TP3 - Développement d'une Classe en TDD

M2103 - Prog. Obj.

Département Informatique - IUT de Bordeaux

Dans ce TP vous allez développer une classe `Rationnel` en utilisant la méthode *Test Driven Development* (TDD, développement piloté par les tests), qui repose sur le principe *Test First* de la méthode agile *Extreme Programming* (XP).

Développement piloté par les tests

Avec **TDD**, l'écriture de *tests automatisés* (code de test) dirige l'écriture du code source. C'est une technique très efficace pour livrer de logiciels bien construits, avec une suite de *tests de non-régression* qui permet de contrôler, après chaque modification, que tout marche encore.

Pendant la phase de développement

1. On écrit d'abord quelques tests
2. On écrit *ensuite* le code qui devrait les satisfaire (au moins partiellement)
3. On lance l'exécution des tests
4. On corrige la première erreur rencontrée, on améliore,
5. On passe à la fonctionnalité suivante.

La classe Rationnel

Il s'agit des fractions comme 1/2, 22/7, -8/3, etc.

Vous devez fournir une classe qui implémente ces fractions, avec des opérations comme la somme, le produit etc. ainsi que la récupération/modification du numérateur et du dénominateur.

La difficulté vient de la *normalisation* : en effet si on déclare un rationnel

```
Rationnel r { 6, -8 };
```

son numérateur sera -3, et son dénominateur 4. Et si on change son dénominateur en -6, on obtiendra 1/2.

Projet fourni

Le projet QTCreator fourni contient

- un fichier `main.cpp` avec quelques tests unitaires Boost qui vous permettront de commencer.
- un début de classe `Rationnel`

Le source est accepté à la compilation, mais bien évidemment, les tests échouent.

Travail demandé

1. Modifier le code de `Rationnel.cpp` pour que les tests réussissent (sans modifier les tests !)
2. Décommenter le test suivant dans `main.cpp`
3. Ajouter les déclarations de nouvelles fonctions nécessaires dans `Rationnel.h`
4. Implémenter le code
5. Tester
6. Passer au suivant.

Les fonctionnalités sont, dans l'ordre

- la construction
- les accesseurs
- la normalisation de signe
- la réduction
- l'addition

Pour la simplification vous aurez certainement besoin de la fonction pgcd, fournie dans le projet.

Fonctionnalités à ajouter

Pour la suite, vous écrivez vous-mêmes les tests

- la comparaison (== et !=)
- la conversion en chaine : `Rationnel(-3,4).toString()` doit produire `"-3/4"`. Et c'est mieux que 4/1 donne `"1"` plutôt que `"4/1"`.
- la valeur du rationnel (sous forme de nombre réel)
- sa valeur absolue,
- le produit de 2 rationnels
- etc.