Projet 1 : Encapsulation des données et des traitements. Représentation des nombres rationnels

Résumé

L'objectif de ce premier projet est de vous familiariser avec les objets. La classe que vous devrez écrire devra regrouper les données et traitements nécessaires pour représenter des nombres rationnels, et interagir avec d'autres instances de la même classe.

Problème

On veut représenter par des objets les nombres *rationnels*, c'est-à-dire les nombres pouvant s'écrire sous la forme $\frac{p}{q}$, où p est un entier relatif et q un entier strictement positif. Par exemple, $\frac{2}{3}$, $6 = \frac{6}{1}$, $\frac{40}{-24} = \frac{-5}{3}$ sont des nombres rationnels.

Toutefois, pour des raisons mystérieuses, le commanditaire du programme souhaite représenter ces nombres rationnels sous la forme suivante :

$$n + \frac{a}{b}$$

en imposant les contraintes suivantes :

- n est un entier relatif (appelé *partie entière* du nombre rationnel)
- a un entier positif
- b un entier strictement positif
- $a < b$ (de sorte que la fraction $\frac{a}{b}$, appelée *partie décimale* du rationnel, soit comprise entre 0 inclus et 1 exclus)
- a et b sont premiers entre eux (c'est-à-dire qu'il n'existe pas d'entier $d > 1$ qui soit un diviseur à la fois de a et de b).

Par exemple, $\frac{2}{3}$ s'écrira $0 + \frac{2}{3}$; de même on écrira $\frac{40}{24} = 1 + \frac{2}{3}$, ou $\frac{28}{-16} = -2 + \frac{1}{4}$, etc.

Un objet rationnel doit pouvoir s'additionner ou se multiplier avec un autre rationnel (en retournant le résultat), tester s'il est nul, se comparer avec un autre rationnel, et donner son inverse s'il n'est pas nul.
Il doit pouvoir s'afficher conformément à la représentation énoncée ci-dessus.

Solution : la classe Rationnel

Q 1. Déterminez les attributs que doit avoir la classe Rationnel.

Q 2. Écrivez la méthode `public String toString()` qui donne la représentation d'un rationnel sous forme de chaîne de caractères. Écrivez ensuite les trois constructeurs suivants et testez-les :

- `public Rationnel(int n, int a, int b)` construit le rationnel $n + \frac{a}{b}$
- `public Rationnel(int a, int b)` construit le rationnel $\frac{a}{b}$
- `public Rationnel(int n)` construit le rationnel $n + 0$

Dans un premier temps, vous ne ferez aucune vérification sur le respect des spécifications portant sur a, b et n.

Q 3. Écrivez une méthode `public boolean estNul()` qui renvoie `true` si l'instance courante représente 0.

Q 4. Écrivez la méthode `public Rationnel inverse()` qui calcule et renvoie l'inverse de l'instance courante.

Q 5. Écrivez les méthodes `public Rationnel ajouter(Rationnel r)` et `public Rationnel multiplier(Rationnel r)` qui calculent et renvoient respectivement la somme et le produit de l'instance courante et du rationnel passé en paramètre.

Q 6. Écrivez une méthode permettant de comparer l'instance courante avec un autre rationnel.

Q 7. Que se passe-t-il si l'on utilise le programme de test suivant ? Est-ce conforme aux spécifications ? Comment remédier au problème ?

```
public class Test1 {
    public static void main(String [] args) {
        System.out.println(new Rationnel(3, 4, 5)) ;
        System.out.println(new Rationnel(5, 4, 3)) ;
        System.out.println(new Rationnel(-2, 3)) ;
        System.out.println(new Rationnel(3, -2)) ;
        System.out.println(new Rationnel(3, 6, -2)) ;
        System.out.println(new Rationnel(3)) ;
        System.out.println(new Rationnel(1,3)) ;
    }
}
```

Q 8. On souhaite récupérer séparément la partie entière et la partie décimale des objets rationnels. Quels problèmes se posent ? Proposez, implémentez et testez des solutions.

Une petite parenthèse sur les nombres « réels »

On rappelle que la représentation des nombres dits réels est faite sur un nombre fixe d'octets (variable, pour chaque langage, selon le type). Cela a plusieurs conséquences :

- dans tous les langages : il est impossible de stocker de façon exacte des valeurs rationnelles dont l'écriture décimale comporte plus de chiffres que ne peut le fournir le type utilisé. C'est *a fortiori* le cas pour des fractions dont l'écriture décimale est infinie, comme $\frac{1}{3} = 0,3333...$
- dans la plupart des langages : les opérations successives, même inverses (par exemple multiplier par 100 puis diviser par 100) conduisent souvent à des erreurs d'arrondi.

En outre, Java est **très mauvais** pour des calculs même élémentaires. Il suffit pour s'en convaincre d'exécuter le programme suivant :

```
public class TestCalcul{
    public static void main(String[] args) {
        double x = 0 ;
        for (int i = 0; i<10; i++)
            x = x + 0.1 ;
        System.out.println(x) ;
    }
}
```

... édifiant, non ?