LU2IN002 - Introduction à la programmation orientée-objet

Christophe Marsala

SCIENCES SORBONNE UNIVERSITÉ

Cours 8 – 18 novembre 2022

Programme du jour

1. Interfaces
2. Packages

Plan du cours

1. Interfaces
2. Packages

Certaines situations posent problèmes

- Ecrire des classes
  - avec la possibilité de sauvegarder leur résultat
  - avec la possibilité d'afficher le résultat sous différents format (écran, page web,...)

  $\Longrightarrow$ type de comportements utile pour des classes qui ne sont pas sur la même branche de la hiérarchie ?

$\Rightarrow$ MAIS limite de l'héritage : pas d'héritage multiple en JAVA

- une classe ne peut hériter que d'une seule classe

$\Rightarrow$ un autre outil existe : les interfaces

Interface : définition et usage

Usage

Une interface définit un comportement :

- un cahier des charges (e.g. Vehicule, Circuit...)
- une propriété (e.g. Serializable, Clonable...)

Elle donne la signature des méthodes à implémenter.

Ce que contient une interface

- signatures de méthodes (comme des méthodes abstraites)
- Mais (<java 1.8) :
  - pas de code
  - pas d'attribut

$\Longrightarrow$ Une interface ressemble à une classe abstraite pure : classe abstraite sans attribut ni méthode concrète

$\Longrightarrow$ MAIS ce n'est pas une classe !

Interface : exemples & syntaxe

1. Vu de l'extérieur de l'objet...
   exemple : qu'est-ce qui caractérise un véhicule ?

Interface : exemples & syntaxe

qu'est-ce qui caractérise un véhicule ? vu de l'extérieur de l'objet

- accélérer, freiner, tourner
- son état actuel (position, direction, vitesse, dérapage)
- observation des propriétés (capacités de braquage, vmax...)

```
1  public interface Véhicule { // caractéristiques d'un véhicule
2     // pour le pilotage
3     public void accelerer(double d);
4     public void freiner(double d);
5     public void tourner(double d);
6
7     // pour connaître son état actuel
8     public double getVitesse();
9     public Vecteur getPosition();
10    public Vecteur getDirection();
11 }
```

$\Longrightarrow$ Une classe Voiture respecte l'interface Vehicule si elle implémente toutes les méthodes déclarées dans l'interface

Interface : exemples & syntaxe (2)

Les interfaces pour énoncer des propriétés pour des objets

Par exemple : qu'est ce qu'un objet qui serait sauvegardable ?

Interface : exemples & syntaxe (2)

Les interfaces pour énoncer des propriétés pour des objets
Qu'est ce qu'un objet qui serait sauvegardable ?

Réponse :

- c'est un objet qui peut être sauvegardé sur disque
- c'est un objet capable de répondre à la méthode suivante public void save(String filename)

- Définissons une interface précisant ce comportement :

```
1  public interface Sauvegardable {
2     public void save(String filename);
3  }
```

$\Longrightarrow$ Une classe dont les instances seront sauvegardables doit implémenter la méthode déclarée dans cette interface

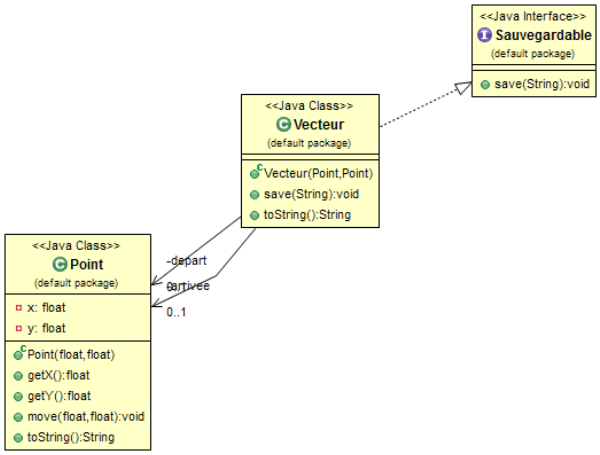
Interface : exemples & syntaxe (2)

Définir un objet sauvegardable : exemple avec la classe Vecteur.

- un objet sauvegardable implémente l'interface Sauvegardable.
- la classe Vecteur doit contenir l'implémentation de la méthode : public void save(String filename)
  $\Longrightarrow$ respect du contrat "sauvegardable"
- une instance de Vecteur pourra donc se sauvegarder : respect d'un cahier des charges

```
1  public class Vecteur implements Sauvegardable{
2     ...
3     public void save(String filename){
4        ... // instructions à réaliser
5     }
6
7  }
```


Interface & héritage : représentation UML



Interface & héritage : propriétés

- Une classe ne peut hériter que d'**une seule classe mère**
- mais une classe peut implémenter **plusieurs interfaces**

- Exemple pour un logiciel de dessin
- interfaces : Sauvegardable (méthode save), Deplacable (méthode move)...

```
1  public class Polygone extends Figure
2        implements Sauvegardable, Deplacable{
3  (...)
4    // Hérite des méthodes de Figure
5    // doit implémenter les méthodes save() et move()
6  }
```

Rem : si une classe implémente une interface, toute sa descendance (classe fille, etc.) hérite des méthodes correspondantes
$\Longrightarrow$ toutes les classes descendantes implémentent donc aussi l'interface par héritage

Interfaces et variables

- Il est possible de déclarer une variable d'un type interface
  - principe de subsomption
    - par exemple : tableau à partir d'une interface
  - mais jamais d'instanciation d'une interface
- Seules les méthodes de l'interface sont accessibles
- Exemple : soit les classes qui implémentent Sauvegardable :
  - classes Vecteur, Point, Figure,...
  - classes Personne, Menagerie,...

```
1 Sauvegardable[] tab = new Sauvegardable[42];
2 tab[0] = new Vecteur();
3 tab[3] = new Point();
4 tab[7] = new Menagerie(12);
5 ...
6 for (int i=0; i<tab.length;i++)
7     if (tab[i] != null)
8         tab[i].save("fichier␣"+i);
```

- Très pratique pour appliquer un traitement identique (ici save) à un ensemble de classes non liées par héritage

Interface et héritage

- Une interface peut hériter d'une autre interface
  - c'est un héritage pas une implémentation...

```
1 public interface Positionnable{
2   public Vecteur getPosition();
3 }
4
5 public interface Deplacable extends Positionnable{
6   public void move(Vecteur v);
7 }
```

- Une classe qui implémente Deplacable doit fournir
  - une définition pour la fonction move
  - une définition pour la fonction getPosition

Les interfaces : bilan

- Une interface permettent de définir un comportement
  - définition d'un cahier des charges à respecter
  - énoncé des propriétés requises pour un objet
  - elle déclare un ensemble de méthodes
  - une classe choisit de respecter ce comportement : implements
- Une interface n'est pas une classe
  - mais elle peut hériter d'une autre interface : extends
- $\Longrightarrow$ deux types de hiérarchies en Java
  - hiérarchie des classes : classe mère Object
  - hiérarchie des interfaces
- (hors programme LU2IN002) Nouvelles versions de Java (≥ 8)
  - attributs public, static, final
  - méthodes static
  - méthodes default

Plan du cours

Introduction

Bonne architecture = beaucoup de petites classes...
... chacune étant ciblée, lisible, ré-utilisable
$\Rightarrow$ Le répertoire de projet devient rapidement illisible !

Solution = arborescence de répertoires

- Sous-répertoires associés aux concepts de bas niveaux,
- Sous-sous-répertoires de test

Création de Packages de classes

Exemple

Gestion d'une course de voiture autonomes

1. Réfléchir à un découpage de bas niveau :
   - **Circuit**
   - **Voiture**
   - Autonome $\Rightarrow$ gestion de l'**IA** / **stratégies**
2. Ajouter les outils (transverses)
   - Gestion de la **géométrie**
   - Gestion des fichiers (sauvegardes/chargements)
   - Interface graphique (IHM)
3. Package de test :

Idée :

valider le fonctionnement de chaque objet indépendamment du reste du projet (dans la mesure du possible).

$\Rightarrow$ **sous-répertoire de test** dans chaque package principal

Créer un package

Package java

- Un Package est un est ensemble de classes mises dans un même répertoire.
- $\Longrightarrow$ les classes d'un même package forment "une famille" : nouveau type de visibilité

- Définition d'un package

```
1 package nomdupackage ; // ne début de fichier de classe
```

- Importer une classe d'un package

```
1 import nomdupackage.LaClasseVoulue ;
```

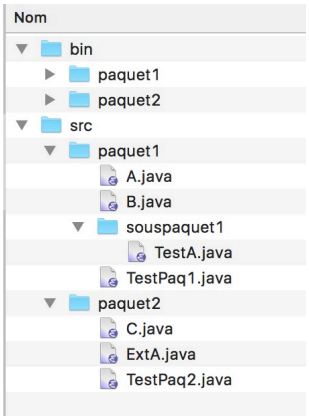
- Importer toutes les classes d'un package

```
1 import nomdupackage.* ;
```

$\Longrightarrow$ Convention : le nom d'un package est en minuscules

Déclarations obligatoires

Arborescence :



1. Déclaration de paquet

```
1 // Fichier A.java
2 package paquet1;
3 public class A {
4 ...
```

2. Déclaration d'import (pour les classes de paquets différents)

```
1 package paquet2;
2 import paquet1.A;
3 public class ExtA extends A{
4     public ExtA() {
5         super();
6     }
```

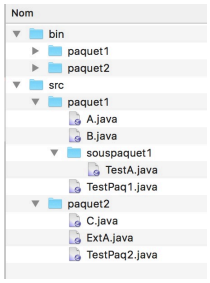
3. Sous-package

```
1 package paquet1.souspaquet1;
2 public class TestA {
3     public static void main(String[] args) {
4         // tests specifiques a A
```

4. Classe JDK

```
1 import java.util.ArrayList;
```


Compilation / exécution du code



- **Compilation** (position = racine)
  - Spécification d'un répertoire cible : -d
  - Spécification du répertoire de gestion des sources : -cp

```
> javac -cp src -d bin src/paquet1/TestPaq1.java
```

⇒Compile l'exécutable + toutes les dépendances

- **Exécution**
  - Instruction pour se positionner dans le répertoire d'exécution : -cp
  - Chemin avec des . (pas des /)

```
> java -cp bin paquet1.TestPaq1
```

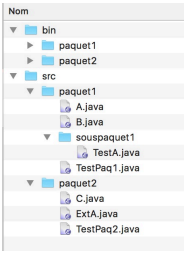
ou

```
> cd bin
> java paquet1.TestPaq1
```


Niveaux de visibilité

Introduction des packages = subtilités sur la visibilité

```
1  package paquet1;
2  public class A {
3    public int i;     // public
4    protected int j;  // protected
5    private int k;    // private
6    int n;            // package (nouveau)
7
8    public A(){
9        i=1; j=2; k=3; n=4;
10   }
11 }
```



Visibilités des attributs de A depuis :

		public i	protected j	private k	n
Même répertoire	B, TestPaq1	√	√	×	√
Classe fille	ExtA	√	√	×	×
Autres cas	C, TestPaq2	√	×	×	×
	TestA	√	×	×	×