TP 2: Classes Time et Rational

Exercice 1: Une classe Time

Pour cet exercice vous devez implementer une classe qui peut représenter l'heure. Chaque objet de votre classe doit contenir trois valeurs (pour représenter les heures, minutes, secondes). La signature de la classe doit contenir au moins les méthodes suivantes:

1. Deux constructeurs, un par défaut, et un qui prend comme paramètres trois `int` (heures, minutes, secondes).
2. Une méthode `void tick()` qui avance l'heure actuelle par une seconde.
3. Une méthode `String toString()` qui retourne une forme de l'objet qu'on peut afficher.

Compteur Cyclique

Pour implémenter la classe Time il vous faudra une structure de données qui représente des entier `modulo` une valeur, c'est-à-dire, un compteur cyclique : en fait, les valeurs stockées pour les minutes, secondes, sont toujours entre 0 et 59, alors que la valeur stockée pour l'heure est toujours entre 0 et 23. **Avant la class Time** programmez alors une class `Compteur` qui implément un compteur module une valeur, et utilisez cette classe dans votre implémentation de la classe Time. Quelques méthodes à inclure dans la class Compteur:

1. Une méthode `inc()` qui augmente la valeur du compteur par 1. Attn: il serait une bonne idée de signaler si cette augmentation a comme effet la re-initialisation du compteur.
2. Contructeurs, et une méthode `toString()`.

Exercice 2: Rational

Pour cet exercice vous devez programmer une classe qui représente les nombres rationnels. Un nombre rationnel peut être représenté par deux `int` (numérateur et dénominateur). La motivation pour adopter une telle classe (à la place de `double` par exemple) et que, si on n'utilise des valeurs au-delà de la valeur max de int, on evite les erreurs de précision associées avec la représentation habituelle des nombres réels.

Vous devez programmer une classe qui contient au moins:

1. Des constructeurs et une méthode `toString()`.
2. Des méthodes pour faire des opérations élémentaires avec des objets de type `Rational`, comme addition, soustraction, multiplication, inverse, etc.
3. Une méthode `toDouble()` qui retourne une version approximative de type double de la valeur d'un rationnel.

4. Une méthode `simplify()` qui retourne un rationnel avec la même valeur, mais où le nominateur et denominateur sont divisé par leurs plus grand commun diviseur.

Une partie de la signature de votre classe peut être:

```
1  public class Rational{
2          public Rational(int x, int y)
3          public Rational(int p){ this(p,1)
4          //Divide by gcd
5          public Rational simplify()
6          public String toString()
7          //Basic operations
8          public Rational add(Rational r)
9          public Rational mult(Rational r)
10         public Rational inv() //return 1/r
11         public Rational add(int r)
12         public Rational mult(int r)
13         public Rational div(Rational r)
14         public Rational sub(Rational r)
15         //Convert to double
16         public double toDouble()
17 }
```

Pour tester votre classe, utilisez le programme suivant:

```
1  public class RationalDemo {
2          public static void main(String [] args){
3                  Rational r = new Rational(0);
4                  //Geometric progression
5                  for(int i=0,j=1; i<15; i++, j*=2){
6                          r=r.add( new Rational(1,j));
7                          System.out.println(r);
8                          System.out.println(r.toDouble());
9                  }
10                 //pi approximation
11                 r = new Rational(0);
12                 for(int i=1;i<25; i+=2){
13                         r = r.add(new Rational(i).inv().mult(
14                                 new Rational(((i+1)%4==0?-1:1))));
15                         System.out.println(r.toDouble()*4);
16                 }
17                 //simplification test
18                 r = new Rational(1);
19                 for(int i=1;i<10;i++){
20                         r = r.div(new Rational(i+2,i+3));
21                         System.out.println(r);
22                 }
23         }
24 }
```