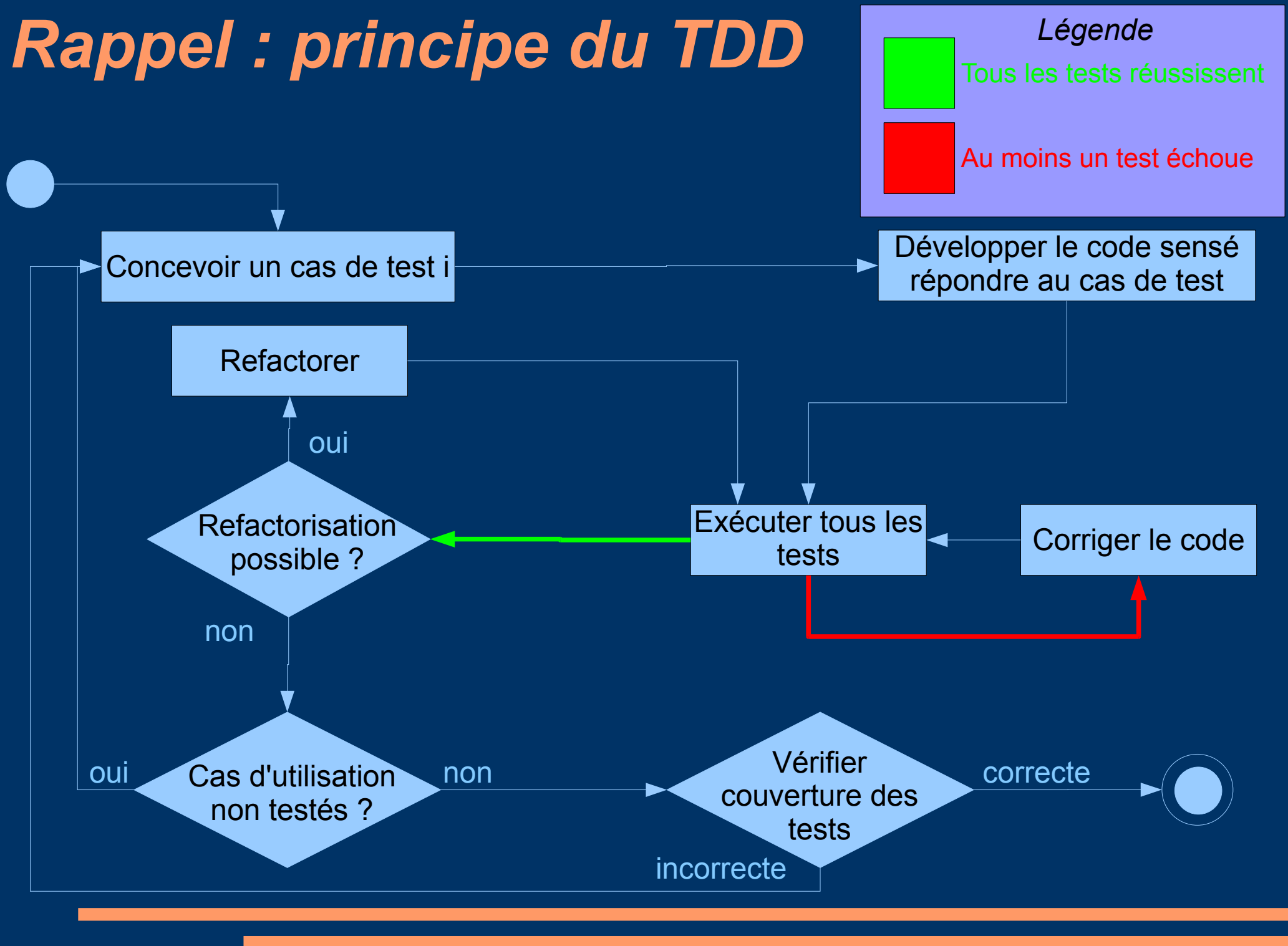
Test Driven Development (TDD) appliqué au problème du triangle

Bruno Mermet

2010

Rappel : principe du TDD



Le problème du Triangle (The Art of Software Testing)

- Énoncé

  Écrire en java un programme qui demande à l'utilisateur de rentrer 3 données (des entiers) correspondant aux longueurs des côtés d'un triangle. Le programme doit alors préciser si le triangle est équilatéral, isocèle ou scalène.

- Contraintes
  - Procéder en utilisant du TDD (Test-Driven Development)
  - Définir un code fonctionnant sans lecture clavier

Profil de la méthode

- Pour que le contrôle des paramètres soit testable, la méthode principale devra prendre directement en paramètre ce qu'on peut lire au clavier, à savoir des chaînes de caractères
- Le type de triangle sera rendu sous la forme d'une valeur dans un type énuméré.

```
package tdd;


public class Triangle {
    public enum Type {EQUILATERAL,
ISOCELE, SCALENE};

    public static Type
typeTriangle(String cote1, String
cote2, String cote3) {
        return Type.EQUILATERAL;
    }
}
```

On choisit une valeur quelconque, juste pour que la compilation ait lieu

Squelette de la classe de test

```
package tdd;

import org.junit.Test;
import static org.junit.Assert.*;
import tdd.Triangle.Type;

public class TriangleTest {

    public TriangleTest() {
    }

    @Test
    public void testTypeTriangleScalene() {
        fail("A implanter");
    }
}
```

Cas de test 1

- « 2, 3, 4 » est un triangle scalène

```
    @Test
    public void testTypeTriangleScalene() {
        System.out.println("typeTriangle");
        String cote1 = "2";
        String cote2 = "3";
        String cote3 = "4";
        Type expResult = Type.SCALENE;
        Type result =
Triangle.typeTriangle(cote1, cote2,
cote3);
        assertEquals(expResult, result);
    }
```

Cas de test 1 : correction

```
public static Type typeTriangle(String
cote1, String cote2, String cote3) {
        return Type.SCALENE;
    }
```

Cas de test 1 : nouvelle exécution

```
    @Test
    public void testTypeTriangleScalene() {
        System.out.println("typeTriangle");
        String cote1 = "2";
        String cote2 = "3";
        String cote3 = "4";
        Type expResult = Type.SCALENE;
        Type result =
Triangle.typeTriangle(cote1, cote2,
cote3);
        assertEquals(expResult, result);
    }
```

Cas de test 2

- « toto, 3, 4 » n'est pas un triangle

```
@Test(expected=NumberFormatException.class)
public void testTypeTriangleErreur() throws Exception {
    System.out.println("typeTriangle erreur");
    String cote1 = "toto";
    String cote2 = "3";
    String cote3 = "4";
    Triangle.typeTriangle(cote1, cote2, cote3);
}
```

Cas de test 2 : correction

```
public static Type typeTriangle(String cote1, String cote2, String cote3) throws
NumberFormatException {
        int longueurCote1;
        longueurCote1 = Integer.parseInt(cote1);

        return Type.SCALENE;
    }
```

Cas de test 3

- « -1, 3, 4 » n'est pas un triangle

```
@Test(expected=NumberFormatException.class)
public void testTypeTriangleErreur2() throws Exception {
    System.out.println("typeTriangle erreur");
    String cote1 = "-1";
    String cote2 = "3";
    String cote3 = "4";
    Triangle.typeTriangle(cote1, cote2, cote3);
}
```

Cas de test 3 : correction

```
public static Type typeTriangle(String cote1, String cote2, String cote3) throws
NumberFormatException {
        int longueurCote1;
        longueurCote1 = Integer.parseInt(cote1);
        if (longueurCote1 <= 0) {
            throw new NumberFormatException();
        }

        return Type.SCALENE;
    }
```

Cas de test 4

- « 3, -1, 4 » n'est pas un triangle

```
@Test(expected=NumberFormatException.class)
public void testTypeTriangleErreur3() throws Exception {
    System.out.println("typeTriangle erreur");
    String cote1 = "3";
    String cote2 = "-1";
    String cote3 = "4";
    Triangle.typeTriangle(cote1, cote2, cote3);
}
```

Cas de test 4 : correction

```
public static Type typeTriangle(String cote1, String cote2, String cote3) throws
NumberFormatException {
        int longueurCote1;
        longueurCote1 = Integer.parseInt(cote1);
        if (longueurCote1 <= 0) {
            throw new NumberFormatException();
        }
        int longueurCote2;
        longueurCote2 = Integer.parseInt(cote2);
        if (longueurCote2 <= 0) {
            throw new NumberFormatException();
        }

        return Type.SCALENE;
    }
```

Cas de test 4 : correction ! Redondance !

```
public static Type typeTriangle(String cote1, String cote2, String cote3) throws
NumberFormatException {
        int longueurCote1;
        longueurCote1 = Integer.parseInt(cote1);
        if (longueurCote1 <= 0) {
            throw new NumberFormatException();
        }
        int longueurCote2;
        longueurCote2 = Integer.parseInt(cote2);
        if (longueurCote2 <= 0) {
            throw new NumberFormatException();
        }

        return Type.SCALENE;
    }
```

Cas de test 4 : correction Refactorisation

```
    public static Type typeTriangle(String cote1, String cote2, String cote3) throws
NumberFormatException {
        int longueurCote1 = convertitLongueurSiValide(cote1);
        int longueurCote2 = convertitLongueurSiValide(cote2);
        int longueurCote3 = convertitLongueurSiValide(cote3);
        return Type.SCALENE;
    }

    private static int convertitLongueurSiValide(String longueurTexte) throws
NumberFormatException{
        int longueur;
        longueur = Integer.parseInt(longueurTexte);
        if (longueur <= 0) {
            throw new NumberFormatException();
        }
        return longueur;
    }
```

Les tests passent, notre « refactorisation » est validée

Structure du code modifée
Refactoriser les cas de test

Principe

- Sur la méthode « type triangle », on ne garde qu'un test avec une chaîne et un test avec une valeur négative, pour des arguments différents
- On rajoute des tests pour la nouvelle méthode
  - 2 valeurs entières correctes
  - 1 valeur « texte »
  - 1 valeur négative
  - 1 valeur nulle

Nouvelle liste de tests (1)

```
@Test
public void testTypeTriangleScalene() {
    System.out.println("typeTriangle scalène");
    String cote1 = "2"; String cote2 = "3"; String cote3 = "4";
    Type expResult = Type.SCALENE; Type result = Triangle.typeTriangle(cote1, cote2, cote3);
    assertEquals(expResult, result);
}
@Test(expected=NumberFormatException.class)
public void testTypeTriangleErreur() throws Exception {
    System.out.println("typeTriangle erreur");
    String cote1 = "toto"; String cote2 = "3"; String cote3 = "4";
    Triangle.typeTriangle(cote1, cote2, cote3);
}
@Test(expected=NumberFormatException.class)
public void testTypeTriangleErreur2() throws Exception {
    System.out.println("typeTriangle erreur");
    String cote1 = "1";
    String cote2 = "-3";
    String cote3 = "4";
    Triangle.typeTriangle(cote1, cote2, cote3);
}
```

Nouvelle liste de tests (2)

```
@Test
public void testConvertitLongueur() throws Exception {
    System.out.println("convertitLongueur");
    int resultatAttendu = 4;
    int resultat = Triangle.convertitLongueurSiValide("4");
    assertEquals(resultatAttendu,resultat);
    int resultatAttendu2 = 100;
    int resultat2 = Triangle.convertitLongueurSiValide("100");
    assertEquals(resultatAttendu2,resultat2);
}
@Test(expected=NumberFormatException.class)
public void testConvertitLongueurErreur1() throws Exception {
    System.out.println("convertitLongueur");
    Triangle.convertitLongueurSiValide("toto");
}
@Test(expected=NumberFormatException.class)
public void testConvertitLongueurErreur3() throws Exception {
    System.out.println("convertitLongueur");
    Triangle.convertitLongueurSiValide("-1");
}
@Test(expected=NumberFormatException.class)
public void testConvertitLongueurErreur2() throws Exception {
    System.out.println("convertitLongueur");
    Triangle.convertitLongueurSiValide("0");
}
```

Nos 7 cas de test passent

Cas de test 5

- « 3, 3, 3 » est équilatéral

```
@Test
 public void testTypeTriangleEquilateral() {
     System.out.println("typeTriangle scalène");
     String cote1 = "3";
     String cote2 = "3";
     String cote3 = "3";
     Type expResult = Type.EQUILATERAL;
     Type result = Triangle.typeTriangle(cote1, cote2, cote3);
     assertEquals(expResult, result);
 }
```

Cas de test 5 : correction

```
    public static Type typeTriangle(String cote1, String cote2, String cote3) throws
NumberFormatException {
        int longueurCote1 = convertitLongueurSiValide(cote1);
        int longueurCote2 = convertitLongueurSiValide(cote2);
        int longueurCote3 = convertitLongueurSiValide(cote3);
        if (longueurCote1 == longueurCote2 && longueurCote2 == longueurCote3) {
            return Type.EQUILATERAL;
        }
        return Type.SCALENE;
    }
```

Cas de test 5 : méthode à 2 rôles => Refactorisation !

- La méthode
  - Convertit les longueurs en entiers
  - Détermine le type du triangle
- 2 rôles => 2 méthodes

```
    public static Type typeTriangle(String cote1, String cote2, String cote3) throws
NumberFormatException {
        int longueurCote1 = convertitLongueurSiValide(cote1);
        int longueurCote2 = convertitLongueurSiValide(cote2);
        int longueurCote3 = convertitLongueurSiValide(cote3);
        if (longueurCote1 == longueurCote2 && longueurCote2 == longueurCote3) {
            return Type.EQUILATERAL;
        }
        return Type.SCALENE;
    }
```

Cas de test 5 : Refactorisation

```
    public static Type typeTriangle(String cote1, String cote2, String cote3) throws
NumberFormatException {
        int longueurCote1 = convertitLongueurSiValide(cote1);
        int longueurCote2 = convertitLongueurSiValide(cote2);
        int longueurCote3 = convertitLongueurSiValide(cote3);
        return typeTriangle(longueurCote1, longueurCote2, longueurCote3);
    }

    protected static Type typeTriangle(int cote1, int cote2, int cote3)  {
        if (cote1 == cote2 && cote2 == cote3) {
            return Type.EQUILATERAL;
        }
        return Type.SCALENE;
    }
```

Les tests passent, notre refactorisation est validée

Cas de test 5 : Modification des cas de test ?

La structure du code a changé

- Modification des cas de test ?
- Non étant donné que la nouvelle méthode est entièrement testée par l'ancienne sans changement sur la pertinence des tests

Cas de test 5 : condition compliquée => Refactorisation !

```
    public static Type typeTriangle(String cote1, String cote2, String cote3) throws
NumberFormatException {
        int longueurCote1 = convertitLongueurSiValide(cote1);
        int longueurCote2 = convertitLongueurSiValide(cote2);
        int longueurCote3 = convertitLongueurSiValide(cote3);
        return typeTriangle(longueurCote1, longueurCote2, longueurCote3);
    }

    protected static Type typeTriangle(int cote1, int cote2, int cote3)  {
        if (cote1 == cote2 && cote2 == cote3) {
            return Type.EQUILATERAL;
        }
        return Type.SCALENE;
    }
```

Cas de test 5 : Refactorisation

```
public static Type typeTriangle(String cote1, String cote2, String cote3) throws
NumberFormatException {
        int longueurCote1 = convertitLongueurSiValide(cote1);
        int longueurCote2 = convertitLongueurSiValide(cote2);
        int longueurCote3 = convertitLongueurSiValide(cote3);
        return typeTriangle(longueurCote1, longueurCote2, longueurCote3);
    }

    protected static Type typeTriangle(int cote1, int cote2, int cote3)  {
        if (estEquilateral(cote1,cote2,cote3)) {
            return Type.EQUILATERAL;
        }
        return Type.SCALENE;
    }

    protected static boolean estEquilateral(int c1, int c2, int c3) {
        return c1 == c2 && c2 == c3;
    }
```

Les tests passent, notre refactorisation est validée
(idem, on ne modifie pas les tests)

Cas de test 6

- « 3, 3, 4 » est isocèle

```
@Test
 public void testTypeTriangleEquilateral() {
     System.out.println("typeTriangle scalène");
     String cote1 = "3";
     String cote2 = "3";
     String cote3 = "4";
     Type expResult = Type.ISOCELE;
     Type result = Triangle.typeTriangle(cote1, cote2, cote3);
     assertEquals(expResult, result);
 }
```

Cas de test 6 : correction

```
protected static Type typeTriangle(int cote1, int cote2, int cote3)  {
    if (estEquilateral(cote1,cote2,cote3)) {
        return Type.EQUILATERAL;
    }
    else if (estIsoceleSiNonEquilateral(cote1, cote2, cote3)) {
        return Type.ISOCELE;
    }
    return Type.SCALENE;
}

protected static boolean estIsoceleSiNonEquilateral(int c1, int c2, int c3) {
    return c1 == c2;
}
```

Cas de test 7

- « 3, 4, 3 » et « 4, 3, 3 » sont isocèles

```
@Test
 public void testTypeTriangleIsocele() {
    System.out.println("typeTriangle scalène");
    String cote1 = "3"; String cote2 = "3"; String cote3 = "4";
    Type expResult = Type.ISOCELE;
    Type result = Triangle.typeTriangle(cote1, cote2, cote3);
    assertEquals(expResult, result);
    String cote1a = "3"; String cote2a = "4"; String cote3a = "3";
    Type expResulta = Type.ISOCELE;
    Type resulta = Triangle.typeTriangle(cote1a, cote2a, cote3a);
    assertEquals(expResulta, resulta);
    String cote1b = "4"; String cote2b = "3"; String cote3b = "3";
    Type expResultb = Type.ISOCELE;
    Type resultb = Triangle.typeTriangle(cote1b, cote2b, cote3b);
    assertEquals(expResultb, resultb);
 }
```

Cas de test 7 : correction

```
protected static boolean estIsoceleSiNonEquilateral(int c1, int c2, int c3) {
    return c1 == c2 || c1 == c3 || c2 == c3;
}
```

Cas de test 8, 9 et 10

Et Quid de « 2,3,10 », « 2,10,2 » ou encore « 2,3,5 » ?
Ce ne sont pas des triangles !

```
@Test(expected=Exception.class)
public void testTypeNonTriangle1() throws Exception {
    System.out.println("non Triangle");
    String cote1 = "2"; String cote2 = "3"; String cote3 = "10";
    Triangle.typeTriangle(cote1, cote2, cote3);
}
@Test(expected=Exception.class)
public void testTypeNonTriangle2() throws Exception {
    System.out.println("non Triangle");
    String cote1 = "2"; String cote2 = "10"; String cote3 = "2";
    Triangle.typeTriangle(cote1, cote2, cote3);
}
@Test(expected=Exception.class)
public void testTypeNonTriangle3() throws Exception {
    System.out.println("non Triangle");
    String cote1 = "2"; String cote2 = "3"; String cote3 = "5";
    Triangle.typeTriangle(cote1, cote2, cote3);
}
```

Cas de test 8 : correction

```
protected static Type typeTriangle(int cote1, int cote2, int cote3)  {
    int[] cotes = new int[3];
    cotes[0] = cote1;
    cotes[1] = cote2;
    cotes[2] = cote3;
    Arrays.sort(cotes);
    if (cotes[2] >= cotes[0]+cotes[1]) {
        throw new RuntimeException("Pas un triangle");
    }
    if (estEquilateral(cote1,cote2,cote3)) {
        return Type.EQUILATERAL;
    }
    else if (estIsoceleSiNonEquilateral(cote1, cote2, cote3)) {
        return Type.ISOCELE;
    }
    return Type.SCALENE;
}
```

Refactorisation

- Problème

  De nombreuses méthodes s'appliquent à nos longueurs de côtés, nécessitant un passage de 3 paramètres, ce qui fait beaucoup

- Solution

  Passer par une instance de la classe Triangle, avec les longueurs stockées dans des variables d'instance

Code refactorisé (1)

```
public class Triangle {
    public enum Type {EQUILATERAL, ISOCELE, SCALENE};

    private int[] cotesTries;

    protected Triangle(int cote1, int cote2, int cote3) {
        cotesTries = new int[3]; cotesTries[0] = cote1; cotesTries[1] = cote2; cotesTries[2] = cote3;
        Arrays.sort(cotesTries);
    }

    public static Type typeTriangle(String cote1, String cote2, String cote3) throws RuntimeException {
        int longueurCote1 = convertitLongueurSiValide(cote1);
        int longueurCote2 = convertitLongueurSiValide(cote2);
        int longueurCote3 = convertitLongueurSiValide(cote3);
        Triangle t = new Triangle(longueurCote1, longueurCote2, longueurCote3);
        return t.typeTriangle();
    }

    protected Type typeTriangle()  {
        if (cotesTries[2] >= cotesTries[0]+cotesTries[1]) {
            throw new RuntimeException("Pas un triangle");
        }
        if (estEquilateral()) {return Type.EQUILATERAL;}
        else if (estIsoceleSiNonEquilateral()) {return Type.ISOCELE;}
        return Type.SCALENE;
    }
```

Code refactorisé (2)

```
    protected boolean estEquilateral() {
        return cotesTries[0] == cotesTries[1] && cotesTries[1] == cotesTries[2];
    }

    protected boolean estIsoceleSiNonEquilateral() {
        return cotesTries[0] == cotesTries[1] || cotesTries[0] == cotesTries[2] || cotesTries[1] ==
cotesTries[2];
    }

    protected static int convertitLongueurSiValide(String longueurTexte) throws
NumberFormatException{
        int longueur;
        longueur = Integer.parseInt(longueurTexte);
        if (longueur <= 0) {
            throw new NumberFormatException();
        }
        return longueur;
    }
}
```

Tous les tests passent, validant notre refactorisation

Refactorisation : simplification

- Nouvelle hypothèse rajoutée par la dernière version du code

  Les côtés sont triés

- Application

  Les recherches des triangles équilatéraux et isocèles peuvent être simplifiées

Code refactorisé

Avant

```
    protected boolean estEquilateral() {
        return cotesTries[0] == cotesTries[1] && cotesTries[1] == cotesTries[2];
    }

    protected boolean estIsoceleSiNonEquilateral() {
        return cotesTries[0] == cotesTries[1] || cotesTries[0] == cotesTries[2] || cotesTries[1] ==
cotesTries[2];
    }
```

Après

```
    protected boolean estEquilateral() {
        return cotesTries[0] == cotesTries[2];
    }

    protected boolean estIsoceleSiNonEquilateral() {
        return cotesTries[0] == cotesTries[1] || cotesTries[1] == cotesTries[2];
    }
```

Tous les tests passent, validant notre refactorisation

Fin ?

- Plus de cas de tests envisagés
- Test de couverture
  - Classe tdd.Triangle, 100% (26 / 26)
- Écriture du programme principal
  - Problème : rendre le programme principal testable
  - Solution
    - Permettre de modifier l'entrée
    - Permettre de modifier la sortie

Classe MainTriangle

```
package tdd;

import java.io.FileNotFoundException; import java.util.Scanner; import tdd.Triangle.Type;
public class MainTriangle {
    private static Scanner s = new Scanner(System.in);
    private static PrintStreamString sortie;

    public static void main(String[] args) {
        String cote1 = s.nextLine(); String cote2 = s.nextLine(); String cote3 = s.nextLine();
        Type type = Triangle.typeTriangle(cote1, cote2, cote3);
        switch(type) {
            case EQUILATERAL:
                System.out.println("votre triangle est équilatéral"); break;
            case ISOCELE:
                System.out.println("votre triangle est isocèle"); break;
            case SCALENE:
                System.out.println("votre triangle est scalène"); break;
        }
    }
    static void setScanner(String chaine) {s = new Scanner(chaine);}
    static void setSortie() throws FileNotFoundException{
        sortie = new PrintStreamString();
        System.setOut(sortie);
    }
    static String getAffichage() {
        return sortie.getText();
    }
}
```

Classe PrintStreamString

```
package tdd;

import java.io.FileNotFoundException;
import java.io.PrintStream;

public class PrintStreamString extends PrintStream {
    private String text;
    public PrintStreamString() throws FileNotFoundException {
        super("FichierTest");
    }

    @Override
    public void println(String chaine) {
        text = chaine;
    }

    protected String getText() {
        return text;
    }
}
```

Remarque

Cette classe ne permet de ne tester qu'une ligne de sortie. Tester plusieurs lignes aurait pu être implanté :

- soit avec un tableau de String, voire de StringBuilder
- soit en passant directement par un fichier en redéfinissant une méthode renvoyant sous la forme d'une chaîne le contenu du fichier

Classe MainTriangleTest

```
package tdd;

import java.io.FileNotFoundException;
import org.junit.Test;
import static org.junit.Assert.*;

public class MainTriangleTest {

    @Test
    public void testMain() throws FileNotFoundException {
        System.out.println("main");
        String[] args = null;
        MainTriangle.setScanner("3\n4\n5\n");
        MainTriangle.setSortie();
        MainTriangle.main(args);
        assertEquals("votre triangle est scalène",MainTriangle.getAffichage());
        MainTriangle.setScanner("3\n3\n3\n");
        MainTriangle.main(args);
        assertEquals("votre triangle est équilatéral",MainTriangle.getAffichage());
        MainTriangle.setScanner("3\n4\n3\n");
        MainTriangle.main(args);
        assertEquals("votre triangle est isocèle",MainTriangle.getAffichage());
    }
}
```

Cas de test 11 et 12

Et si l'utilisateur rentre des données non valides ou un non-triangle ?

```
@Test
public void testMainInvalide() throws FileNotFoundException {
    System.out.println("main");
    String[] args = null;
    MainTriangle.setScanner("toto\n4\n5\n");
    MainTriangle.setSortie();
    MainTriangle.main(args);
    assertEquals("la longueur du côté 1 n'est pas valide",MainTriangle.getAffichage());
}
@Test
public void testMainNontriangle() throws FileNotFoundException {
    System.out.println("main");
    String[] args = null;
    MainTriangle.setScanner("1\n2\n3\n");
    MainTriangle.setSortie();
    MainTriangle.main(args);
    assertEquals("il ne s'agit pas d'un triangle",MainTriangle.getAffichage());
}
```

Les 2 tests génèrent des erreurs !

Cas de test 11 & 12

- Problème
  - Notre code renvoie une exception non traitée
  - Notre exception est trop générale pour être utilisable
- Solution : Refactoring
  - On va créer nos propres exceptions

Refactorisation
Nouvelles classes d'exception

```
package tdd;
public class TriangleException extends RuntimeException {
    public TriangleException(String texte) {
        super(texte);
    }
}
```

```
package tdd;
public class CoteInvalideException extends TriangleException {
    public CoteInvalideException(int n) {
        super("la longueur du côté "+n+ " n'est pas valide");
    }
}
```

```
package tdd;
public class NonTriangleException extends TriangleException {
    public NonTriangleException() {
        super("il ne s'agit pas d'un triangle");
    }
}
```

Refactorisation
Modification de la classe Triangle

```
public static Type typeTriangle(String cote1, String cote2, String cote3) throws TriangleException {
        int longueurCote1 = convertitLongueurCoteN(cote1,1);
        int longueurCote2 = convertitLongueurCoteN(cote2,2);
        int longueurCote3 = convertitLongueurCoteN(cote3,3);
        Triangle t = new Triangle(longueurCote1, longueurCote2, longueurCote3);
        return t.typeTriangle();
    }
protected static int convertitLongueurCoteN(String longueurTexte, int n) throws
CoteInvalideException {
        int longueur;
        try {
            longueur = Integer.parseInt(longueurTexte);
            if (longueur <= 0) {throw new NumberFormatException();}
        } catch (NumberFormatException nfe) {throw new CoteInvalideException(n);}
        return longueur;
    }
protected Type typeTriangle()  {
        if (cotesTries[2] >= cotesTries[0]+cotesTries[1]) {throw new NonTriangleException();}
        if (estEquilateral()) {return Type.EQUILATERAL;}
        else if (estIsoceleSiNonEquilateral()) {return Type.ISOCELE;}
        return Type.SCALENE;
    }
```

2 tests génèrent des erreurs : on attendait des NumberFormatException

Modification des cas de test

```
@Test(expected=CoteInvalideException.class)
 public void testTypeTriangleErreur() throws Exception {
    System.out.println("typeTriangle erreur");
    String cote1 = "toto";
    String cote2 = "3";
    String cote3 = "4";
    Triangle.typeTriangle(cote1, cote2, cote3);
 }

 @Test(expected=CoteInvalideException.class)
 public void testTypeTriangleErreur2() throws Exception {
    System.out.println("typeTriangle erreur");
    String cote1 = "1";
    String cote2 = "-3";
    String cote3 = "4";
    Triangle.typeTriangle(cote1, cote2, cote3);
 }
```

Tous les cas de test de Triangle passent

On a encore des échecs pour MainTriangle !!

Classe MainTriangle corrigée

```
package tdd;
import java.io.FileNotFoundException; import java.util.Scanner; import tdd.Triangle.Type;
public class MainTriangle {
    private static Scanner s = new Scanner(System.in); private static PrintStreamString sortie;

    public static void main(String[] args) {
        String cote1 = s.nextLine(); String cote2 = s.nextLine(); String cote3 = s.nextLine();
        try {
            Type type = Triangle.typeTriangle(cote1, cote2, cote3);
            switch (type) {
                case EQUILATERAL: System.out.println("votre triangle est équilatéral");break;
                case ISOCELE: System.out.println("votre triangle est isocèle");break;
                case SCALENE: System.out.println("votre triangle est scalène");break;
            }
        } catch (TriangleException te) {
            System.out.println(te.getMessage());
        }
    }
    static void setScanner(String chaine) {s = new Scanner(chaine);}
    static void setSortie() throws FileNotFoundException{sortie = new PrintStreamString();System.setOut(sortie);}
    static String getAffichage() {return sortie.getText();}
}
```

Tous les tests passent !

Tests de couverture

tdd.CoteInvalideException	100 %	(2 / 2)
tdd.MainTriangle	91 %	(21 / 23)
tdd.NonTriangleException	100 %	(2 / 2)
tdd.PrintStreamString	100 %	(5 / 5)
tdd.Triangle	100 %	(33 / 33)
tdd.TriangleException	100 %	(2 / 2)

- tdd.MainTriangle : pourquoi pas 100% ?
  - Le dernier break du *switch* est court-circuité
  - Le constructeur par défaut n'est pas appelé puisqu'aucune instance n'est créée