JUnit

Correction commentée de l'exercice « Equation »

Yvan Maillot

La classe Equation

```
public class Equation {
    private int nbRacines;
    private double x1, x2;
    /* Résoudre l'équation axx+bx+c=0
     * @param a le coefficient de degré 2,
     *                un entier différent de 0
     * @param b le coefficient de degré 1
     * @param c le coefficient de degré 0
     */
    public Equation(int a, int b, int c) {
    }

    public int getNbRacines() {return nbRacines;}
    public double getX1() {return x1;}
    public double getX2() {return x2;}
}
```

Le test de Equation

```
public void testEquation() {
  System.out.println("Test équation");
  for (int i = 0; i < 100000; i++) {
    int a;
    do {
      a = random.nextInt(101) - 50;
    } while (a == 0);
    int b = random.nextInt(101) - 50;
    int c = random.nextInt(101) - 50;
    Equation e = new Equation(a, b, c);
    if (e.getNbRacines() > 0) {
      double x1 = e.getX1(), x2 = e.getX2();
      assertEquals(0.0, a * x1 * x1 + b * x1 + c, 1e-10);
      assertEquals(0.0, a * x2 * x2 + b * x2 + c, 1e-10);
    }}}
```

Erratum

```
public void testEquation() {
  System.out.println("Test équation");
  for (int i = 0; i < 100000; i++) {
    int a;
    do {
      a = random.nextInt(101) - 50;
    } while (a == 0);
    int b = random.nextInt(101) - 50;
    int c = random.nextInt(101) - 50;
    Equation e = new Equation(a, b, c);
// Si, par erreur, getNbRacines() retourne toujours 0, le
// test passera quelles que soient les racines trouvées !
    if (e.getNbRacines() > 0) {
      double x1 = e.getX1(), x2 = e.getX2();
      assertEquals(0.0, a * x1 * x1 + b * x1 + c, 1e-10);
      assertEquals(0.0, a * x2 * x2 + b * x2 + c, 1e-10);
    }}}
```

Humanum est

```
public void testEquation() {
  System.out.println("Test équation");
// Commence comme la diapo précédente
  ...
// Il n'y a pas d'autres solutions que re-calculer le
// discriminant pour savoir si le nombre de racines
// calculé est juste.
    double delta = b * b - 4 * a * c;
    if (delta < 0) {
       assertEquals(0, e.getNbRacines());
    } else if (delta == 0) {
        assertEquals(1, e.getNbRacines());
    } else {
        assertEquals(2, e.getNbRacines());
    }
    if (e.getNbRacines() > 0) {
      double x1 = e.getX1(), x2 = e.getX2();
      assertEquals(0.0, a * x1 * x1 + b * x1 + c, 1e-10);
      assertEquals(0.0, a * x2 * x2 + b * x2 + c, 1e-10);
    }}}
```

Le constructeur

```
    public Equation(int a, int b, int c) {
        double delta = b * b - 4 * a * c;
        if (delta < 0) {
            nbRacines = 0;
        } else if (delta == 0) {
            nbRacines = 1;
// Un piège (-b/(2*a)) à l'origine d'erreurs fréquentes
            x1 = x2 = (-b*1.0) / (2 * a);
        } else {
            nbRacines = 2;
            double rdelta = Math.sqrt(delta);
// Autre erreur possible : diviser par 2 * a
            x1 = (-b - rdelta)/ (2 * a);
            x2 = (-b + rdelta)/ (2 * a);
        }
    }
```

Exception possible

```
...
for (int i = 0; i < 100000; i++) {
int a;
//do {
  a = random.nextInt(101) - 50;
//} while (a == 0);
...
try {
  Equation e = new Equation(a, b, c);
} catch (Exception ex) {
  assertTrue(ex instanceof DataFormatException);
  assertEquals(0, a);
}
```

Lancement de l'exception

```
public Equation(int a, int b, int c) throws
    DataFormatException {
    if (a == 0) throw new DataFormatException();

    double delta = b * b - 4 * a * c;
    if (delta < 0) {
        nbRacines = 0;
    } else if (delta == 0) {
        nbRacines = 1;
        x1 = x2 = (-b*1.0) / (2 * a);
    } else {
        nbRacines = 2;
        double rdelta = Math.sqrt(delta);
        x1 = (-b - rdelta)/ (2 * a);
        x2 = (-b + rdelta)/ (2 * a);
    }
}
```