

Les interfaces

P.O.O.

LMI 2 – Semestre 4 – Option Info

Année 2008-09

La notion de package

Les interfaces

La notion de package

Les interfaces

Pour le moment, nos applications Java sont constituées de

- ▶ un ensemble de classes
- ▶ chaque classe est dans un fichier
- ▶ le nom du fichier est *exactement identique* au nom de la classe, aux majuscules près.

Dans les projets plus grands :

- ▶ les classes peuvent être rangées dans différents répertoires et sous-répertoires
- ▶ les classes qui sont dans un même répertoire sont dans un même paquetage ou **package**
- ▶ le nom d'un **package** est composé des noms des répertoires formant le chemin d'accès à ses classes, séparés par un `.`
- ▶ les noms des répertoires commencent par une minuscule.

Déclaration de package

- ▶ La définition des classes commence par la désignation du package dans lequel la classe se trouve.
- ▶ Cette indication est *optionnelle*.
- ▶ Si cette instruction est présente, elle doit être la première instruction du fichier.

```
package etudiant;
```

```
class Etudiant{...  
}
```

Les très gros projets sont plus généralement organisés en une hiérarchie de répertoires à plusieurs niveaux.

```
package monrep.monsousrep;
```

Le chemin `monrep.monsousrep` doit indiquer la liste des sous-répertoires à traverser pour atteindre cette classe.

API : Application Programming Interface

C'est l'ensemble des classes Java déjà programmées auxquelles vous avez accès.

Elles sont organisées en packages et sous-packages. En voici quelques uns :

- ▶ `java.lang`
- ▶ `java.util`
- ▶ `java.awt`
- ▶ `java.awt.event`
- ▶ `javax.swing`
- ▶ ...

- ▶ Par défaut, accès à toutes les classes définies :
 - ▶ dans `java.lang`
 - ▶ dans votre répertoire courant
- ▶ Pour utiliser des classes provenant d'autres `packages`, on utilise l'instruction `import`.
- ▶ Par exemple

```
import java.util.Date ;
```
- ▶ Plus généralement

```
import rep1.rep2.UneClasse ;
import rep1.rep3.* ;
```
- ▶ Les instructions `import` doivent se trouver après l'instruction `package`, si elle est présente, et avant la définition de la classe.

Utiliser les classes d'un package

Pour pouvoir utiliser une classe `MaClasse` d'un package `monPackage`, il faut :

- ▶ que `MaClasse` soit définie `public` dans le package `monPackage` ;
- ▶ que le code qui veut utiliser `MaClasse` l'indique par une instruction `import monPackage.MaClasse ;`
ou bien
`import monPackage.* ;`.

- ▶ Dans un même package, les classes sont définies **public** ou **rien**.
- ▶ Si une classe est déclarée **public**, alors elle est accessible en dehors du package auquel elle appartient.
- ▶ Par défaut, (i.e. sans **public**), l'accès à la classe sera dit **package** et la classe ne sera accessible que par les classes du même package.

La variable `classpath` permet au compilateur `javac` et à l'interprète `java` de savoir où se trouvent les classes utilisées.

Par défaut, `classpath` contient `.` et le chemin d'accès aux API Java.

La notion de package

Les interfaces

Décomposer un problème en sous-problèmes :

- ▶ c'est définir le contrat à respecter par chaque partie
- ▶ c'est définir les protocoles de communication entre objets

contrat $\equiv$ interface

Une interface n'est pas une classe :

- ▶ elle **n'implémente aucune méthode**.
- ▶ elle **ne peut pas être instanciée** (pas de **new**) ;

Une interface ne peut contenir que

- ▶ la signature de méthodes (**public**)
- ▶ la déclaration de constantes (**public static final**)

Exemple

```
public interface Article {  
    public String getDescriptif();  
    public int getQuantite();  
    public float getPrix();  
    public Article ajouteQuantite(int nb);  
    public Article retireQuantite(int nb);  
}
```

Déclaration d'une interface

- ▶ Une interface est dans un fichier du même nom que l'interface ;
- ▶ Une interface peut être déclarée dans un package ;
- ▶ Une interface est `public` ou rien (i.e. *package*) ;
- ▶ Un fichier qui contient une définition d'interface peut contenir des instructions `import` ;

Implémentation d'une interface

Une classe, si elle implémente une interface, doit fournir une implémentation pour **toutes** les méthodes définies dans l'interface.

Une classe peut implémenter plusieurs interfaces.

Syntaxiquement :

```
class NomDeLaClasse implements NomDUneInterface,  
NomDUneAutreInterface{ ...}
```

Exemple

```
public class Crayon implements Article{
    private String nom;
    private String couleur;
    private int qte;
    private float prix;
    public Crayon(String nom, String couleur, float prix){
        this.nom = nom;
        this.couleur = couleur;
        this.prix = prix;
    }
    public Crayon(String nom, String couleur,
                  float prix, int qte){
        this(nom, couleur, prix);
        if (qte > 0)
            this.qte = qte;    }
```

Exemple

```
public int getQuantite(){  
    return qte;    }  
public float getPrix(){  
    return prix;    }  
public String getDescriptif( ){  
    return nom+" - "+couleur;    }  
public String getNom(){  
    return nom;    }  
public String getCouleur(){  
    return couleur;    }
```

Exemple

```
public Article ajouteQuantite(int nb){  
    if (nb > 0)  
        return new Crayon(nom,prix,qte+nb);  
    return null;  
}  
public Article retireQuantite(int nb){  
    if (nb < 0)  
        return null;  
    if (qte >= nb)  
        return new Crayon(nom,couleur,prix, qte - nb);  
    return new Crayon(nom,couleur,prix);    }  
} //Crayon
```

Exemple

```
public class Vetement implements Article{
    private String type;
    private String couleur;
    private int taille;
    private int qte;
    private float prix;
    public Vetement(String type, String couleur,
                    int taille, float prix){
        this.type = type;
        this.taille = taille;
        this.couleur = couleur;
        this.prix = prix;    }
    public Vetement(String type, String couleur, int taille,
                    float prix, int qte){
        this(type, couleur, taille, prix);
        (qte > 0)
        this.qte = qte;    }
}
```



Exemple

```
public int getQuantite(){  
    return qte;    }  
public float getPrix(){  
    return prix;    }  
public String getType( ){  
    return type;    }  
public String getCouleur(){  
    return couleur;    }  
public int getTaille(){  
    return taille;    }  
public String getDescriptif( ){  
    return nom+" - "+couleur+" - taille : "+taille;    }  
}
```

Exemple

```
public Article ajouteQuantite(int nb){  
    if (nb > 0)  
        qte += nb;  
    return this;  
}
```

```
public Article retireQuantite(int nb){  
    if (nb > 0){  
        if (qte >= nb)  
            qte = qte - nb;  
        else qte = 0;  
    }  
    return this;  
}
```

```
}//Vetement
```

- ▶ Les définitions d'interface créent des noms de types tout comme le font les définitions de classe.
- ▶ On peut utiliser le nom d'une interface comme le nom de type d'une variable ;
- ▶ Toute instance d'une classe qui implémente cette interface peut être affectée à cette variable.

```
Article a1 = new Vetement("robe","jaune",38);  
Article a2 = new Crayon("crayon d'aquarelle","rouge",3);  
Vetement v = new Vetement("pantalon","gris",44);  
a1.ajouteQuantite(2);  
v.ajouteQuantite(1);  
System.out.println("Taille : "+v.getTaille());  
System.out.println("Taille : "+a1.getTaille()); //Refus
```

Une classe peut implémenter plusieurs interfaces.

Que se passe-t-il si un même nom de méthode se trouve défini dans plusieurs interfaces ?

Plusieurs cas :

- ▶ même signature. *Pas de conflit.*
- ▶ les signatures diffèrent par les paramètres. *Pas de conflit : surcharge de méthodes*
- ▶ les signatures ne diffèrent que par leur type de retour. *Conflit !
Il n'est pas possible qu'une classe implémente ces deux interfaces en même temps.*