TD1

Exercice – faute, erreur, défaillance

1. Soit le programme erroné suivant :

```java
// Effects: if x==null throw NullPointerException
//   else return the index of the last 0 in x
//   Return -1 if 0 does not occur in x
public static int lastZero (int[] x) {
  for (int i = 0; i < x.length; i++) {
    if (x[i] == 0) {
      return i;
    }
  }
  return -1;
}
```

Pour `x = [0,1,0]`, ce programme provoque une défaillance (résultat attendu : 2).

(a) Identifiez la faute.

(b) Si possible, identifiez un cas de test qui n'exécute pas la faute.

(c) Si possible, identifiez un cas de test qui exécute la faute, mais n'aboutit pas à un état d'erreur.

(d) Si possible, identifiez un cas de test qui entraîne une erreur, mais pas une défaillance.

(e) Pour le cas de test donné, identifiez le premier état d'erreur et décrivez-le

(f) Corrigez la faute et vérifiez que le test donné produit maintenant le résultat attendu.

Corrigé ▶

(a) for (int i=0; i < x.length; i++) : la boucle devrait parcourir le tableau du dernier au premier élément

(b) Impossible, toutes les entrées exécutent la faute (même null)

(c) x doit être null ou un tableau à un élément nul

(d) x doit être vide ou un tableau à un élément non nul (car la dernière valeur de la variable i est -1 dans le code correct, mais 1 dans le code incorrect, juste avant de quitter la boucle) ou un tableau à plus d'un élément avec au plus un zéro (puisque la valeur de la variable i monte au lieu de descendre)

(e) Le premier état erroné est atteint lorsque la variable i a la valeur 0 au lieu de la taille du tableau moins 1 : x=[0,1,0], i=0, juste après l'affectation int i=0

(f) for (int i=x.length-1; i >= 0; i--)

◀

2. Soit le programme erroné suivant :

```java
// Effects: if x==null throw NullPointerException
//   else return the number of elements in x that are either odd
//        or positive (or both)
```

```java
public static int oddOrPos(int[] x) {
  int count = 0;
  for (int i = 0; i < x.length; i++) {
    if (x[i]%2 == 1 || x[i] > 0) {
      count++;
    }
  }
  return count;
}
```

Pour `x = [-3,-2,0,1,4]`, ce programme provoque une défaillance (résultat attendu : 3).

(a) Mêmes questions que précédemment.

Corrigé ▶

(a) x[i]%2 == 1 : en Java, le modulo sur un nombre négatif est négatif

(b) x doit être null ou vide,

(c) x doit être un tableau non vide sans entier impair négatif

(d) Toute entrée qui entraîne une erreur provoque une défaillance, car ici les états erronés ne sont pas réparables par un traitement ultérieur (s'il y a un entier impair négatif dans x, tous les états suivants, c-a-d après count++, seront erronés, peu importe ce qui se trouve dans x)

(e) Le premier état erroné est atteint lorsque la variable count n'est pas incrémenté pour la valeur -3 de x : x=[-3,-2,0,1,4], i=0, count=0, juste à la fin de l'instruction if

(f) x[i]%2 == -1 : la conditionnelle if doit prendre en compte les entiers négatifs (les entiers impairs positifs sont pris en compte par x[i] > 0)

◀

3. Soit le programme erroné suivant :

```java
// Effects: if x==null throw NullPointerException
//   else return the index of the last element in x that equals y
//   If no such element exists, return -1
public static int findLast (int[] x, int y) {
  for (int i=x.length-1; i > 0; i--) {
    if (x[i] == y) {
      return i;
    }
  }
  return -1;
}
```

Pour `x = [2,3,5]` et `y=2`, ce programme provoque une défaillance (résultat attendu : 0).

(a) Mêmes questions que précédemment.

Corrigé ▶

(a) i > 0 : la boucle for ne prend pas en compte le premier élément de x

(b) x doit être null

(c) x doit être vide ou un tableau avec au moins de y ou un tableau avec un seul y qui n'est pas le premier élément

(d) x doit être un tableau non vide sans y

(e) Le premier état erroné est atteint lorsque la variable i a la valeur 0 : x=[2,3,5], y=2, i=0, à la fin de la boucle

(f) i >= 0

◀

4. Soit le programme erroné suivant :

```
// Effects: if x==null throw NullPointerException
//   else return the number of positive elements in x
public static int countPositive (int[] x) {
  int count = 0;
  for (int i=0; i < x.length; i++) {
    if (x[i] >= 0) {
      count++;
    }
  }
  return count;
}
```

Pour `x = [-4,2,0,2]`, ce programme provoque une défaillance (résultat attendu : 2).

(a) Mêmes questions que précédemment.

Corrigé ▶

(a) x[i] >= 0 : la conditionnelle ne doit pas prendre en compte les zéros

(b) x doit être null ou vide

(c) x doit être un tableau non vide sans zéro

(d) Toute entrée qui entraîne une erreur provoque une défaillance

(e) Le premier état erroné est atteint lorsque la variable count prend la valeur 2 : x =[-4,2,0,2], i=2, count=2, à la fin de l'instruction if

(f) x[i] > 0

◀

Exercice – le problème du triangle (*Weinberg-Myers Triangle Test*)

Soit un programme qui lit trois nombres réels et détermine s'ils correspondent aux longueurs des côtés d'un triangle. Si ces trois nombres ne permettent pas de former un triangle, le programme affiche un message d'erreur approprié. Dans le cas d'un triangle, le programme affiche le type du triangle : scalène (tous les côtés ont une longueur différente), équilatéral (tous les côtés ont la même longueur) ou isocèle (seulement deux côtés ont la même longueur), ainsi que la nature de son plus grand angle : aigu ($< 90°$), droit ($= 90°$) ou obtus ($> 90°$). Pour rappel, un ensemble de trois nombres peut valablement correspondre aux longueurs des côtés d'un triangle si :

- Chaque nombre de l'ensemble est strictement positif
- La somme de deux des nombres de l'ensemble est strictement supérieur au troisième, quelles que soient les permutations des nombres

1. Faites une analyse partitionnelle sur le domaine des données d'entrée et donnez un cas de test (donnée de test + résultat attendu) par classe d'équivalence. **Corrigé** ▶

Classes d'équivalence et données de test :

	Aigu	*Obtus*	*Droit*
Scalène	$6,5,3$	$5,6,10$	$3,4,5$
Isocèle	$6,1,6$	$7,4,4$	$\sqrt{2},2,\sqrt{2}$
Equilatéral	$4,4,4$	*impossible*	*impossible*

+ Triangle invalide : $1,2,8$

Cette dernière classe d'équivalence peut être raffinée pour prendre en compte les différentes conditions non respectées (nombres négatifs + somme de deux des nombres inférieure ou égale au troisième). De plus, pour le triangle isocèle et le triangle invalide, pensez aux permutations de la donnée de test !

◀

2. Proposez d'autres cas de test avec les valeurs aux limites. **Corrigé** ►

Pas un triangle	$1, 1, 2$	*triangle invalide*
Un seul point	$0, 0, 0$	*triangle invalide*
Une des longueurs est nulle	$4, 0, 3$	*triangle invalide*
Presque triangle	$1, 2, 3.00001$	*triangle invalide*
Très petit triangle	$0.001, 0.001, 0.001$	*équilatéral*
Très grand triangle	$88888, 88888, 88888$	*équilatéral*
Presque équilatéral	$3.00001, 3, 3$	*isocèle (aigu)*
Presque isocèle	$2.99999, 3, 4$	*scalène (aigu)*
Presque droit	$3, 4, 5.00001$	*scalène (obtus)*
Données négatives	$-3, -3, 5$	*entrée invalide*

◄