

Exercice 0

Le crible d'Érathostène

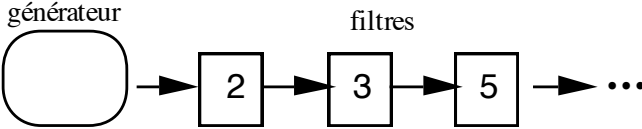
Donner une réalisation en Ruby du crible d'Erathostène, un algorithme qui permet de trouver l'ensemble des nombres premiers inférieurs ou égaux à n , n étant donné.

Principe :

Un générateur envoie des nombres de 2 à n de façon incrémentale. Une série de filtres sont placés de façon à récupérer les valeurs envoyées par les filtres précédents. Lorsqu'un nombre x est récupéré par un filtre i , si x est divisible par i , alors x n'est pas premier. Sinon x passe sur le filtre suivant. Si x n'a rencontré aucun diviseur à la sortie de la chaîne des filtres, on construit au bout de la chaîne un filtre de valeur x .

Traduire en java le code Ruby de l'exercice sur le crible d'Erathostène.

On fournira sur la ligne de commande le nombre n

La classe Generateur	La Classe Filtre
<pre>load "Filtre.rb" class Generateur def initialize() @premierFiltre=Filtre.valeur(2) end def to_s() "\nLa Liste des Filtres Générés est :\n"+@premierFiltre.to_s()+"\n" end def lanceToi(unNombre) 3.upto(unNombre) do i @premierFiltre.aToiDeTraiter(i) end end end</pre>	<pre>class Filtre private_class_method :new def Filtre.valeur(unNombre) new(unNombre) end def initialize(valeur) @valeur=valeur @suivant=nil end def aToiDeTraiter(unNombre) if unNombre==@valeur print(unNombre, " est premier\n") else if unNombre % @valeur ==0 print(unNombre, " n'est pas premier\n") else if @suivant!=nil @suivant.aToiDeTraiter(unNombre) else print(unNombre, " est premier\n") @suivant=Filtre.valeur(unNombre) end end end end def to_s() res = @valeur.to_s()+": " if @suivant res+=@suivant.to_s() end return res end end</pre>
	

Exercice 1

Le but de cet exercice est d'écrire la classe `EnsembleDeChiffres` permettant de représenter et manipuler des ensembles de chiffres.

Un ensemble de chiffres est représenté par un tableau `present` de 10 booléens, `present[i]` ayant la valeur `true` si l'entier `i` appartient à l'ensemble et `false` sinon. (Si certains avaient pensé (?) à utiliser un `BitSet`, nous ne retiendrons pas cette solution)

Par exemple l'ensemble `{0,3,4,6,7}` sera représenté par le tableau ci dessous

0	1	2	3	4	5	6	7	8	9
true	false	false	true	true	false	true	True	false	False

Les opérations définies sur un ensemble de chiffres sont : l'ajout d'un chiffre à l'ensemble, le retrait d'un chiffre à l'ensemble, la vérification qu'un chiffre est présent dans l'ensemble, le calcul de l'intersection de l'ensemble avec un autre ensemble, le calcul de l'union de l'ensemble avec un autre ensemble, le test de l'inclusion de l'ensemble dans un autre ensemble et l'affichage à l'écran de l'ensemble.

Le squelette de la classe `EnsembleDeChiffres` est donc le suivant :

```
public class EnsembleDeChiffres {
    private boolean[] present ;
    // constructeur
    public EnsembleDeChiffres()
        // construit et initialise present de sorte que l'objet EnsembleDeChiffres
        // créé corresponde à l'ensemble vide

    // méthodes
    public void ajoute(int i)
        // marque la présence du chiffre i dans l'ensemble
    public void retire(int i)
        // marque le chiffre i comme n'étant plus présent dans l'ensemble
    public void tirerAuHasard()
        // initialise au hasard l'ensemble
        // (la présence ou non de chaque chiffre est tirée au hasard)
    public boolean appartient(int i)
        // retourne true si le chiffre i appartient à l'ensemble false sinon
    public EnsembleDeChiffres intersectionAvec(EnsembleDeChiffres e)
        // construit et retourne un nouvel ensemble qui est le résultat de
        // l'intersection de l'ensemble sur lequel s'applique la méthode
        // et de l'ensemble e passé en paramètre
    public EnsembleDeChiffres unionAvec(EnsembleDeChiffres e)
        // construit et retourne un nouvel ensemble qui est le résultat de
        // l'union de l'ensemble receveur avec l'ensemble e
    public boolean estInclusDans(EnsembleDeChiffres e)
        // retourne true si l'ensemble est inclus dans l'ensemble e, false sinon
    public void affiche()
        // affiche l'ensemble à l'écran, sous la forme suivante : {0 4 6 9}
}
```

Rappels :

La méthode `random()` de la classe `Math` retourne une valeur réelle (un `double`) tiré au hasard dans l'intervalle `[0,1[`. Pour utiliser la classe `Math` il faut *importer* le *package* dans laquelle elle est définie (`java.lang.Math;`)

Exercice 2

Le but de cet exercice est d'écrire la classe `Rationnel`. Un objet de type `Rationnel` représente un rationnel au sens mathématique du terme (c'est-à-dire une fraction (p/q)).

Le squelette de la classe `Rationnel` est le suivant :

```
public class Rationnel
{
    private int p;
    private int q;

    // constructeur
    public Rationnel(int p,int q)

    // méthodes
    public int num()
        // Renvoie la valeur du numérateur du rationnel
    public int den()
        // Renvoie la valeur du dénominateur du rationnel
    public double toDouble()
        // Renvoie le double le plus proche du rationnel représenté par l'objet
    public Rationnel somme(Rationnel r)
        // Renvoie un nouvel objet Rationnel somme du rationnel sur lequel
        // s'applique la méthode et du rationnel r (this + r)
    public Rationnel difference(Rationnel r)
        // Renvoie un nouvel objet Rationnel différence du rationnel sur lequel
        // s'applique la méthode et du rationnel r (this - r)
    public Rationnel produit(Rationnel r)
        // Renvoie un nouvel objet Rationnel produit du rationnel sur lequel
        // s'applique la méthode et du rationnel r (this * r)
    public Rationnel quotient(Rationnel r)
        // Renvoie un nouvel objet Rationnel quotient du rationnel sur lequel
        // s'applique la méthode et du rationnel r (this / r)
    public int compareTo(Rationnel r)
        // Renvoie un entier :
        // < 0 si le rationnel sur lequel s'applique la méthode est plus petit
        //    que r
        // = 0 s'ils sont égaux
        // > 0 si r est plus petit
    public Rationnel abs()
        //Renvoie le rationnel valeur absolue du rationnel appelant.
}
```

Question 1 :

Écrire la classe `Rationnel` à partir du squelette donné.

Question 2 :

Écrire une méthode `public static Rationnel moyenne(Rationnel[] t)` qui renvoie la moyenne des rationnels du tableau `t`, calculée de façon exacte grâce à la classe `Rationnel`.

Question 3 :

On souhaite résoudre de façon exacte le système d'équations suivant :

$$\begin{cases} ax + by + c = 0 \\ dx + ey + f = 0 \end{cases}$$

On suppose que les coefficients a, b, c, d, e et f sont des entiers.

On forme d'abord le déterminant du système $g = ae - bd$. Si $g \neq 0$, le système possède une seule solution donnée par :

$$x = \frac{bf - ce}{g} \text{ et } y = \frac{cd - af}{g}$$

Écrire la méthode qui résout le système d'équations décrit par les paramètres entiers :

```
public static Rationnel[] systeme(int a, int b, int c, int d, int e, int f).
```

Si ce système n'a pas de solution unique, la méthode devra renvoyer un tableau vide. Sinon, elle devra renvoyer, sous forme d'un tableau de `Rationnel`, les deux `Rationnel` x et y calculés exactement (avec `tab[0]` = un objet de type `Rationnel` représentant x et `tab[1]` = un objet de type `Rationnel` représentant y).

Question 4 :

La principale limitation des rationnels est qu'ils ne peuvent pas représenter de façon exacte certains nombres réels.

L'exemple le plus simple est $\sqrt{2}$ qui est un irrationnel. Pour représenter de façon approchée une racine carrée, on utilise la propriété suivante. Étant donné un réel $x > 0$, la suite u_n définie comme suit converge (très rapidement) vers \sqrt{x} :

$$\begin{cases} u_0 = \frac{x}{2} \\ u_k = \frac{1}{2} \left(u_{k-1} + \frac{x}{u_{k-1}} \right) \text{ pour } k > 0 \end{cases}$$

Écrire la méthode `public static Rationnel sqrt(Rationnel r, Rationnel precision)` qui renvoie la valeur approchée de \sqrt{r} , calculée en utilisant la suite définie plus haut. On arrête le calcul de la suite quand $|u_{k-1} - u_k| < \text{precision}$, à ce moment là, u_k est la valeur approchée de \sqrt{r} avec une précision de `precision`.

Attention : on veut un résultat `Rationnel`, pour cela on ne manipule que des `Rationnel`, pas de réels.

Question 5 :

En utilisant la méthode `sqrt` qui vient d'être définie, écrire la méthode suivante :

```
public static Rationnel std(Rationnel[] t, Rationnel precision) qui renvoie l'écart-type du tableau de Rationnel t.
```

On rappelle que l'écart-type de la liste de valeur (t_0, \dots, t_{n-1}) est donné par :

$$\sqrt{\frac{1}{n} \sum_{i=0}^{n-1} (t_i - m)^2},$$

où m désigne la moyenne de la liste.

Question 6 :

Écrire la méthode suivante :

```
public static Rationnel[] trinome(Rationnel[] coeffs, Rationnel precision) qui calcule la ou les racine(s) réelle(s) du trinôme dont les coefficients sont rangés dans le tableau coeffs (coeffs[i] correspond au coefficient de  $x^i$ ). S'il n'existe pas de racine réelle, le tableau renvoyé devra être vide.
```