Programmation Orientée Objet

TP d'introduction, première partie – Soyons rationnel (n'ayons l'air de rien)

TP préparé par : Samah Bouzidi

Résumé

L'objectif de ce TP est de découvrir les bases de la programmation orientée objet avec Java. Nous nous attacherons à construire et à manipuler une classe très simple pour mettre en pratique les notions de classe, d'objet, d'attribut, de méthode, de rétention d'information (*encapsulation* en anglais), de constructeur, ...

Vous connaissez probablement tous la définition de l'ensemble $\mathbb{Q}$ des rationnels. Un nombre rationnel est un nombre qui peut être défini par un ratio n/d, où n est un entier, et d est un entier non nul. Nous allons dans ce TP construire une classe représentant les nombres rationnels.

1 On commence doucement

Question 1 Première version d'une classe `Rational`
1.a - Créez une classe `Rational`, possédant deux attributs entiers `num` et `denom`.
1.b - Écrivez un programme de test qui crée une fraction $3/2$, affiche son numérateur, et affiche son dénominateur (dans la console).
1.c - Compilez tout ça avec le compilateur `javac`, et exécutez avec la machine virtuelle `java`.

Question 2 Représentation textuelle d'un rationnel
2.a - Modifiez votre classe `Rational` afin d'y ajouter une méthode `toString()` sans paramètre, et renvoyant une chaîne de caractères (type `String`) représentant le nombre rationnel (par exemple, cette méthode, appelée sur le rationnel $3/2$, devrait renvoyer la chaîne de caractères `"3 / 2"`).
2.b - Modifiez votre programme de test pour utiliser la méthode `toString()` créée précédemment. Compilez et exécutez.

2 Encapsuler pour régner

Question 3 Modifiez votre programme de test pour créer un autre rationnel de dénominateur égal à 0. Est-ce possible ? Si oui, en quoi est-ce un problème ?

Question 4 Proposez une solution pour empêcher la *création* et la *manipulation* d'objets rationnels ayant pour dénominateur 0. Modifiez votre classe `Rational` en conséquence.

3 Des opérations arithmétiques

Question 5 Pour rappel (au cas où...), la multiplication de deux rationnels est définie de la manière suivante : $\frac{n}{d} \times \frac{n'}{d'} = \frac{nn'}{dd'}$.
5.a - Ajoutez à votre classe `Rational` une méthode `mult` prenant en paramètre un autre rationnel, et le multipliant au rationnel sur lequel elle est appliquée. La méthode ne renvoie rien (type de retour `void`).
Cette opération modifie l'état de l'objet sur lequel la méthode est invoquée : `a.mult(b)` revient à

faire $a \leftarrow a * b$. Il n'est pas possible en Java de redéfinir des opérateurs (comme en C++), sinon ce serait équivalent à `a *= b`.

5.b - Modifiez votre programme de test en créant un second rationnel ($1/3$ par exemple), en le multipliant au premier, et en affichant le résultat. Compilez et exécutez.

Question 6 Mêmes questions pour l'addition. Pour rappel, l'addition de deux rationnels est définie de la manière suivante : $\frac{n}{d} + \frac{n'}{d'} = \frac{nd'+n'd}{dd'}$.

4 Des fractions irréductibles

Question 7 En plus d'interdire la création et la manipulation de rationnels dont le dénominateur est à 0, nous souhaitons que tout rationnel soit toujours sous sa forme irréductible. Il s'agit d'un *invariant de classe* : à tout instant, dès sa création, un objet `Rational` est toujours sous forme irréductible.

Pour rappel, un rationnel $\frac{n}{d}$ est sous forme irréductible si et seulement si $\text{pgcd}(n, d) = 1$, avec :

```
fonction pgcd(a,b)
    si b est égal à 0 renvoyer a
    sinon renvoyer pgcd(b, reste(a,b))
```

Modifiez votre classe `Rational` en conséquence (ou créez un nouvelle classe `ReducedRational`). Testez.

5 Des vecteurs de rationnels

Pour cette partie, vous aurez besoin de manipuler des tableaux d'éléments en Java. Pour avoir quelques éléments sur la manière dont cela fonctionne en Java.

Question 8 Notre classe `Rational` est sympathique, mais bien seule pour l'instant. Nous allons l'utiliser pour manipuler des vecteurs de nombres rationnels (par exemple $\langle 1/2, 3/2\rangle$, ou encore $\langle 1/6, 6/4, 33/1\rangle$).

8.a - Créez une classe `Vector` représentant un vecteur de rationnels (de dimension fixée à la création), et donc ayant pour attribut un tableau d'éléments de type `Rational`. Réfléchissez bien en particulier au(x) paramètre(s) dont vous avez besoin pour l'initialisation d'un objet de cette classe, et aux visibilités des attributs.

8.b - Ajoutez à votre classe une méthode `toString()`, sans paramètre et renvoyant une chaîne de caractères (type `String`) représentant le vecteur. Par exemple, cette méthode appelée sur le vecteur $\langle 1/2, 3/2\rangle$ devrait renvoyer la chaîne de caractères `"( 1 / 2, 3 / 2 )"`.

8.c - Compilez et testez sur un vecteur de dimension 2 par exemple.

Question 9 Intéressons-nous maintenant à l'accès aux composantes du vecteur.

9.a - Ajoutez à votre classe une méthode `void set(int i, Rational r)` qui modifie le $i^{\text{ème}}$ rationnel du vecteur de telle sorte qu'il vaille désormais `r` (une erreur est levée si i excède la dimension du vecteur).

9.b - Écrivez un programme de test qui exécute le pseudo-code suivant :

- $\overrightarrow{v} \leftarrow \langle 0/1, 0/1\rangle$;
- $a \leftarrow 2/3$ (`a` est un `Rational`)
- $v_1 \leftarrow a$ (affecte r au premier élément du vecteur)

— Afficher $\overrightarrow{v}$;
— $b \leftarrow 3/2$ (`b` est un `Rational`)
— $a \leftarrow a * b$ (grâce à la méthode `mult` de la classe `Rational`. `a` vaut désormais $3/3 == 1$)
— Afficher $\overrightarrow{v}$;

Compilez et exécutez. Que constatez-vous ? Est-ce un problème ? Si oui comment l'expliquez-vous et comment le résoudre ?

9.c - Écrivez une méthode `Rational get(int i)` prenant en paramètre un entier i et renvoyant le rationnel correspondant à la $i^{\text{ème}}$ composante du vecteur (ou une erreur si i excède la dimension du vecteur).

9.d - Écrivez un programme de test qui exécute le pseudo-code suivant :
— $\overrightarrow{v} \leftarrow \langle 1/2, 1/2 \rangle$;
— Afficher $\overrightarrow{v}$;
— $a \leftarrow v_1$ (récupérer la première composante de $\overrightarrow{v}$) ;
— $a \leftarrow a * 1/3$ (grâce à la méthode `mult` de la classe `Rational`.)
— Afficher $\overrightarrow{v}$.

Compilez et exécutez. Que constatez-vous ? Est-ce un problème ? Si oui comment l'expliquez-vous et comment le résoudre ?

Question 10 Nous allons maintenant doter notre classe `Vector` quelques opérations basiques d'un espace vectoriel sur $\mathbb{Q}$:

10.a - Ajoutez à la classe `Vector` une méthode de multiplication par un rationnel. Cette méthode prend en paramètre un `Rational`, et multiplie toutes les composantes du vecteur par ce rationnel.

10.b - Ajoutez une méthode `add`, additionnant un `Vector` donné en paramètre à l'objet de type `Vector` sur lequel elle est invoquée. Pour rappel, l'addition de deux vecteurs *de même taille* se fait en additionnant leurs composantes respectives. La méthode échoue si le vecteur passé en paramètre n'est pas de même dimension que le vecteur sur lequel la méthode est appelée.

6 Des vecteurs de taille dynamique (optionnel)

Question 11 Nous voulons maintenant enrichir notre modèle de vecteur pour rendre sa dimension dynamique. Concrètement, nous désirons créer une classe `ExtensibleVector` modélisant un vecteur de rationnels de dimension variable, et possédant au moins les méthodes suivantes :
— une méthode `void add(Rational r)` qui ajoute le rationnel passé en paramètre en fin de vecteur : par exemple, cette méthode, appelée sur le vecteur $\langle 1/2, 1/3 \rangle$ avec le paramètre $1/4$ modifie le vecteur en $\langle 1/2, 1/3, 1/4 \rangle$.
— une méthode `int getDimension()` renvoyant la dimension (la longueur) du vecteur.

Imaginez une solution simple pour développer une telle classe. Développez cette solution et analysez la complexité des méthodes `get`, `add` et `getDimension`.