

TP 2 : Nombres rationnels

Préambule

But : Le but du TP est de programmer :

- deux implémentations du type abstrait **Rationnel** ;
- un programme client qui permettra de tester vos implémentations et de vérifier qu'il est simple de passer d'une implémentation à l'autre et même de mélanger des instances des deux implémentations.

Important

Pré-conditions et assertions :

- les pré-conditions sur les paramètres d'une fonction *doivent être vérifiées lors de l'instruction d'appel* ;
- pour « protéger » vos fonctions, vous testerez systématiquement leurs pré-conditions avec des *assertions* (voir document de préparation du premier TP).

Références et instances :

- dans tout votre programme (implémentations et client), les instances de rationnels seront obligatoirement désignées par des références de type **Rationnel** (le type de l'interface) quelle que soit leur implémentation ;

1 Le type abstrait Rationnel

1.1 Préparation

Le type abstrait **Rationnel** est destiné à modéliser des nombres rationnels (ou fractions) ; sa spécification, sous forme d'interface java se trouve dans le fichier `/share/esir1/prog/tp02/types/Rationnel.java` ; vous copierez ce fichier dans votre projet à l'intérieur d'un paquetage nommé `types`. Vous pouvez aussi copier le fichier `util/Outils.java` dans le paquetage `util`.

Remarques :

1. En java, les fichiers d'un paquetage **P** doivent être placés dans un répertoire nommé **P** ;
2. le constructeur à 2 paramètres possède une *pré-condition* : dénominateur non nul et une *post-condition* : fraction irréductible et dénominateur strictement positif ; cette post-condition correspond à l'état « *normalisé* » souhaité de chaque fraction à l'issue de son initialisation.

1.2 Implémentation simple

Dans la première implémentation (**RationnelSimple**), un rationnel sera représenté par deux entiers (le numérateur et le dénominateur, quelle surprise !) ; programmez cette implémentation dans le paquetage `rationnel`.

1.3 Programme client

1.3.1 première version

Le programme client sera placé dans le paquetage `rationnel`, dans un *nouveau fichier*.

Programmez les fonctions :

1. **static** **Rationnel** `lireRationnel` (**Scanner** `input`) : cette fonction effectue la saisie d'un numérateur et d'un dénominateur (non nul) puis crée et renvoie le rationnel correspondant ; dans cette première version, la fonction crée une instance de **RationnelSimple**.
2. fonction `main` : programmez une itération qui va lire au terminal une suite de rationnels (arrêt quand la valeur du rationnel saisi vaut zéro) et effectue les traitements suivants :

- afficher le dernier rationnel lu (grâce à la méthode `toString`);
- calculer et afficher sa somme avec le rationnel lu à l'itération précédente (zéro à la première itération);
- calculer et afficher son inverse si c'est possible;
- afficher sa valeur réelle;
- afficher (uniquement par appel de `compareTo`) s'il est plus petit, plus grand, ou égal à ce rationnel précédent;
- afficher (à l'aide de `equals`) s'il est égal à ce rationnel précédent.

NB : pas besoin de tableau, il suffit de mémoriser deux rationnels à chaque itération.

exemple :

```
courant = 2/3; 2/3 + 0 = 2/3; inverse = 3/2; valeur = 0.6666666666666666; 2/3 > 0; 2/3 ≠ 0
courant = 3/5; 3/5 + 2/3 = 19/15; inverse = 5/3; valeur = 0.6; 3/5 < 2/3; 3/5 ≠ 2/3
courant = 3/5; 3/5 + 3/5 = 6/5; inverse = 5/3; valeur = 0.6; 3/5 = 3/5; 3/5 = 3/5
```

1.3.2 deuxième version

Dans le même fichier que la première version, programmez ce qui suit :

1. **static** `Rationnel makeRationnel(int num, int den);`
cette fonction crée et renvoie une instance de `RationnelSimple` initialisée avec les paramètres de la fonction; on suppose que le dénominateur est non nul (pré-condition). Le but de cette fonction est de simplifier le changement d'implémentation.
À partir de maintenant, toutes les instances de rationnels créées dans le client devront l'être avec cette fonction (il faudra donc modifier `lireRationnel`).
2. **static void** `afficher(Rationnel [] lesRationnels, int nb);`
cette fonction affiche (fraction et valeur) les `nb` premiers éléments d'un tableau de rationnels; le tableau est supposé créé et initialisé avant l'appel et (bien sûr) $0 \leq nb \leq lesRationnels.length$.
3. modifiez la fonction `main` pour :
 - (a) créer un tableau de rationnels (de capacité « suffisante »);
 - (b) ajouter dans ce tableau chaque rationnel lu, ainsi que chacun des rationnels calculés; *l'ajout de chaque rationnel devra se faire de telle sorte que les éléments du tableau soient en permanence classés par valeur croissante*; cet ajout sera obligatoirement programmé dans une nouvelle fonction :

```
/**
 * insérer le rationnel nouveau dans le tableau lesRationnels
 * @pre : tableau trié (ordre croissant)
 * @param nb : nombre d'éléments dans le tableau avant ajout
 * @pre : 0 ≤ nb < lesRationnels.length
 * @post : tableau trié (ordre croissant)
 */
static void insererRationnel(Rationnel nouveau, Rationnel [ ] lesRationnels, int nb);
```

- (c) afficher le contenu du tableau à la fin de chaque itération.
4. **static** `Rationnel sommeRationnels(Rationnel [] lesRationnels, int nb);`
cette fonction calcule et renvoie la somme des `nb` premiers éléments d'un tableau de rationnels; le tableau est supposé créé et initialisé avant l'appel et (bien sûr) $0 \leq nb \leq lesRationnels.length$.
5. ajoutez à la fonction `main` l'appel de la fonction précédente et l'affichage du résultat.

1.4 Classe Couple

Programmez dans le paquetage `util` la classe générique `Couple` qui possède les fonctionnalités décrites en annexe (pas d'interface pour cette classe).

1.5 Implémentation avec un couple

Dans la deuxième implémentation (`RationnelCouple`), un rationnel sera représenté par un *couple d'entiers*; programmez cette implémentation dans le paquetage `rationnel`.

Modifiez le programme client pour qu'il utilise cette nouvelle implémentation à la place de la première : quel est le seul changement à effectuer dans le programme client ? Normalement, avec les mêmes données, vous devriez obtenir les mêmes résultats.

1.6 Programme client, troisième version

Modifiez la fonction `makeRationnel` afin de créer tantôt une instance de `RationnelSimple`, tantôt une instance de `RationnelCouple` (par exemple, en fonction de la parité du numérateur du rationnel à créer).

Pour vérifier le type des instances créées par les différentes opérations, faites afficher le nom de la classe par chaque constructeur.

Annexe

Spécification du TA Couple générique

Modélise un couple de valeurs de deux types quelconques `T1` et `T2` ; pas de spécification sous forme d'interface ; implémentation à programmer dans le paquetage `rationnel`.

- initialiser un couple avec deux valeurs ; pas de constructeur sans paramètre !
- accesseurs : valeur de la première, de la deuxième composante du couple (méthodes `getFirst` et `getSecond`) ;
- mutateurs : modifier la première, la deuxième composante du couple (`setFirst` et `setSecond`) ;
- égalité de deux couples : méthode `equals` ; attention au type du paramètre.