Exercice 26-a : Fractions

Programmez la classe `Fraction` qui représente une fraction mathématique !

La classe doit avoir le constructeur que voici :

- `Fraction(int pNumerator, int pDenominator)`

On suppose (sans vérification) que le dénominateur est différent de zéro, sinon les résultats des calculs sont imprévisibles. Ajouter aussi le *setter* suivant (pas de duplication de code!) :

- `setFraction(int pNumerator, int pDenominator)`

Cette classe doit avoir les méthodes suivantes :

- `int getNumerator()`
- `int getDenominator()`
- `double getDecimal()` retourne la valeur décimale de la fraction
- `String toString()`
  retourne un texte de la forme : numérateur/dénominateur (*valeur décimale*)
  si le dénominateur est 1, alors il n'est pas affiché
  Exemples :
  ```
  1/2 (0.5)
  125/100 (1.25)
  23 (23.0)
  ```
- `int gcd() / int lcm()`
  calcule et retourne le PGCD (resp. le PPCM) de a et b en utilisant une méthode au choix
- `simplify()`
  simplifie la fraction en utilisant la méthode ci-dessus. Une fraction est toujours à simplifier après chaque manipulation!

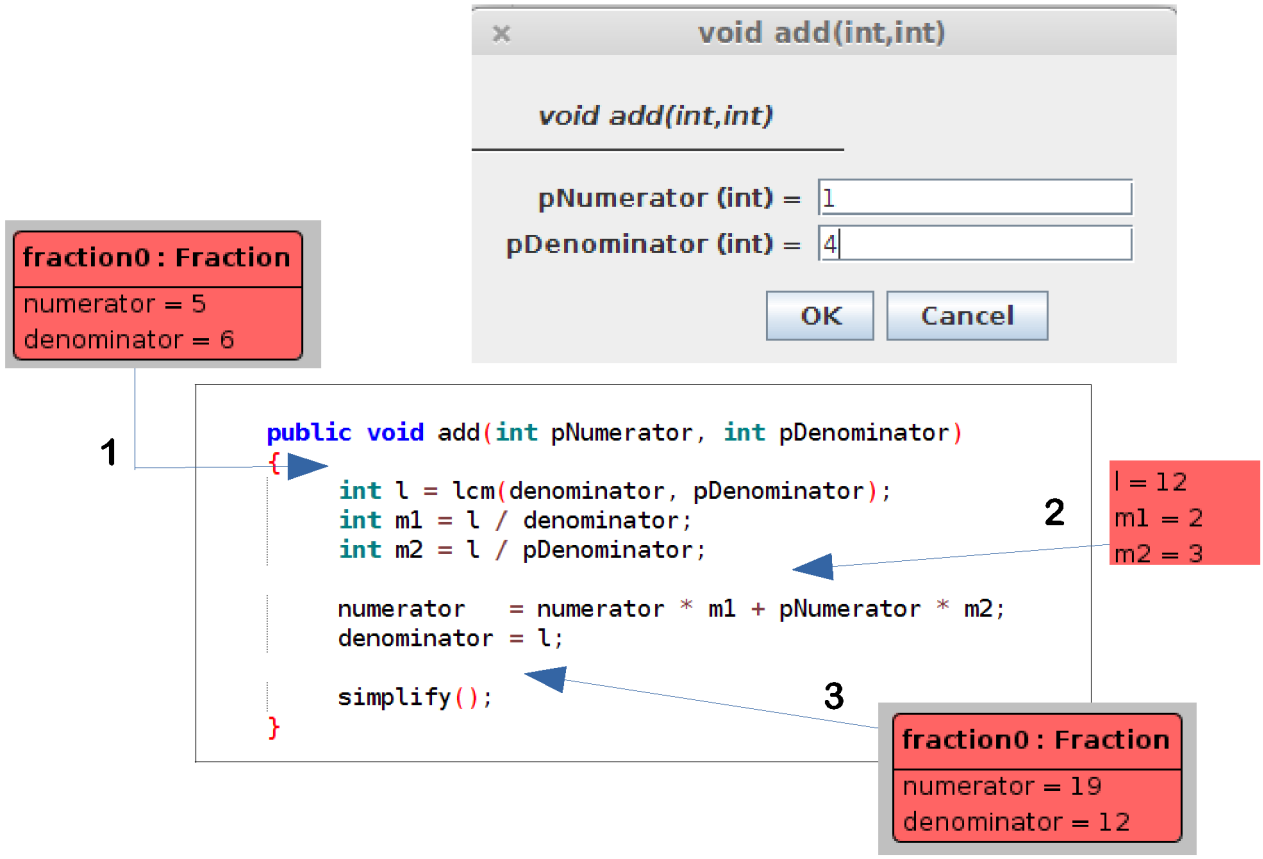
Les 4 méthodes `add, subtract, multiply, divide` permettent d'effectuer les opérations de base sur les fractions. Ces méthodes prennent à chaque fois le nominateur et le dénominateur comme paramètres et modifient la fraction (elles sont du type `void`).
Les deux paramètres sont la fraction à ajouter, soustraire, etc. à la fraction actuelle.

Pour tester ces méthodes faites une classe de test !

Fraction
− numerator : int
− denominator : int
+ Fraction(pNumerator : int, pDenominator : int)
+ setFraction(pNumerator : int, pDenominator : int) : void
+ getNumerator() : int
+ getDenominator() : int
+ getDecimal() : double
+ toString() : String
+ gcd(a : int, b : int) : int
+ lcm(a : int, b : int) : int
+ simplify() : void
+ add(pNumerator : int, pDenominator : int) : void
+ subtract(pNumerator : int, pDenominator : int) : void
+ multiply(pNumerator : int, pDenominator : int) : void
+ divide(pNumerator : int, pDenominator : int) : void

Explications sur l'exercice 26a

On effectue un appel de méthode : `add(1, 4)` sur l'objet `fraction0`
C'est les deux attributs qui contiennent le résultat (c'est à dire qui sont modifiés).



Étapes :

1. la fraction vaut $\frac{5}{6}$ (les attributs `numerator` et `denominateur` contiennent les valeurs 5 et 6 respectivement)
2. les calculs sont faits en utilisant ensemble les attributs et les paramètres ; les variables locales `l`, `m1` et `m2` contiennent les valeurs 12, 2 et 3 respectivement
3. les attributs de la fraction (objet `fraction0`) contiennent le résultat de l'addition (les attributs `numerator` et `denominateur` contiennent les valeurs 19 et 12 respectivement) et la fraction vaut $\frac{19}{12}$

Exercice 26-b

Ajoutez les méthodes `add, subtract, multiply, divide` qui permettent d'effectuer des calculs entre deux objets du type `Fraction`. On peut ajouter, soustraire, diviser et multiplier la fraction actuelle à/par une deuxième fraction passée comme paramètre. Pensez à réutiliser les méthodes existantes autant que possible !

La nouveauté pour cet exercice est qu'on peut aussi **passer un objet comme paramètre**, de la même façon qu'on peut passer des nombres ou des textes comme paramètres.

Fraction
− numerator : int
− denominator : int
+ Fraction(pNumerator : int, pDenominator : int)
+ setFraction(pNumerator : int, pDenominator : int) : void
+ getNumerator() : int
+ getDenominator() : int
+ getDecimal() : double
+ toString() : String
+ gcd(a : int, b : int) : int
+ lcm(a : int, b : int) : int
+ simplify() : void
+ add(pNumerator : int, pDenominator : int) : void
+ subtract(pNumerator : int, pDenominator : int) : void
+ multiply(pNumerator : int, pDenominator : int) : void
+ divide(pNumerator : int, pDenominator : int) : void
+ add(pF : Fraction) : void
+ subtract(pF : Fraction) : void
+ multiply(pF : Fraction) : void
+ divide(pF : Fraction) : void

Pour tester ces méthodes faites une classe de test !

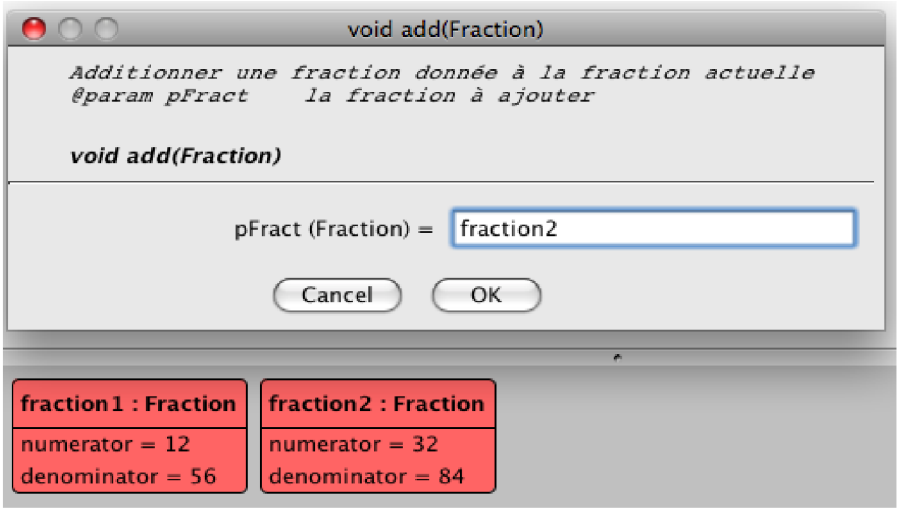
Exercice 26-c [Exercice IMPORTANT]

Enlevez les méthodes d'addition, soustraction, multiplication et division avec les deux paramètres entiers.

Fraction
− numerator : int
− denominator : int
+ Fraction(pNumerator : int, pDenominator : int)
+ setFraction(pNumerator : int, pDenominator : int) : void
+ getNumerator() : int
+ getDenominator() : int
+ getDecimal() : double
+ toString() : String
+ gcd(a : int, b : int) : int
+ lcm(a : int, b : int) : int
+ simplify() : void
+ add(pF : Fraction) : void
+ subtract(pF : Fraction) : void
+ multiply(pF : Fraction) : void
+ divide(pF : Fraction) : void

Pour tester ces méthodes faites une classe de test ! Ou alors, voir page suivante... mais c'est moins facile, car il faut toujours recréer des objets, définir les attributs et appeler les méthodes !

Pour tester ces méthodes en Unimozer, il faut que vous ayez créé deux instances de la classe `Fraction`.
Ensuite, vous pouvez par exemple ajouter une fraction à l'autre, en passant le nom de la deuxième fraction comme paramètre à la méthode `add` de la première fraction.



Exercice 26-d

Ajoutez les méthodes `add, subtract, multiply, divide` qui prennent **2 fractions** comme paramètres et retournent une nouvelle fraction. La fraction courante ainsi que les fractions f1 et f2 ne sont pas modifiées.

- `Fraction add(Fraction pF1, Fraction pF2)` $f_{res} = f_1 + f_2$
- `Fraction subtract(Fraction pF1, Fraction pF2)` $f_{res} = f_1 - f_2$
- `Fraction multiply(Fraction pF1, Fraction pF2)` $f_{res} = f_1 * f_2$
- `Fraction divide(Fraction pF1, Fraction pF2)` $f_{res} = f_1 / f_2$

Fraction
− numerator : int
− denominator : int
+ Fraction(pNumerator : int, pDenominator : int)
+ setFraction(pNumerator : int, pDenominator : int) : void
+ getNumerator() : int
+ getDenominator() : int
+ getDecimal() : double
+ toString() : String
+ gcd(a : int, b : int) : int
+ lcm(a : int, b : int) : int
+ simplify() : void
+ add(pF : Fraction) : void
+ subtract(pF : Fraction) : void
+ multiply(pF : Fraction) : void
+ divide(pF : Fraction) : void
+ add(pF1 : Fraction, pF2 : Fraction) : Fraction
+ subtract(pF1 : Fraction, pF2 : Fraction) : Fraction
+ multiply(pF1 : Fraction, pF2 : Fraction) : Fraction
+ divide(pF1 : Fraction, pF2 : Fraction) : Fraction

Notions requises : classe, objet, méthode, paramètre, type, attribut, void, constructeur
Structures requises : if, (for || while)