
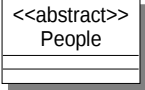


### 1. La modélisation statique

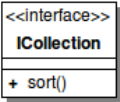

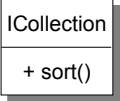
Les codes Java d'implémentation des concepts UML ne sont que des exemples. Il existe quelques fois plusieurs implémentations possibles et équivalentes.

#### Classe

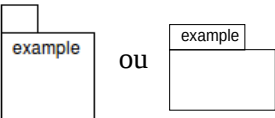
	UML	Java
Classe concrète		<pre>public final class Student {     ... }</pre>
Classe abstraite		<pre>public abstract class People {     ... }</pre>

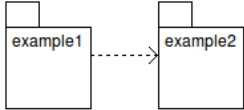
#### Interface

Trois représentations sont possibles. Les deux premières sont à privilégier parce qu'elles sont plus claires. La dernière peut être confondue avec une classe.

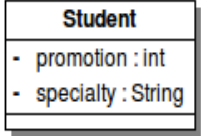
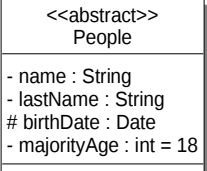
	UML	Java
Classe stéréotypée		<pre>public interface ICollection {     public void sort(); }</pre>
Icône		
Classe avec 2 compartiments		

#### Paquet

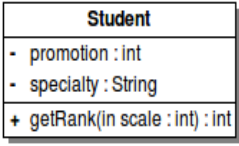
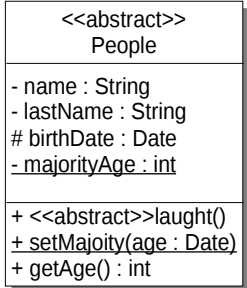
	UML	Java
Paquet		<pre>package example;</pre>

Importation		<pre>package example1 ; import example2.*;</pre>	<pre>package example2;</pre>
-------------	---	--	------------------------------

## Attribut

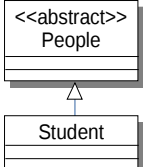
	UML	Java
Attribut privé		<pre>public final class Student {     private int _promotion;     private String _specialty;     ... }</pre>
Attribut protégé et attribut statique		<pre>public abstract class People {     private String _name;     private String _lastName;     protected Date _birthDate;     private static int _majorityAge = 18;     ... }</pre>

## Opération

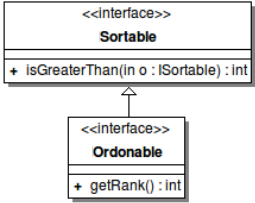
	UML	Java
Opération avec valeur de retour et paramètre		<pre>public final class Student {     private int _promotion;     private String _specialty;     public int getRank( int scale ) {         ...     } }</pre>
Opération statique, Opération abstraite		<pre>public abstract class People {     private String _name;     private String _lastName;     protected Date _birthDate;     private static int _majorityAge;      public abstract void laught();     public static void setMajoity( Date date ) {         ...     }     public int getAge() {         ...     } }</pre>

## 2. Relation

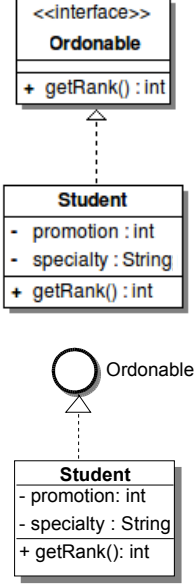
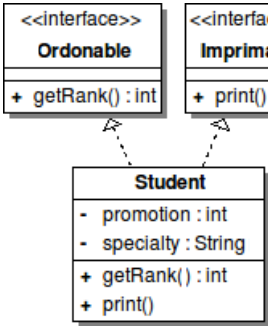
### Généralisation

	UML	Java
Héritage de classe		<pre>public abstract class People {     ... }  public final class Student extends People {     ... }</pre>

### 3 ■ Correspondance UML Java

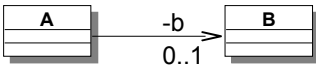
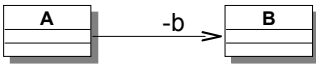
Héritage d'interface	 <pre> classDiagram     class Sortable {         &lt;&lt;interface&gt;&gt;         + isGreaterThan(in o : Sortable) : int     }     class Ordonable {         &lt;&lt;interface&gt;&gt;         + getRank() : int     }     Sortable &lt; -- Ordonable         </pre>	<pre> public interface Sortable {     public void isGreaterThan( Sortable o ); }  public interface Ordonable extends Sortable {     public int getRank(); }         </pre>
----------------------	--	--

### Réalisation

	UML	Java
Réalisation d'une interface	 <pre> classDiagram     class Ordonable {         &lt;&lt;interface&gt;&gt;         + getRank() : int     }     class Student {         - promotion : int         - specialty : String         + getRank() : int     }     Ordonable .. .. Student         </pre>	<pre> public interface Ordonable {     public int getRank(); }  public final class Student implements Ordonable {     private int _promotion;     private String _specialty;      public int getRank() {         ...     } }         </pre>
Réalisation de plusieurs interfaces	 <pre> classDiagram     class Ordonable {         &lt;&lt;interface&gt;&gt;         + getRank() : int     }     class Imprimable {         &lt;&lt;interface&gt;&gt;         + print()     }     class Student {         - promotion : int         - specialty : String         + getRank() : int         + print()     }     Ordonable .. .. Student     Imprimable .. .. Student         </pre>	<pre> public interface Imprimable {     public void print(); }  public interface Ordonable {     public int getRank(); }  class Student implements Ordonable, Imprimable {     private int _promotion;     private String _specialty;      public int getRank() {         ...     }      public void print() {         ...     } }         </pre>

ENSICAEN - Spécialité Informatique

### Association

	UML	Java
Association navigable de multiplicité 0..1	 <pre> classDiagram     class A     class B     A --&gt; B : -b         </pre>	<pre> public final class A {     private B _b;     ... }  public final class B { }         </pre>
Association navigable de multiplicité 1	 <pre> classDiagram     class A     class B     A --&gt; B : -b         </pre>	<pre> public final class A {     private B _b = new B();     ... }  public final class B { }         </pre>

Association avec une multiplicité fixée		<pre>public final class A {     private B[] _bs = new B[10];     ... }</pre>
Association avec une multiplicité quelconque		<pre>public final class A {     private List&lt;B&gt; _bs = new ArrayList&lt;B&gt;();     ... }</pre>
Association multiple ordonnée		<pre>public final class A {     private Set&lt;B&gt; _bs = new TreeSet&lt;B&gt;();     ... } public final class B implements Comparable&lt;B&gt; {     int compareTo( B b ) { } }</pre>
Association qualifiée		<pre>public final class A {     private Map&lt;B&gt; _bs = new HashMap&lt;B&gt;();     ... }</pre>
Association sans navigabilité (bidirectionnelle)		<pre>public final class A {     private B _b; } public final class B {     private A _a; }</pre>
Association réflexive (bidirectionnelle)		<pre>public final class People {     private List&lt;People&gt; _employees;     private People _employer; }</pre>

### Agrégation et composition

	UML	Java
Agrégation		<pre>public final class Voiture {     private List&lt;Pneu&gt; _pneus = new ArrayList&lt;Pneu&gt;(); }</pre>
Composition		<pre>public final class Voiture {     private final Chassis _chassis;     private Voiture() {         _chassis = new Chassis();     }     ... } public final class Chassis { }</pre>
Composition comme une classe interne		<pre>public final class Voiture {     private final Chassis _chassis;     private Voiture() {         _chassis = new Chassis();     }     class Chassis {     }     ... }</pre>

### Dépendance

	UML	Java
Lien de dépendance entre classes		<pre>public final class A {     ...     public use( B b ) {     } } public final class B {     public void method() {         C c;     } } public final class C { }</pre>

## 5 ■ Correspondance UML Java

### Classe d'association

	UML	Java
Association promue au rang de classe. (abstraction hypostatique)		<pre>public final class Student { }  public final class Professor { }  public final class Teach {     private String _course;     private int _year;      private Professor _professor;     private Student _student;     ... }</pre>

## 3. La modélisation dynamique

### Séquences

ENSICAEN - Spécialité Informatique

	UML	Java
Diagramme de séquences		<pre>public final class A {     public void scenario() {         b.operationB1();         operationA1();         b.operationB2();     }     private void operationA1() { ... } }  public final class B {     ...     public void operationB1() { ... }     public boolean operationB2() { ... } }</pre>

### Communication

	UML	Java
Diagramme de communication		<pre>public final class Library {     private Librarian _lib;     private Member _a123;      public void register( int idLib ) {         Member a = _a123.find(idLib);         _lib.SetMembers(a);     } }</pre>

### États-transitions

	UML	Java
Diagramme états-transitions		<pre>public final class A {     Enum States {state1, state2, state3 }     private Enum States _state;     public void do( Event e ) {         switch (_state) {             case state1 : ...             case state2 : ...         }     } }</pre> <p>(Voir le cours de 2A sur le patron de conception État.)</p>