Algorithmique Programmation FIP

Langages et Techniques de Programmation

ING39

Septembre 2016

Debrief Tp1

La spécification (contrat) javadoc donnée

```
/**
 * Renvoie un tableau ayant les cases de t
 * decalees d'un indice a droite, sauf la derniere
 * case, mise au debut du tableau resultat.
 * Si t est null, leve une IllegalArgumentException ;
 * si t est de taille 0, renvoie un tableau de taille 0
 * <p> Pre-condition: t different de null </p>
 * @param t le tableau sur lequel se fera le decalage
 * @return un nouveau tableau avec les cases de t decal
 * la derniere mise au debut.
 * @throws IllegalArgumentException si t est null
 */
public static int [] decaleDroite(int [] t) {
      return null;
}
```

Quels tests pour cette spécification ?

t	résultat attendu	cas testé
`null`	`IllegalArgumentException`	tableau nul
{ }	{ }	tableau de taille zéro
{1}	{1}	tableau à une case
{ 1, 5}	{ 5,1}	tableau à deux cases
{ 2,5,9}	{9, 2, 5}	tableau à 3 cases
{ 2, 4, 50, 17, 9}	{ 9, 2, 4, 50, 17}	tableau à plus de 3 cases

Schéma de test généré par Netbeans

```
/**
 * Test of decaleDroite method, of class ShiftTableau.
@Test
 public void testDecaleDroite() {
     System.out.println("decaleDroite");
     int[] t = null;
     int[] expResult = null;
     int[] result = ShiftTableau.decaleDroite(t);
     assertArrayEquals(expResult, result);
     // TODO review the generated test code and remove t
     // default call to fail.
    fail("The test case is a prototype.");
}
```

Cette méthode échoue toujours : à remplacer par un vrai cas de test.

Cas sur tableau de taille un

```
/**
 * Test sur tableau avec une seule case.
 */
 @Test
  public void testDecaleDroiteSingleton() {
      int[] t = {1};
      int[] expResult = {1};
      int[] result = ShiftTableauOK.decaleDroite(t);
      assertArrayEquals(expResult, result);
  }
```

On donne un nom de méthode qui renseigne sur le cas de test.

On ajoute un commentaire qui le décrit.

Cas d'un tableau de taille zéro

```
/**
 * Test sur tableau de taille 0.
 */
@Test
 public void testDecaleDroiteTaille0() {
      int[] t = new int[0];
      int[] expResult = t;
      int[] result = ShiftTableauOK.decaleDroite(t);
      assertArrayEquals(expResult, result);
 }
```

On donne un nom de méthode qui renseigne sur le cas de test.

On ajoute un commentaire qui le décrit.

Cas d'un tableau null

```
/**
* Test avec t=null.
*/
@Test(expected=IllegalArgumentException.class)
 public void testDecaleDroiteNul() {
      int[] t = null;
      ShiftTableauOK.decaleDroite(t);
  }
```

Une exception est attendue dans ce cas. Ne pas oublier d'ajouter `.class`

Cas tableau à 2 et 3 cases

```
/* Test sur tableau deux cases. */
 @Test
  public void testDecaleDroiteDeux() {
      int[] t = {1,5};
      int[] expResult = {5,1};
      int[] result = ShiftTableauOK.decaleDroite(t);
      assertArrayEquals(expResult, result);
  }
  /* Test of sur tableau 3 cases.  */
 @Test
  public void testDecaleDroiteTrois() {
      int[] t = {2,5, 9};
      int[] expResult = {9,2,5};
      int[] result = ShiftTableauOK.decaleDroite(t);
      assertArrayEquals(expResult, result);
  }
```

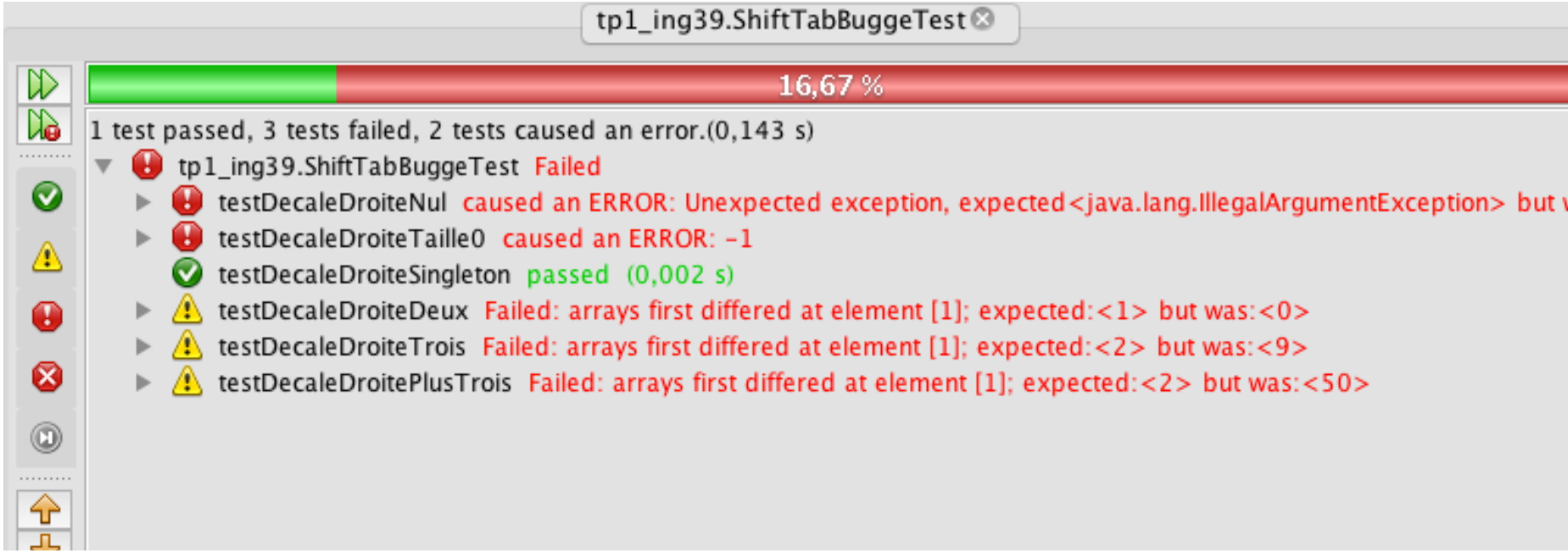
Cas tableau à plus de 3 cases

```
/**
 * Test sur tableau a plus de 3 cases.
 */
@Test
public void testDecaleDroitePlusTrois() {
    int[] t = {2,4,50,17,9};
    int[] expResult = {9,2,4,50,17};
    int[] result = ShiftTableauOK.decaleDroite(t);
    assertArrayEquals(expResult, result);
}
```

Une implantation buggée

```
public static int [] decaleDroite(int [] t) {
    int [] nouveau = new int [t.length];
    for (int i=0; i<t.length-1; i++){
         nouveau[i] = t[i+1];
    }
    nouveau[0]=t[t.length-1];
    return nouveau;
}
```

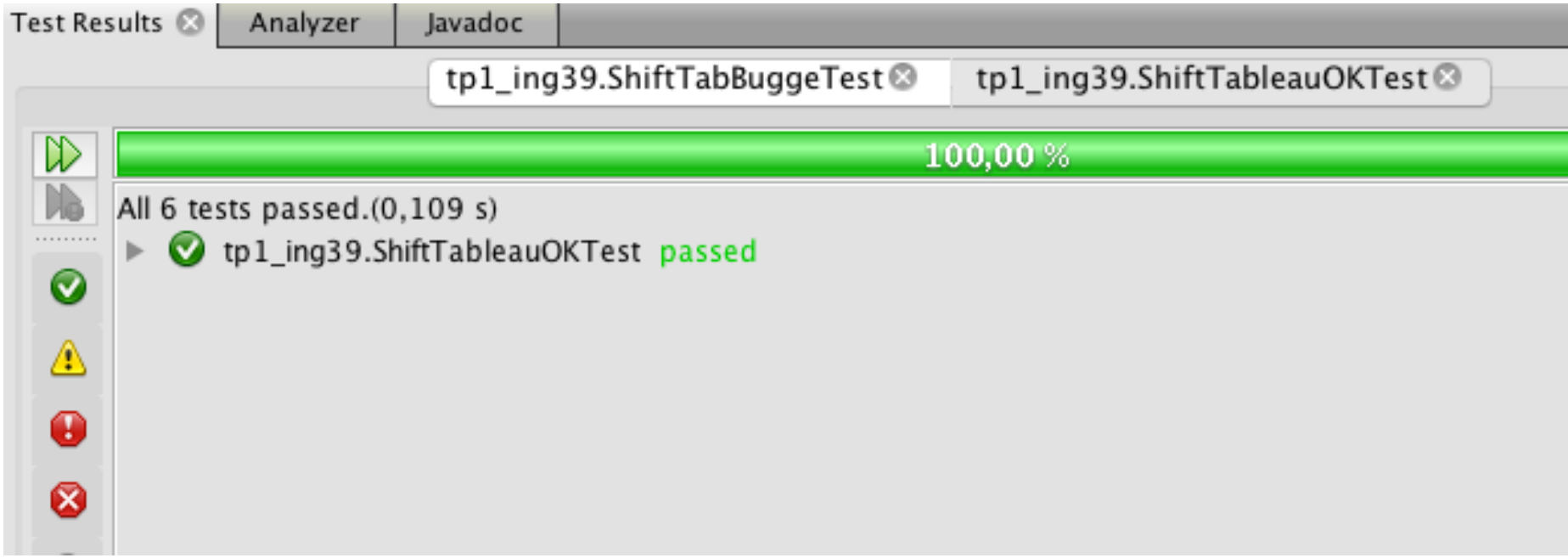
Tests sur implantation buggée



Une implantation correcte

```
public static int [] decaleDroite(int [] t) {
    if (t==null)
         throw new IllegalArgumentException();
    int [] nouveau = new int [t.length];
    if (t.length==0)
          return nouveau;
    nouveau[0] = t[t.length-1];
    for (int i=0; i<=t.length-2; i++){
         nouveau[i+1] = t[i];
    }
    return nouveau;
}
```

Tests sur implantation correcte



Autre exemple : sommeTab

```
/**
 * Retourne un tableau dont les composantes sont
 * obtenues par addition des composantes t1[i] et
 * t2[i] des tableaux {@code t1} et {@code t2}.
 * @param t1 un tableau d'entiers
 * @param t2 un tableau d'entiers
 * @return
 * pre : t1 et t2 sont de taille identique
 * @throws IllegalArgumentException si t1 et t2 ne
 * sont pas de meme taille.
 * @throw NullPointerException si t1 ou t2 est null.
 */
static int [] sommeTab(int []t1, int[] t2){...}
```

sommeTab : une implantation

```
static int [] sommeTab(int []t1, int[] t2){
      if (t1.length != t2. length)
          throw new IllegalArgumentException();
      int [] r = new int [t1.length];
      for(int i= 0; i< t1.length; i++){
          r[i]= t1[i] + t2[i];
      }
      return r;
  }
```

Cette implantation correspond à la spécification ?

Quelques cas de test pour `sommeTab`

1. t1 est nul $\Rightarrow$ `NullPointerException`
2. t2 est nul $\Rightarrow$ `NullPointerException`,
3. 2 tableaux non nuls mais de tailles différentes $\Rightarrow$ `IllegalArgumentException`
4. 2 tableaux de taille 0 $\Rightarrow$ tableau de taille 0
5. 2 tableaux de même taille $\Rightarrow$ le tableau avec les bonnes additions

Définir un jeu (table) de tests en Java

Pour tester $f(x, y)$, nous disposons d'une table (jeu de tests) :

x	y	résultat attendu
x_1	y_1	r_1
x_2	y_2	r_2
x_3	y_3	r_3

1. définir cette table en Java, dans le fichier de tests,
2. écrire des méthodes pour jouer les tests de cette table,
3. pour ajouter des nouveau tests, nous les ajoutons dans la table.

Définir une table avec les arguments pour les tests

Pour simplifier, supposons que x,y sont des entiers :

```
  int [][] argsF = {
           {x1,y1}, // arguments cas de test 1
           {x2,y2}, // arguments cas de test 2
           {x3,y3}, // arguments cas de test 2
 }
```

Nous utilisons un tableau qui contient 3 tableaux.
Dans chaque tableau il y a les deux arguments x_i, y_i pour tester la fonction.

Définir une table avec les résultats attendus

Supposons que le résultat attendu est un double.
On mettra à la place i, le résultat r_i attendu pour les arguments x_i, y_i :

```
  double [] resF = {
            r1,           // attendu cas de test 1
            r2,           // attendu cas de test 2
            r3,          //  attendu cas de test 2
 }
```

Une méthode pour tester $f(x_i, y_i)$

Supposons que le résultat attendu est un double.

```
void fTestGen(int i){
    int x= argsF[i][0];
    int y = argsF[i][1];
    double attendu = resF[i];
    Assert.assertEquals("f, cas "+i,
            attendu,Tp.f(x, y), 0.001);
}
```

Une méthode pour jouer tous les cas avec Junit

```
@Test
     public void testFJeu() {
          for(int i=0; i< argsF.length; i++ ){
               fTestGen(i);
          }
     }
```

Table d'arguments pour `sommeTab`

La table avec les arguments (qui ne lèvent pas d'exception) :

```
int [][][] argsSommeT = {
        {{},{}},                    // 2 tableaux de taille 0
        {{1},{3}},                 // 2 tableaux taille 1
        {{0,2}, {0,0}},   // 2 tableaux taille 2 avec 0's
        {{-1,5,10}, {-2,-6,4}}  // 2 tableaux taille 3 a
    };
```

Table de résultats attendus pour `sommeTab`

La table avec les arguments (qui ne lèvent pas d'exception) :

```
int [][] resSommeT = {
        {}, {4} , {0,2}, {-3, -1, 14}
    };
```

Les méthodes pour jouer les tests `sommeTab`

```
void sommeTabTestGen(int i){
        int t1[] = argsSommeT[i][0];
        int t2[] = argsSommeT[i][1];
        int [] attendu = resSommeT[i];
        Assert.assertArrayEquals("sommeTab, cas "+i,
                attendu,
                Tp2.sommeTab(t1, t2));
    }
```

Les méthodes pour jouer les tests (suite)

```
@Test
    public void testSommeTabJeu() {
        for(int i=0; i< argsSommeT.length; i++ ){
            sommeTabTestGen(i);
        }
    }
```