



TP2 JAVA

Concepts abordés

- Héritage, Généricité, Polymorphisme, Collections, Interface

Consignes générales de travail

- Commencez chaque TP dans un nouveau *Projet Java* (ici *TP1bis*)
- Créez toujours un (ou plusieurs) paquetages pour contenir ses classes (ne jamais utiliser le paquetage par défaut).
- Écrivez chaque nouvelle classe dans son propre fichier (portant le nom de la classe).
- Pour ce TP, déclarez systématiquement vos attributs en *private* et vos méthodes en *package friendly* ou *public*.
- Définissez systématiquement une méthode `toString():String` pour chaque classe que vous écrivez.
- Lisez chaque exercice en entier avant de commencer vos réponses.

Pour mener à bien ces exercices, il est impératif d'ouvrir

1) la Javadoc

<http://docs.oracle.com/javase/6/docs/api/>

2) le tutoriel de Sun sur le collections

<http://docs.oracle.com/javase/tutorial/collections/index.html>

Exercice 1: Héritage, Polymorphisme, Généricité

(maximum 1 heure)

I Classe Animal

Compléter et ranger dans un package zoo.

```
public class Animal
{
    private int poids;
    private String nom;

    // constructeurs par défaut, à 2 paramètres
    // les 2 accesseurs en lecture et écriture pour tous les attributs
    // les méthodes toString; hashCode, equals
}
```

II Classe Chien

Définir et ranger dans un package zoo.

- Définir une classe Chien sous-classe de Animal. La classe Chien a un attribut (variable d'instance) maître, défini comme un objet chaînes de caractères.
- Définir un constructeur permettant d'initialiser les 3 attributs d'un chien.
- Ecrire les 2 accesseurs en lecture et écriture
- Ecrire les méthodes toString; hashCode, equals

III Classe Utilisation

```
public static void main (String args[])
{
    // définir une instance de Chien
    // afficher le nom du chien, son poids et le nom de maître
    // définir une instance d'Animal
    // définir deux autres instances de Chien
    // définir un tableau contenant les 4 instances ci-dessus
    // Faire afficher les noms des 4 animaux dans une boucle
}
```

Question : expliquer l'affichage obtenu

Collections: Choisir une liste de type ArrayList puis une liste de type LinkedList comme container à la place d'un tableau.

IV Classe Chat

Définir et ranger dans un package zoo.

- Définir une classe Chat sous-classe de Animal. La classe Chat a un attribut (variable d'instance) race, défini comme un objet chaînes de caractères.
- Définir un constructeur permettant d'initialiser les 3 attributs d'un chat.
- Ecrire les 2 accesseurs en lecture et écriture
- Ecrire les méthodes toString; hashCode, equals

V Classe Utilisation

Reprendre et compléter la classe Utilisation définie précédemment en ajoutant un chat....

VI Classe ABSTRAITE Animal

- Rendre abstraite la classe Animal définie précédemment en ajoutant une méthode abstraite manger().
- Reprendre et compléter les classes Chien et Chat et Utilisation de manière à ce que les messages '*Le chien mange du Pal*' et '*Le chat mange du Ronron*' s'affichent correctement.

VII Classe container Zoo

Définir une classe Zoo:

- ayant un attribut de type List pour une liste d'animaux (juste une référence)
- un constructeur permettant d'allouer la mémoire pour cette référence
- une méthode d'ajout d'un animal dans le zoo
- Ecrire les méthodes toString; hashCode, equals

Puis tester tout cela dans la classe Utilisation.

Temps cumulé maxi: 1 heure

VIII Générer un .jar exécutable et contenant les fichiers sources.

Créer un fichier archive contenant tous les fichiers sources .java du projet : ceci vous sera notamment utile quand un rendu vous sera demandé...

Produire un fichier .jar contenant tous les fichiers sources (.java) du projet :

- . *menu contextuel du projet > Export > JAR File*
- . *cocher «Export generated class files and resources»*
- . *cocher «Export Java sources files and resources»*
- . *renseigner «Select the export destination JAR file» (choisir par exemple le bureau (Desktop) comme répertoire cible, puis saisir un nom de fichier .jar, par exemple TP2_zoo.jar),*
- . *puis > next > next et, dans cette dernière page (JAR Manifest Specification), sélectionner la classe (en l'occurrence Essais) qui contient le programme principal,*
- . *valider par finish.*

Le fichier .jar est maintenant créé (sur le bureau si c'est le choix qui a été fait).

Pour vérifier que ce fichier contient bien les fichiers sources du projet : ouvrir ce fichier comme s'il s'agissait d'un fichier .zip ou .rar, et vérifier qu'il contient bien tous les fichiers .java du projet.

Exercice 2: Ranger les arguments d'un programme Java dans une classe Collection (List, LinkedList et ArrayList**)**
(maximum 20 minutes)

1. Expliquer ce que fait le code suivant :

```
import java.util.List;
import java.util.ArrayList;
import java.util.LinkedList;
...
public static void main(String[] args) {
    List list;
    if (args.length == 0)
        list = new ArrayList<String>();
    else
        list = new LinkedList<String>();

    for(String s:args)
        list.add(s);
}
...
```

Quelle différence y a t'il entre **LinkedList** et **ArrayList** ?

2. A quoi sert l'interface List ?
3. Expliquer ce que fait le code suivant ::

```
public static void main(String[] args) {
    List list = new ArrayList<String>();
    Collections.addAll(list, args);

    for(int i=0; i<list.size(); i++) {
        String s = (String)list.get(i);
        System.out.printf("%s:%d\n", s, s.length());
    }
}
```

Temps cumulé maxi: 1 heure et 20 minutes

Exercice 3: PolyLine

(maximum 30 minutes)

Pour cette exercice, vous utiliserez la classe Point définie en annexe.

Le but de cet exercice est de définir une classe `PolyLine` qui représente une ligne brisée définie par une succession de points.

On souhaite que des `PolyLine` différentes puissent être définies par un nombre de points différents mais que le nombre maximum de points soit défini lors de la création de chaque `PolyLine`.

Pour définir les points de la `PolyLine` on définira une méthode `add` qui permet d'ajouter un point à la `PolyLine`.

Pour tout l'exercice vous écrirez un main de test dans la classe `PolyLineTest`.

Indication les points seront stockés dans un tableau de points

```
private final Point[] points;
```

1. Est-il intéressant de stocker le nombre maximum de points dans un champ statique ?
2. Ecrire le constructeur ainsi que la méthode `add()`. Sachant que l'on ne va pas écrire de code spécifique si l'on dépasse le nombre maximum de points autorisés, que va t'il se passer ?
3. Ecrire une méthode `pointCapacity` renvoyant le nombre de points maximum de la `PolyLine`.
4. Comment gérer le débordement du tableau de points si on ajoute un point à une `PolyLine` possédant déjà son nombre maximum de points. Implémenter votre solution.
5. Ecrire une méthode `pointCount` qui renvoie le nombre de points constituant actuellement la `PolyLine`.
6. Ecrire une méthode `contains` qui renvoie vrai si le point passé en argument est un des points constitutifs de la `PolyLine`.
7. Que se passe t'il si l'on fait un `contains` avec `null` en argument ?
Et un `add` avec `null` en argument ?

Temps cumulé maxi: 1 heure et 50 minutes

Exercice 4 - FreePolyLine

(maximum 20 minutes)

On souhaite simplifier la vie de l'utilisateur d'une `PolyLyne` en lui évitant d'indiquer le nombre maximum de points. Pour cela on va créer une classe `FreePolyLine`

Pour cela, on se propose de stocker les points dans une `java.util.LinkedList`.

1. Ré-écrire le constructeur et la méthode `add`.
2. Que faire de la méthode `pointCapacity` ?
3. Réécrire `pointCount` et `contains` en regardant la doc de `java.util.LinkedList`

Temps cumulé maxi: 2 heures et 10 minutes

Exercice 5 - Table de hachage, classe Collection `HashMap`

(maximum 30 minutes)

On souhaite afficher l'animal préférée de Bob, Alice ou June.

Alice a pour animal préféré : `edith` le singe.

Bob a pour animal préféré : `izard` le chamoix.

June a pour animal préféré : `gold` le poisson rouge.

- Ecrire un programme simple qui prend en paramètre le nom d'un enfant (parmi Bob, Alice et June) et affiche son animal préféré.

Exemple en ligne de commande:

```
java Animal bob
```

```
L'animal préféré de bob est izard le chamoix
```

- Ré-écrire le code utilisant une table de hachage `java.util.HashMap` et ses méthodes d'accès.

Temps cumulé maxi: 2 heures et 40 minutes

Exercice 6 - Table de hachage, classe Collection HashMap

(maximum 10 minutes)

Sachant que la classe `Pair` est définie comme ceci :

```
public class Pair<U, V> {
    private final U first;
    private final V second;

    public Pair(U first, V second) {
        this.first=first;
        this.second=second;
    }
    public U getFirst() {
        return first;
    }
    public V getSecond() {
        return second;
    }
    @Override
    public String toString() {
        return first+" "+second;
    }
}
```

Pourquoi le code ci-dessous n'affiche pas ce qu'il faut ?

```
HashMap<Pair<String,String>,String> map
    = new HashMap<Pair<String,String>,String>();

map.put(new Pair<String,String>("jean-paul","sartre"), "mort");
map.put(new Pair<String,String>("elvis","presley"), "vivant");
map.remove(new Pair<String,String>("elvis","presley"));

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

Que doit-on faire pour résoudre le problème ?

Temps cumulé maxi: 3 heures

ANNEXE

La classe Point

```

public class Point
{
    private final float x;
    private final float y;

    public static void main(String[] args)
    {
        Point p1 = new Point(0.0F, 0.0F);
        System.out.println(p1.x+" "+p1.y);
        Point p2 = new Point(0.0F, 0.0F);
        System.out.println(p1.equals(p2));
        Point p3 = new Point(4.5F, 3.6F);
        System.out.println(p3.toString());
    }

    public Point(final float x, final float y) { this.x = x ; this.y = y ; }

    public Point(final Point p) { this(p.x, p.y) ; }

    public float getX() { return x ; }

    public float getY() { return y ; }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + Float.floatToIntBits(x);
        result = prime * result + Float.floatToIntBits(y);
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Point other = (Point) obj;
        if (x != other.x)
            return false;
        if (y != other.y)
            return false;
        return true;
    }

    @Override
    public String toString() {
        return "Point [x=" + x + ", y=" + y + "]";
    }
}

```