Ensemble ordonné

Exercice 1 : Définition d'un ensemble

L'objectif de cet exercice est de définir des ensembles d'entiers qui pourront être utilisés dans différents contextes.

Nous prendrons comme exemple d'utilisation de l'ensemble, le calcul des nombres premiers par une variante de l'algorithme dit du « crible d'Ératosthène ». Ce n'est cependant qu'un exemple.

Pour trouver tous les nombres premiers de 2 à MAX, le principe du crible d'Ératosthène est le suivant :

1. Construire l'ensemble contenant tous les entiers de 2 à MAX ;
2. Extraire et afficher le plus petit élément de l'ensemble car c'est un nombre premier ;
3. Enlever de l'ensemble tous les multiples de ce nombre premier ;
4. Continuer en 2 jusqu'à ce que l'ensemble soit vide.

1.1 Spécifier en UML puis écrire en Java `Ensemble` qui décrit un ensemble d'entiers.

Pour simplifier, on définira seulement les opérations sur les ensembles qui sont utiles pour le crible d'Ératosthène, c'est-à-dire ajouter un élément, supprimer un élément, savoir si un élément est présent ou non, savoir si l'ensemble est vide ou non, connaître la cardinalité de l'ensemble, donner le plus petit élément de l'ensemble. On ne considérera donc pas les opérations sur les ensembles telles que l'union, l'intersection, l'inclusion, etc.

1.2 Écrire l'algorithme qui calcule les nombres premiers en utilisant le principe du crible d'Ératosthène (et l'ensemble défini, bien sûr !) ainsi qu'un programme qui l'utilise.

1.3 On hésite sur la manière d'implanter ces ensembles. On envisage d'utiliser un tableau pour stocker les éléments ou de les chaîner entre eux.

1.3.1 Dessiner le diagramme de classe qui fait apparaître ces deux implantations des ensembles.

1.3.2 Pour les deux implantations de l'ensemble, écrire la méthode qui :

1. ajoute un élément dans un ensemble,
2. donne le plus petit élément d'un ensemble.

1.3.3 Est-ce que les implantations envisagées sont efficaces ? Quelles implantations seraient plus efficaces ?

Exercice 2 : Généraliser les ensembles du crible d'Ératosthène

Dans l'exercice 1, nous avons eu le souci de développer une classe Ensemble réutilisable. Cependant, nous nous sommes limités à des ensembles d'entiers. Il serait intéressant de disposer d'ensemble de points, de nombres réels, etc.

Proposer une solution pour atteindre cet objectif.

TOB
TD2.1

Ensemble ordonné

Exercice 1 : Définition d'un ensemble

2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~ ... MAX

Ensemble de nombres naturels.

1/ Ensemble Entier

Void Ensemble (MAX Entier)

void AjouterElement (e Entier)
post (E.present(e))

void SupprimerElement (e Entier)
post (! E.present(e))

boolean Present (Entier e)

boolean EstVide ( )

Entier Cardinal ( )

Entier PPe ( )
pre (! Estvide ( ))

Ensemble Entier
<< requête >>
int getMinimum
boolean estPresent (int)
int getCardinal
boolean estVide
<< commandes >>
void addElement (int)
void deleteElement (int)
<<constructeur >>
Ensemble Entier (int taille)

Norme Javabeans : pour une requête -> getMinimum
isPresent
getCardinal
isVide isEmpty
add
delete

2/
```
@pre : maximum >= 10
void cribler (int maximum) {
    Ensemble Entier support = new Ensemble Entier (maximum - 1)
    for (int i = 2 ; i <= maximum ; i++) {
        support. addElement (i);
    }
    do {
        int p = support. getMinimum ( );
        System.out.println (p);
        For (int i = p ; i <= maximum ; i += p) {
```

```
            support.deleteElement(i);
        }
    } while (!support.estVide());
```

Cette méthode doit être dans une classe

```
public class CribleEratosthene {
    public static void main(String args[]) {
        cribler(1000);
    }
    private static void cribler(int...) {
        :
    }
```

3/a)

```
public interface EnsembleEntier {
    boolean estVide();          boolean estPresent(int);
    int getMinimum();
    int getCardinal();
    void addElement(int);
    void deleteElement(int);
}

public class EnsembleTableauEntier implements EnsembleEntier {
    private int support[], private int cardinal;
    public EnsembleTableauEntier(int taille) {
        support = new int[taille]; cardinal = 0;
    }
    boolean estVide() {                 int cardinal() {
        return (this.cardinal == 0);        return this.cardinal
    }                                   }
```

TOB
TD 2.2

```
int minimum () {
    return support[0];
}

void delElement (int valeur) {
    int position;
    for (position=0; position < cardinal
        && support[position] != valeur;
        position ++) {}
    if (position < cardinal) {
        for ( ; position < (cardinal -1);
            position ++) {
            support[position] = support[position+1];
        }
        cardinal = cardinal - 1;
    }
}

void ajouterElement (int valeur) {
    int position;
    for (position=0; support[position] <= valeur;
        ++position) {}
    if (support[position] != valeur) {
        for (int indice = cardinal -1; indice > position;
            indice --) {
            support[indice+1] = support[indice];
        }
        support[position] = valeur;
        cardinal ++; }
}
```