Examen de Java - Modèle 3

Durée : 2 heures.
Documents : Machines autorisées sans connexion

I. Questions de Cours (5 points).

1. Par quel concept le comportement d'un objet (sa variation dans le temps) est-il proposé au monde extérieur ?

Par des méthodes

2. Le code suivant crée un objet Point et un objet Rectangle.

```
...
Point point = new Point(2,4);
Rectangle rectangle = new Rectangle(point, 20, 20);
point = null;
...
```

Combien reste-t-il de références actives à la fin de l'exécution de ce code ?

2

3. Une classe abstraite peut-elle hériter d'une classe concrète ?

Oui, si elle ajoute des méthodes non résolues (abstraites)

4. Exceptions et erreurs

Associer les libellés suivants (colonne gauche) aux situations (colonne droite) correspondantes :

int[] A; A[0] = 0;	a. Aucune exception
La Machine Virtuelle Java commence à exécuter votre programme, mais la JVM ne parvient pas à trouver les les classes de la plate-forme Java. (elles resident usuellement dans classes.zip ou rt.jar.)	b. Exception vérifiée
Un programme lit un flux et atteint le marqueur de fin de fichier.	c. Erreur de JVM
Un programme, avant de refermer un flux arrivé au marqueur de fin de fichier, tente de relire le flux.	d. Erreur de compilation

A. {{
int[] A;
A[0] = 0;
}}

[[d]]

B. {{
La Machine Virtuelle Java commence à exécuter votre programme, mais la JVM ne parvient pas à trouver les les classes de la plate-forme Java. (elles resident usuellement dans classes.zip ou rt.jar.)
}}

[[c]]

C. {{
Un programme lit un flux et atteint le marqueur de fin de fichier.
}}

[[a]]

D. {{

Un programme, avant de refermer un flux arrivé au marqueur de fin de fichier, tente de relire le flux.
}}

[[b]]

5. Quelle est la fonction de la variable d'environnement CLASSPATH ? Différence avec PATH ?

CLASSPATH est une liste des répertoires à partir desquels les classes nécessaires à l'exécution d'un programme peuvent être recherchées. PATH est la liste des répertoire où les commandes système (ou exécutables disponibles en tant que commandes) peuvent être trouvés.

II. Exercice (5 points).
Theme : Interfaces

```
class PageWeb{
   static long tempsDeRelaxation;
   String URL;
   String contenu;
   long dateDernierAcces;

   public PageWeb(String u, String c, String d){
      // COMPLETER //
   }

   public raffraichir(long dateCourante){
      // COMPLETER //
   }

   public static String obtenirContenu(URL){
      // NE PAS CHERCHER A IMPLEMENTER CETTE METHODE (1) //
   }

   public static long obtenirDateCourante(){
      // NE PAS CHERCHER A IMPLEMENTER CETTE METHODE (1) //
   }
}
```

1. Compléter la classe

```
class PageWeb{
   static long tempsDeRelaxation;
   String URL;
   String contenu;
   long dateDernierAcces;

   public PageWeb(String u, String c, long d){
      URL = u;
      contenu = c;
      dateDernierAcces = d;
   }
   public void raffraichir(long dateCourante){
      if (dateCourante > dateDernierAcces){
         contenu = PageWeb.obtenirContenu(URL);
      }
   }
   public static String obtenirContenu(String url){
      // NE PAS CHERCHER A IMPLEMENTER CETTE METHODE (1) //
      return new String(); // pour pouvoir compiler uniquement
   }

   public static long obtenirDateCourante(){
      // NE PAS CHERCHER A IMPLEMENTER CETTE METHODE (1) //
      return 01; // pour pouvoir compiler uniquement;
   }

}
```

2. Définir une interface qui permet de rendre cette classe "Cachable". On ne peut mettre qu'une seule page dans le cache.
   a. On ajoute à la classe un membre "static PageWeb cache;" qui constitue l'unique cache.
   b. On doit pouvoir mettre une page en cache.
   c. Un objet doit pouvoir obtenir son contenu du cache si c'est la même URL et si le cache n'est pas trop vieux. On modifie la méthode de raffraichissement.

```
/*
 * 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 interface Cachable {
   long getCacheDate() throws NothingCachedException;
   void cache(PageWeb page);
   PageWeb getCached() throws NothingCachedException;
}
```

Ceci est une proposition d'interface qui répond au problème précis posé par le cahier des charges.

3. Proposer un exemple de scénario d'utilisation.

```
/*
 * 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 PageWeb implements Cachable {
   static PageWeb cache;
   static long tempsDeRelaxation;
   String URL;
   String contenu;
   long dateDernierAcces;

   public PageWeb(String u, String c, long d){
      URL = u;
      contenu = c;
      dateDernierAcces = d;
   }

   public PageWeb(PageWeb p){
      URL = p.URL;
      contenu = p.contenu;
      dateDernierAcces = p.dateDernierAcces;
   }

   public void raffraichir(long dateCourante){
      if (dateCourante > dateDernierAcces){
         try {
            getCacheDate();
         } catch (NothingCachedException e) {
            contenu = PageWeb.obtenirContenu(URL);
            dateDernierAcces = dateCourante;
            cache(this);
         }
      }
   }

   public static String obtenirContenu(String url){
      // NE PAS CHERCHER A IMPLEMENTER CETTE METHODE (1) //
      return new String(); // pour pouvoir compiler uniquement
   }

   public static long obtenirDateCourante(){
      // NE PAS CHERCHER A IMPLEMENTER CETTE METHODE (1) //
      return 0l; // pour pouvoir compiler uniquement;
   }

   public long getCacheDate() throws  NothingCachedException {
      if (cache == null) throw (new NothingCachedException());
      if (cache.dateDernierAcces > obtenirDateCourante() - tempsDeRelaxation){
         cache = null;
      }
      return cache.dateDernierAcces;
   }

   public void cache(PageWeb page) {
      cache = new PageWeb(page);
      cache.dateDernierAcces = obtenirDateCourante();
   }

   public PageWeb getCached() throws NothingCachedException {
```

```
      if (cache == null) throw (new NothingCachedException());
      if (cache.dateDernierAcces > obtenirDateCourante() - tempsDeRelaxation){
         cache = null;
         throw (new NothingCachedException());
      }
      return cache;
   }
}
```

(1) Note 1 : Vous admettez que ces méthodes vous renvoient des objets valides si vous les appelez.

III. Problème (7 points).
Thème : Jeu de pièces de construction

On désire développer un modèle d'un jeu de construction (type Lego) qui met en relations trois entités :

Une classe qui modélise les pieces :

- un coût de fabrication (double)
- un nom de piece en chaîne
- une couleur en chaîne
- des méthodes d'accesseur
- une méthode toString() qui permet d'exprimer la piece

```
/*
 * Created on 4 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 construction;

public class Piece {
   // un coût de fabrication (double)
   double cout;

   // un nom de piece en chaîne
   String nom;

   // une couleur en chaîne
   String couleur;

   // un constructeur (nécessaire)
   public Piece(String n, String c1, double c2){
      nom = n;
      couleur = c1;
      cout = c2;
   }

   // des méthodes d'accesseur
   public double getCout(){
      return cout;
   }

   public String getNom(){
      return nom;
   }

   public String getCouleur(){
      return couleur;
   }

   // une méthode toString() qui permet d'exprimer la piece
   public String toString(){
      String str = "";
      str += "piece(";
      str += nom + "," + couleur + "," + cout;
      str += ")\n";
      return str;
   }

}
```

Une classe qui modélise le catalogue des pièces fabricables :

- une collection des pièces disponibles
- un constructeur qui construit un catalogue vide
- une méthode qui permet d'enregistrer des pieces uniques dans le jeu
- une méthode qui permet de tirer une boîte au hazard en ne retirant pas les pieces du jeu et renvoie la boîte.
- une méthode toString() qui permet d'exprimer le catalogue

```
/*
 * Created on 4 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 construction;

public class Catalogue {
   // une collection des pièces disponibles
   // option 1 : réalisation avec un tableau
    Piece[] pieces;

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

   // un constructeur qui construit un catalogue vide
    public Catalogue(){
       // option 1
        pieces = new Piece[0];

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

   // une méthode qui permet d'enregistrer des pieces uniques dans le jeu
    public boolean addPiece(Piece p){
        boolean res = false;

       // option 1 : Syntaxe Java 5.0
        for(Piece unePiece : pieces){
            if (p == unePiece) return false;
        }

        Piece[] nouvellesPieces = new Piece[pieces.length + 1];
        System.arraycopy(pieces, 0, nouvellesPieces, 0, pieces.length);
        nouvellesPieces[pieces.length] = p;
        pieces = nouvellesPieces;
        res = true;

        return res;

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

   // une méthode qui permet de tirer une boîte au hazard en ne retirant pas les pieces du jeu et renvoie la boîte.
    public Boite makeBoite(){
        Boite b = new Boite();

        if (pieces.length == 0) return null;
        for(int i = 0 ; i < Boite.TAILLE_MAX ; i++ ){

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

           try{
              b.addPiece(pieces[indiceHazard]);
           }
           catch (TropCherException ex){
              // simple arrêt du remplissage
              break;
           }

          // option 2 : vecteur
          /*
```

```
        double hazard = Math.random();
        int indiceHazard = (int)Math.round(hazard * pieces.size());

        try{
           m.addPiece((Piece)(pieces.elementAt(indiceHazard)));
        }
        catch (TropCherException ex){
           break;
        }
         */

      }
      return b;
   }

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

      // option 1 et 2 : tableau et vecteur grace à la syntaxe de parcours
      // de collections (légère modif à faire si le Vector est générique)
      str += "jeu[\n";
      for(Piece p : pieces){
         str += "\t" + p.toString() + "\n";
      }
      str += "]\n";

      return str;
   }
}
```

Une classe qui modélise une boîte d'un certain nombre de pieces

- la taille d'une boîte (statique)
- le coût maximum d'une boîte (statique)
- la collection des pieces de la boîte
- un constructeur qui consruit une boîte vide
- une méthode qui permet d'ajouter une piece à la boîte jusqu'à un certain coût de fabrication.
- une méthode qui calcule le coût de fabrication de la boîte
- une méthode toString() qui permet d'exprimer la boîte

```
/*
 * Created on 4 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 construction;

public class Boite {
   // la taille d'une boîte (statique)
   public static final int TAILLE_MAX = 15;

   // le coût maximum d'une boîte (statique)
   public static final double COUT_MAX = 15;

   // la collection des pieces de la boîte
   // option 1 : tableau statique
   Piece[] boîte;

   // option 2 : Vecteur
   /*
   Vector boîte;
   */

   // un constructeur qui consruit une boîte vide
   public Boite(){
      // option 1
      boîte = new Piece[0];

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

   // une méthode qui permet d'ajouter une piece à la boîte jusqu'à un certain coût de fabrication.
   public boolean addPiece(Piece p) throws TropCherException {
      boolean res = false;

      // option 1 : Syntaxe Java 5.0
```

```
        if (coutBoite() + p.getCout() > Boite.COUT_MAX){
            throw (new TropCherException());
        }

        Piece[] nouvelleBoite = new Piece[boîte.length + 1];
        System.arraycopy(boîte, 0, nouvelleBoite, 0, boîte.length );
        nouvelleBoite[boîte.length] = p;
        boîte = nouvelleBoite;

        return res;

       // option 2 : Syntaxe Java avec collection Vector
       /*
        *
        if (coutBoite() + p.getCout() < Boite.COUT_MAX){
          boîte.add(p);
        }
        else{
           throw new (TropCherException());
        }
       */
    }

    // une méthode qui calcule le coût de fabrication de la boîte
    public double coutBoite(){
        double cout = 0.0;

       // syntaxe Java 5.0, boîte est un tableau
        for(Piece p : boîte){
            cout += p.getCout();
        }

       // syntaxe Java 5.0, boîte est un Vector
       /*
      for(Object o : boîte){
         cout += (Piece)(o.getCout());
      }
       */

        return cout;
    }

    // une méthode toString() qui permet d'exprimer la boîte
    public String toString(){
        String str = "";
        str += "boîte[\n";

       // option 1 : boîte est un tableau
        for(Piece p : boîte){
            str += p.toString() + "\n";
        }

       // option 2 : boîte est un Vector
       /*
      for(Object o : boîte){
         str += (Piece)(o.toString()) + "\n";
      }
       */

        str += "]\n";
        return str;
    }

}
```

On réalisera également une fonction principale (main) qui :

- crée un certain nombre de pieces,
- crée un catalogue et y enregistre les pieces
- obtient une boîte de 15 pièces maximum et l'imprime.
- imprime le coût réel de la boîte et la marge obtenue (coût plafond - coût réel).

```
  public static void main(String[] args){
      Catalogue cat = new Catalogue();
      cat.addPiece(new Piece("2", "rouge", 0.5));
      cat.addPiece(new Piece("2", "bleu", 0.5));
      cat.addPiece(new Piece("2", "blanc", 0.5));
      cat.addPiece(new Piece("2", "noir", 0.5));
      cat.addPiece(new Piece("3 en ligne", "rouge", 0.8));
      cat.addPiece(new Piece("3 en ligne", "bleu", 0.8));
      cat.addPiece(new Piece("3 en ligne", "blanc", 0.8));
```

```
      cat.addPiece(new Piece("3 en ligne", "noir", 0.8));
      cat.addPiece(new Piece("4 en ligne", "rouge", 1.0));
      cat.addPiece(new Piece("4 en ligne", "bleu", 1.0));
      cat.addPiece(new Piece("4 en ligne", "blanc", 1.0));
      cat.addPiece(new Piece("4 en ligne", "noir", 1.0));
      cat.addPiece(new Piece("4 en bloc", "rouge", 0.95));
      cat.addPiece(new Piece("4 en bloc", "bleu", 0.95));
      cat.addPiece(new Piece("4 en bloc", "blanc", 0.95));
      cat.addPiece(new Piece("4 en bloc", "noir", 0.95));
      cat.addPiece(new Piece("6 en bloc", "rouge", 0.95));
      cat.addPiece(new Piece("6 en bloc", "bleu", 1.1));
      cat.addPiece(new Piece("6 en bloc", "blanc", 1.1));
      cat.addPiece(new Piece("6 en bloc", "noir", 1.1));
      cat.addPiece(new Piece("8 en ligne", "rouge", 1.5));
      cat.addPiece(new Piece("8 en ligne", "bleu", 1.5));
      cat.addPiece(new Piece("8 en ligne", "blanc", 1.5));
      cat.addPiece(new Piece("8 en ligne", "noir", 1.5));
      cat.addPiece(new Piece("12 en bloc", "rouge", 2.5));
      cat.addPiece(new Piece("12 en bloc", "bleu", 2.5));
      cat.addPiece(new Piece("12 en bloc", "blanc", 2.5));
      cat.addPiece(new Piece("12 en bloc", "noir", 2.5));

      System.out.println(cat);

      Boite b = cat.makeBoite();

      System.out.println(b);

      System.out.println("cout : " + b.coutBoite());
      System.out.println("marge : " + (Boite.COUT_MAX - b.coutBoite()));

   }
```

Ce programme donne en sortie :

```
jeu[
        piece(2,rouge,0.5)
        piece(2,bleu,0.5)
        piece(2,blanc,0.5)
        piece(2,noir,0.5)
        piece(3 en ligne,rouge,0.8)
        piece(3 en ligne,bleu,0.8)
        piece(3 en ligne,blanc,0.8)
        piece(3 en ligne,noir,0.8)
        piece(4 en ligne,rouge,1.0)
        piece(4 en ligne,bleu,1.0)
        piece(4 en ligne,blanc,1.0)
        piece(4 en ligne,noir,1.0)
        piece(4 en bloc,rouge,0.95)
        piece(4 en bloc,bleu,0.95)
        piece(4 en bloc,blanc,0.95)
        piece(4 en bloc,noir,0.95)
        piece(6 en bloc,rouge,0.95)
        piece(6 en bloc,bleu,1.1)
        piece(6 en bloc,blanc,1.1)
        piece(6 en bloc,noir,1.1)
        piece(8 en ligne,rouge,1.5)
        piece(8 en ligne,bleu,1.5)
        piece(8 en ligne,blanc,1.5)
        piece(8 en ligne,noir,1.5)
        piece(12 en bloc,rouge,2.5)
        piece(12 en bloc,bleu,2.5)
        piece(12 en bloc,blanc,2.5)
        piece(12 en bloc,noir,2.5)
]

boîte[
        piece(8 en ligne,noir,1.5)
        piece(4 en bloc,blanc,0.95)
        piece(3 en ligne,blanc,0.8)
        piece(4 en bloc,blanc,0.95)
        piece(6 en bloc,noir,1.1)
        piece(3 en ligne,noir,0.8)
        piece(6 en bloc,noir,1.1)
        piece(4 en ligne,noir,1.0)
        piece(4 en ligne,bleu,1.0)
        piece(6 en bloc,blanc,1.1)
        piece(4 en ligne,bleu,1.0)
        piece(4 en ligne,bleu,1.0)
        piece(2,rouge,0.5)
```

```
        piece(6 en bloc,noir,1.1)
        piece(6 en bloc,rouge,0.95)
]

cout : 14.85

marge : 0.15000000000000036
```

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)