iUT ORSAY

Java 2007 – Héritage

Jean-Pierre Fournier

http://www.iut-orsay.fr/~fournier

Deux formes d'héritage

- une classe peut étendre une classe existante
  - parce que les objets que l'on veut créer sont déjà partiellement définis dans cette autre classe (relation "sorte-de")
  - parce que cette dernière décrit des attributs et des méthodes utiles (éviter de refaire ce qui est déjà fait)

```
class A{
   private int x;
   private Object ref;
   …
   public int methode(…){…}
   }
class B extends A{
…
}
```

les instances de la classe B comporteront les attributs x et ref, et posséderont la méthode…

Utiliser le travail déjà fait

- vous voulez réaliser une GUI (Graphical User Interface), donc des fenêtres avec une bordure, une couleur de fond, une certaine police de caractères, etc…
- les classes `java.awt.Frame` et `javax.swing.JFrame` proposent de telles fenêtres et ont été validées par de nombreux utilisateurs,

```
class MaFenetre extends Frame {…}
```

Mise en commun d'attributs

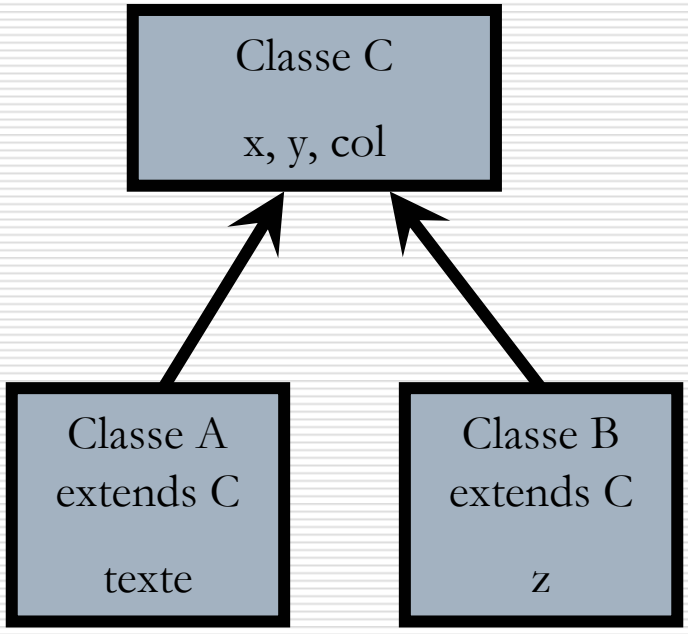
- Si deux classes à construire comportent des attributs communs

```
class A{
    private int x, y;
    private Color col;
    private String
       texte;
    }
et class B{
    private int x, y, z;
    private Color col;
    }
```

- on évitera les duplications en rassemblant les éléments communs dans une classe de base

```
class C{
    private int x, y;
    private Color col;
    }
class A extends C{
    private String
       texte;
    }
class B extends C{
    private int z;
    }
```

Et les méthodes



- dans la classe C, les méthodes qui travaillent sur x, y, col,
- dans la classe A, les méthodes qui ont besoin de texte,
- dans la classe B, les méthodes qui ont besoin de z.

Classes abstraites

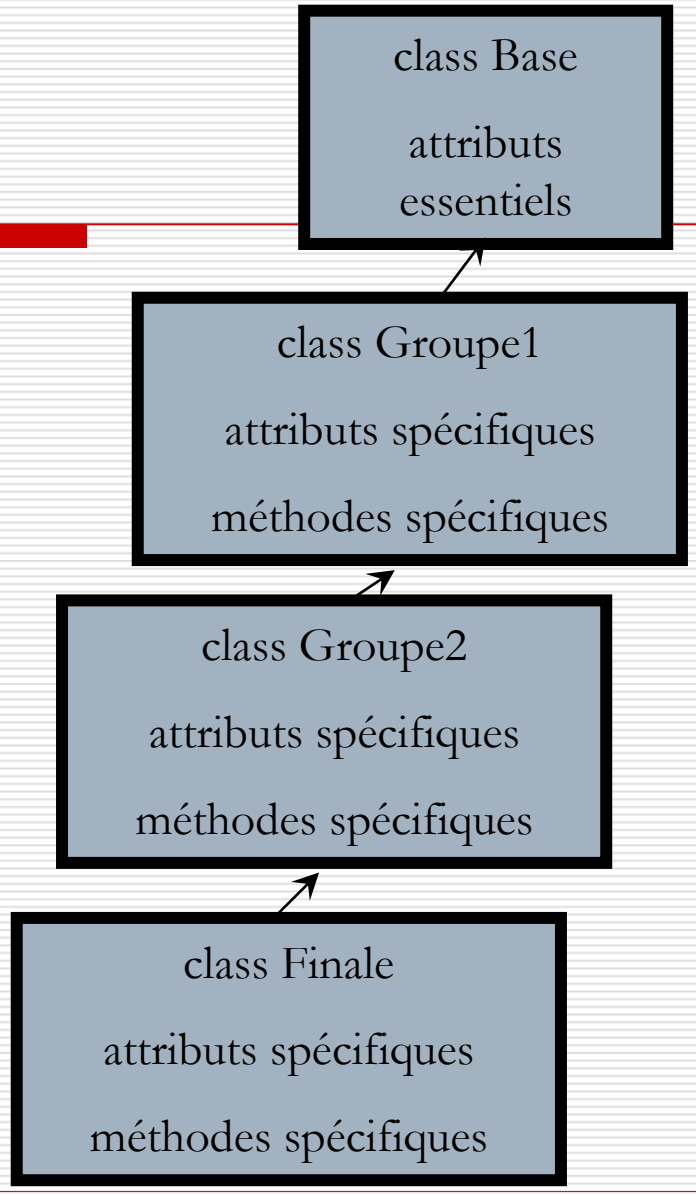
- il est possible que la classe de base ne puisse être employée seule
- exemple : class Voiture extends Vehicule{…}, class Cycle extends Vehicule{…}, class Velomoteur extends Cycle{…}, class Velo extends Cycle{…}
- les classes Vehicule et Cycle seront probablement abstraites, on n'écrira jamais new Vehicule() ou new Cycle()…

Réduction de volume

- Il est malsain qu'une classe contienne beaucoup d'attributs et de longues méthodes
- rassembler par paquets les attributs liés
- rassembler par paquets les méthodes liées

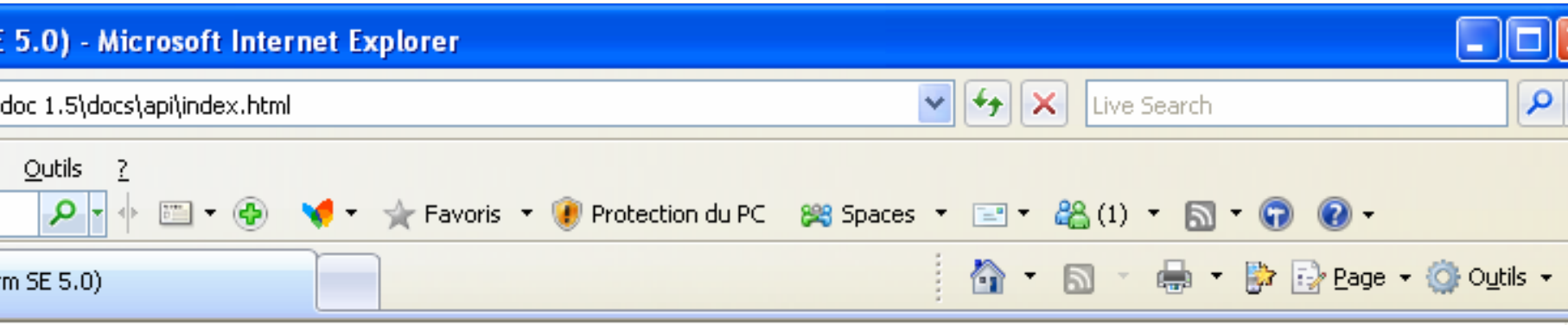
Réduction de volume

- Dans la classe Finale, tous les attributs et toutes les méthodes sont présents
- Chaque classe est de taille raisonnable, testable séparément



Les interfaces

- attention : ne pas confondre les interfaces et les GUI (interfaces homme-machine)
- une classe peut se conformer à une normalisation de noms de méthodes
  - pour éviter que des traitements similaires ne portent des noms différents
  - parce qu'elle doit employer des méthodes qui existent déjà (sur un serveur par exemple)
  - pour respecter une normalisation existante (w3c par exemple)
- Par exemple, l'interface **`Comparable`** spécifie que, pour que deux objets soient considérés comme comparables par les outils standards, leur classe doit contenir une méthode **`compareTo(...)`**
- Une interface ne contient que des descriptions de méthodes, pas d'algorithme !



va.lang

nterface Comparable<T>

ll Known Subinterfaces:

Delayed, Name, ScheduledFuture<V>

ll Known Implementing Classes:

Authenticator.RequestorType, BigDecimal, BigInteger, Boolean, Byte, ByteBuffer, Calendar, Character, CharBuffer, Charset, CollationKey, CompositeName, CompoundName, Date, Date, Double, DoubleBuffer, ElementType, Enum, File, Float, FloatBuffer, Formatter.BigDecimalLayoutForm, FormSubmitEvent.MethodType, GregorianCalendar, IntBuffer, Integer, JTable.PrintMode, KeyRep.Type, LdapName, Long, LongBuffer, MappedByteBuffer, MemoryType, ObjectStreamField, Proxy.Type, Rdn, RetentionPolicy, RoundingMode, Short, ShortBuffer, SSLEngineResult.HandshakeStatus, SSLEngineResult.Status, String, Thread.State, Time, Timestamp, TimeUnit, URI, UUID

Exemple d'interface (suite)

Method Summary

int	`compareTo(T o)` Compares this object with the specified object for order.

Method Detail

compareTo

```
int compareTo(T o)
```

Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

In the foregoing description, the notation `sgn(`*expression*`)` designates the mathematical *signum* function, which is defined to return one of `-1`, `0`, or `1` according to whether the value of *expression* is negative, zero or positive. The implementor must ensure `sgn(x.compareTo(y)) == -sgn(y.compareTo(x))` for all `x` and `y`. (This implies that `x.compareTo(y)` must throw an exception iff `y.compareTo(x)` throws an exception.)

The implementor must also ensure that the relation is transitive: `(x.compareTo(y)>0 && y.compareTo(z)>0)` implies `x.compareTo(z)>0`.

Finally, the implementer must ensure that `x.compareTo(y)==0` implies that `sgn(x.compareTo(z)) == sgn(y.compareTo(z))`, for all `z`.

It is strongly recommended, but *not* strictly required that `(x.compareTo(y)==0) == (x.equals(y))`. Generally speaking, any class that implements the `Comparable` interface and violates this condition should clearly

Les interfaces et les classes

- Une classe peut annoncer qu'elle implémente une interface : `class MaClasse implements MonInterface{…}`
- Elle doit alors proposer toutes les méthodes spécifiées dans l'interface et leur donner un corps (algorithme)

Les interfaces riches

- Certaines interfaces contiennent beaucoup de méthodes
- `Window`**`Listener`** propose 6 méthodes
- Si une classe veut implémenter `Window`**`Listener`**, elle aura l'obligation de donner un corps aux 6 méthodes
- La classe `Window`**`Adapter`** fournit 6 implémentations vides

Method Summary

void	`windowActivated(WindowEvent e)` Invoked when the Window is set to be the active Window.
void	`windowClosed(WindowEvent e)` Invoked when a window has been closed as the result of calling
void	`windowClosing(WindowEvent e)` Invoked when the user attempts to close the window from the w
void	`windowDeactivated(WindowEvent e)` Invoked when a Window is no longer the active Window.
void	`windowDeiconified(WindowEvent e)` Invoked when a window is changed from a minimized to a norm
void	`windowIconified(WindowEvent e)` Invoked when a window is changed from a normal to a minimize
void	`windowOpened(WindowEvent e)` Invoked the first time a window is made visible.