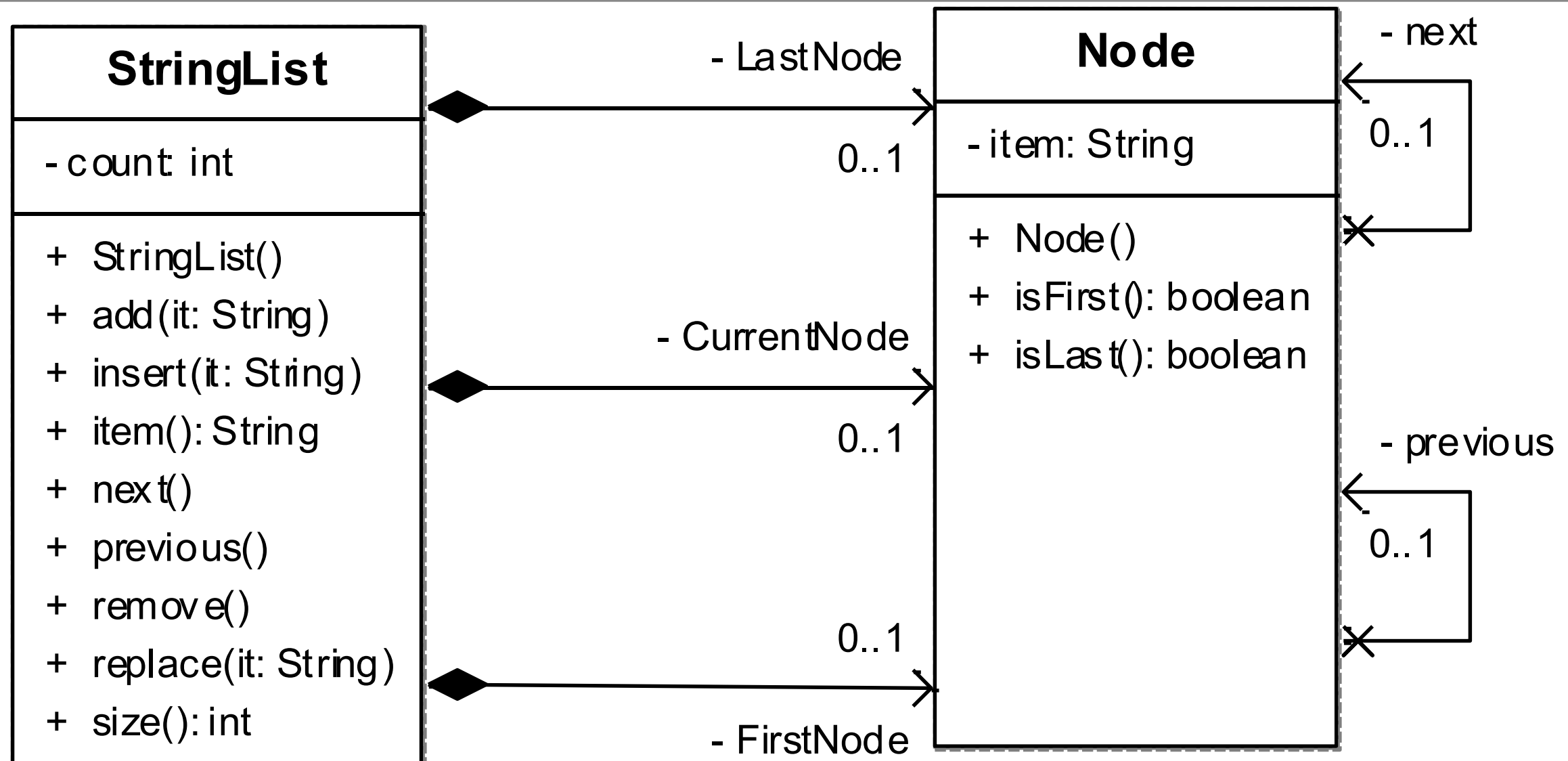
- Problème pour l'oracle :
  - Encapsulation : les attributs sont souvent privés
  - Difficile de récupérer l'état d'un objet
- Penser au test au moment du développement (« testabilité »)
  - prévoir des accesseurs en lecture sur les attributs privés
  - des méthodes pour accéder à l'état de l'objet

# Cas de test unitaire

---

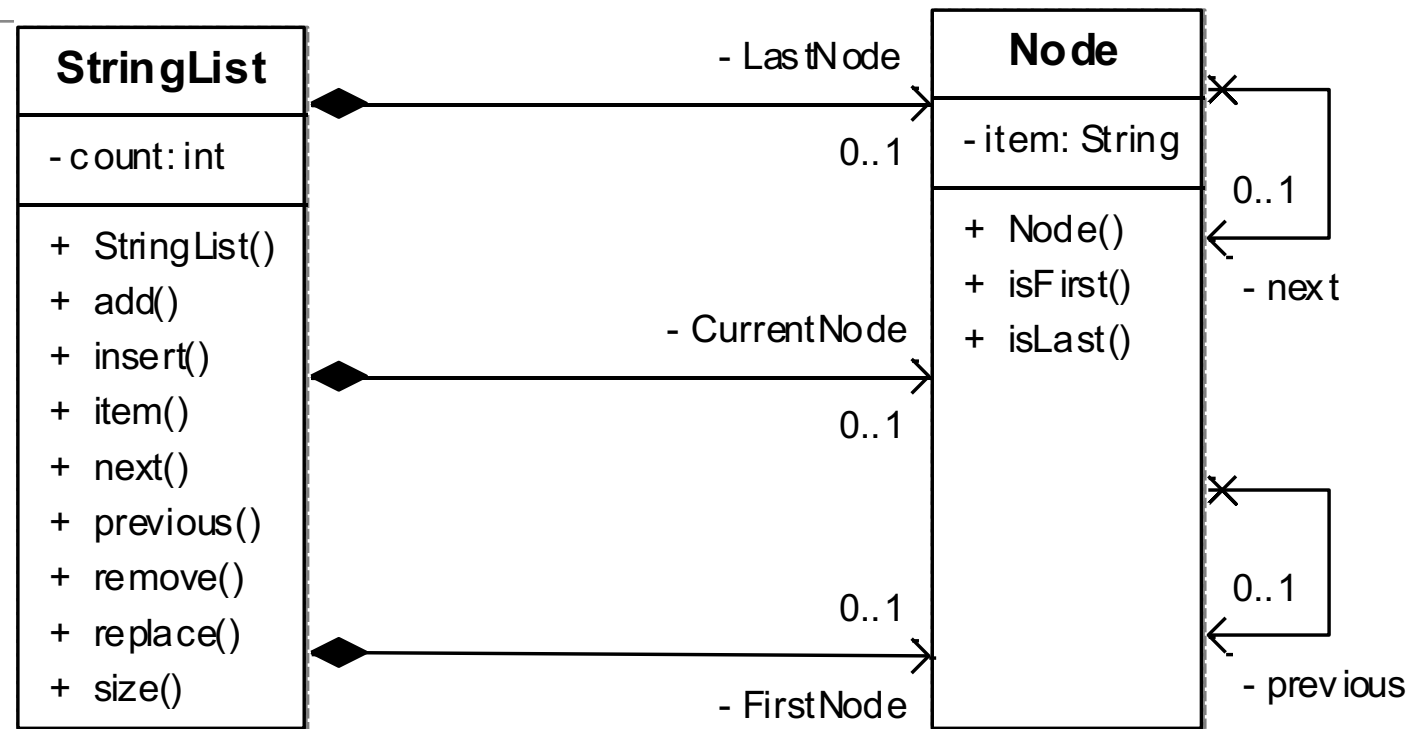
- Cas de test = une méthode
- Corps de la méthode
  - Configuration initiale
  - Une donnée de test
    - un ou plusieurs paramètres pour appeler la méthode testée
  - Un oracle
    - il faut construire le résultat attendu
    - ou vérifier des propriétés sur le résultat obtenu
- Une classe de test pour une classe testée
  - Regroupe les cas de test
  - Il peut y avoir plusieurs classes de test pour une classe testée

# Exemple : test de StringList



- Créer une classe de test qui manipule des instances de la classe **StringList**
- Au moins 9 cas de test (1 par méthode publique)
- Pas accès aux attributs privés : `count`, `LastNode`, `CurrentNode`, `FirstNode`

# Exemple : insertion dans une liste



spécification du cas  
de test

```
//first test for insert: call insert  
//and see if current element is the  
//one that's been inserted
```

initialisation

```
public void testInsert1() {
```

```
    list.add("first");
```

```
    list.add("second");
```

```
    list.insert("third");
```

```
    assertTrue(list.size() == 3);
```

```
    assertTrue(list.item() == "third");
```

```
}
```

appel avec donnée de test  
oracle

# JUnit

---

- Origine
  - Xtreme Programming (test-first development)
  - framework de test écrit en Java par E. Gamma et K. Beck
  - open source: [www.junit.org](http://www.junit.org)
- Objectifs
  - test d'applications en Java
  - faciliter la création des tests
  - tests de non régression

# JUnit : un framework

---

*« Un framework est un ensemble de classes et de collaborations entre les instances de ces classes. »*

<http://st-www.cs.uiuc.edu/users/johnson/frameworks.html>

# JUnit : un framework

---

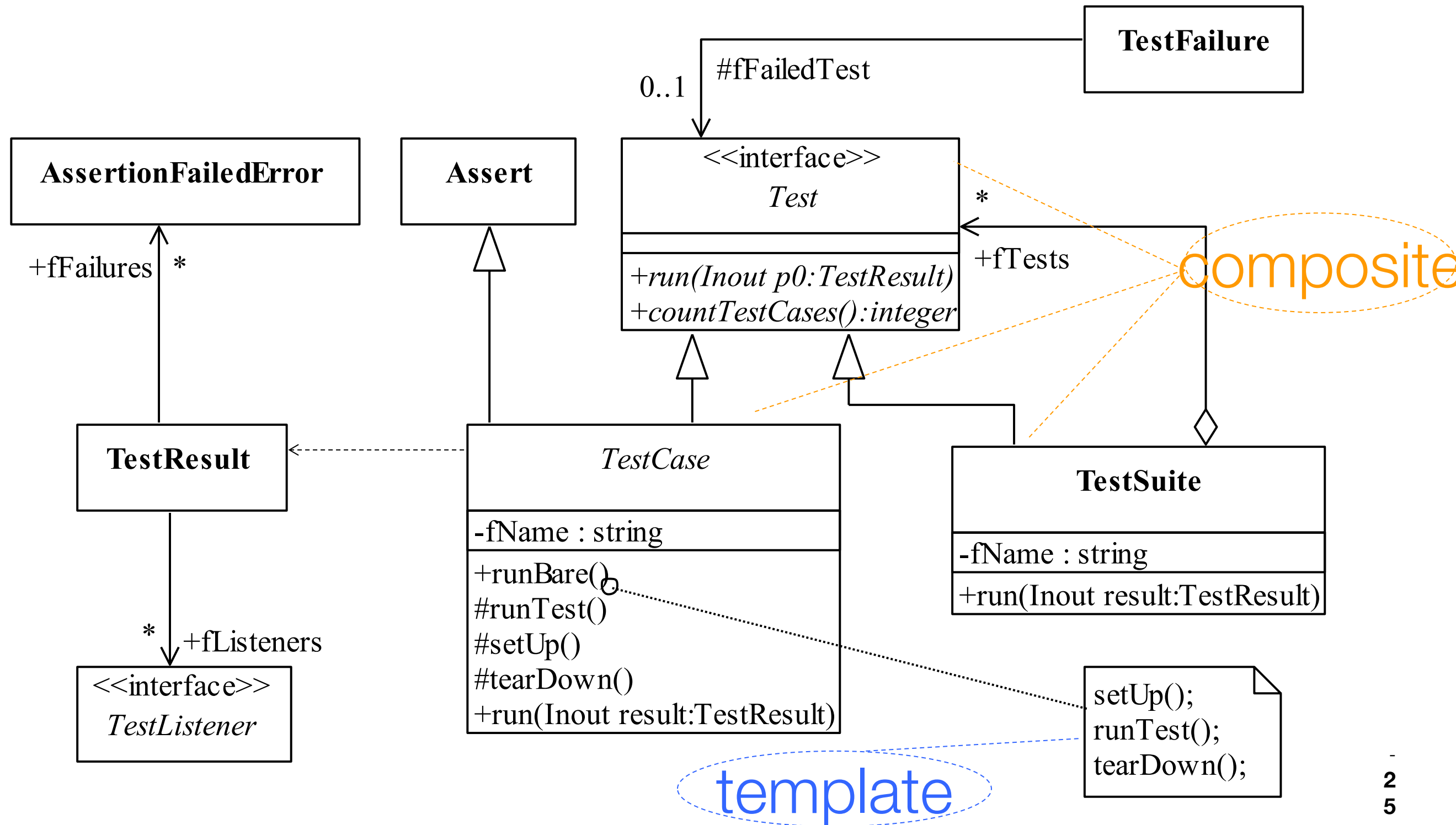
- Le source d'un framework est disponible
- Ne s'utilise pas directement: il se spécialise

Ex: pour créer un cas de test on hérite de la classe  
TestCase

Ⓟ Un framework peut être vu comme un programme à  
« trous » qui offre la partie commune des traitements et  
chaque utilisateur le spécialise pour son cas particulier.



# JUnit : un framework



# JUnit v3 : codage (1/5)

---

- Organisation du code des tests
  - cas de Test: TestCase
    - setUp() et tearDown()
    - les méthodes de test
  - suite de Test: TestSuite
    - Méthodes de test
    - Cas de test
    - Suite de Test

# Exemple : une classe StringList

---

```
public class StringList {  
  
    private Node currentNode;  
    private Node firstNode;  
    private int count;  
  
    public StringList(){  
        count=0;  
    }  
  
    public String item(){  
        return this.currentNode.getItem();  
    }  
  
    public void next(){...}  
    public void previous(){ ... }  
    public int size(){...}  
    public void add (String it){...}  
    public void replace (String it){...}  
  
}
```

# JUnit

---

- Une classe de test regroupe des cas de test pour une classe

```
import org.junit.Before;
import org.junit.Test;

public class TestStringList {
    //déclaration des instances
    private StringList list;

    //setUp()
    //tearDown()
    //méthodes de test
    //main()
}
```

# JUnit

---

## - la méthode setUp:

*//appelée avant chaque cas de test*

*//permet de factoriser la construction de l'état initial*

@Before

```
protected void setUp() throws Exception {  
    list = new StringList();  
}
```

## - la méthode tearDown:

*//appelée après chaque cas de test*

*//permet de défaire « l'état du monde »*

@After

```
protected void tearDown() throws Exception {  
    super.tearDown();  
}
```

# JUnit

---

- les méthodes de test:

```
//test add two elements  
@Test  
public void testAdd2 () {  
    list.add("first");  
    list.add("second");  
    assertTrue(list.size() == 2);  
    assertTrue(list.item() == "second");  
}
```

# Ecrire 4 cas de test pour List

---

```
package test;

public class List {

    private int[] tab = new int[10];
    private int current = -1;

    public void add(int i) {
        current++;
        tab[current]=i;
    }

    public int length() {
        return current+1;
    }

    public int current() {
        return tab[current];
    }

    public int get(int i) {
        return tab[i];
    }

}
```

```
public class TestList {  
    private List l;  
    @Before  
    public void setUp() throws Exception {  
        l = new List();  
    }  
    @After  
    public void tearDown() throws Exception {  
    }  
    @Test  
    public void testinit() {  
        Assert.assertEquals(0, l.length());  
    }  
    @Test  
    public void testadd() {  
        l.add(1);  
        Assert.assertEquals(1, l.get(0));  
    }  
    @Test  
    public void testadd2() {  
        l.add(1);  
        l.add(2);  
        Assert.assertEquals(2, l.length());  
    }  
    @Test  
    public void testcurrent() {  
        l.add(1);  
        Assert.assertEquals(1, l.current());  
    }  
}
```



# Ecrire 2 cas de test pour readData

---

```
public class Stream {
    BufferedReader reader;

    public Stream(String fileNameData) throws FileNotFoundException {
        this.reader = new BufferedReader(new FileReader(fileNameData));
    }

    public void readData() {
        try {
            String line = reader.readLine();
            while (line != null) {
                System.out.println(parseLine(line));
                line = reader.readLine();
            }
            reader.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    protected double parseLine(String line) {
        String[] split = line.split(";");
        return Double.parseDouble(split[0]) / Double.parseDouble(split[1]);
    }

    public static void main(String[] args) throws FileNotFoundException {
        Stream s = new Stream(args[0]);
        s.readData();
    }
}
```

```
private final ByteArrayOutputStream outContent = new ByteArrayOutputStream();
private final ByteArrayOutputStream errContent = new ByteArrayOutputStream();

@Before
public void setUp() throws Exception {
    System.setOut(new PrintStream(outContent));
    System.setErr(new PrintStream(errContent));
}

@Test
public void testOK(){
    try {
        File f = new File("ok.text");
        //FileOutputStream s = new FileOutputStream(f);
        FileWriter fstream = new FileWriter(f);
        BufferedWriter out = new BufferedWriter(fstream);
        out.write("4;4");
        out.close();
        Stream s = new Stream(f.getName());
        s.readData();
        assertEquals("1.0\n", outContent.toString());
    } catch (Exception e) {
        System.err.println("Error: " + e.getMessage());
    }
}
```

---

```
@Test
public void testnOK() {
    try {
        File f = new File("ok.text");
        //FileOutputStream s = new FileOutputStream(f);
        FileWriter fstream = new FileWriter(f);
        BufferedWriter out = new BufferedWriter(fstream);
        out.write("4,4");
        out.close();
        Stream s = new Stream(f.getName());
        s.readData();
        assertFalse(errContent.size()==0);
    } catch (Exception e) {
        System.err.println("Error: " + e.getMessage());
    }
}
```

# Les assertions de JUnit

---

- `assertTrue(...)`, `assertFalse(...)`
- `assertEquals(Object, Object)`
- `assertSame(Object, Object)`
- `assertNull(...)`
- `assertEquals(double expected, double actual, double delta)`



**Voir la liste des méthodes static de la classe `Assert`**

# JUnit v3 : détail d'implémentation

---

- Pour exécuter une suite de tests, JUnit utilise l'introspection

```
public TestSuite (final Class theClass){  
    ...  
    Method[] = theClass.getDeclaredMethods  
    ...  
}  
private boolean isTestMethod(Method m) {  
    String name= m.getName();  
    Class[] parameters= m.getParameterTypes();  
    Class returnType= m.getReturnType();  
    return parameters.length == 0 && name.startsWith("test") &&  
    returnType.equals(Void.TYPE);  
}
```

# JUnit version 4

---

- Fonctionne avec Java 5+
- Utilisation intensive des annotations
- Tests paramétrés, timeouts, etc

# JUnit v4 : classe et méthode de test

---

- Classe de test :
  - `import org.junit.Test;`
  - `import static org.junit.Assert.*;`
- Méthodes de test :
  - Nom de méthode quelconque
  - Annotation `@Test`
  - Publique, type de retour `void`
  - Pas de paramètre, peut lever une exception
  - Annotation `@Test(expected = Class)` pour indiquer l'exception attendue
  - Annotation `@Ignore` pour ignorer un test

# JUnit v4 : test paramétrés

---

```
@RunWith(Parameterized.class)
public class FibonacciTest {
    @Parameters
    public static Collection<Object[]> data() {
        return Arrays.asList(new Object[][] {
            {0, 0}, {1, 1}, {2, 1}, {3, 2}, {4, 3}, {5, 5}, {6, 8});
        }

    private int fInput;

    private int fExpected;

    public FibonacciTest(int input, int expected) {
        fInput= input;
        fExpected= expected;
    }

    @Test
    public void test() {
        assertEquals(fExpected, Fibonacci.compute(fInput));
    }
}
```



# JUnit v5 : test exceptions

---

@Rule

```
public ExpectedException thrown =  
ExpectedException.none();
```

@Test

```
public void shouldTestExceptionMessage() throws  
IndexOutOfBoundsException {  
    List<Object> list = new ArrayList<Object>();  
  
    thrown.expect(IndexOutOfBoundsException.class);  
    thrown.expectMessage("Index: 0, Size: 0");  
    list.get(0); // execution will never get past this  
line  
}
```

# Junit v5 : Assumptions

---

```
public class AssumptionsDemo {
    @Test
    void testOnlyOnDeveloperWorkstation() {
        assumeTrue("DEV".equals(System.getenv("ENV")),
            () -> "Aborting test: not on developer workstation");
        // remainder of test
    }

    @Test
    void testInAllEnvironments() {
        assumingThat("CI".equals(System.getenv("ENV")),
            () -> {
                // perform these assertions only on the CI server
                assertEquals(2, 2);
            });

        // perform these assertions in all environments
        assertEquals("a string", "a string");
    }
}
```

# JUnit

---

- Permet de structurer les cas de test
  - cas de test / suite de test
- Permet de sauvegarder les cas de test
  - important pour la non régression
  - quand une classe évolue on ré-exécute les cas de test

# JUnit

---

- [www.junit.org](http://www.junit.org)
  - <http://junit.org/junit5/docs/current/user-guide/>
- Avantages
  - gratuit
  - simple
  - Intégré aux IDE et outils de build
- Inconvénients
  - exploitation des résultats (pas d'historique...)

---

## **SOME EXAMPLES FROM THE FIELD**

# Loads of JUnit test cases available

---

- Lots of good examples
  - well-designed test cases
  - complex test data
  - original checkers
  - complex oracles

# Regular test case

---

```
// In  
org.apache.commons.math3.linear.RealVectorFormatAbstractTest
```

```
@Test  
public void testSimpleWithDecimals() {  
    ArrayRealVector c = new ArrayRealVector(new  
double[] {1.23, 1.43, 1.63});  
    String expected =  
        "{1"      + getDecimalCharacter() +  
        "23; 1"    + getDecimalCharacter() +  
        "43; 1"    + getDecimalCharacter() +  
        "63}";  
    String actual = realVectorFormat.format(c);  
    Assert.assertEquals(expected, actual);  
}
```

# Test exceptional case

---

```
// In org.apache.commons.math3.filter.KalmanFilterTest

@Test(expected=MatrixDimensionMismatchException.class)
public void testTransitionMeasurementMatrixMismatch() {

    // A and H matrix do not match in dimensions

    // A = [ 1 ]
    RealMatrix A = new Array2DRowRealMatrix(new double[] { 1d });
    // no control input
    RealMatrix B = null;
    // H = [ 1 1 ]
    RealMatrix H = new Array2DRowRealMatrix(new double[] { 1d, 1d });
    // Q = [ 0 ]
    RealMatrix Q = new Array2DRowRealMatrix(new double[] { 0 });
    // R = [ 0 ]
    RealMatrix R = new Array2DRowRealMatrix(new double[] { 0 });

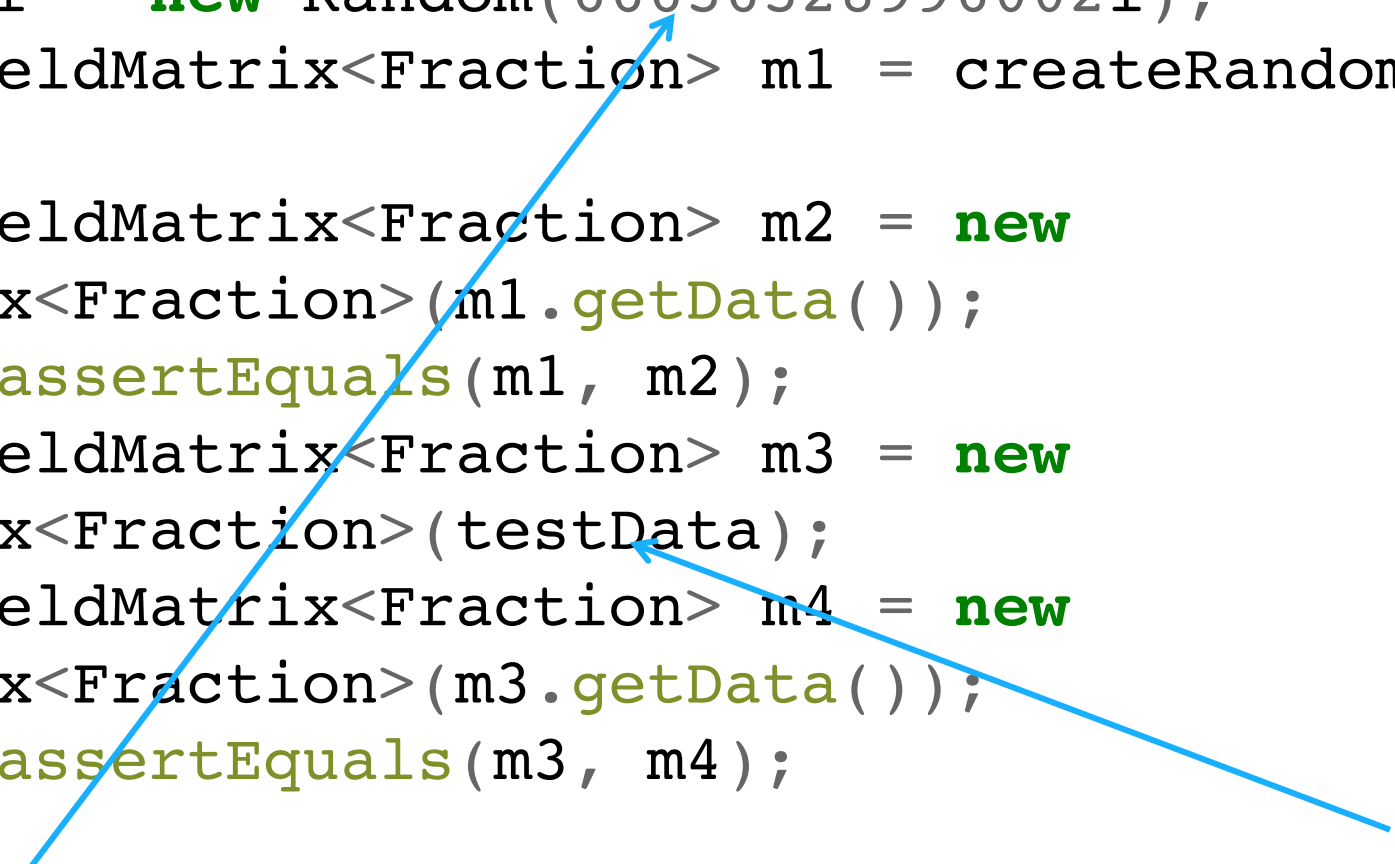
    ProcessModel pm
        = new DefaultProcessModel(A, B, Q,
                                   new ArrayRealVector(new double[] { 0 }), null);
    MeasurementModel mm = new DefaultMeasurementModel(H, R);
    new KalmanFilter(pm, mm);
    Assert.fail("transition and measurement matrix should not be compatible")
}
18
```



# Well-designed test case

```
// In org.apache.commons.math3.linear.BlockFieldMatrixTest

/** test copy functions */
@Test
public void testCopyFunctions() {
    Random r = new Random(666363289960021);
    BlockFieldMatrix<Fraction> m1 = createRandomMatrix(r,
47, 83);
    BlockFieldMatrix<Fraction> m2 = new
BlockFieldMatrix<Fraction>(m1.getData());
    Assert.assertEquals(m1, m2);
    BlockFieldMatrix<Fraction> m3 = new
BlockFieldMatrix<Fraction>(testData);
    BlockFieldMatrix<Fraction> m4 = new
BlockFieldMatrix<Fraction>(m3.getData());
    Assert.assertEquals(m3, m4);
}
```



fix the seed for repeatability

use data instead of files

# Strong test case

---

```
// In org.apache.commons.math3.random.UnitSphereRandomVectorGeneratorTest
```

```
@Test
```

```
public void test2DDistribution() {
```

```
    RandomGenerator rg = new JDKRandomGenerator();
```

```
    rg.setSeed(173992254321);
```

```
    UnitSphereRandomVectorGenerator generator = new UnitSphereRandomVectorGenerator(2, rg);
```

```
    // In 2D, angles with a given vector should be uniformly distributed
```

```
    int[] angleBuckets = new int[100];
```

```
    int steps = 1000000;
```

```
    for (int i = 0; i < steps; ++i) {
```

```
        final double[] v = generator.nextVector();
```

```
        Assert.assertEquals(2, v.length);
```

```
        Assert.assertEquals(1, length(v), 1e-10);
```

```
        // Compute angle formed with vector (1,0)
```

```
        // Cosine of angle is their dot product, because both are unit length
```

```
        // Dot product here is just the first element of the vector by construction
```

```
        final double angle = FastMath.acos(v[0]);
```

```
        final int bucket = (int) (angleBuckets.length * (angle / FastMath.PI));
```

```
        ++angleBuckets[bucket];
```

```
    }
```

```
    // Simplistic test for roughly even distribution
```

```
    final int expectedBucketSize = steps / angleBuckets.length;
```

```
    for (int bucket : angleBuckets) {
```

```
        Assert.assertTrue("Bucket count " + bucket + " vs expected " + expectedBucketSize,  
                           FastMath.abs(expectedBucketSize - bucket) < 350);
```

```
    }
```

```
}
```

# Strong test data

---

```
//In org.apache.commons.math3.special.GammaTest
```

```
/**
 * Reference data for the {@link Gamma#logGamma(double)} function. This data
 * was generated with the following <a
 * href="http://maxima.sourceforge.net/">Maxima</a> script.
 *
 * <pre>
 * kill(all);
 *
 * fpprec : 64;
 * gamln(x) := log(gamma(x));
 * x : append(makelist(bfloat(i / 8), i, 1, 80),
 *      [0.8b0, 1b2, 1b3, 1b4, 1b5, 1b6, 1b7, 1b8, 1b9, 1b10]);
 *
 * for i : 1 while i <= length(x) do
 *   print("{", float(x[i]), ",", float(gamln(x[i])), "},");
 * </pre>
 */
```

```
private static final double[][] LOG_GAMMA_REF = {
    { 0.125 , 2.019418357553796 },
    { 0.25  , 1.288022524698077 },
    { 0.375 , .8630739822706475 },
    { 0.5   , .5723649429247001 },
    { 0.625 , .3608294954889402 },
    { 0.75  , .2032809514312954 },
    { 0.875 , .08585870722533433 },
    { 0.890625 , .07353860936979656 },
    ...124 more lines
};
```

# Strong checker

---

```
// org.apache.commons.math3.geometry.euclidean.twod.PolygonsSetTest
private void checkVertices(Vector2D[][] rebuiltVertices,
                           Vector2D[][] vertices) {
    // each rebuilt vertex should be in a segment joining two original vertices
    for (int i = 0; i < rebuiltVertices.length; ++i) {
        for (int j = 0; j < rebuiltVertices[i].length; ++j) {
            boolean inSegment = false;
            Vector2D p = rebuiltVertices[i][j];
            for (int k = 0; k < vertices.length; ++k) {
                Vector2D[] loop = vertices[k];
                int length = loop.length;
                for (int l = 0; (! inSegment) && (l < length); ++l) {
                    inSegment = checkInSegment(p, loop[l], loop[(l + 1) % length], 1.0e-10);
                }
            }
            Assert.assertTrue(inSegment);
        }
    }
    // each original vertex should have a corresponding rebuilt vertex
    for (int k = 0; k < vertices.length; ++k) {
        for (int l = 0; l < vertices[k].length; ++l) {
            double min = Double.POSITIVE_INFINITY;
            for (int i = 0; i < rebuiltVertices.length; ++i) {
                for (int j = 0; j < rebuiltVertices[i].length; ++j) {
                    min = FastMath.min(vertices[k][l].distance(rebuiltVertices[i][j]),
                                         min);
                }
            }
            Assert.assertEquals(0.0, min, 1.0e-10);
        }
    }
}
```

# Interesting checker

---

```
// In org.apache.commons.math3.TestUtils
```

```
/**
 * Verifies that the relative error in actual vs. expected is less than or
 * equal to relativeError. If expected is infinite or NaN, actual must be
 * the same (NaN or infinity of the same sign).
 *
 * @param msg message to return with failure
 * @param expected expected value
 * @param actual observed value
 * @param relativeError maximum allowable relative error
 */
public static void assertRelativelyEquals(String msg, double expected,
    double actual, double relativeError) {
    if (Double.isNaN(expected)) {
        Assert.assertTrue(msg, Double.isNaN(actual));
    } else if (Double.isNaN(actual)) {
        Assert.assertTrue(msg, Double.isNaN(expected));
    } else if (Double.isInfinite(actual) || Double.isInfinite(expected)) {
        Assert.assertEquals(expected, actual, relativeError);
    } else if (expected == 0.0) {
        Assert.assertEquals(msg, actual, expected, relativeError);
    } else {
        double absError = FastMath.abs(expected) * relativeError;
        Assert.assertEquals(msg, expected, actual, absError);
    }
}
```

# Test case antipatterns

---

- PMD has a set of rules for JUnit
  - <http://pmd.sourceforge.net/pmd-4.3.0/rules/junit.html>
  - **JUnitTestsShouldIncludeAssert**
  - **TestClassWithoutTestCases**
  - **UnnecessaryBooleanAssertion**
  - ...
- **Test case antipatterns**
  - **The Free Ride / Piggyback**
  - **Happy Path**
  - **The Hidden Dependency**
  - <http://stackoverflow.com/questions/333682/unit-testing-anti-patterns-catalogue>

# Antipattern: Piggybacking

```
// In org.apache.commons.lang3.ArrayUtilsRemoveMultipleTest
```

```
@Test
public void testRemoveAllObjectArray() {
    Object[] array;
    array = ArrayUtils.removeAll(new Object[] { "a" }, 0);
    assertEquals(ArrayUtils.EMPTY_OBJECT_ARRAY, array);
    assertEquals(Object.class, array.getClass().getComponentType());
    array = ArrayUtils.removeAll(new Object[] { "a", "b" }, 0, 1);
    assertEquals(ArrayUtils.EMPTY_OBJECT_ARRAY, array);
    assertEquals(Object.class, array.getClass().getComponentType());
    array = ArrayUtils.removeAll(new Object[] { "a", "b", "c" }, 1, 2);
    assertEquals(new Object[] { "a" }, array);
    assertEquals(Object.class, array.getClass().getComponentType());
    array = ArrayUtils.removeAll(new Object[] { "a", "b", "c", "d" }, 1, 2);
    assertEquals(new Object[] { "a", "d" }, array);
    assertEquals(Object.class, array.getClass().getComponentType());
    array = ArrayUtils.removeAll(new Object[] { "a", "b", "c", "d" }, 0, 3);
    assertEquals(new Object[] { "b", "c" }, array);
    assertEquals(Object.class, array.getClass().getComponentType());
    array = ArrayUtils.removeAll(new Object[] { "a", "b", "c", "d" }, 0, 1, 3);
    assertEquals(new Object[] { "c" }, array);
    assertEquals(Object.class, array.getClass().getComponentType());
    array = ArrayUtils.removeAll(new Object[] { "a", "b", "c", "d", "e" }, 0, 1, 3);
    assertEquals(new Object[] { "c", "e" }, array);
    assertEquals(Object.class, array.getClass().getComponentType());
    array = ArrayUtils.removeAll(new Object[] { "a", "b", "c", "d", "e" }, 0, 2, 4);
    assertEquals(new Object[] { "b", "d" }, array);
    assertEquals(Object.class, array.getClass().getComponentType());

    ...
}
```

# Antipattern: happy path

---

```
// In org.easymock.tests.CapturesMatcherTest
```

```
@Test
public void test() throws Exception {

    matcher.appendTo(buffer);
    assertEquals("capture(Nothing captured yet)", buffer.toString());

    assertTrue(matcher.matches(null));

    matcher.validateCapture();

    clearBuffer();
    matcher.appendTo(buffer);
    assertEquals("capture(null)", buffer.toString());

    assertTrue(matcher.matches("s"));

    matcher.validateCapture();

    clearBuffer();
    matcher.appendTo(buffer);
    assertEquals("capture([null, s])", buffer.toString());
}
```



# Antipattern: useless assert

---

```
// In org.apache.commons.math3.genetics.RandomKeyTest
```

```
@Test
public void testRandomPermutation() {
    // never generate an invalid one
    for (int i=0; i<10; i++) {
        DummyRandomKey drk = new
DummyRandomKey(RandomKey.randomPermutation(20));
        Assert.assertNotNull(drk);
    }
}
```



Assertion cannot fail

# Flaky test

The test checks that mocking behaves properly on multi-threaded flows

```
// In a test class
public class Test {
    private static final int THREAD_COUNT = 10;

    @Test
    public void testThreadSafe() throws Throwable {
        final IMethods mock = createMock(IMethods.class);
        expect(mock.oneArg("test")).andReturn("result").times(THREAD_COUNT);

        replay(mock);

        final Callable<String> replay = new Callable<String>() {
            public String call() throws Exception {
                return mock.oneArg("test");
            }
        };

        final ExecutorService service = Executors.newFixedThreadPool(THREAD_COUNT);
        final List<Callable<String>> tasks = new ArrayList<>();
        final List<Future<String>> results = new ArrayList<>();
        for (final Future<String> future : service.invokeAll(tasks)) {
            results.add(future);
        }
        assertEquals(results, Collections.singletonList("result"));
        verify(mock);
    }
}
```

create 10 threads

creates 10 copies of the replay mock

assign 10 tasks to the threads

check that the mocks return "result"

check that the 10 mocks have run

# Flaky test

---

```
// In org.easymock.tests2
public class ThreadingTest {

    private static final int THREAD_COUNT = 10;

    @Test
    public void testThreadSafe() throws Throwable {
```

When we run the test case 20  
times in a row, it fails once or  
twice...

```
        return mock.oneArg("test");
    }
};
final ExecutorService service = Executors.newFixedThreadPool(THREAD_COUNT);
final List<Callable<String>> tasks = Collections.nCopies(THREAD_COUNT, replay);
final List<Future<String>> results = service.invokeAll(tasks);
for (final Future<String> future : results) {
    assertEquals("result", future.get());
}
verify(mock);
}
```

# Flaky test

```
// In org.easymock.tests2
public class ThreadingTest {
    newFixedThreadPool
    private
    public static ExecutorService newFixedThreadPool(int nThreads)
    @Test
    public Creates a thread pool that reuses a fixed number of threads operating off a shared
        unbounded queue. At any point, at most nThreads threads will be active processing tasks.
    fi If additional tasks are submitted when all threads are active, they will wait in the queue until
    ex a thread is available. If any thread terminates due to a failure during execution prior to
    re shutdown, a new one will take its place if needed to execute subsequent tasks. The threads
        in the pool will exist until it is explicitly shutdown.
    fi Parameters:
        nThreads - the number of threads in the pool
    Returns:
        the newly created thread pool
    fi Throws:
    fi IllegalArgumentException - if nThreads <= 0
    }
    verify(mock);
}
```

# Root cause

---

*//In org.easymock.internal.ReplayState*

```
public Object invoke(final Invocation invocation)
                                throws Throwable {

    behavior.checkThreadSafety();
    if (behavior.isThreadSafe()) {
        // If thread safe, synchronize the mock
        lock.lock();
        try {
            return invokeInner(invocation);
        }
    }

    return invokeInner(invocation);
}
```

---

(very short intro.)

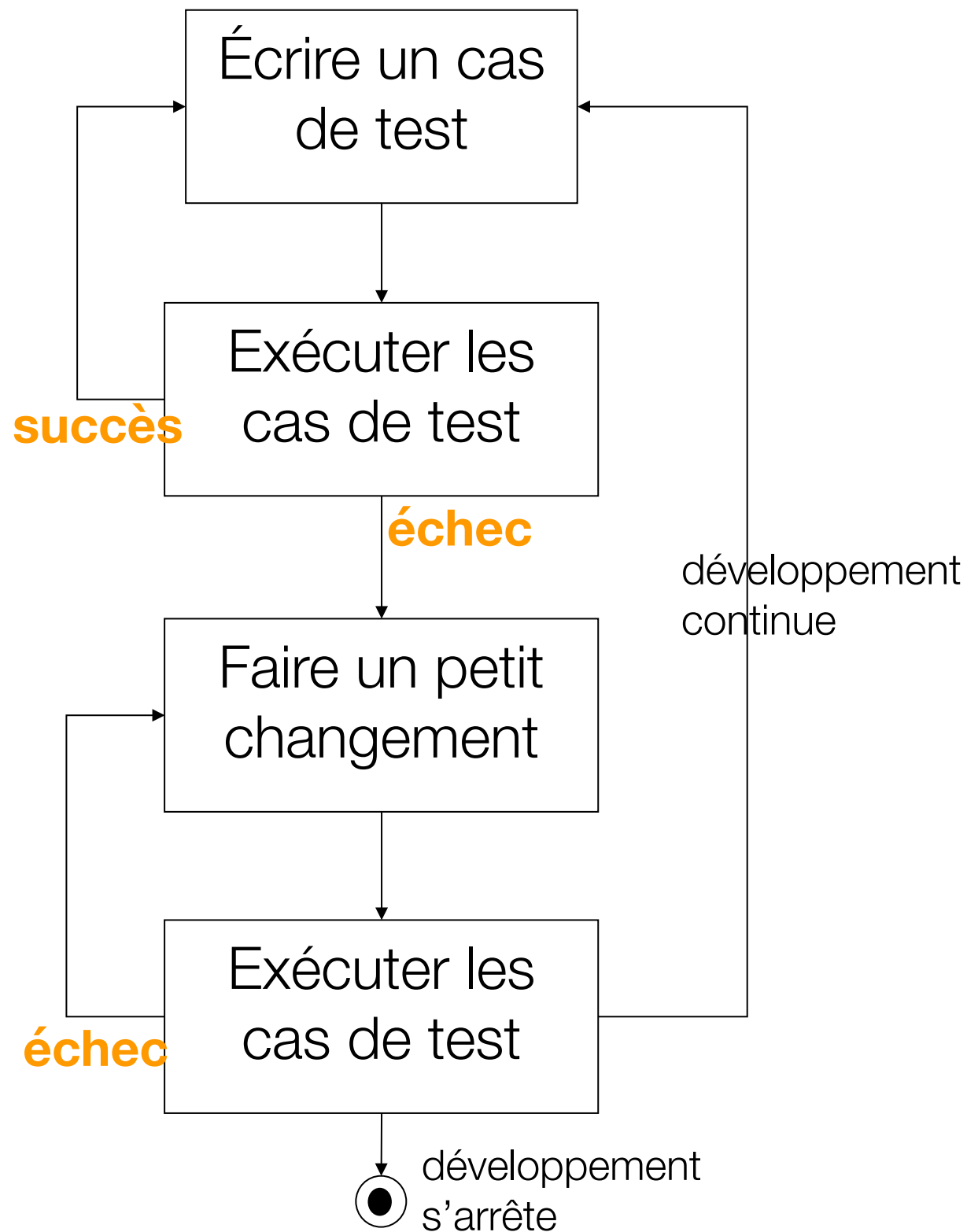
# TEST-DRIVEN DEVELOPMENT

# Test-first development

---

- Xtreme programming
- Écrire les cas de test avant le programme
  - les cas de test décrivent ce que le programme doit faire
  - avant d'écrire le code, les cas de test échouent
  - quand on développe, les cas de test doivent passer
- Ensuite on développe la partie du programme qui fait passer le cas de test

# Test-first development



Exemple : ajout dans une liste chaînée

```
public void testAdd(){
    list.add("first");
    assertTrue(list.size()==1);
}

public void add (String it){
    public void add (String it){
        Node node = new Node();
        node.setItem(it);
        node.setNext(null);
        if (firstNode == null) {
            node.setPrevious(null);
            this.setFirstNode(node);
        }
        lastNode = node;
        this.setCurrentNode(node);
    }
}
```



# Test-first development

---

- Les cas de test servent de support à la documentation
  - des exemples d'utilisation du code
- Tester avec une intention précise
  - qu'est-ce qu'on teste?
  - pourquoi on le teste?
  - quand est-ce assez testé?
- Retours rapides sur la qualité du code
  - itérations courtes dans le cycle de développement
  - on exécute le code tout de suite (avant même de l'avoir écrit)
  - On ne code que quand un test a échoué

# Test-first development

---

- Les cas de test spécifient ce que le programme doit faire mais pas comment
  - il faut associer TDD à des refactorings fréquents
    - revoir la structure du code
    - ne pas oublier la conception
- Grande importance du test de non-régression
  - quand on refactorise les cas de test qui passaient doivent continuer à passer

# Test-first development

---

- Importance d'un environnement pour l'exécution automatique des tests
  - pouvoir exprimer facilement des test unitaires
  - pouvoir les exécuter rapidement
  - pouvoir réexécuter les cas de test
- JUnit + d'autres outils pour automatiser le test et permettre TDD

- 
- <http://pmd.sourceforge.net/pmd-4.3.0/rules/junit.html>
    - **JUnitTestsShouldIncludeAssert**
    - **TestClassWithoutTestCases**
    - **UnnecessaryBooleanAssertion**
    - ...

---

```
private final ByteArrayOutputStream outContent = new ByteArrayOutputStream();
private final ByteArrayOutputStream errContent = new ByteArrayOutputStream();
```

```
@Before
```

```
public void setUp() throws Exception {
    System.setOut(new PrintStream(outContent));
    System.setErr(new PrintStream(errContent));
}
```

```
@Test
```

```
public void testOK() throws FileNotFoundException {
    try {
        File f = new File("ok.text");
        //FileOutputStream s = new FileOutputStream(f);
        FileWriter fstream = new FileWriter(f);
        BufferedWriter out = new BufferedWriter(fstream);
        out.write("4;4");
        out.close();
        Stream s = new Stream(f.getName());
        s.readData();
        assertEquals("1.0\n", outContent.toString());
    } catch (Exception e) {
        System.err.println("Error: " + e.getMessage());
    }
}
```

---

```
@Test
public void testnOK() throws FileNotFoundException {
    try {
        File f = new File("ok.text");
        //FileOutputStream s = new FileOutputStream(f);
        FileWriter fstream = new FileWriter(f);
        BufferedWriter out = new BufferedWriter(fstream);
        out.write("4,4");
        out.close();
        Stream s = new Stream(f.getName());
        s.readData();
        assertNotNull(errContent);
    } catch (Exception e) {
        System.err.println("Error: " + e.getMessage());
    }
}
```