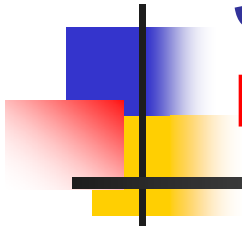
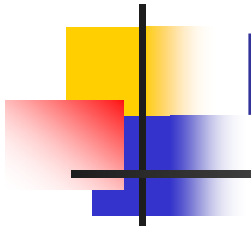


Java

Licence professionnelle CISII, 2009-2010

Cours 6 : le paquetage (package)

Cours inspiré des cours de Richard Grin, Antoine Tabbone et Hazel Everett



Le paquetage

- Définition
  - Les classes Java sont regroupées en paquetages (*packages* en anglais)
  - Ils correspondent aux « bibliothèques » des autres langages
  - Les paquetages offrent un niveau de modularité supplémentaire pour
    - réunir des classes suivant un centre d'intérêt commun
    - la protection des attributs et des méthodes

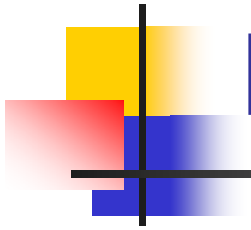
Attribution d'un nom

- Se fait au niveau du fichier source
- Exemple

```
//Fichier Vehicule..java
package ex1;
public class Vehicule {
...
}

//Fichier Voiture_Composee.java
package ex1;
public class Voiture_Composee {
  ...
}
```

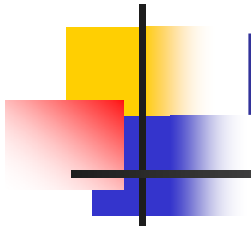
```
//Fichier Voiture_Derivee.java
package ex1;
public class Voiture_Derivee
    extends Vehicule {...

}

//Fichier Main.java
package ex1;

public class Main {
  public static void main(String[]
    args) {
...
  }
}
```

- De cette manière, tous les fichiers seront rassemblés comme si ils étaient un
- De plus, cela protège les classes de l'extérieur. En effet, si vous êtes sous Eclipse, par ex., vous pouvez construire un projet contenant plusieurs packages sans conflit de nom entre des classes de même nom



Le paquetage

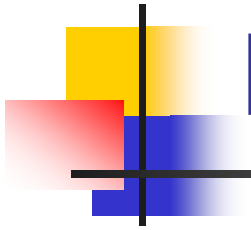
- Quelques paquetages du SDK
  - **java.lang** : classes de base de Java
  - **java.util** : utilitaires
  - **java.io** : entrées-sorties
  - **java.awt** : interface graphique
  - **javax.swing** : interface graphique avancée
  - **java.applet** : applets
  - **java.net** : réseau
  - **java.rmi** : distribution des objets



Le paquetage

- Intervention du paquetage dans le nommage d'une classe en Java
  - Le nom complet d'une classe (*qualified name* dans la spécification du langage Java) est le nom de la classe préfixé par le nom du paquetage :

    java.util.ArrayList
  - Une classe du même paquetage peut être désignée par son nom « terminal »
    - les classes du paquetage **java.util** peuvent désigner la classe ci-dessus par « **ArrayList** »
  - Une classe d'un autre paquetage doit être désignée par son nom complet, c.à.d avec le préfixe



Le paquetage

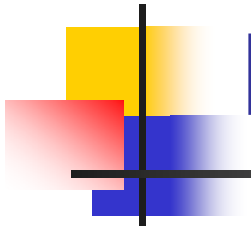
- Importer une classe d'un paquetage
  - Pour pouvoir désigner une classe d'un autre paquetage par son nom terminal, il faut l'importer

```
import java.util.ArrayList;
public class Classe {
...
ArrayList liste = new ArrayList();
...
```

  - On peut utiliser une classe sans l'importer
    - l'importation permet seulement de raccourcir le nom d'une classe dans le code :

➔ dans ce cas, il faut indiquer le nom total

```
java.util.ArrayList Liste = new java.util.ArrayList();
```

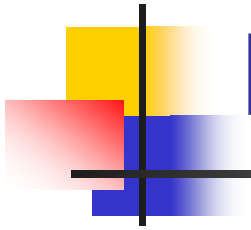


Le paquetage

- Importer toutes les classes d’un paquetage
  - On peut importer toutes les classes d'un paquetage :

    ```
    import java.util.*;
    ```

  - Les classes du paquetage **java.util** sont implicitement importées



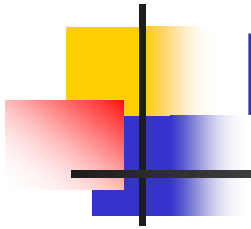
Le paquetage

- Lever une ambiguïté
  - On aura une erreur à la compilation si
    - 2 paquetages ont une classe qui a le même nom
  - Exemple
    - Ces deux paquetages ont la classe List en commun

```
import java.awt.*;
import java.util.*;
```

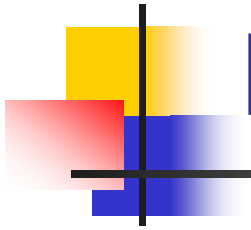
  - On peut le faire, mais pour lever l'ambiguïté, on devra à chaque fois donner le nom complet de la classe. Par exemple,

```
java.util.List l = getListe();
```



Le paquetage

- Importer des constantes **static**
  - Depuis le JDK 5.0 on peut importer des variables ou méthodes statiques d'une classe ou d'une interface
    - ➔ On allège ainsi le code, par exemple, pour l'utilisation des fonctions mathématiques de la classe **java.lang.Math**
  - A utiliser avec précaution pour ne pas nuire à la lisibilité du code (il peut être plus difficile de savoir d'où vient une constante ou méthode)



Le paquetage

- Exemple d'import static

```
Import static java.lang.Math.*; // importer tous les
  //membres statiques de la classe java.lang.Math
Import java.util.*;
Public class Machin {
...
X = max(sqrt(abs(y)), sin(y));  // au lieu de Math.sqrt,
  //Math.sin...
```

- On peut importer une seule variable ou méthode :

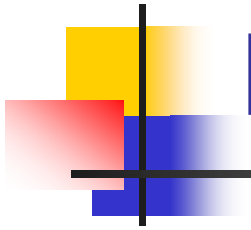
```
Import static java.lang.Math.PI;
X = 2* PI;
```

Le paquetage

- Ajout d'une classe dans un paquetage
  - Il suffit d'indiquer le nom du package au début du fichier
  - Exemple :

```
//Fichier Lait.java
package ex2;
public class Lait extends
   Liquide{
  public void
   imprimer(){
   System.out.println("je
   suis du lait");
  }
}
```

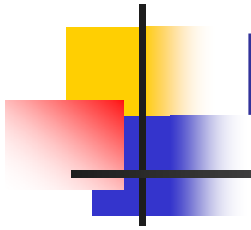
```
//Fichier Tasse.java
package ex2;
public class Tasse {
  private Liquide l;
  public void AjouterLiquide
    (Liquide l) {
    this.l = l;
  }
  public void imprimer() {
    l.imprimer();
  }
}
```



Le paquetage

- Sous-paquetage
  - Un paquetage peut avoir des sous-paquetages
  - Par exemple
    - **java.awt.event** est un sous-paquetage de **java.awt**
  - L'importation des classes d'un paquetage n'importe pas les classes des sous-paquetages ;
  - on devra écrire par exemple :

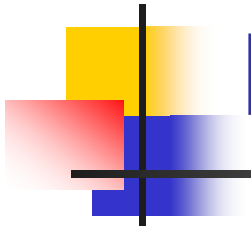
```
import java.awt.*;
import java.awt.event.*;
```



Le paquetage

- Nom d'un paquetage
  - Le nom d'un paquetage est hiérarchique :

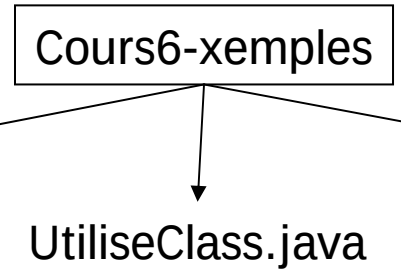
    java.awt.event
  - Il est conseillé de préfixer ses propres paquetages par son adresse Internet :
    - Fr.unice.toto.liste
    - Com.oreilly.projets.LivresJava



Le paquetage

- Placement d'un paquetage
  - N'importe où
    - Cependant, il faut respecter le chemin dans les import
  - Dans Eclipse
    - Créer un package
    - Puis créer les classes au fur et à mesure dans le package
    - Ensuite Eclipse s'arrange pour trouver le compilateur et le chemin du paquetage
  - Compilation
    - Il faut respecter la hiérarchie des import

Exemple complet



coursJava

monPaquet

DansPaquetage.java

```
package coursJava.monPaquet;
public class DansPaquetage
{
  public void faire()
  {
    System.out.println("bonjour
    depuis monPaquet");
  }
}
```

UtiliseClass.java

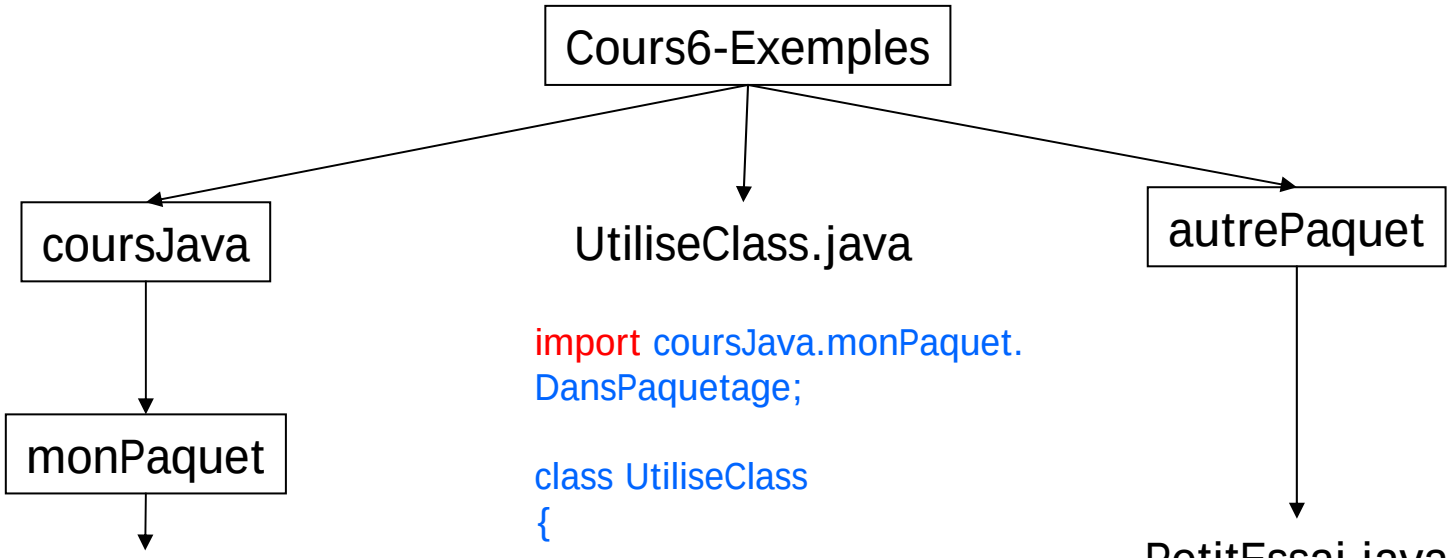
```
import coursJava.monPaquet.
DansPaquetage;

class UtiliseClass
{
  public static void main
   (String[] arg)
  {
    (new DansPaquetage())
    .faire();
  }
}
```

autrePaquet

PetitEssai.java

```
package autrePaquet;
import coursJava.monPaquet.*;
class PetitEssai
{
  public static void main(String[] arg)
  {
    (new DansPaquetage()).faire();
  }
}
```

- Exemple complet : compilation
  - Pour UtiliseClass.java
    - on se met dans Cours6-Exemples
    - Javac UtiliseClass.java ➔ UtiliseClass.class et DansPaquetage.class
  - Pour PetitEsai.java
    - on se met également dans Cours6-Exemples pour respecter la hiérarchie indiquée dans les import
    - Javac autrePaquet/PetitEssai.java



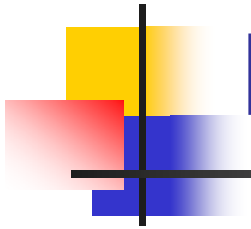
DansPaquetage.java

```
package coursJava.monPaquet;
public class DansPaquetage
{
  ...
}
```

```
import coursJava.monPaquet.
DansPaquetage;

class UtiliseClass
{
  ...
}
```

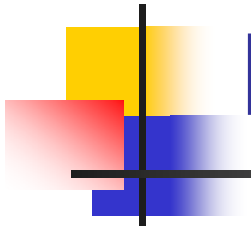
PetitEssai.java

```
package autrePaquet;
import coursJava.monPaquet.*;
class PetitEssai
{
  ...
}
```



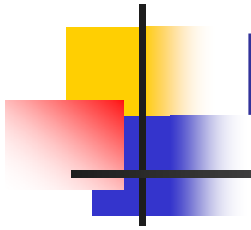
Le paquetage

- Encapsulation d'une classe dans un paquetage
  - Si la définition de la classe commence par **public class**
    - la classe est accessible de partout
  - Sinon, la classe n'est accessible que depuis les classes du même paquetage



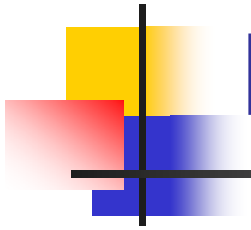
Exercice

- Cours6-TD6
  - Exercices 1-3



Le paquetage

- Archive Java : JAR (Java Archive)
  - Avec la croissance des réseaux, les applications sont vouées à voyager, il faut donc s'assurer :
    - qu'elles restent intactes
    - que tous les éléments (packages) de l'application soient présents
    - de rendre l'application la moins gourmande en espace
  - Ainsi, Java propose l'utilitaire **jar** dans le JDK, un utilitaire permettant de rassembler les différentes classes (fichiers .class) d'une application au sein d'une archive compressée, appelé package, afin d'en assurer l'intégrité et la taille



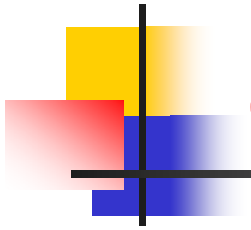
Le paquetage

- Archive Java : JAR (Java Archive)
  - Grâce à cet utilitaire, il est possible d'appeler à partir d'une page Web l'ensemble des classes d'une applet en faisant uniquement référence à l'archive (dont l'extension est .jar)

Fichiers Jar

Opérations de base

- Création d'un fichier Jar
  - jar cfv fichier.jar fichier_inclus$_1$ ... fichier_inlcus$_n$
    - *c* indique qu'il faut *créer* une archive Jar
    - *f* indique que le résultat sera redirigé dans un *fichier*
    - *v* (pour *verbose*) fait afficher les commentaires associés à l'exécution de la commande, en particulier, les noms des éléments ajoutés au fichier d'archive au fur et à mesure qu'ils y sont ajoutés
    - *fichier.jar* est le nom du fichier d'archive créé
    - les *fichier_inclus1 ... fichier_inlcus$_n$* sont une suite de noms de fichiers qui seront inclus dans l'archive ; ces noms peuvent utiliser des * ; s'ils font référence à des répertoires, leur contenu sera récursivement inclus dans l'archive

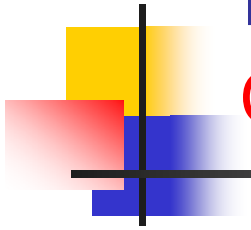


Fichiers Jar

Opérations de base

- **Création d'un fichier Jar**
  - Soit le répertoire cours6/monJar contenant
    - Point.class, Rectangle.class et surfRectangle.class
    - Manifest.txt contenant la ligne :

      Main-Class: surfRectangle
    - jar cfv monJar.jar *.class
      - génère un fichier d'archive, monJar.jar placé dans le répertoire courant
      - La commande génère également un fichier MANIFEST pour cette archive qui sera dans monJar.jar



Fichiers Jar

Opérations de base

- Visualisation du contenu d'un fichier Jar
  - La commande de base pour visualiser le contenu d'un fichier Jar est :
    - jar tf fichier.jar
  - Où :
    - *t* fait afficher la *table* du contenu de l'archive Jar
    - *f* indique que l'archive Jar à visualiser est contenu dans le *fichier* passé en paramètre
    - *fichier.jar* est le nom du fichier d'archive créé

Fichiers Jar

Opérations de base

- Extraction du contenu d'un fichier Jar
  - jar xfv fichier.jar [fichier_a_extraire1 ... fichier_a_extrairen]

- Où :
  - *x* indique qu'il faut *extraire* le contenu d'une archive Jar
  - *f* indique que l'archive Jar considérée est un *fichier* passé en paramètre
  - *v* (pour *verbose*) fait afficher les commentaires associés à l'exécution de cette commande
  - *fichier.jar* est le nom du fichier d'archive considéré
  - éventuellement les *fichier_a_extraire1 ... fichier_a_extrairen* qui sont une suite de noms de fichiers qui seront extrait de l'archive ; si ces noms ne sont pas spécifiés, l'ensemble du contenu de l'archive sera extrait.
- le contenu extrait est placé dans le répertoire courant
- Le fichier Jar demeure, quant à lui, inchangé

Fichiers Jar

Opérations de base

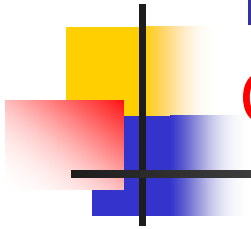
- Modification d'un fichier Jar
  - jar uf fichier.jar [nouveau_fichier1 ... nouveau_fichiern]

- Où :
  - *u* (*update*) indique qu'il faut *mettre à jour* une archive Jar
  - *f* indique que l'archive Jar considérée est un *fichier* passé en paramètre
  - *fichier.jar* est le nom du fichier d'archive considéré
  - éventuellement les *nouveau_fichier1 ... nouveau_fichiern* qui sont une suite de noms de fichiers à ajouter à l'archive.
- Cette commande permet d'ajouter des fichiers à une archive Jar ou de mettre à jour des fichiers déjà contenus dans l'archive

Fichiers Jar

Opérations de base

- Utilisation d'une librairie Jar
  - Créez un répertoire *Visage* et copiez-y le fichier *AppliVisage1.java*
  - Placez-vous dans le répertoire *Visage* et exécutez la commande de compilation suivante :
    - javac AppliVisage1.java
      - Que constatez-vous ? Expliquez.
  - Copiez la librairie *MyLib.jar* dans le répertoire *Visage* puis exécutez la commande de compilation suivante :
    - javac -classpath MyLib.jar AppliVisage1.java
      - Expliquez
  - Donnez la ligne de commande qui permet d'exécuter le programme Java *AppliVisage1*



Fichiers Jar

Opérations de base

- Exécution d'une application Java en ligne de commande à partird'un fichier Jar
  - Créez un répertoire *Hello* et copiez-y le fichier Jar *Hello1.jar*
    - Quel est le contenu du fichier *Hello1.jar* ?
  - Lancez l'application *HelloWorld* en exécutant la commande suivante :
    - java -classpath Hello1.jar HelloWorld
  - Copiez le fichier Jar *Hello2.jar* puis exécutez la commande suivante :
    - java -jar Hello2.jar
  - Que fait cette commande ?
  - Expliquez pourquoi c'est l'application *HelloWorld* qui a été exécutée par cette commande et pas un des autres programmes inclus dans le fichier Jar *Hello2.jar* (voir le contenu du fichier *META-INF/MANIFEST.MF* inclus dans *Hello2.jar*)

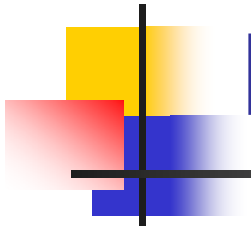
Fichiers Jar

Opérations de base

- Modification du MANIFEST

  jar umf ajout-manifest fichier.jar [nouveau_fichier1 ... nouveau_fichiern]

  - *u* (*update*) indique qu'il faut *modifier* une archive Jar
  - *m* (*manifest*) indique que des options vont être ajoutées au MANIFEST de l'archive Jar créée
  - *f* indique que l'archive Jar sera créée dans un *fichier*
  - *ajout-manifest* est le nom d'un fichier texte qui contient les options que l'on souhaite ajouter au MANIFEST (ex. *Main-Class: classname* ou *Class-Path: servlet.jar infobus.jar acme/beans.jar*)
  - *fichier.jar* est le nom du fichier d'archive considéré
- Faites les changements nécessaires pour que la commande java -jar Hello1.jar lance l'exécution de la classe *AppliVisage1* (au lieu de *HelloWorld* actuellement)
- Donnez la ligne de commande qui permet d'effectuer ce changement.



Le paquetage

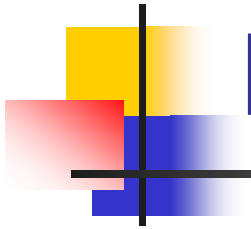
- Sous Eclipse
  - Pour ajouter un jar qui se trouve dans le workspace
    - bouton droit sur le projet -> propriétés -> java build path -> onglet librairies : Add jar
  - Pour un ajouter un jar hors du workspace
    - bouton droit sur le projet -> propriétés -> java build path -> onglet librairies : Add external jar

Fichiers Jar

Opérations de base

- Synthèse

Opération effectuée	Commande
Création d'un fichier Jar	jar cf fichier.jar fichier_inclus$_1$ … fichier_inlcus$_n$
Visualisation du contenu d'un fichier Jar	jar tf fichier.jar
Extraction du contenu d'un fichier Jar	jar xf fichier.jar
Extraction de certains fichiers d'un fichier Jar	jar xf fichier.jar fichier_extrait$_1$ … fichier_ extrait $_n$
Exécution d'une application contenue dans un Jar	java –classpath fichier.jar classe_principale
Exécution d'une application contenue dans un Jar (classe_principale doit être spécifiée dans le fichier MANIFEST)	java –classpath fichier.jar
Exécution d'une applet contenue dans un Jar	<applet code=*AppletClassName.class* archive="*JarFileName.jar*" width=*width* height=*height*> </applet>



Exercice

- Cours6-TD6
  - Exercice 4