

## Examen - 26 Janvier 2009

2 heures - seul document autorisé : un aide mémoire sous forme d'une feuille A4 recto verso

### 1 Le problème du sac à dos

On cherche à remplir un sac à dos qui ne peut supporter plus d'un certain poids, avec tout ou partie d'un ensemble d'objets ayant chacun un poids et une valeur. Les objets mis dans le sac à dos doivent maximiser la valeur totale, sans dépasser le poids maximum.

#### 1.1 La classe `Objet` (environ 10 mn)

La première étape est de définir une classe `Objet` qui définit un objet que l'on peut mettre dans le sac à dos. La variable d'instance `poids` est le poids de l'objet sous forme d'un entier, la variable d'instance `valeur` est un entier qui contient la valeur de l'objet.

```
public class Objet {
    private int poids;
    private int valeur;
    public Objet (int poidsMaximum, int valeurMaximum) {}
    public int getPoids () {}
    public int getValeur () {}
    public String toString () {}
}
```

Programmer la classe `Objet` en respectant la documentation suivante :

1. Le constructeur `Objet (int poidsMaximum, int valeurMaximum)` construit un objet ayant un poids aléatoire compris entre 0 et `poidsMaximum` ainsi qu'une valeur aléatoire comprise entre 0 et `valeurMaximum`. On rappelle que la fonction `double Math.random ()` renvoie un double entre 0.0 et 1.0.
2. Les méthodes "observateurs" `int getPoids ()` et `int getValeur ()` retournent respectivement le poids et la valeur de l'objet appelant.
3. La méthode `String toString ()` renvoie une chaîne décrivant l'objet appelant sous la forme `[poids, valeur]`.

#### 1.2 La classe `ListeObjets` (environ 10 mn)

La classe `ListeObjets` représente une liste d'objets à l'aide d'une `ArrayList`.

```

import java.util.*;
public class ListeObjets {
    private ArrayList<Objet> liste;
    public ListeObjets (int n, int pmaxi, int vmaxi) {}
    public Objet get (int i) {}
    public int size () {}
    public String toString () {}
    public static void main (String[] args) {
        ListeObjets l = new ListeObjets (30, 100, 100);
        System.out.println (l);
    }
}

```

Programmer la classe `ListeObjets` en respectant la documentation suivante :

1. Le constructeur `ListeObjets (int n, int pmaxi, int vmaxi)` construit une liste de  $n$  objets dont les poids sont tirés aléatoirement entre 0 et  $pmaxi$  et les valeurs tirées aléatoirement entre 0 et  $vmaxi$ .
2. La méthode `Objet get (int i)` retourne l'objet d'indice  $i$  dans la liste d'objets.
3. La méthode `int size ()` retourne le nombre d'objets dans la liste d'objets.
4. La méthode `String toString()` retourne une chaîne de caractères représentant la liste d'objets.

### 1.3 La classe Individu

La classe `Individu` représente une liste de choix à l'aide d'une `ArrayList` nommée `liste`. Un choix est représenté par un `Integer` : une valeur de 0 correspond à choisir de ne pas prendre l'objet dans le sac à dos alors qu'une valeur de 1 correspond à choisir de prendre l'objet dans le sac à dos. L'élément d'indice  $i$  de la liste de choix correspond au choix du  $i$ ème objet de la liste d'objets de la variable statique `listeObjets`. La variable statique  $p$  représente le poids maximum que peut contenir le sac à dos. La classe `Individu` implante l'interface `Comparable` car nous allons par la suite trier les individus.

```

import java.util.*;
public class Individu implements Comparable {
    private ArrayList<Integer> liste;
    private static int p;
    private static ListeObjets listeObjets;
    public Individu () {}
    public static void setProblem (ListeObjets l, int w) {}
    public void set (int i, int val) {}
    public int get (int i) {}
    public int size () {}
    public int evaluation () {}
    public int compareTo (Object obj) {}
    public int optimum (int profondeur) {}
    public String toString () {}
    public static void main (String[] args) {
        ListeObjets l = new ListeObjets (25, 100, 100);
        setProblem (l, 750);
    }
}

```

```

        Individu ind = new Individu ();
        System.out.println (l);
        System.out.println (ind.optimum (0));
    }
}

```

Programmer la classe `Individu` en respectant la documentation suivante :

1. Le constructeur `Individu ()` construit la liste de choix `liste` dont la taille est la même que celle de la liste `listeObjets`. Chaque choix est tiré aléatoirement et vaut soit 0 soit 1.
2. La méthode `static void setProblem (ListeObjets l, int w)` affecte les variables statiques `listeObjets` et `p` respectivement avec les paramètres `l` et `w`.
3. La méthode `void set (int i, int val)` affecte un *Integer* correspondant à `val` à l'élément d'indice `i` de la liste de choix `liste`.
4. La méthode `int get (int i)` retourne un entier correspondant au choix d'indice `i` dans la liste de choix.
5. La méthode `int size ()` retourne le nombre de choix de la liste de choix.
6. La méthode `int evaluation ()` retourne l'évaluation de la liste de choix. On calcule pour cela le poids total et la valeur totale de tous les objets choisis (les objets de `listeObjets` dont le choix correspondant à leur indice vaut 1). Si le poids total dépasse `p`, l'évaluation retourne 0. Sinon l'évaluation renvoie la valeur totale.
7. Comme la classe `Individu` implante l'interface `Comparable`, on doit écrire la méthode `int compareTo (Object obj)` qui compare deux individus. La variable `obj` doit être transtypée vers un individu et la méthode renvoie une valeur négative si l'évaluation de l'individu passé en paramètre est plus grande que celle de l'individu appelant, une valeur nulle si les deux évaluations sont égales et une valeur positive si l'individu appelant a une meilleure évaluation que l'individu passé en paramètre .
8. La méthode `int optimum (int profondeur)` est une fonction récursive qui retourne l'évaluation de la meilleure configuration possible d'objets dans le sac à dos. Elle doit être appelée la première fois avec une valeur initiale de `profondeur` égale à 0. L'arrêt de la récursivité se fait quand `profondeur` est égale à la taille de la liste de choix, dans ce cas la méthode renvoie l'évaluation de la liste de choix. Dans le cas contraire il y a deux appels récursifs. Avant le premier appel récursif le choix d'indice `profondeur` est mis à 0, puis on fait un appel récursif à `profondeur + 1` dont on mémorise le résultat. Avant le deuxième appel récursif le choix d'indice `profondeur` est mis à 1. Le résultat renvoyé est le maximum des résultats des deux appels récursifs.
9. La méthode `String toString()` retourne une chaîne de caractères représentant la liste des choix ainsi que son évaluation.
10. On demande de plus d'écrire une nouvelle classe nommée `OrdreDecroissant` qui implante l'interface `Comparator` et qui permettra de comparer deux `Individus` pour les classer dans l'ordre décroissant de leur évaluation. La méthode `int compare (Object o1, Object o2)` de cette classe transtypera `o1` et `o2` vers des `Individus` puis renverra une valeur positive si le deuxième a une meilleure évaluation que le premier, une valeur nulle s'ils sont égaux, une valeur négative sinon.

## 1.4 La classe ListeIndividus

La classe `ListeIndividus` représente une liste d'Individus à l'aide d'une `ArrayList`.

```
import java.util.*;
public class ListeIndividus {
    private ArrayList<Individu> liste;
    public ListeIndividus (int n) {}
    public Individu get (int i) {}
    public void tri () {}
    public ListeIndividus reproduction () {}
    public int generations (int nbIndividus, int nbGenerations) {}
    public String toString () {}
    public static void main (String[] args) {
        ListeObjets l = new ListeObjets (25, 100, 100);
        Individu.setProblem (l, 750);
        Individu ind = new Individu ();
        ListeIndividus listeInd = new ListeIndividus (30);
        System.out.println (l);
        System.out.println ("optimum = " + ind.optimum (0));
        listeInd.generations (1000, 30);
    }
}
```

Programmer la classe `ListeIndividus` en respectant la documentation suivante :

1. Le constructeur `ListeIndividus (int n)` construit une liste de  $n$  individus tirés aléatoirement.
2. La méthode `Objet get (int i)` retourne l'individu d'indice  $i$  dans la liste d'individus.
3. La méthode `void tri ()` trie la liste d'individus en ordre décroissant à l'aide de la méthode `void sort (List list, Comparator c)` de la classe `Collections`.
4. La méthode `ListeIndividus reproduction ()` renvoie une nouvelle liste d'individus créée à partir de la liste `liste` et qui a la même taille (le même nombre d'individus). Chaque individu de la nouvelle liste est créé à partir de deux individus `ind1` et `ind2`, tirés aléatoirement dans le premier quart de la liste `liste`. On choisit aussi aléatoirement pour chaque nouvel individu un indice  $i$  compris entre 0 et la taille d'un individu. Le nouvel individu comportera les choix d'indices 0 à  $i$  de `ind1` et les choix d'indices  $i$  à sa taille de `ind2`. Par exemple si `ind1` vaut 11111111, `ind2` vaut 00000000 et  $i$  vaut 4, le nouvel individu vaudra 11110000.
5. La méthode `void generations (int nbIndividus, int nbGenerations)` crée une liste d'individus `nbGenerations` fois. La première liste d'individus nommée `listeInd` est créée aléatoirement et comporte `nbIndividus`. Ensuite, à chaque génération, la liste d'individus `listeInd` est triée à l'aide de la fonction `tri ()`, puis `listeInd` est affectée avec le résultat de la méthode `reproduction ()` appliquée à `listeInd`.
6. La méthode `String toString()` retourne une chaîne de caractères représentant la liste d'individus.