Examen de Java - Modèle 2

I. Questions de Cours (5 points).

1. Comment s'appelle le "modèle" syntaxique qui permet de décrire la structure d'un "objet logiciel" ?

Une classe

2. Soit le programme :

```
public class SomethingIsWrong {
    public static void main(String[] args) {
        Rectangle myRect;
        myRect.width = 40;
        myRect.height = 50;
        System.out.println("myRect's area is " + myRect.area());
    }
}
```

Quel est le problème dans ce code ?

constructeur de MyRect pas appelé (new manquant)

3. Considérez les deux classes suivantes :

```
public class ClassA {
    public void methodOne(int i) {
    }
    public void methodTwo(int i) {
    }
    public static void methodThree(int i) {
    }
    public static void methodFour(int i) {
    }
}

public class ClassB extends ClassA {
    public static void methodOne(int i) {
    }
    public void methodTwo(int i) {
    }
    public void methodThree(int i) {
    }
    public static void methodFour(int i) {
    }
}
```

Quelle méthode de la classe B "remplace" une méthode de la classe A ?

methodTwo

4. Quel est l'inconvénient d'écrire un gestionnaire d'exceptions trop général ? (3 lignes)

Ne pas clairement voir quel est l'origine de l'erreur, plus difficile à débugger, pas de stratégies correctives ciblées

5. Cette interface est-elle valide ?

```
public interface Marker {
}
```

A quoi peut-elle servir ?

Oui parfaitement, et elle peut servir à marquer certaines classes comme appartenant à la "famille" des classes marquées

II. Exercice (5 points).
Thème : Entrées/sorties

```
class ArbreFractal{
   int facteurDeBranche; // indique le nombre de sous-branches créées à chaque génération
   double facteurDeFeuilles;
   double dispersionAngleBranche;
   String famille;
   double nombreDeBranchesTotal;
   int nombreDeGenerations;
```

```
   public ArbreFractal(double db, double ff, double dab, String f, double ndb ){
      // COMPLETER //
   }

   public Vector genererArbre(){
      // NE PAS CHERCHER A IMPLEMENTER CETTE METHODE //
   }

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

1. Compléter

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.BufferedReader;
import java.io.Reader;
import java.io.Writer;
import java.util.Vector;

/*
 * Created on 2 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 ArbreFractal{
   int facteurDeBranche; // indique le nombre de sous-branches créées à chaque génération
   double facteurDeFeuilles;
   double dispersionAngleBranche;
   String famille;
   double nombreDeBranchesTotal;
   int nombreDeGenerations;

   public ArbreFractal(int fdb, double fdf, double dab, String f, int ng ){
      facteurDeBranche = fdb;
      facteurDeFeuilles = fdf;
      dispersionAngleBranche = dab;
      famille = f;
      nombreDeGenerations = ng;

      // arbre non généré
      nombreDeBranchesTotal = 0;
}

   public Vector genererArbre(){
      // NE PAS CHERCHER A IMPLEMENTER CETTE METHODE //
      return new Vector();
      // juste pour que ça compile
   }

   public void calculerTotalBranches(){
      // la génération augmente le nombre de branches d'un facteur multiplicatif facteurDeBranche à chaque génération :
      nombreDeBranchesTotal = facteurDeBranche;
      for(int i = 1 ; i < nombreDeGenerations ; i++){
         nombreDeBranchesTotal *= facteurDeBranche;
      }
   }

}
```

2. Organiser une sauvegarde fichier de l'objet par la méthode qu'il vous plaira, en fournissant un Stream de référence :
   a. save(xxxxOutputStream s);
   b. load(xxxxInputStream i);

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.BufferedReader;
import java.io.Reader;
import java.io.Writer;
import java.util.Vector;
```

```
/*
 * Created on 2 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 ArbreFractal{
   int facteurDeBranche; // indique le nombre de sous-branches créées à chaque génération
   double facteurDeFeuilles;
   double dispersionAngleBranche;
   String famille;
   double nombreDeBranchesTotal;
   int nombreDeGenerations;

   ...

   // la méthode la plus efficace est bien sûr le flux d'objet
   public void save(ObjectOutputStream s) throws IOException{
      s.writeObject(this);
   }

   // lors de la récupération il faut penser que la lecture construit un objet
   public void load(ObjectInputStream s) throws IOException, ClassNotFoundException{
      ArbreFractal f;
      f = (ArbreFractal) s.readObject();
      this.dispersionAngleBranche = f.dispersionAngleBranche;
      this.facteurDeBranche = f.facteurDeBranche;
      this.facteurDeFeuilles = f.facteurDeFeuilles;
      this.famille = f.famille;
      this.nombreDeBranchesTotal = f.nombreDeBranchesTotal;
      this.nombreDeGenerations = f.nombreDeGenerations;
   }

   // une méthode plus explicite utilise les DataStream
   public void save(DataOutputStream s) throws IOException{
      s.writeInt(facteurDeBranche);
      s.writeDouble(facteurDeFeuilles);
      s.writeDouble(dispersionAngleBranche);
      s.writeChars(famille);
      s.writeInt(nombreDeGenerations);
      s.writeDouble(nombreDeBranchesTotal);
}

   // lors de la récupération il faut penser que la lecture doit se faire dans le même ordre
   public void load(DataInputStream s) throws IOException{
      facteurDeBranche = s.readInt();
      facteurDeFeuilles = s.readDouble();
      dispersionAngleBranche = s.readDouble();
      famille = s.readUTF();
      nombreDeGenerations = s.readInt();
      nombreDeBranchesTotal = s.readDouble();
   }

   // la méthode la moins efficace est l'écriture texte, surtout pour la récupération
   // mais elle produit un fichier "lisible"
   public void save(Writer w) throws IOException{
      //option 1 : écriture basique
      w.write("fdb:" + facteurDeBranche + "\n");
      w.write("fdf:" + facteurDeFeuilles + "\n");
      w.write("dab:" + dispersionAngleBranche + "\n");
      w.write("fam:" + famille + "\n");
      w.write("ndg:" + nombreDeGenerations + "\n");
      w.write("nbt:" + nombreDeBranchesTotal + "\n");

      // option 2 : utilisation d'un Formatter
      /*
     Formatter f = new Formatter(w);
     f.format("fdb:%s%n",facteurDeBranche);
     f.format("fdf:%s%n", facteurDeFeuilles);
     f.format("dab:%s%n", dispersionAngleBranche);
     f.format("fam:%s%n", famille);
     f.format("ndg:%s%n", nombreDeGenerations);
     f.format("nbt:%s%n", nombreDeBranchesTotal);
      */
   }

   // la récupération suppose un décodage de chaque donnée après extraction de la ligne
   public void load(Reader r) throws IOException{
      BufferedReader br = new BufferedReader(r);
      this.dispersionAngleBranche = Double.parseDouble(br.readLine().substring(4));
      this.facteurDeBranche = Integer.parseInt(br.readLine().substring(4));
      this.facteurDeFeuilles = Double.parseDouble(br.readLine().substring(4));
      this.famille = br.readLine().substring(4);
      this.nombreDeBranchesTotal = Double.parseDouble(br.readLine().substring(4));
      this.nombreDeGenerations = Integer.parseInt(br.readLine().substring(4));
```

```
    }

}
```

III. Problème (7 points).
Thème : Scrabble

On désire développer un modèle d'un jeu de scrabble :

Une classe qui modélise les lettres :

- une table statique des fréquences des lettres (voir annexe fournie)
- une valeur en points
- une lettre (chaîne)
- des méthodes d'accesseur
- une méthode toString() qui permet d'exprimer la lettre

```
package scrabble;
import java.util.HashMap;

/*
 * Created on 2 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 Lettre {
   // une table statique des fréquences des lettres (voir annexe fournie)

   public static HashMap<String,Integer> frequences;

   static {
        frequences = new HashMap<String,Integer>();
        frequences.put("A", 13);
        frequences.put("B", 2);
        frequences.put("C", 2);
        frequences.put("D", 2);
        frequences.put("E", 18);
        frequences.put("F", 3);
        frequences.put("G", 2);
        frequences.put("H", 2);
        frequences.put("I", 3);
        frequences.put("J", 2);
        frequences.put("K", 1);
        frequences.put("L", 4);
        frequences.put("M", 6);
        frequences.put("N", 6);
        frequences.put("O", 5);
        frequences.put("P", 6);
        frequences.put("Q", 2);
        frequences.put("R", 6);
        frequences.put("S", 8);
        frequences.put("T", 6);
        frequences.put("U", 8);
        frequences.put("V", 4);
        frequences.put("W", 1);
        frequences.put("X", 1);
        frequences.put("Y", 1);
        frequences.put("Z", 1);
   }

   // une valeur en points
   int points;

   // une lettre (chaîne) ou Char
   String lettre;

   public Lettre(String lettre, int points){
      this.points = points;
      this.lettre = lettre;
   }

   // des méthodes accesseur
   public int getPoint(){
      return points;
   }

   public String getLettre(){
      return lettre;
```

```
   }

   public char getLettre(char foo){
      return lettre.charAt(0);
   }

   // une méthode toString() qui permet d'exprimer la lettre
   public String toString(){
      return lettre + "(" + points + ")";
   }
}
```

Pas de grande difficultés pour cette classe où il suffisait de suivre ls consigne et utiliser ce qui était donné. On peut remarquer la double implémentation de getLettre.

Une classe qui modélise le sac de lettres :

- une collection des lettres.
- un constructeur qui construit un sac vide
- une méthode qui permet de peupler le sac avec des lettres en fonction de la fréquence
- une méthode qui permet de tirer une main au hazard en retirant les lettres du sac et renvoie la main
- une méthode toString() qui permet d'exprimer le sac.

```
package scrabble;

/*
 * Created on 2 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 Sac {
   // taille du sac. Non précisé, mais une bonne méthode pour générer le sac
   public static final int TAILLE_SAC = 50;

   //   la collection des lettres du sac
   // option 1 : tableau statique
   Lettre[] sac;

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

   //   un constructeur qui construit un sac vide
   public Sac(){
      // option 1
      sac = new Lettre[0];

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

   // une méthode qui permet de peupler le sac avec des lettres en
   // fonction de la fréquence
   // la meilleure façon de faire est d'utiliser la table des fréquences comme une
   // table de lettres où on répeterait autant de fois que la fréquence la lettre.
   // On utilise une main pour celà, ce qui uppose avroi écrit la main.
   public void peupler(){

      // on crée une main pour avoir un tableau de lettres selon les fréquences
      Main m = new Main();

      for(char c = 'A' ; c < 'Z' ; c++ ){
         for(int i = 0 ; i < Lettre.frequences.get("" + c) ; i++){
            m.addLettre(new Lettre(Character.toString(c), 1));
         }
      }

      // on récupère un tableau de lettres
      Lettre[] mainFrequence = m.getLettres();
      System.out.println("taille pioche : " + mainFrequence.length);

      for(int i = 0 ; i < Sac.TAILLE_SAC ; i++){
        // on tire au hazard
         double hazard = Math.random();
         int indiceHazard = (int)Math.floor(hazard * mainFrequence.length);

        // on ajoute la lettre tirée au sac
         Lettre[] nouveauSac = new Lettre[sac.length + 1];
```

```
            System.arraycopy(sac, 0, nouveauSac, 0, sac.length );
            nouveauSac[sac.length] = mainFrequence[indiceHazard];
            sac = nouveauSac;
        }
    }

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

        if (sac.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 * sac.length);

            m.addLettre(sac[indiceHazard]);

            // retirer du sac : code récupéré de Main
            Lettre[] nouveauSac = new Lettre[sac.length - 1];
            System.arraycopy(sac, 0, nouveauSac, 0, indiceHazard);
            System.arraycopy(sac, indiceHazard + 1, nouveauSac, indiceHazard,
                                        sac.length - indiceHazard - 1);
            sac = nouveauSac;

            // option 2 : collections
            /*
            double hazard = Math.random();
            int indiceHazard = (int)Math.floor(hazard * sac.size());

            m.addLettre((Lettre)(sac.elementAt(indiceHazard)));
            sac.remove(indiceHazard);
            */

        }
        return m;
    }

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

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

        return str;
    }

}
```

La méthode peupler() est la plus difficile. En effet, le respect des fréquences de lettres dans le sac n'est pas un procédé trivial.

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

- la taille d'une main (statique, 7 par défaut)
- la collection des lettres de la main
- un constructeur qui consruit une main vide
- une méthode qui permet d'ajouter une lettre à la main
- une méthode qui permet de tirer un mot au hazard dans la main, étant un nombre de lettres donné
- une méthode toString() qui permet d'exprimer la main

```
/*
 * Created on 2 juin 2007
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */
package scrabble;

public class Main {

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

```
//   la collection des lettres de la main
// option 1 : tableau statique
Lettre[] main;

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

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

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

// accesseur sur le tableau (nécessaire pour la génération)
public Lettre[] getLettres(){
  return main;
}

//   une méthode qui permet d'ajouter une lettre à la main
public boolean addLettre(Lettre c){
    boolean res = false;

   // option 1 : tableaux

    Lettre[] nouvelleMain = new Lettre[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
   /*
   main.add(c);
   */
}


// une méthode qui permet de tirer un mot au hazard dans la main,
// étant un nombre de lettres donné
public String tirerMot(int lettres){
   // selectionner au hazard une carte, c'est tirer son indice

    Lettre[] tirage = new Lettre[main.length];
    System.arraycopy(main, 0, tirage, 0, main.length);
    String str = "";

   // option 1 : tableau
    for (int i = 0 ; i < lettres ; i++){
       double hazard = Math.random();
       int indiceHazard = (int)Math.floor(hazard * tirage.length);

      // ajoute la lettre au mot en cours
       str += tirage[indiceHazard].getLettre();

       Lettre[] nouvelleMain = new Lettre[tirage.length - 1];
       System.arraycopy(tirage, 0, nouvelleMain, 0, indiceHazard);
       if (indiceHazard < tirage.length - 1){
          System.arraycopy(tirage, indiceHazard + 1, nouvelleMain,
                           indiceHazard, tirage.length - indiceHazard - 1);
       }
       tirage = nouvelleMain;
    }

   // option 2 : collection
   /*
   tirage = main.clone();
   String str = "";
   for (int i = 0 ; i < lettres ; i++){
      double hazard = Math.random();
      int indiceHazard = (int)Math.round(hazard * main.size());

      str += (Lettre)main.elementAt(indiceHazard).getLettre();
      tirage.remove(indiceHazard);
   }
   */
    return str;
}

//   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(Lettre l : main){
         str += "\t" + l.toString() + "\n";
      }
      str += "]\n";

      return str;
   }

}
```

On réalisera également une fonction principale (main) qui :

- crée un certain nombre de lettres,
- peuple le sac et l'imprime
- tire une main de 7 lettres et l'imprime
- sort 3 mots au hazard (les mots sont simplement une suite de n lettres).

```
  public static void main(String[] args){
     Lettre l = new Lettre("A", 1);

     Sac s = new Sac();
     s.peupler();

     System.out.println(s);

     Main m = s.makeMain();

     System.out.println(m);

     System.out.println(m.tirerMot(3) + "\n");
     System.out.println("--\n");
     System.out.println(m.tirerMot(4) + "\n");
     System.out.println("--\n");
     System.out.println(m.tirerMot(5) + "\n");
  }
```

La sortie effective de ce code est :

```
taille pioche : 114
sac[
        M(1)
        M(1)
        A(1)
        V(1)
        D(1)
        A(1)
        A(1)
        I(1)
        Q(1)
        A(1)
        O(1)
        L(1)
        J(1)
        L(1)
        M(1)
        S(1)
        R(1)
        U(1)
        A(1)
        O(1)
        E(1)
        P(1)
        P(1)
        H(1)
        O(1)
        V(1)
        E(1)
        E(1)
        U(1)
        E(1)
        T(1)
        E(1)
        M(1)
        M(1)
```

```
        I(1)
        A(1)
        E(1)
        R(1)
        R(1)
        E(1)
        P(1)
        U(1)
        A(1)
        U(1)
        I(1)
        U(1)
        V(1)
        E(1)
        A(1)
        D(1)
]

main[
        A(1)
        I(1)
        E(1)
        D(1)
        O(1)
        P(1)
        U(1)
]

AEI

--

UDPI

--

EUPIO
```

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)

Pour la table de fréquence, on utilise une Map, initialisée par un bloc statique :

```
public static Map<String,Integer> frequences;

    static {
       frequences = new Map<String,Integer>();
       frequences.put("E", 12);
       frequences.put("A", 9);
       ...
    }
```

Pour lire la fréquence d'une lettre, il suffit d'écrire :

```
int freqE = Lettre.frequences.get("E").intValue();
```