Examen de Java - Modèle 4

Durée : 2 heures
Documents : portables autorisés sans connexion

I. Questions de Cours (5 points).

1. Comment se nomme le procédé général qui, en programmation objet, consiste à enterrer des variables et masquer le fonctionnement interne d'un objet ?

L'encapsulation

2. Un programmeur peut-il explicitement détruire un object ? Si oui, comment ? Si non, pourquoi ?

Pas directement, mais en annulant toutes les références à l'objet, on rend l'objet éligible à la destruction

3. Qu'est-ce qui permet à une classe d'apparaître comme plusieurs identités d'objets différents ?

Implémenter plusieurs interfaces

4. Toutes les exceptions doivent-elles être capturées ou transmises ?

non, seules les exceptions vérifiées doivent l'être

5. Comment s'appelle la variable d'environnement système qui détermine toutes les racines absolues possibles pour les classes ?

CLASSPATH

II. Exercice (5 points).
Thème : Exceptions

```
class Alarme{
   public static int OFF = 0;
   public static int ARMEE = 1;
   public static int DECLENCHEE = 2;

   int etat;
   String code;

   public Alarme(String code){
      // COMPLETER AU MIEUX //
   }

   public void armer(){
      // COMPLETER AU MIEUX //
   }

   public int désarmer(String code){
      // COMPLETER AU MIEUX : si le code est pas le bon, on déclenche,
      //  sinon on désactive, renvoie l'état
   }
}
```

1. Compléter le code

```
/*
```

```
 * 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
 */

class Alarme{
   public static int OFF = 0;
   public static int ARMEE = 1;
   public static int DECLENCHEE = 2;

   int etat;
   String code;
   public Alarme(String code){
      etat = OFF;
      this.code = code;
   }

   public void armer(){
      etat = ARMEE;
   }

   public int désarmer(String code){
      if (code.equals(this.code)){
         etat = OFF;
      }
      else{
         etat = DECLENCHEE;
      }
     // si on veut rendre un résultat sur l'opération :
      return etat;
   }
}
```

2. Transformez la classe pour intégrer les exceptions :
   a. Au moment de la construction, si on oublie de mettre le code (chaîne vide ou null)
   b. Au moment du désarmement, si le code est pas bon (déclenchement par exception).
   c. Ecrire la séquence try...catch nécessaire pour celui qui utilise votre alarme.

```
 /*
 * 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
 */

class Alarme{
   public static int OFF = 0;
   public static int ARMEE = 1;
   public static int DECLENCHEE = 2;

   int etat;
   String code;

   public Alarme(String code) throws CodeInvalideException {
      if (code.equals("") || code == null) throw (new CodeInvalideException());
      etat = OFF;
      this.code = code;
   }

   public void armer(){
      etat = ARMEE;
   }

   public int désarmer(String code) throws CodeErronéException{
      if (code.equals(this.code)){
         etat = OFF;
```

```
      }
      else{
         etat = DECLENCHEE;
         throw (new CodeErronéException());
      }
     // si on veut rendre un résultat sur l'opération :
      return etat;
   }

 }
```

```
public class CodeInvalideException extends AlarmeException {
}
```

```
public class CodeErronéException extends AlarmeException {
}
```

```
public class AlarmeException extends AlarmeException {
}
```

Commentaire sur les classes d'exception : C'est toujours une bonne idée que de créer une superclasse "famille" pour toutes ses exceptions.

```
  public static void main(String[] args){
      Alarme a;
      try {
         a = new Alarme("AZERTY");
         a.armer();
         a.désarmer("*****");
      } catch (CodeInvalideException e) {
         System.out.println("Lis le manuel !!");
      } catch (CodeErronéException e) {
         System.out.println("J'appele les flics !!");
      }
   }
```

III. Problème (12 points).
Thème : Jeu du tierce

On désire développer un modèle du tiercé qui met en relation trois entités :

Une classe qui modélise les chevaux :

- un indice de "forme" (double)
- un nom de cheval (chaîne)
- un nom de jockey (chaîne)
- des méthodes d'accesseur
- une méthode toString() qui permet d'exprimer la forme

```
/*
 * 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 tierce;

public class Cheval {
   // un indice de "forme"  (double)
   double forme;

   // un nom de cheval (chaîne)
```

```
   String nom;

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

   public Cheval(double f, String n, String j){
      forme = f;
      nom = n;
      jockey = j;
   }

   // des méthodes d'accesseur
   public String getNom(){
      return nom;
   }

   public double getForme(){
      return forme;
   }

   public String getJockey(){
      return jockey;
   }

   // une méthode toString() qui permet d'exprimer la forme
   public String toString(){
      String str = "";
      str += "cheval[";
      str += nom + "," + jockey + "," + forme;
      str += "]";
      return str;
   }
}
```

Une classe qui modélise un paddock de chevaux disponibles :

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

```
/*
 * 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 tierce;

public class Paddock {
   // une collection de chevaux
   // option 1 : réalisation avec un tableau
   Cheval[] paddock;

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

   // un constructeur qui construit un paddock vide
   public Paddock(){
      // option 1
      paddock = new Cheval[0];

      /*
       * option 2
```

```
     *
    paddock = new Vector();
     */
  }

  // une méthode qui permet d'enregistrer des chevaux uniques dans le paddock
     public boolean addCheval(Cheval c){
        boolean res = false;

       // option 1 : Syntaxe Java 5.0
        for(Cheval unCheval : paddock){
           if (c == unCheval) return false;
        }

        Cheval[] nouveauPaddock = new Cheval[paddock.length + 1];
        System.arraycopy(paddock, 0, nouveauPaddock, 0, paddock.length);
        nouveauPaddock[paddock.length] = c;
        paddock = nouveauPaddock;
        res = true;

        return res;

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

  // une méthode qui permet de tirer une course au hazard en retirant les chevaux
  // du paddock et renvoie la course
     public Course makeCourse(){
        Course c = new Course();

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

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

           c.addCheval(paddock[indiceHazard]);
           // retirer du paquet : code récupéré de Main
           Cheval[] nouveauPaddock = new Cheval[paddock.length - 1];
           System.arraycopy(paddock, 0, nouveauPaddock, 0, indiceHazard);
           System.arraycopy(paddock, indiceHazard + 1, nouveauPaddock, indiceHazard,
                                                  paddock.length - indiceHazard - 1);
           paddock = nouveauPaddock;

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

         m.addCheval((Cheval)(paddock.elementAt(indiceHazard)));
         paddock.remove(indiceHazard);
          */


        }
        return c;
     }

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

       // option 1 et 2 : tableau et vecteur grace à la syntaxe de parcours
       // de collections. quelques modifs si Vector est générique (Object)
        str += "paddock[\n";
        for(Cheval c : paddock){
```

```
            str += "\t" + c.toString() + "\n";
         }
         str += "]\n";

         return str;
      }

}
```

Une classe qui modélise une course dans laquelle un certain nombre de chevaux vont participer

- la taille d'une course (statique)
- la collection des chevaux de la course
- un constructeur qui consruit une course vide
- une méthode qui permet d'ajouter un cheval à la course
- une méthode qui permet de tirer un tiercé au hazard et imprimer au passage la moyenne de "forme" du tiercé
- une méthode toString() qui permet d'exprimer la course

```
/*
 * 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 tierce;

public class Course {
   //   le nombre de chevaux d'une course (statique)
   public static final int taille = 10;

   //   la collection des chevaux de la course
   // option 1 : tableau statique
   Cheval[] course;

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

   //   un constructeur qui consruit une course vide
   public Course(){
      // option 1
      course = new Cheval[0];

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

   //   une méthode qui permet d'ajouter un cheval à la course
   public boolean addCheval(Cheval c){
      boolean res = false;

      // option 1 : Syntaxe Java 5.0
      for(Cheval unCheval : course){
         if (c == unCheval) return false;
      }

      Cheval[] nouvelleCourse = new Cheval[course.length + 1];
      System.arraycopy(course, 0, nouvelleCourse, 0, course.length );
      nouvelleCourse[course.length] = c;
      course = nouvelleCourse;

      return res;
```

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

// une méthode qui permet de tirer un tiercé au hazard dans la course
public Course tirerTierce(){
    Course c = new Course();

    // option 1 : tableau
    for (int i = 0 ; i < 3 ; i++){
      // selectionner au hazard un cheval, c'est tirer son indice
       double hazard = Math.random();
       int indiceHazard = (int)Math.floor(hazard * course.length);

      // ajoute la lettre au mot en cours
       c.addCheval(course[indiceHazard]);

       Cheval[] nouvelleCourse = new Cheval[course.length - 1];
       System.arraycopy(course, 0, nouvelleCourse, 0, indiceHazard);
       if (indiceHazard < course.length - 2){
          System.arraycopy(course, indiceHazard + 1, nouvelleCourse,
                                   indiceHazard, course.length - indiceHazard - 1);
       }
       course = nouvelleCourse;
    }

    // option 2 : collection
    /*
    tirage = course.clone();
    String str = "";
    for (int i = 0 ; i < 3 ; i++){
       double hazard = Math.random();
       int indiceHazard = (int)Math.round(hazard * course.size());

      c.addCheval((Cheval)(course.elementAt(indiceHazard)));
      course.remove(indiceHazard);
   }
    */
    return c;
}

// une méthode calculant la moyenne de forme de la course (non sépcifiée, mais utile)
public double formeMoyenne(){
    double sommeforme = 0;
    int i = 0;
    for(Cheval c : course){
       i++;
       sommeforme += c.getForme();
    }

    if (i == 0) return 0;
    return (sommeforme / i);
}

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

    // option 1 et 2 : tableau et vecteur grace à la syntaxe de parcours
    // de collections (une petit modif mineure à faire si Vector est générique)
    str += "main[\n";
    for(Cheval c : course){
       str += "\t" + c.toString() + "\n";
    }
    str += "]\n";
```

```
      return str;
   }

}
```

On réalisera également une fonction principale (main) qui :

- crée un certain nombre de chevaux,
- crée un paddock et y enregistre les chevaux
- tire une main de 10 chevaux et l'imprime
- sort trois tiercés en imprimant les estimations.

```
      public static void main(String[] args){
         Paddock p = new Paddock();
         p.addCheval(new Cheval("Arthur", "Matthieu Grand", 1.0));
         p.addCheval(new Cheval("Mustang", "René Brilloin", 0.9));
         p.addCheval(new Cheval("Carmino", "Pierre Julien", 0.8));
         p.addCheval(new Cheval("Fier Alezan", "Jean Flouet", 1.1));
         p.addCheval(new Cheval("Cartagène", "Luc Trouille", 1.05));
         p.addCheval(new Cheval("Salambo", "Jacques Trouille", 0.85));
         p.addCheval(new Cheval("Artaban", "Jules Trouille", 0.78));
         p.addCheval(new Cheval("Priscilla", "Kevin Bruche", 1.11));
         p.addCheval(new Cheval("Georgio", "Fabrice Jeanoult", 1.11));
         p.addCheval(new Cheval("Balaton", "Jean-Pierre Jeanoult", 0.95));
         p.addCheval(new Cheval("Simonet", "Jean Simonet", 1.14));
         p.addCheval(new Cheval("Falco", "Simon Pierre", 1.14));

         System.out.println(p);

         Course c = p.makeCourse();

         System.out.println(c);

         Course t = c.tirerTierce();
         System.out.println(t);
         System.out.println("forme : " + t.formeMoyenne());

         t = c.tirerTierce();
         System.out.println(t);
         System.out.println("forme : " + t.formeMoyenne());

         t = c.tirerTierce();
         System.out.println(t);
         System.out.println("forme : " + t.formeMoyenne());
   }
```

La sortie de ce programme est :

```
paddock[
cheval[Arthur,Matthieu Grand,1.0]
cheval[Mustang,René Brilloin,0.9]
cheval[Carmino,Pierre Julien,0.8]
cheval[Fier Alezan,Jean Flouet,1.1]
cheval[Cartagène,Luc Trouille,1.05]
cheval[Salambo,Jacques Trouille,0.85]
cheval[Artaban,Jules Trouille,0.78]
cheval[Priscilla,Kevin Bruche,1.11]
cheval[Georgio,Fabrice Jeanoult,1.11]
cheval[Balaton,Jean-Pierre Jeanoult,0.95]
cheval[Simonet,Jean Simonet,1.14]
cheval[Falco,Simon Pierre,1.14]
]

course[
cheval[Mustang,René Brilloin,0.9]
```

```
cheval[Artaban,Jules Trouille,0.78]
cheval[Balaton,Jean-Pierre Jeanoult,0.95]
cheval[Georgio,Fabrice Jeanoult,1.11]
cheval[Arthur,Matthieu Grand,1.0]
cheval[Carmino,Pierre Julien,0.8]
cheval[Falco,Simon Pierre,1.14]
cheval[Cartagène,Luc Trouille,1.05]
cheval[Priscilla,Kevin Bruche,1.11]
cheval[Salambo,Jacques Trouille,0.85]
]

course[
cheval[Falco,Simon Pierre,1.14]
cheval[Artaban,Jules Trouille,0.78]
cheval[Salambo,Jacques Trouille,0.85]
]

forme : 0.9233333333333333

course[
cheval[Arthur,Matthieu Grand,1.0]
cheval[Carmino,Pierre Julien,0.8]
cheval[Priscilla,Kevin Bruche,1.11]
]

forme : 0.9700000000000001

course[
cheval[Balaton,Jean-Pierre Jeanoult,0.95]
cheval[Mustang,René Brilloin,0.9]
cheval[Georgio,Fabrice Jeanoult,1.11]
]

forme : 0.9866666666666667
```

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)