Examen de Java - Modèle 1

I. Questions de Cours (5 points).

1. Un objet décrit souvent une portion du monde réel, représenté numériquement. Quelles sont les deux composantes de cette description ?

Un état, et un (ou des) comportement(s)

2. Considérez la classe suivante :

```
public class IdentifyMyParts {
    public static int x = 7;
    public int y = 3;
}
```

Soit le code suivant ajouté à la classe IdentifyMyParts

```
IdentifyMyParts a = new IdentifyMyParts();
IdentifyMyParts b = new IdentifyMyParts();
a.y = 5;
b.y = 6;
IdentifyMyParts.x = 1;
b.x = 2;
System.out.println("a.y = " + a.y);
System.out.println("b.y = " + b.y);
System.out.println("IdentifyMyParts.x = " + a.x);
System.out.println("b.x = " + b.x);
```

Quelle est la sortie de ce programme ?

a.y=5
b.y=6
IdentifyMyParts.x=2
b.x=2

3. Pourquoi une classe finale ne peut être simultanément abstraite ?

Parce qu'une classe abstraite doit être dérivée pour pouvoir être utilisée, ce que final empêche

4. Voici un segment de code :

```
try {
} catch (Exception e) {
   ...
} catch (ArithmeticException a) {
   ...
}
```

Un tel segment de code compile-t-il ? Précisez.

Non, car le deuxième catch capture une exception plus spécialisée que la première : le code est inatteignable

5. L'interface suivante est-elle correcte ?

```
public interface SomethingIsWrong {
    void aMethod(int aValue){
        System.out.println("Hi Mom");
    }
}
```

Non, pas de code dans une interface

II. Exercice (5 points).
Thème : Factorisation

```
class Navet{
   double poids;
   double prixKilo;
   String categorie;
   boolean epluche = false;

   public Navet(double poids, double prix, String categorie){
      // COMPLETER //
   }

   public void peler(){
      // COMPLETER //
   }
```

```
   public double peser(){
      // COMPLETER //
   }
}
```

```
class Carotte{
   double poids;
   double prixKilo;
   double diametre;
   boolean epluche = false;

   public Carotte(double poids, double prix, double diametre){
      // COMPLETER //
   }

   public void eplucher(){
      // COMPLETER //
   }

   public double peser(){
      // COMPLETER //
   }
}
```

```
class PetitPois{
   double poids;
   double prixKilo;
   double nombregrainsmoyen;
   boolean ecosse = false;

   public PetitPois(double poids, double prix, double diametre){
      // COMPLETER //
   }

   public int ecosser(){
      // COMPLETER //
   }

   public double peser(){
      // COMPLETER //
   }
}
```

1. Compléter le code des trois classes (intelligemment).

Première étape : implémentation des fonctions triviales

```
/*
 * Created on 1 juin 2007
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

class Navet{
   double poids;
   double prixKilo;
   String categorie;
   boolean epluche = false;

   public Navet(double poids, double prix, String categorie){
      this.poids = poids;
      this.prixKilo = prix;
      this.categorie = categorie;
   }

   public void peler(){
      epluche = true;
   }

   public double peser(){
      return poids * prixKilo;
   }
}
```

```
/*
 * Created on 1 juin 2007
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

public class Carotte {
   double poids;
   double prixKilo;
   double diametre;
   boolean epluche = false;
```

```
   public Carotte(double poids, double prix, double diametre){
      this.poids = poids;
      prixKilo = prix;
      this.diametre = diametre;
   }
   public void eplucher(){
      epluche = true;
   }

   public double peser(){
      return poids * prixKilo;
   }

}
```

```
/*
 * Created on 1 juin 2007
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

public class PetitPois {
   double poids;
   double prixKilo;
   double nombregrainsmoyen;
   boolean ecosse = false;

   public PetitPois(double poids, double prix, double nombregrains){
      this.poids = poids;
      prixKilo = prix;
      nombregrainsmoyen = nombregrains;
   }
   public void ecosser(){
      ecosse = true;
   }

   public double peser(){
      return poids * prixKilo;
   }

}
```

2. Trouvez une superclasse intéressante et factorisez.

La superclasse commune est bien sûr Legume !!

```
/*
 * Created on 1 juin 2007
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

/**
 * Factorization tracklog
 *
 * STEP1
 * create class Legume
 * add field poids
 * add field prixKilo
 * add constructor(poids, prixKilo)
 * add constructor assignation to field poids
 * add constructor assignation to field prixKilo
 *
 * STEP2
 * add peser
 *
 * STEP3
 * add abstract method preparer()
 * make Legume abstract
 **/

abstract public class Legume {
   double poids;
   double prixKilo;

   public Legume(double poids, double prix){
      this.poids = poids;
      this.prixKilo = prix;
   }

   public double peser(){
      return poids * prixKilo;
   }

   abstract void preparer();
}
```

La classe Légume factorise les trois classes Navet, Carotte et PetitPois selon les procédures suivantes, en commençant par la classe Carotte et en appliquant un procédé similaire aux autres classes :

```
/*
 * Created on 1 juin 2007
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

/**
 * Factorization tracklog
 *
 * STEP 1 fields and constructors
 * field facto poids -> class Legume
 * field facto prixKilo -> class Legume
 * field delete poids
 * field delete prixKilo
 * extends Legume
 * delete constructor poids assignement
 * delete constructor prixKilo assignement
 * add super(poids,prixKilo)
 *
 * STEP 2 other methods
 * export peser -> Legume
 *
 * STEP 3 resolve abstraction
 * add method preparer
 * make preparer() call eplucher()
 **/

public class Carotte extends Legume{
   double diametre;
   boolean epluche = false;

   public Carotte(double poids, double prix, double diametre){
      super(poids, prix);
      this.diametre = diametre;
   }

   public void eplucher(){
      epluche = true;
   }

   @Override
   void preparer() {
      eplucher();
   }
}
```

```
/*
 * Created on 1 juin 2007
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

/**
 * Factorization tracklog
 *
 * STEP1
 * from (Carotte)
 * field facto poids -> class Legume
 * field facto prixKilo -> class Legume
 * field delete poids
 * field delete prixKilo
 * extends Legume
 * delete constructor poids assignement
 * delete constructor prixKilo assignement
 * add super(poids,prixKilo)
 * (/Carotte)
 *
 * STEP2
 * from (Carotte)
 * delete peser()
 * (/Carotte)
 *
 * STEP3
 * from (Carotte)
 * add preparer()
 * make preparer() call peler()
 * (/Carotte)
 **/

class Navet extends Legume{
   String categorie;
   boolean epluche = false;

   public Navet(double poids, double prix, String categorie){
      super(poids, prix);
      this.categorie = categorie;
   }
```

```
   public void peler(){
      epluche = true;
   }

   @Override
   void preparer() {
      peler();
   }
}
```

```
/*
 * Created on 1 juin 2007
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

/**
 * Factorization tracklog
 *
 * STEP 0 Resolve errors
 * turn ecosser() void
 *
 * STEP 1 fields and constructors
 * from (Carotte)
 * field facto poids -> class Legume
 * field facto prixKilo -> class Legume
 * field delete poids
 * field delete prixKilo
 * extends Legume
 * delete constructor poids assignement
 * delete constructor prixKilo assignement
 * add super(poids,prixKilo)
 * (/Carotte)
 *
 * STEP 2 Other methods
 * from (Carotte)
 * delete peser()
 * (/Carotte)
 *
 * STEP 3 resolve abstraction
 * add method preparer
 * make preparer() call ecosser()
 **/

public class PetitPois extends Legume {
   double nombregrainsmoyen;
   boolean ecosse = false;

   public PetitPois(double poids, double prix, double nombregrainsmoyen){
      super(poids,prix);
      this.nombregrainsmoyen = nombregrainsmoyen;
   }

   public void ecosser(){
      ecosse = true;
   }

   @Override
   void preparer() {
      ecosser();
   }
}
```

III. Problème (7 points).
Thème : Jeu de cartes

On désire développer un modèle d'un jeu de cartes qui met en relations trois entités :

Une classe qui modélise les cartes :

- une valeur en points
- un nom de carte (chaîne)
- une famille (chaîne)
- des méthodes d'accesseur
- une méthode toString() qui permet d'exprimer la carte

```
/*
 * Created on 1 juin 2007
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

public class Carte {
   public static final Carte AS_COEUR = new Carte("Coeur", "As", 1);
   public static final Carte SEPT_COEUR = new Carte("Coeur", "Sept", 7);
   public static final Carte HUIT_COEUR = new Carte("Coeur", "Huit", 8);
```

```
   public static final Carte NEUF_COEUR = new Carte("Coeur", "Neuf", 9);
   public static final Carte DIX_COEUR = new Carte("Coeur", "Dix", 9);
   public static final Carte VALET_COEUR = new Carte("Coeur", "Valet", 11);
   public static final Carte DAME_COEUR = new Carte("Coeur", "Dame", 12);
   public static final Carte ROI_COEUR = new Carte("Coeur", "Roi", 13);

   public static final Carte AS_TREFLE = new Carte("Trefle", "As", 1);
   public static final Carte SEPT_TREFLE = new Carte("Trefle", "Sept", 7);
   public static final Carte HUIT_TREFLE = new Carte("Trefle", "Huit", 8);
   public static final Carte NEUF_TREFLE = new Carte("Trefle", "Neuf", 9);
   public static final Carte DIX_TREFLE = new Carte("Trefle", "Dix", 9);
   public static final Carte VALET_TREFLE = new Carte("Trefle", "Valet", 11);
   public static final Carte DAME_TREFLE = new Carte("Trefle", "Dame", 12);
   public static final Carte ROI_TREFLE = new Carte("Trefle", "Roi", 13);

   public static final Carte AS_PIQUE = new Carte("Pique", "As", 1);
   public static final Carte SEPT_PIQUE = new Carte("Pique", "Sept", 7);
   public static final Carte HUIT_PIQUE = new Carte("Pique", "Huit", 8);
   public static final Carte NEUF_PIQUE = new Carte("Pique", "Neuf", 9);
   public static final Carte DIX_PIQUE = new Carte("Pique", "Dix", 9);
   public static final Carte VALET_PIQUE = new Carte("Pique", "Valet", 11);
   public static final Carte DAME_PIQUE = new Carte("Pique", "Dame", 12);
   public static final Carte ROI_PIQUE = new Carte("Pique", "Roi", 13);

   public static final Carte AS_CARREAU = new Carte("Carreau", "As", 1);
   public static final Carte SEPT_CARREAU = new Carte("Carreau", "Sept", 7);
   public static final Carte HUIT_CARREAU = new Carte("Carreau", "Huit", 8);
   public static final Carte NEUF_CARREAU = new Carte("Carreau", "Neuf", 9);
   public static final Carte DIX_CARREAU = new Carte("Carreau", "Dix", 9);
   public static final Carte VALET_CARREAU = new Carte("Carreau", "Valet", 11);
   public static final Carte DAME_CARREAU = new Carte("Carreau", "Dame", 12);
   public static final Carte ROI_CARREAU = new Carte("Carreau", "Roi", 13);

   //   une valeur en points
   int valeur;

   //   un nom de carte (chaîne)
   String nom;

   //   une famille (chaîne)
   String famille;

   public Carte(String f, String n, int v){
      valeur = v;
      nom = n;
      famille = f;
   }

   //   des méthodes d'accesseur

   public int getValeur(){
      return valeur;
   }

   public String getNom(){
      return nom;
   }

   public String getFamille(){
      return famille;
   }

   //   une méthode toString() qui permet d'exprimer la carte
   public String toString(){
      String str = "";

      str += nom + " de " + famille + "(" + valeur + ")";

      return str;
   }
}
```

Commentaire : la seule particularité de la correction est l'emploi d'objets statiques constants pour "faire" les cartes. On pouvait bien entiendu construire des cartes dans la méthode main de la classe Paquet, mais cette façon de faire (une classe propose ses propres "instances prédéfinies") est bien plus jolie.

Une classe qui modélise le paquet (jeu) de cartes :

- une collection des cartes du jeu
- un constructeur qui construit un jeu vide
- une méthode qui permet d'enregistrer des cartes uniques dans le jeu
- une méthode qui permet de tirer une main au hazard en retirant les cartes du jeu et renvoie la main
- une méthode toString() qui permet d'exprimer le jeu

```
/*
 * Created on 1 juin 2007
 *
```

```
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

public class Paquet {
   //   une collection des cartes du jeu
   // option 1 : réalisation avec un tableau
    Carte[] cartes;

   /* option 2 : avec les collections
    Vector cartes;
    */

   //   un constructeur qui construit un jeu vide
    public Paquet(){
       // option 1
        cartes = new Carte[0];

       /*
        * option 2
        *
       cartes = new Vector();
        */
    }

   //   une méthode qui permet d'enregistrer des cartes uniques dans le jeu
    public boolean addCarte(Carte c){
        boolean res = false;

       // option 1 : Syntaxe Java 5.0
        for(Carte uneCarte : cartes){
            if (c == uneCarte) return false;
        }

        Carte[] nouvellesCartes = new Carte[cartes.length + 1];
        System.arraycopy(cartes, 0, nouvellesCartes, 0, cartes.length);
        nouvellesCartes[cartes.length] = c;
        cartes = nouvellesCartes;
        res = true;

        return res;

       // option 2 : Syntaxe Java avec collection Vector
       /*
      if (!cartes.contains(c)){
         cartes.add(c);
      }
       */
    }

   //   une méthode qui permet de tirer une main au hazard en retirant les cartes du jeu et renvoie la main
    public Main makeMain(){
        Main m = new Main();

        if (cartes.length == 0) return null;
        for(int i = 0 ; i < Main.taille ; i++ ){

          // option 1 : tableau
           double hazard = Math.random();
           int indiceHazard = (int)Math.floor(hazard * cartes.length);

           m.addCarte(cartes[indiceHazard]);
          // retirer du paquet : code récupéré de Main
           Carte[] nouvelleCartes = new Carte[cartes.length - 1];
           System.arraycopy(cartes, 0, nouvelleCartes, 0, indiceHazard);
           System.arraycopy(cartes, indiceHazard + 1, nouvelleCartes, indiceHazard, cartes.length - indiceHazard - 1);
           cartes = nouvelleCartes;

          // option 2 : tableau
          /*
         double hazard = Math.random();
         int indiceHazard = (int)Math.round(hazard * cartes.size());

         m.addCarte((Carte)(cartes.elementAt(indiceHazard)));
          // retirer du paquet : code récupéré de Main
         cartes.remove(indiceHazard);
          */

        }
        return m;
    }

   //   une méthode toString() qui permet d'exprimer le jeu
    public String toString(){
        String str = "";

       // option 1 et 2 : tableau et vecteur grace à la syntaxe de parcours
       // de collections
        str += "jeu[\n";
        for(Carte c : cartes){
            str += "\t" + c.toString() + "\n";
        }
        str += "]\n";
```

```
        return str;
    }
}
```

On voit immédiatement le grand intérêt de la connaissance des collection.

Le tirage au hazard nécessite un peu de recherche. La fonction Math.random() retourne un double entre 0.0 et 1.0 (exclus). Ce n'est pas convenable pour indicer le jeu et choisir une carte. Il faut ramener ce tirage à un indice entier. C'est ce qui est fait par la multiplication de `hazard` par le nombre d'éléments disponibles. Mais le résultat est également un double qu'il faut arrondir (méthode Math.floor). Le tirage est donc entre 0 et `cartes.length`. Le retour de floor() étant un double il faut le forcer à l'entier (cast) pour devenir un indice.

Une classe qui modélise une main d'un certain nombre de cartes

- la taille d'une main (statique)
- la collection des cartes de la main
- un constructeur qui consruit une main vide
- une méthode qui permet d'ajouter une carte à la main
- une méthode qui permet de jouer une carte au hazard en la remettant dans un jeu donné
- une méthode toString() qui permet d'exprimer la main

```
/*
 * Created on 1 juin 2007
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

public class Main {

   //   la taille d'une main (statique)
   public static final int taille = 7;

   //   la collection des cartes de la main
   // option 1 : tableau statique
   Carte[] main;

   // option 2 : collection
   /*
   Vector main;
   */

   //   un constructeur qui consruit une main vide
   public Main(){
      // option 1
      main = new Carte[0];

      // option 2
      /*
     main = new Vector();
      */
   }

   //   une méthode qui permet d'ajouter une carte à la main
   // si je suis malin, je recopie tel quelle la méthode de Paquet en changeant le nom
   // de la  collection
   public boolean addCarte(Carte c){
      boolean res = false;

      // option 1 : Syntaxe Java 5.0
      for(Carte uneCarte : main){
         if (c == uneCarte) return false;
      }

      Carte[] nouvelleMain = new Carte[main.length + 1];
      System.arraycopy(main, 0, nouvelleMain, 0, main.length );
      nouvelleMain[main.length] = c;
      main = nouvelleMain;

      return res;

      // option 2 : Syntaxe Java avec collection Vector
      /*
     if (!main.contains(c)){
        main.add(c);
     }
      */
   }


   //   une méthode qui permet de jouer une carte au hazard en
   //   la remettant dans un jeu donné
   public void jouerCarte(Paquet p){
      // selectionner au hazard une carte, c'est tirer son indice

      // option 1 : tableau
      double hazard = Math.random();
      int indiceHazard = (int)Math.floor(hazard * main.length);

      // préserver la carte à jouer
      Carte c = main[indiceHazard];
```

```
      // retirer de la main
      Carte[] nouvelleMain = new Carte[main.length - 1];
      System.arraycopy(main, 0, nouvelleMain, 0, indiceHazard);
      System.arraycopy(main, indiceHazard + 1, nouvelleMain,
                       indiceHazard, main.length - indiceHazard - 1);
      main = nouvelleMain;

      p.addCarte(c);

      // option 2 : collection
      /*
      double hazard = Math.random();
      int indiceHazard = (int)Math.round(hazard * main.size());

     Carte c = (Carte)main.elementAt(indiceHazard);
     main.remove(indiceHazard);
     p.addCarte(c);
      */
   }

  //   une méthode toString() qui permet d'exprimer la main
   public String toString(){
      String str = "";

      // option 1 et 2 : tableau et vecteur grace à la syntaxe de parcours
      // de collections
      str += "main[\n";
      for(Carte c : main){
         str += "\t" + c.toString() + "\n";
      }
      str += "]\n";

      return str;
   }
}
```

On réalisera également une fonction principale (main) qui :

- crée un certain nombre de cartes,
- crée un jeu et y enregistre les cartes
- tire une main de cinq cartes et l'imprime
- joue trois cartes.

```
   public static void main(String[] args){
      Paquet p = new Paquet();
      p.addCarte(Carte.AS_TREFLE);
      p.addCarte(Carte.SEPT_TREFLE);
      p.addCarte(Carte.HUIT_TREFLE);
      p.addCarte(Carte.NEUF_TREFLE);
      p.addCarte(Carte.DIX_TREFLE);
      p.addCarte(Carte.VALET_TREFLE);
      p.addCarte(Carte.DAME_TREFLE);
      p.addCarte(Carte.ROI_TREFLE);
      p.addCarte(Carte.AS_CARREAU);
      p.addCarte(Carte.SEPT_CARREAU);
      p.addCarte(Carte.HUIT_CARREAU);
      p.addCarte(Carte.NEUF_CARREAU);
      p.addCarte(Carte.DIX_CARREAU);
      p.addCarte(Carte.VALET_CARREAU);
      p.addCarte(Carte.DAME_CARREAU);
      p.addCarte(Carte.ROI_CARREAU);
      p.addCarte(Carte.AS_COEUR);
      p.addCarte(Carte.SEPT_COEUR);
      p.addCarte(Carte.HUIT_COEUR);
      p.addCarte(Carte.NEUF_COEUR);
      p.addCarte(Carte.DIX_COEUR);
      p.addCarte(Carte.VALET_COEUR);
      p.addCarte(Carte.DAME_COEUR);
      p.addCarte(Carte.ROI_COEUR);
      p.addCarte(Carte.AS_PIQUE);
      p.addCarte(Carte.SEPT_PIQUE);
      p.addCarte(Carte.HUIT_PIQUE);
      p.addCarte(Carte.NEUF_PIQUE);
      p.addCarte(Carte.DIX_PIQUE);
      p.addCarte(Carte.VALET_PIQUE);
      p.addCarte(Carte.DAME_PIQUE);
      p.addCarte(Carte.ROI_PIQUE);

      System.out.println(p);

      Main m = p.makeMain();

      System.out.println(m);

      m.jouerCarte(p);

      System.out.println(m);
```

```
      m.jouerCarte(p);

      System.out.println(m);

      m.jouerCarte(p);

      System.out.println(m);

     // on réimprime le paquet pour vérifier qu'on a bien
     // toutes les cartes.
      System.out.println(p);

   }
```

Dont la sortie produit la séquence suivante :

```
jeu[
        As de Trefle(1)
        Sept de Trefle(7)
        Huit de Trefle(8)
        Neuf de Trefle(9)
        Dix de Trefle(9)
        Valet de Trefle(11)
        Dame de Trefle(12)
        Roi de Trefle(13)
        As de Carreau(1)
        Sept de Carreau(7)
        Huit de Carreau(8)
        Neuf de Carreau(9)
        Dix de Carreau(9)
        Valet de Carreau(11)
        Dame de Carreau(12)
        Roi de Carreau(13)
        As de Coeur(1)
        Sept de Coeur(7)
        Huit de Coeur(8)
        Neuf de Coeur(9)
        Dix de Coeur(9)
        Valet de Coeur(11)
        Dame de Coeur(12)
        Roi de Coeur(13)
        As de Pique(1)
        Sept de Pique(7)
        Huit de Pique(8)
        Neuf de Pique(9)
        Dix de Pique(9)
        Valet de Pique(11)
        Dame de Pique(12)
        Roi de Pique(13)
]

main[
        Huit de Carreau(8)
        Sept de Trefle(7)
        Sept de Pique(7)
        Neuf de Pique(9)
        Neuf de Trefle(9)
        Valet de Coeur(11)
        As de Carreau(1)
]

main[
        Huit de Carreau(8)
        Sept de Trefle(7)
        Neuf de Pique(9)
        Neuf de Trefle(9)
        Valet de Coeur(11)
        As de Carreau(1)
]

main[
        Sept de Trefle(7)
        Neuf de Pique(9)
        Neuf de Trefle(9)
        Valet de Coeur(11)
        As de Carreau(1)
]

main[
        Sept de Trefle(7)
        Neuf de Pique(9)
        Neuf de Trefle(9)
        Valet de Coeur(11)
]

jeu[
        As de Trefle(1)
        Huit de Trefle(8)
```

```
        Dix de Trefle(9)
        Valet de Trefle(11)
        Dame de Trefle(12)
        Roi de Trefle(13)
        Sept de Carreau(7)
        Neuf de Carreau(9)
        Dix de Carreau(9)
        Valet de Carreau(11)
        Dame de Carreau(12)
        Roi de Carreau(13)
        As de Coeur(1)
        Sept de Coeur(7)
        Huit de Coeur(8)
        Neuf de Coeur(9)
        Dix de Coeur(9)
        Dame de Coeur(12)
        Roi de Coeur(13)
        As de Pique(1)
        Huit de Pique(8)
        Dix de Pique(9)
        Valet de Pique(11)
        Dame de Pique(12)
        Roi de Pique(13)
        Sept de Pique(7)
        Huit de Carreau(8)
        As de Carreau(1)
]
```

Annexes :

La fonction de tirage aléatoire est Math.random() fonction statique du paquetage java.lang

Pour les collections on peut utiliser un tableau (plus long à coder) ou la classe java.util.Vector avec les méthodes
v.add()
v.elementAt(int)
v.remove(int)
v.elements() (fournit une collection, explorable par next(), hasNext())