L3 : Programmation par objets
Travaux pratiques

Nihal Pekergin et Sovanna Tan
sovanna.tan@univ-paris12.fr
nihal.pekergin@univ-paris12.fr

2008-2009

Avertissement

Les TP seront notés. Lors de la dernière séance, les étudiants présenteront tous leurs TP. Le code source devra être commenté et les commentaires devront être mis en forme avec `javadoc`.

Liens utiles

- Les outils de développement en ligne de commande `JDK` se trouvent sur http://java.sun.com/javase/downloads/
- La documentation de référence de l'API se trouve sur le site http://java.sun.com/reference/api/
- La documentation sur `javadoc` se trouve sur http://java.sun.com/j2se/javadoc/
- La page des TP est http://www.univ-paris12.fr/lacl/tan/Java/java.html

1 TP n°1 : les outils

Les TP auront lieu au CMC sous Linux. L'environnement de développement en Java de Sun fonctionne aussi sous Windows.

1.1 Compilation d'un programme avec les outils en ligne de commande

1. Modification du fichier `.bashrc` pour ajouter les commandes `javac` et `java` dans le `path` pour les versions de Fedora (distribution linux) antérieures à Fedora 8. Pour les anciennes Fedora, le `JDK` est installé dans le répertoire `/opt`.
2. Créer un dossier `Java` sur votre espace de travail (`mkdir Java`).
3. Créer deux sous dossier dans le dossier `Java` appelés `src` et `build`
4. Créer un dossier `premier` dans le dossier `src`
5. Créer le fichier `Premier.java` dans le dossier `premier` avec le contenu suivant

```
package premier;
public class Premier{
  public static void main(String[] args){
    System.out.println("Mon premier programme java");
  }
}
```

6. Lancer une fenêtre de commande.
7. Compilation : dans le dossier `~/Java/src/premier`, lancer la commande
   `javac -d ../../build *.java`.
   Cette commande compile tous les fichiers source du répertoire courant et place les fichiers compilés (avec une extension `.class`) dans le répertoire `~/Java/build/premier`. Lorsque le dossier `~/Java/build/premier` n'existe pas, le compilateur `javac` le créé.
8. Exécution : depuis le dossier `~/Java/src/premier`, lancer
   `java -classpath ../../build premier.Premier`

1.2 Implantation du premier TD

1. Créer un dossier `td1` dans le dossier `~/Java/src`. Les classes devront faire partie du paquetage `td1`.
2. Programmer Question 1 Feuille 1. Donner les affichages demandés.
3. Ajouter les lignes suivantes et donner les affichages :

```
q.translate(2,2);
System.out.println(q.x + q.y + p.x + p.y);
q= new Point (5,5);
System.out.println(q.x + q.y + p.x + q.y);
```

4. Programmer Question 1 Feuille 2 (s'il reste du temps).

2 TP n°2 : La classe Message, utilisation de la classe String de l'API

Lorsque les droits d'accès ne sont pas précisés mettre le droit `public`.

1. Dans un paquetage `tp2`, créer une classe `Message`, permettant de manipuler des messages dotée d'un champ privé `contenu` de type `String`. La classe proposera au moins :
   - un constructeur à un argument permettant d'initialiser un message à partir d'un argument de type String contenant le texte du message ;
   - une méthode `affiche` qui affiche le contenu du message.
2. Ajouter à la classe `Message` une méthode statique `main` permettant de tester la classe. Compiler et exécuter.
3. Quel sera le résultat de l'exécution des instructions suivantes ?

```
Message m1=new Message("bonjour") ;
Message m2=new Message("bonjour") ;
Message m3=m1 ;
System.out.println((m2==m1)) ;
System.out.println((m3==m1)) ;
```

4. Ajouter à la classe `Message` :
   - un accesseur `getContenu` de type `String` qui renvoie le message ;
   - un accesseur `setContenu` de type `void` qui prend un argument de type `String` permettant de modifier le message ;
   - un constructeur sans argument créant un message avec un texte vide.

   Modifier le programme afin de tester l'ensemble des nouvelles fonctionnalités de la classe.
5. Quel sera le résultat de l'exécution des instructions suivantes ?

```
Message m1=new Message("bonjour") ;
Message m2=m1;
m2.setContenu("bonsoir") ;
m2.affiche( ) ;
m1.affiche( ) ;
```

Pour chacune des questions qui suivent, il est fortement conseillé de consulter soigneusement la documentation concernant la classe `String`. L'original se trouve sur http://java.sun.com/javase/6/docs/api/java/lang/String.html

6. `if(){}else{}`
   Ajouter à la classe `Message` une méthode qui renvoie un booléen, avec la valeur `false` si le message fait strictement plus de 6 caractères et `true` dans le cas contraire.
7. `for(;;) , while(){}, do{}while()`
   Ajouter à la classe `Message` une méthode provoquant l'affichage du message avec un caractère de moins à chaque ligne, comme ceci :

   ```
   Hello
   Hell
   Hel
   He
   H
   ```

8. Ajouter une méthode affichant une pyramide avec le premier caractère du message :

   ```
   HHHHHHH
    HHHHH
     HHH
      H
   ```

   La hauteur de la pyramide pourra éventuellement dépendre d'un paramètre de la méthode.
9. Ajouter une méthode qui met en majuscule tous les caractères du message.
10. Ajouter une méthode qui permet de déterminer si le message contient une chaîne de caractères passée en paramètre.
11. Ecrire les commentaires au format `javadoc` dans le fichier `Message.java`.
12. Lancer la commande `javadoc -private -d ../../docs/tp2 Message.java` et regarder le contenu du dossier `~/Java/docs/tp2` obtenu.

3 TP n°3 : Héritage et délégation

Lorsque les droits d'accès ne sont pas précisés, mettre le droit `public`. Toutes les classes des TP n°3, 5, 6 et 7 seront écrites dans un paquetage `jbp12`.

3.1 La classe Groupe

1. Ecrire une classe `Groupe` dans le paquetage `jbp12` avec deux champs privés de type `String`, `nomGroupe` et `paysGroupe`.
2. Ecrire un constructeur complet qui prend en argument deux `String` et qui initialise les deux champs.
3. Ecrire un constructeur à un argument de type `String` qui initialise le nom du groupe et qui utilise le constructeur complet.

4. Ecrire un constructeur sans argument, le constructeur par défaut qui initialise les deux champs avec une chaîne vide en utilisant le constructeur complet.
5. Ecrire les accesseurs `String getNomGroupe()`, `void setNomGroupe(String n)`, `String getPaysGroupe()` et `void setPaysGroupe(String p)`.
6. Ecrire une méthode `void affiche()` qui affiche le contenu des champs sur la sortie standard.
7. Tester les constructeurs et les méthodes dans le programme principal.

3.2 La classe Musicien

1. Ecrire une classe `Musicien` avec des champs privés, deux champs de type `String`, `nom` et `prenom` ainsi qu'un champ `instruments` de type tableau de `String`.
2. Ecrire le constructeur complet `Musicien(String n, String p, String[] i)` et le constructeur par défaut qui initialise les chaînes de caractères avec la chaîne vide et le tableau à `null`.
3. Ecrire les accesseurs en lecture et écriture pour chacun des champs.
4. Ecrire des méthodes `void ajouterInstrument( String i)` et `void supprimerInstrument( String i)` qui modifient le tableau des instruments.
5. Ecrire une méthode `void affiche()` qui affiche le contenu des champs sur la sortie standard.
6. Tester les constructeurs et les méthodes dans le programme principal.

3.3 La classe GroupeMusiciens

1. Ecrire une classe `GroupeMusiciens` dérivée de la classe `Groupe` qui contient un champ privé de type `Vector<Musicien>`.
2. Ecrire un constructeur à deux arguments qui initialise le nom et le pays du groupe et qui créé un vecteur vide.
3. Ecrire un constructeur par défaut qui initialise les chaînes de caractères avec la chaîne vide et créé un vecteur vide.
4. Ecrire un accesseur `Vector<Musicien> getMusiciens()`.
5. Ecrire une méthode `void affiche()` qui affiche le contenu des champs sur la sortie standard. Pour afficher le contenu du vecteur, on utilisera le code suivant :

```
Iterator<Musicien> q=musiciens.iterator();
while(q.hasNext())
  q.next().affiche();
```

6. Ecrire une méthode `void trierMusiciens()` qui trie les musiciens par ordre alphabétique des noms en utilisant la méthode statique `public static <T extends Comparable<? super T>> void sort(List<T> list)` de la classe `Collections`. Pour cela modifier la classe `Musicien`, dire qu'elle implémente l'interface `Comparable<Musicien>` et ajouter la méthode suivante :

```
public int compareTo(Musicien m){
  return(this.nom.compareTo(m.getNom()));
}
```

7. Tester les constructeurs et les méthodes dans le programme principal.

3.4 La classe MorceauS

1. Ecrire une classe `MorceauS` comprenant les champs statiques suivants :

   ```
   public static final byte INCONNU=0;
   public static final byte REGGAE=1;
   public static final byte ROCK=2;
   public static final byte PUNK=3;
   public static final byte RAP=4;
   public static String[] genresChaine={"inconnu","reggae","rock","punk","rap"};
   ```

   des champs `titre` et `fichier` de type `String`, un champ `groupe` de type `GroupeMusiciens`, un champs `annee` de type entier et un champ `genre` de type `byte`.

2. Ecrire une méthode protégée `boolean anneeValide(String an)` qui vérifie que la chaîne `an` est constituée de chiffre en utilisant la méthode statique `static boolean isDigit(char ch)` de la classe `Character`.

3. Ecrire un constructeur complet `MorceauS(String t, String f, GroupeMusiciens b,String a, byte ge)` qui initialise les champs si l'année est valide et qui écrit une message d'erreur sur la sortie d'erreur si l'année n'est pas valide. Lorsque le genre `ge` n'est pas un genre défini dans les champs statiques, il doit être initialisé avec la valeur `INCONNU`.

4. Ecrire un constructeur `MorceauS(String s,String f)` qui initialise le champ `titre` et le champ `fichier` et qui initialise le champ `groupe` à `null` et les champs `annee` et `genre` à `0` en utilisant le constructeur complet.

5. Ecrire un constructeur par défaut similaire au précédent qui initialise les champs de type chaîne de caractères avec la chaîne vide.

6. Ecrire les accesseurs `String getTitre()` et `void setTitre(String t)`.

7. Ecrire une méthode `void affiche()` qui affiche tous les champs sauf le champ `groupe` sur la sortie standard.

8. Ecrire une méthode `void afficheGroupe()` qui affiche tous les champs sur la sortie standard.

9. Tester les constructeurs et les méthodes dans le programme principal.

3.5 Jouer les fichier MPEG-3

3.5.1 La bibliothèque JLayer1.0 et la classe MyPlayer

La bibliothèque `JLayer1.0` permet de jouer simplement des fichiers MPEG-3. La version utilisée en TP a été adaptée en suivant les préconisations de l'article http://www.informit.com/guides/content.aspx?g=java&seqNum=290&rl=1. Elle permet d'interrompre un morceau.

Pour l'utiliser, créer un répertoire `~/Java/libs` et télécharger le fichier `jl1.0.jar` dans ce dossier. La classe `MyPlayer` que l'on va utiliser pour jouer les morceaux utilise la bibliothèque `JLayer1.0`

1. Télécharger le fichier `MyPlayer.java` dans le répertoire des fichiers sources du paquetage `jbp12`.

2. Compiler la classe sous Linux avec la commande suivante :

   ```
   javac -d ../../build -classpath .:../../libs/jl1.0.jar MyPlayer.java
   ```

   Sous windows, la commande devient

   ```
   javac -d ..\..\ -classpath .;..\..\libs\jl1.0.jar MyPlayer.java
   ```

3.5.2 La classe Morceau

La classe `Morceau` permet de jouer un fichier MPEG-3. Elle utilise la classe `MyPlayer`.

1. Télécharger le fichier `Morceau` dans le répertoire des fichiers source du paquetage `jbp12`.
2. Créer un répertoire `~/Java/mp3` et mettre un fichier MPEG-3 dedans.
3. Adapter la méthode `main` de la classe `Morceau`
4. Compiler avec la commande

   ```
   javac -d ../../build -classpath .:../../libs/jl1.0.jar *.java
   ```

5. Exécuter avec la commande

   ```
   java  -classpath ../../build:../../libs/jl1.0.jar jbp12.Morceau
   ```

4 TP n°4 : Exceptions

Transformer la classe `MorceauS` du TP n°3 en remplaçant les messages d'erreur par des exceptions et tester la levée d'exception dans le programme principal en ajoutant un bloc `try {} catch() {}`.

5 TP n°5 : Interfaces

A partir des interfaces et classes suivantes :

```
// fichier Tracable.java
package tp4;
interface Tracable{
  static final int DIM_MAX =500;
  void afficher( java.awt.Graphics g);
}
// fichier  Coloriable.java
package tp4;
interface Coloriable{
  void colorier(java.awt.Color c,java.awt.Graphics g);
}
// fichier Figure.java
package tp4;
public abstract class Figure implements Tracable,Coloriable {
  public abstract void afficher(java.awt.Graphics g);
  public abstract void colorier(java.awt.Color c,java.awt.Graphics g);
}
// fichier Carre.java
package tp4;
import java.awt.*;
public class Carre extends Figure {
   private int l;
   private int x;
   private int y;
   private Color c=Color.black;
   public Carre(int cote,int abs, int ord, Color couleur){
     x=abs;
     y=ord;
     c=couleur;
     l=cote;
   }
```

```
   public void afficher(Graphics g){
     if(l<DIM_MAX){
       g.setColor(c);
       g.drawRect(x,y,l,l);
     }
   }
   public void colorier(Color couleur,Graphics g){
     c=couleur;
     if(l<DIM_MAX){
       g.setColor(c);
       g.fillRect(x,y,l,l);
     }
   }
}

// fichier Dessin.java
package tp4;
import java.awt.*;
import javax.swing.*;

class Dessin extends JFrame{
   private JPanel zoneDessin;
   Dessin(){
     zoneDessin=new JPanel();
     getContentPane().add(zoneDessin);
   }
   public void paint(Graphics  g){
     super.paint(g);
     Carre c1=new Carre(100,10,10,Color.red);
     Carre c2=new Carre(80,200,200,Color.red);
     Graphics dg=zoneDessin.getGraphics();
     c2.colorier(Color.blue,dg);
     c1.afficher(dg);
   }
   public static void main(String [] args){
     Dessin d=new Dessin();
     d. setSize(500,500);
     d.setVisible(true);
  }
}
```

1. définir d'autres classes dérivées de `Figure` en utilisant d'autres méhodes de la classe `Graphics`,
2. construire un dessin en stockant ses éléments dans un tableau de `Figure`,
3. afficher le dessin en utilisant une méthode qui dessine tous les éléments d'un tableau de `Figure`.

Ne pas hésiter à compléter ou modifier les classes proposées.

6 TP n°6 : Interface graphique pour le juke-box

La classe `JukeBox` contient une interface pour un `Vector<Morceau>`.

1. Mettre plusieurs fichiers MPEG-3 dans le réperoire `~/Java/mp3`. Adapter la variable `MP3_DIR` dans le fichier `JukeBox.java` et faire fonctionner cette interface avec les classes du TP n°3.

2. Ajouter la gestion du genre et de l'année.
3. Adapter le fichier `JukeBox.java` pour le traitement des exceptions et le faire fonctionner avec les classes du TP n°4.

7 TP n°7 : Juke-box amélioré

Transformer le juke-box de manière à ce que les morceaux soient regroupés par groupe en intégrant les classes `Groupe` et `GroupeMusiciens`. Ne pas hésiter à développer l'arborescence des classes.

8 TP n°8 : Soutenance des TP