TD II

TD 02 : Types, diagrammes UML et Exceptions

II.1 Trinôme du second degré

II.1.a Rappels mathématiques

En mathématiques, lorsque l'on souhaite résoudre une équation du second degré du type:

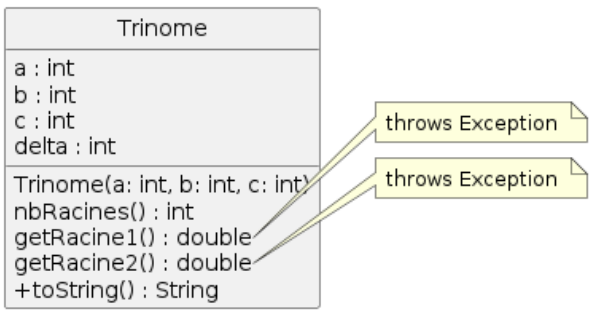
$$ax^2 + bx + c = 0$$

- On calcule $\Delta = b^2 - 4ac$
- Si $\Delta < 0$, il n'y a pas de solution réelle
- Si $\Delta = 0$, il y a une solution double $x = -\frac{b}{2a}$
- Si $\Delta > 0$, il y a deux solutions réelles: $\frac{-b-\sqrt{\Delta}}{2a}$, et $\frac{-b+\sqrt{\Delta}}{2a}$

Cette démarche est incorrecte si $a = 0$, mais dans ce cas, la solution est évidente.

II.1.b Diagramme UML de Classes

Pour écrire un solveur d'équation (1er et second degré) en Java, on considère la classe suivante.



Note

On remarque que les coefficients `a`, `b` et `c` sont des entiers. Ce choix est discutable. Il est réalisé ici pour une gestion des exceptions simplifiée...

Question 1.1
Écrire le début de la classe `Trinome` (déclaration de la classe, attributs et constructeurs).

II.1.c La méthode `toString`

La méthode `public String toString()` renvoie une chaîne de caractère sous la forme :

```
5x^2 + 3x + 2 = 0
5x^2 - 3x -8 = 0
3x -8 = 0
```

Note

La méthode `toString` a ici le *modificateur d'accès* `public`. Dans cette partie de l'UE, il vous est demandé de **ne pas** préciser de *modificateur d'accès*... On fait ici une exception car la méthode `toString` et une méthode définie par Java dans la classe `Object`. Vous comprendrez les implications de cela à partir du chapitre sur l'héritage... En attendant, ici, exceptionnellement, on met un `public` devant `toString`...

Question 1.2
Écrire la méthode `toString` de la classe `Trinome`.

II.1.d La méthode `nbRacines`

Question 1.3
Écrire la méthode `nbRacines`.

Penser aux cas où `a` ou `b` sont nuls...

II.1.e Les méthodes `getRacine1` et `getRacine2`

La méthode `getRacine1`

- lève une `Exception` lorsqu'il n'y a pas de solution à l'équation
- dans le cas où il y a 2 solutions, renvoie $\frac{-b-\sqrt{\Delta}}{2a}$
- dans le cas où il n'y a qu'une solution, la renvoie.

Note

Pour cet exercice, et afin de s'entraîner à gérer les exceptions, on demande de **ne pas** utiliser l'instruction `if (a == 0)`. Au lieu de ça, on vérifiera que l'instruction `1/(2*a)` ne renvoie pas d'exception. Si une `ArithmeticException` est renvoyée, alors cela signifie que `a == 0`, on fait alors le traitement adéquat pour renvoyer le résultat...

La division par zéro ne renvoie une `ArithmeticException` que s'il s'agit d'une division **entière** ! Aussi, l'instruction `(-b - Math.sqrt(delta))/(2*a)` ne lèvera pas d'exception mais renverra `- Infinity`... Il faut donc tester `1/(2*a)` à part...

Question 1.4
Écrire la méthode `getRacine1`.

La méthode `getRacine2`

- lève une `Exception` lorsqu'il n'y a pas de solution à l'équation
- lève une `Exception` (avec un message différent) lorsqu'il n'y a qu'une seule solution à l'équation
- dans le cas où il y a 2 solutions, renvoie $\frac{-b+\sqrt{\Delta}}{2a}$

Note

Comme pour la méthode précédente, on souhaite gérer le cas $a = 0$ par les exceptions.

II.1.f Test du trinôme

On souhaite à présent tester la classe `Trinome`.

Question 1.5
Quelles erreurs seront produites à la compilation et ou à l'exécution si l'on laisse le code tel-quel (en laissant les lignes vides vides...) ? Complétez la méthode `main` pour que le programme compile et ne crash jamais (et affiche les messages d'erreur).

```
1  public class TestTrinome {
2      public static void main(String[] args) {
3          Trinome trinome;
4
5          ----------------------------------------------------------------
6
7          // 5 x^2 + 3x - 8 = 0
8          trinome = new Trinome(5, 3, -8);
9          System.out.println("On considère le trinome : " + trinome.toString());
10         ----------------------------------------------------------------
11
12         System.out.println("Ce trinome a " + trinome.nbRacines() + " racines.")
               ;
13         ----------------------------------------------------------------
14
15         System.out.println("- Racine 1 : " + trinome.getRacine1());
16         System.out.println("- Racine 2 : " + trinome.getRacine2());
17         ----------------------------------------------------------------
18         ----------------------------------------------------------------
19         ----------------------------------------------------------------
20         ----------------------------------------------------------------
21
22
23         ----------------------------------------------------------------
24
25         //  3x - 8 = 0
26         trinome = new Trinome(0, 3, -8);
27         System.out.println("On considère le trinome : " + trinome.toString());
28         ----------------------------------------------------------------
29
30         System.out.println("Ce trinome a " + trinome.nbRacines() + " racines.")
               ;
31
32         ----------------------------------------------------------------
33
34         System.out.println("- Racine 1 : " + trinome.getRacine1());
35             System.out.println("- Racine 2 : " + trinome.getRacine2());
36         ----------------------------------------------------------------
37         ----------------------------------------------------------------
38         ----------------------------------------------------------------
39         ----------------------------------------------------------------
```

```
40
41
42          ------------------------------------------------------------
43          // 5 x^2 + 3x + 2 = 0
44          trinome = new Trinome(5, 3, 2);
45          System.out.println("On considère le trinome : " + trinome.toString());
46          ------------------------------------------------------------
47
48          System.out.println("Ce trinome a " + trinome.nbRacines() + " racines.")
                ;
49          ------------------------------------------------------------
50
51          System.out.println("- Racine 1 : " + trinome.getRacine1());
52              System.out.println("- Racine 2 : " + trinome.getRacine2());
53          ------------------------------------------------------------
54          ------------------------------------------------------------
55          ------------------------------------------------------------
56          ------------------------------------------------------------
57
58      }
59  }
```

II.1.g Encapsulation (premières réflexions...)

Suite à ce programme, on pourrait souhaiter modifier le trinôme existant plutôt que de re-créer une instance à chaque fois (c'est discutable, mais possible)...

Question 1.6
Ecrire les accesseurs (`getA() : double`, `getB() : double` , `getC(): double`, `getDelta(): double`) et les modificateurs (`setA(a: double): void`, `setB(b: double): void`, `setC(c: double): void`) de la classe trinôme.

Question 1.7
Pourquoi ne peut-on pas écrire de modificateur `+setDelta(delta: double): void` ?