TP4 : Exercices de Programmation Objet

Florian Boudin

Module X0IC020 - 2013

Exercice 1 : les classes `Author` et `Book`

Cet exercice a pour but que vous faire construire deux classes que vous allez utiliser conjointement. Il s'agit de créer une classe livre (`Book`) qui regroupe les informations au sujet d'un ouvrage, parmi lesquelles l'auteur correspond à une instance d'une autre classe (`Author`). Les diagrammes des deux classes sont présentés dans les figures ci-dessous.

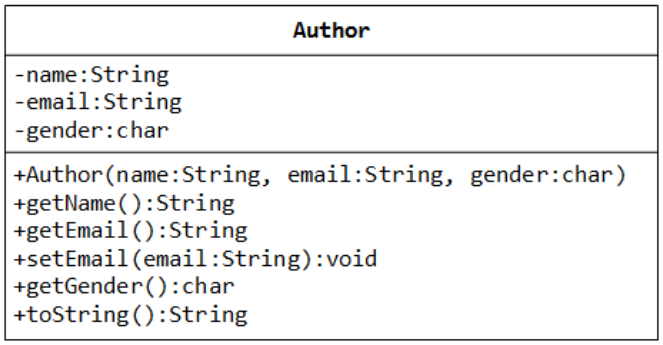

FIGURE 1 – Diagramme de la classe `Author`

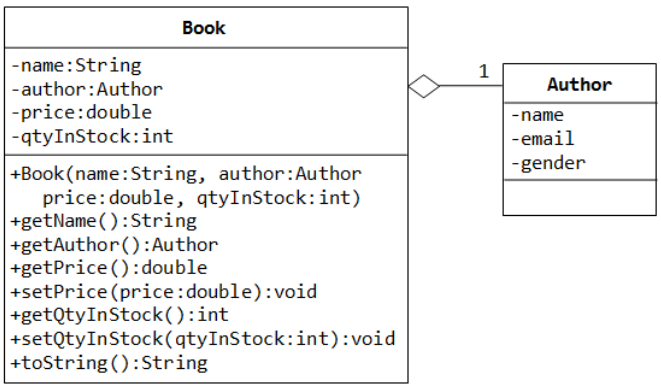

FIGURE 2 – Diagramme de la classe `Book`

La classe `Author` doit contenir :

1. Trois variables d'instances privées : `name`, `email` et `gender` (un caractère qui peut être soit `'m'` ou `'f'`).
2. Un constructeur pour initialiser les variables `name`, `email` et `gender` avec des valeurs données en paramètres.
3. Les méthodes *getters* et *setters* : `getName()`, `getEmail()`, `setEmail()` et `getGender()`. Il n'y a pas de *setters* pour `name` et `gender` puisque ces valeurs ne doivent pas être changées.
4. Une méthode `toString()` qui retourne une description de l'instance sous la forme `"name(email)"`.

Écrire la classe utilisatrice `TestAuthor` afin de tester le constructeur et les méthodes que vous avez créés. Par exemple, essayez de modifier l'adresse email d'un auteur :

```
Author anAuthor = new Author("John Doe", "jd@email.fr", 'm');
System.out.println(anAuthor);     // call toString()
anAuthor.setEmail("doe@nowhere.com")
System.out.println(anAuthor);
```

La classe `Book` doit contenir :

1. Quatre variables d'instances privées : `name`, `author` (du type de la classe `Author` que vous venez de créer), `price` et `qtyInStock`. Cette modélisation suppose que chaque livre est écrit par un seul auteur.
2. Un constructeur qui construit une instance avec tous les paramètres spécifiés.
3. Les méthodes *getters* et *setters* : `getName()`, `getAuthor()`, `getPrice()`, `setPrice()`, `getQtyInStock()` et `setQtyInStock()`. Pour la même raison que dans la classe `Author`, il n'y a pas de *setters* pour `name` et `author`.
4. La méthode `toString()` qui retourne une description de l'instance de `Book` sous la forme `""bookname" written by authorname(email)"`. Vous devez ré-utiliser la méthode `toString()` de la classe `Author`.
5. Les classes `Book` et `Author` contiennent toutes les deux une variable nommée `name`. Elles peuvent être différenciées via l'instance à laquelle on se réfère (e.g. `aBook.name` et `anAuthor.name`). Cependant, il est recommandé d'utiliser les *getters* que vous avez créés, e.g. `aBook.getAuthor().getName()`. Écrire les méthodes `getAuthorName()`, `getAuthorEmail()` et `getAuthorGender()` permettant d'accéder aux informations sur l'auteur du livre.

Écrire la classe utilisatrice `TestBook` afin de tester les constructeurs et les méthodes que vous avez créés. Attention, vous devez créer une instance de `Author` avant de construire une instance de `Book`, e.g.

```
Author anAuthor = new Author(...);
Book aBook = new Book("Java for dummy", anAuthor, 19.95, 1000);
```

Exercice 2 : la classe `MyComplex`

Dans cet exercice, vous allez créer la classe `MyComplex` permettant de modéliser un nombre complexe $(x + yi)$. Le diagramme de classe est présenté dans la figure ci-dessous.

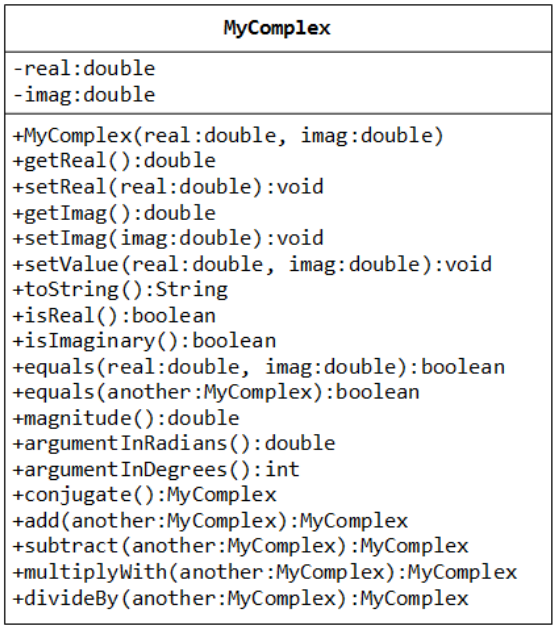

FIGURE 3 – Diagramme de la classe `MyComplex`

La classe `MyComplex` doit contenir :

1. Deux variables d'instances privées : `real` et `imag` qui contiennent la partie réelle et imaginaire d'un nombre complexe.
2. Un constructeur qui construit une instance avec les paramètres de valeurs de la partie réelle et imaginaire.
3. Les méthodes *getters* et *setters* pour la partie réelle et imaginaire.
4. La méthode `setValue(double real, double imag)` pour affecter la valeur d'un nombre complexe
5. La méthode `toString()` qui retourne une description de l'instance de `MyComplex` sous la forme `"(x + yi)"`.
6. Les méthodes `isReal()` et `isImaginary()` qui retournent vrai si le nombre complexe est réel ou imaginaire (indice : `return (imag == 0); //isReal()`)
7. Une méthode `equals(double real, double imag)` qui retourne vrai si le nombre complexe est égal au nombre spécifié en paramètre.
8. Une méthode surchargée `equals(MyComplex aComplex)` qui retourne vrai si le nombre complexe est égal au nombre spécifié en paramètre.
9. Une méthode `magnitude()` qui retourne la magnitude d'un nombre complexe.

   ```
   magnitude(x+yi) = Math.sqrt(x^2 + y^2)
   ```

10. Les méthodes `argumentInRadians()` et `argumentInDegrees()` qui retourne l'argument du nombre complexe en radians (`double`) et en degrés (`int`).

    ```
    arg(x+yi) = Math.atan2(y, x) // (in radians)
    ```

11. Une méthode `conjugate()` qui retourne un nouveau objet MyComplex contenant le conjugué du nombre complexe.

    ```
    conjugate(x+yi) = x - yi
    ```

12. Les méthodes `add(MyComplex aComplex)` et `substract(MyComplex aComplex)` qui additionne et soustrait le nombre complexe avec celui spécifié en paramètre, et retourne un nouveau objet `MyComplex`.

```
(a + bi) + (c + di) = (a+c) + (b+d)i
(a + bi) - (c + di) = (a-c) + (b-d)i
```

13. Les méthodes `multiplyWith(MyComplex aComplex)` et `divideBy(MyComplex aComplex)` qui mutliplie et divise le nombre complexe avec celui spécifié en paramètre, puis modifie la valeur du nombre complexe et le retourne (indice : `return this;`).

```
(a + bi) * (c + di) = (ac - bd) + (ad + bc)i
(a + bi) / (c + di) = [(a + bi) * (c - di)] / (c^2 + d^2)
```

Vous devez ensuite écrire la classe utilisatrice `TestComplex` afin de tester les constructeurs et les méthodes que vous avez créés.

L'application `MyComplexApp`

Ecrire l'application `MyComplexApp` qui utilise la classe `MyComplex`. Cette application doit demander à un utilisateur deux nombres complexes ; afficher leurs valeurs ; vérifier si les nombres sont réels, imaginaires ou égaux ; et réaliser les opérations arithmétiques définies.

```
Enter complex number 1 (real and imaginary part): 1.1 2.2
Enter complex number 2 (real and imaginary part): 3.3 4.4

Number 1 is: (1.1 + 2.2i)
(1.1 + 2.2i) is NOT a pure real number
(1.1 + 2.2i) is NOT a pure imaginary number

Number 2 is: (3.3 + 4.4i)
(3.3 + 4.4i) is NOT a pure real number
(3.3 + 4.4i) is NOT a pure imaginary number

(1.1 + 2.2i) is NOT equal to (3.3 + 4.4i)
(1.1 + 2.2i) + (3.3 + 4.4i) = (4.4 + 6.6000000000000005i)
(1.1 + 2.2i) - (3.3 + 4.4i) = (-2.1999999999999997 + -2.2i)
```

Il y a quelques failles dans le design de la classe `MyComplex` (qui est ici présenté dans un but pédagogique uniquement).

1. La comparaison de deux nombres `double` avec `==` peut être dangereuse. Par exemple, `(2.2+4.4) == 6.6` retourne faux. Un seuil appelé `EPSILON` (10^{-8}) est généralement utilisé pour comparer les nombres réels.
2. Les méthodes lstinlineadd(), `substract()` et `conjugate()` retournent des nouvelles instances tandis que `multiplyWith()` et `divideBy()` modifient l'instance. Il y a donc des inconsistances dans le design de la classe.

Sources

Ce document est une adaptation d'exercices rédigés par Chua Hock-Chuan. Rédigé par Florian Boudin, 2011–12.