IUT Lyon 1 – DUT 1A – DS de Java 24 mai 2016

Durée : 1h50

Instructions : Documents non autorisés, téléphones éteints et prise de parole non-autorisée interdite. **Rendre le sujet avec la copie.**

Pour les bénéficiaires du tiers-temps : vous êtes dispensés de l'écriture de la classe `Poulet` et des méthodes des questions **2a)**, **2c)** et **2g)**. Vous utiliserez toutefois cette classe et ces méthodes dans les autres questions si nécessaire.

Questions de programmation (**20 points**)

A propos de Volailles…

Un éleveur de volaille reçoit d'un fournisseur de jeunes canards et de jeunes poulets qu'il élève jusqu'à ce qu'ils aient la taille nécessaire à leur commercialisation.
Une `Volaille` est caractérisée par son `poids` et un `numéro d'identification` sous forme d'un entier attribué automatiquement, constitué à partir de la variable de classe `dernierNumIdent` initialisée à 1 : la première instance d'une volaille créée portera le numéro 1, la deuxième le numéro 2,… Ce numéro est reporté sur une bague que la volaille porte à sa petite patte. Les volailles arrivent à l'élevage à l'âge de trois semaines. Elles sont baguées et enregistrées dans le système informatique.
Il y a deux sortes de volailles : des `Canard`s et des `Poulet`s.
Le `prix au kilo` (un réel) du canard et celui du poulet sont **deux prix différents**, exprimés en euros par kilo. Le prix actuel pour les canards est de 1.2 euro par kilo, celui des poulets de 1 euro. En revanche, le prix est le même pour tous les bêtes de la même espèce. Ce prix est fixé chaque jour.
Le `poids d'abattage` (un réel) auquel on abat les bêtes est **différents** pour les canards et les poulets, mais c'est le même pour tous les poulets (respectivement, tous les canards). Le poids d'abattage actuel des canards est de 1.5 kilo, celui des poulets de 1.2 kilo.

1. **(5 pts)** Ecrire dans un package nommé `métier`, les classes `Volaille`, `Canard`, et `Poulet`.
   Il faut pouvoir modifier les `prix au kilo` du jour, les `poids d'abattage` des poulets et des canards et les `poids` des volailles.
   On veillera à ce que ces classes soient munies
   - de la méthode *public boolean assezGrosse()* qui détermine si une volaille est éligible à l'abattage;
   - de la méthode *public double prix()* qui détermine le prix actuel d'une volaille en fonction de son poids;
   - des getters et setters que vous jugerez nécessaires ainsi que de la méthode *public String toString()*, proprement redéfinie.

   On veillera également à réutiliser au maximum le code existant et à rendre non-instanciable une classe qui ne doit pas l'être.

2. On souhaite maintenant modéliser l'application élevage, dans laquelle on peut :
   a. représenter l'ensemble des volailles de l'élevage ;
   b. enregistrer une volaille nouvellement arrivée dans l'élevage ;
   c. rechercher une volaille à partir de son numéro d'identification ;
   d. évaluer le prix des volailles de l'élevage envoyées à l'abattoir ;

e. trier les volailles à envoyer à l'abattoir (qui ne feront donc plus partie de l'élevage) ;
f. afficher la liste des volailles;
g. afficher la liste des poulets;
h. afficher la liste des canards.

Travail à faire pour cela :

- Proposer, dans le package `métier`, une classe `Elevage` ayant un attribut d'instance privé de type `List<Volaille>` *: lesVolailles*. Cette classe a également un attribut d'instance privé de type `int` : *nombreDeVolailles* correspondant au nombre de volailles de l'élevage. **(1 pts)**
- Ajouter à cette classe :
  - **a)** un constructeur qui initialise les 2 champs; **(1 pts)**
  - **b)** la méthode *public void ajouter(Volaille v) throws VolailleExistante* qui se charge d'ajouter la volaille *v* dans la liste *listeVolaille* et lève l'exception *VolailleExistante* si la volaille existe déjà. Ecrire la classe d'exception *VolailleExistante*; **(2.5 pts)**
  - **c)** la méthode *public Volaille rechercherVolailleParId(int identité) throws VolailleInexistante* qui recherche une volaille à partir de son numéro d'identification et lève l'exception *VolailleInexistante* si elle n'a pas été trouvé dans le conteneur. La méthode retourne `null` dans ce cas. Il n'est pas demandé d'écrire la classe d'exception *VolailleInexistante*; **(1.5 pts)**
  - **d)** la méthode *public double évaluerElevage()* qui retourne le prix total des volailles de l'élevage prêtes à être envoyées à l'abattoir; **(1.5 pts)**
  - **e)** la méthode *public List<Volaille> envoyerALAbattoir()* qui retourne les volailles à envoyer à l'abattoir et qui supprime ces volailles de l'élevage. La méthode met également à jour le nombre de volailles de l'élevage; **(2 pts)**
  - **f)** la méthode *public String toString()* pour afficher en mode texte les caractéristiques des volailles de l'élevage ainsi que leur type (canard ou poulet); **(1 pts)**
  - **g)** la méthode *public void afficherLesPoulets()* qui affiche en mode texte les poulets de l'élevage. **Attention**, la syntaxe boucle *for* JDK5 (ou boucle *for* optimisée) est imposée; **(1 pts)**
  - **h)** la méthode *public void afficherLesCanards()* qui affiche en mode texte les canards de l'élevage. **Attention**, l'utilisation d'un itérateur est imposée. **(1 pts)**

3. **(2.5 pts)** Dans un package nommé `application` écrire un programme de test au sein d'une classe `AppVolaille` illustrant l'utilisation des méthodes proposées par vos autres classes. En particulier, votre programme proposera l'instanciation d'objets de type `Canard`, `Poulet` et `Elevage` ainsi que l'utilisation des méthodes développées dans la partie 2 en capturant et traitant les exceptions déclenchées par celles-ci. Vous fournirez également une trace d'exécution de votre programme de test.

Annexe : Extrait de la spécification de la classe **`ArrayList<E>`** du package **java.util**

Constructor Summary

ArrayList()	Constructs an empty list.
ArrayList(Collection c)	Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.
ArrayList(int initialCapacity)	Constructs an empty list with the specified initial capacity.

Method Summary

Modifier and Type	Method and Description
void	**add**(int index, Object element) Inserts the specified element at the specified position in this list.
boolean	**add**(Object o) Appends the specified element to the end of this list.
boolean	**addAll**(Collection c) Appends all of the elements in the specified Collection to the end of this list, in the order that they are returned by the specified Collection's Iterator.
boolean	**addAll**(int index, Collection c) Inserts all of the elements in the specified Collection into this list, starting at the specified position.
void	**clear**() Removes all of the elements from this list.
Object	**clone**() Returns a shallow copy of this ArrayList instance.
boolean	**contains**(Object elem) Returns true if this list contains the specified element.
void	**ensureCapacity**(int minCapacity) Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.
Object	**get**(int index) Returns the element at the specified position in this list.
int	**indexOf**(Object elem) Searches for the first occurence of the given argument, testing for equality using the equals method.
boolean	**isEmpty**() Tests if this list has no elements.
int	**lastIndexOf**(Object elem) Returns the index of the last occurrence of the specified object in this list.
Object	**remove**(int index) Removes the element at the specified position in this list.

boolean	**remove**(Object o) Removes the first occurrence of the specified element in this list.
Object	**set**(int index, Object element) Replaces the element at the specified position in this list with the specified element.
int	**size**() Returns the number of elements in this list.
Object[]	**toArray**() Returns an array containing all of the elements in this list in the correct order.
Object[]	**toArray**(Object[] a) Returns an array containing all of the elements in this list in the correct order.
void	**trimToSize**() Trims the capacity of this ArrayList instance to be the list's current size.