

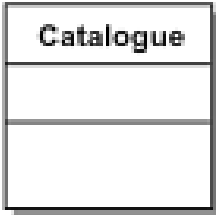


# TRADUIRE UML EN C#/JAVA

1

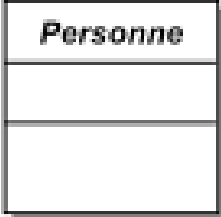
# CLASS

- Chaque classe UML devient un fichier .cs (C#) ou .java (Java), mais le stockage peut être différent si le concepteur le souhaite.

UML	Java
	<pre>public class Catalogue {     ... }</pre>
	C#
	<pre>public class Catalogue {     ... }</pre>



# CLASS ABSTRAITE

- Une classe abstraite est simplement une classe qui ne s'instancie pas directement mais qui représente une pure abstraction afin de factoriser des propriétés. Elle se note *italique*.

UML	Java
	<pre>public abstract class Personne {     ... }</pre>
	C#
	<pre>public abstract class Personne {     ... }</pre>

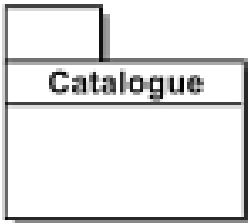
# INTERFACE

- L'interface peut être représenté en UML différemment

UML	Java
 <p>The diagram shows a rectangular box representing an interface. The top section is labeled «interface» and contains the name <b>IAffichable</b>. The bottom section contains the method signature <b>+afficher()</b>.</p>	<pre>public interface IAffichable {     public void afficher(); }</pre>
	C#
 <p>The diagram shows a circle representing an interface, with the name <b>IAffichable</b> written below it.</p>	<pre>public interface IAffichable {     void Afficher(); }</pre>

# PACKAGE

- Le package représente un regroupement de classe.

UML	Java
 A UML package diagram showing a package named 'Catalogue'. The package is represented by a rectangle with a tab on the top-left corner. The name 'Catalogue' is written inside the rectangle.	<code>package catalogue;</code> ...
	C#
	<code>namespace Catalogue</code> { ... }

# ATTRIBUT

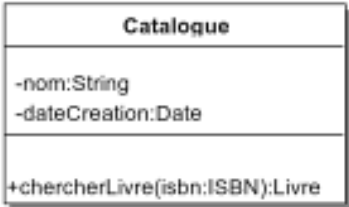
- Les attributs deviennent des variables en C#. Leur type sont soit des types primitifs (int ,string), soit des classes (DateTime , String).

Il faut ne pas oublier d'ajouter le package nécessaire afin que la définition du type ou de la Classe soit connue. La class ArrayList nécessite d'ajouter le package System.Collections.

UML	Java
 <pre>classDiagram     class Catalogue {         -nom:String         -dateCreation:Date     }</pre>	<pre>public class Catalogue {     private String nom;     private Date dateCreation; }</pre>
C#	<pre>public class Catalogue {     private string nom;     private DateTime dateCreation; }</pre>
UML	Java
 <pre>classDiagram     class Personne {         -nom:String         -prenom:String         #dateNaissance:Date         -ageMajorite:int=18     }</pre>	<pre>abstract public class Personne {     private String nom;     private String prenom;     protected Date dateNaissance;     private static int ageMajorite = 18; }</pre>

# OPERATIONS

- Les opérations deviennent des méthodes en Java et C#.
- Leur visibilité est définie avec la même convention que les attributs.
- Les opérations de classes deviennent des méthodes statiques. Les opérations abstraites (en italique) se traduisent par le mot clé *abstract*.

UML	Java
 <pre>classDiagram     class Catalogue {         -nom:String         -dateCreation:Date         +chercherLivre(isbn:ISBN):Livre     }</pre>	<pre>public class Catalogue {     private String nom;     private Date dateCreation;     public Livre chercherLivre(ISBN isbn) {         --     }     ... }</pre>
	C#
	<pre>public class Catalogue {     private string nom;     private DateTime dateCreation;     public Livre ChercherLivre(ISBN isbn) {         --     }     ... }</pre>

# OPERATIONS

UML	Java	
<pre> classDiagram     class Personne {         -nom:String         -prenom:String         #dateNaissance:Date         -ageMajorite:int=18         +calculerDureePret():int         +setAgeMajorite(a:int)         +getAge():int     }         </pre>	<pre> abstract public class Personne {     private String nom;     private String prenom;     protected Date dateNaissance;     private static int ageMajorite = 18;     public abstract int calculerDureePret();     public static void setAgeMajorite(int aMaj) {         ...     }     public int getAge() {         ...     } }         </pre>	
	<th>C#</th>	C#
	<pre> abstract public class Personne {     private String nom;     private String prenom;     protected DateTime dateNaissance;     private static int ageMajorite = 18;     public abstract int calculerDureePret();     public static int AgeMajorite     {         get { return Personne.ageMajorite; }         set { Personne.ageMajorite = value; }     } }         </pre>	

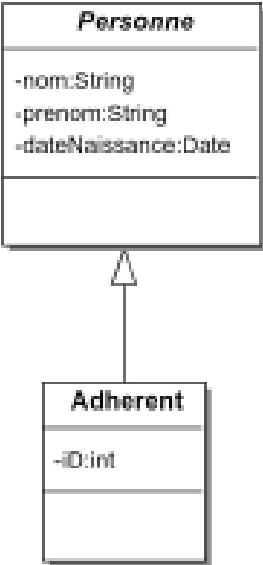


# OPERATIONS

- Un attribut est accessible par l'intermédiaire d'une opération particulière, les accesseurs et modifieurs.
- On appelle ces accesseurs et modifieurs, Propriété en C#. La propriété a le même nom que l'attribut avec une majuscule.

# HÉRITAGE

Afin de préciser de qui hérite une classe, on spécifie le nom du père à côté de la classe fille.

UML	Java
 <pre>classDiagram     class Personne {         -nom:String         -prenom:String         -dateNaissance:Date     }     class Adherent {         -ID:int     }     Personne &lt; -- Adherent</pre>	<pre>public class Adherent extends Personne {     private int iD; }</pre>
	C#
	<pre>public class Adherent : Personne {     private int iD; }</pre>

# RÉALISATION

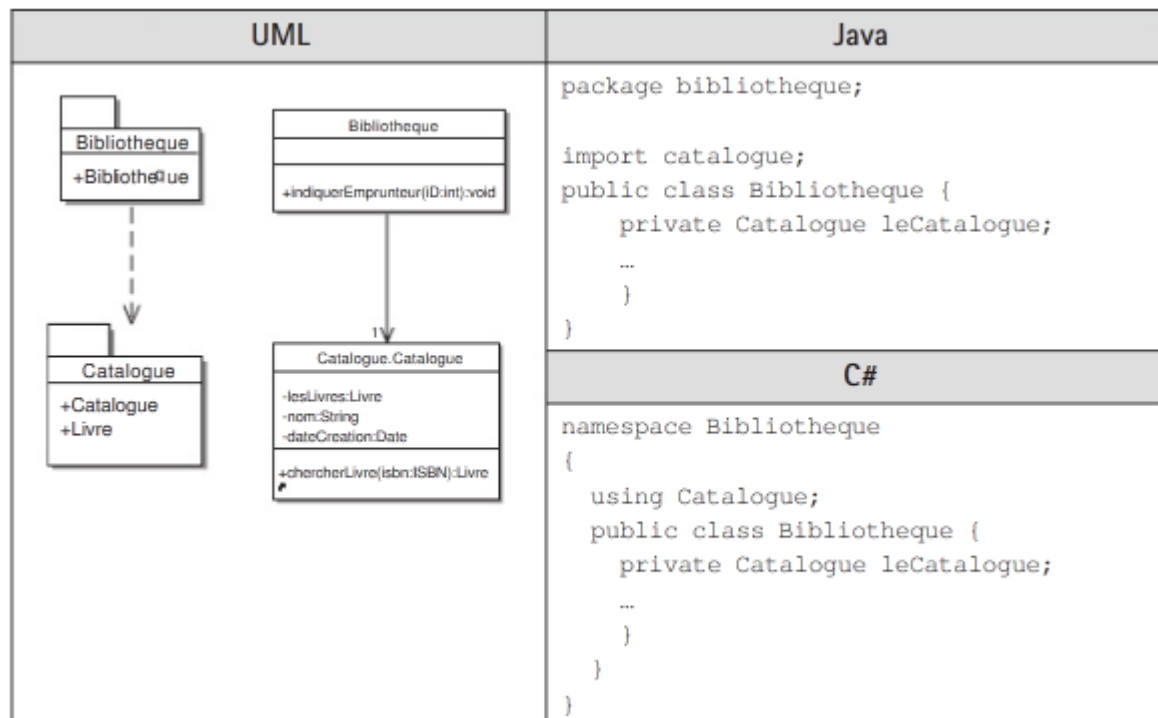
- Une classe UML peut réaliser plusieurs interfaces ou classes. Mais en C#, l'héritage multiple est interdit alors que la réalisation de multiple interface est autorisée.

# RÉALISATION

UML	Java
<pre> classDiagram     class IEmpruntable {         &lt;&lt;interface&gt;&gt;         +emprunter():void         +retourner():void     }     class IImprimable {         &lt;&lt;interface&gt;&gt;         +imprimer():void     }     class Livre {         -titre:String         -auteur:String         -isbn:ISBN         +imprimer():void         +emprunter():void         +retourner():void     }     IEmpruntable .. &gt; Livre     IImprimable .. &gt; Livre         </pre>	<pre> public class Livre implements     IImprimable, IEmpruntable {     private String titre;     private String auteur;     private ISBN isbn;     public void imprimer() {         ...     }      public void emprunter() {         ...     }      public void retourner() {         ...     } }         </pre>
	C#
	<pre> public class Livre :     IImprimable, IEmpruntable {     private string titre;     private string auteur;     private ISBN isbn;     public void Imprimer() {         ...     }      public void Emprunter() {         ...     }      public void Retourner() {         ...     } }         </pre>

# DÉPENDANCE

- La dépendance est un concept très général en UML. Une dépendance entre une classe A et une classe B existe par exemple si A possède une méthode prenant comme paramètre une référence sur une instance de B, ou si A utilise une opération de classe de B. Il n'existe pas de mot-clé correspondant en Java ou en C#.





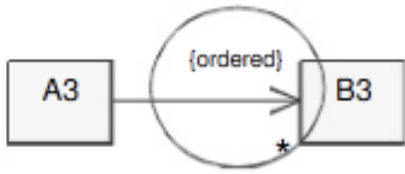
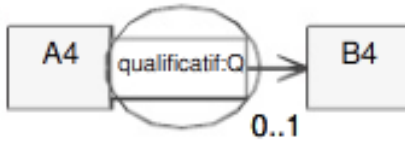
# ASSOCIATION

- Les associations navigables se traduisent en code en prenant compte de la multiplicité de l'extrémité concernée, mais également l'existence d'une contrainte {ordered} ou d'un qualificatif.
- Une association navigable avec une multiplicité 1 se traduit par une variable d'instance , tout comme un attribut, mais avec un type référence vers une instance de classe du modèle au lieu d'un type simple.
- Une multiplicité « \* » va se traduire par un attribut de type collection de références d'objets au lieu d'une simple référence sur un objet. La difficulté consiste à choisir la bonne collection parmi toutes les collections C#.

# ASSOCIATION

- En C#, on va utiliser:
  - `List<T>` en général
  - `SortedSet<T>` si vous devez respecter un ordre et récupérer les objets à partir d'un indice entier
  - `Dictionary<Tkey,TValue>` si vous souhaitez récupérer les objets à partir d'une clé arbitraire.
- Mais on peut également utiliser `ArrayList` et toutes les autres collections.

# ASSOCIATION

UML	Java	C#
	<pre>public class A1 {     private B1 leB1;     ... }</pre>	<pre>public class A1 {     private B1 leB1;     ... }</pre>
	<pre>public class A2 {     private B2 lesB2[];     ... }</pre>	<pre>public class A2 {     private List&lt;B2&gt; lesB2;     ... }</pre>
	<pre>public class A3 {     private List&lt;B3&gt; lesB3;     ... }</pre>	<pre>public class A2 {     private SortedSet&lt;B3&gt; lesB3;     ... }</pre>
	<pre>public class A4 {     private Map&lt;Q, B4&gt; lesB4;     ... }</pre>	<pre>public class A4 {     private Dictionary&lt;Q, B4&gt; lesB4;     ... }</pre>



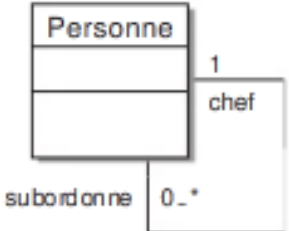
# ASSOCIATION BIDIRECTIONNELLE

- Elle se traduit simplement par une paire de références, une dans chaque classe impliquée dans l'association. Les noms des rôles aux extrémités d'une association servent à nommer les variables de type référence.

UML	Java	C#
<pre>classDiagram     class Homme     class Femme     Homme "0..1" -- "0..1" Femme : mari / epouse</pre>	<pre>public class Homme {     private Femme epouse;     ... }  public class Femme {     private Homme mari;     ... }</pre>	<pre>public class Homme {     private Femme epouse;     ... }  public class Femme {     private Homme mari;     ... }</pre>

# ASSOCIATION RÉFLEXIVE

- Elle se traduit simplement par une référence sur un objet de la même classe.

UML	Java
	<pre>public class Personne {     private Personne subordonne[];     private Personne chef ;     ... }</pre>
	<div>C#</div> <pre>public class Personne {     private Personne[] subordonne;     private Personne chef ;     ... }</pre>

# AGRÉGATION ET COMPOSITION

- L'agrégation est un cas particulier d'association non symétrique exprimant une relation de contenance. L'association peut être nommée « contient », « est composé de ».
- Le codage des agrégations n'est pas fondamentalement différent de celle des associations simples. La seule contrainte est qu'une association ne peut contenir de marque d'agrégation qu'à l'une de ses extrémités.
- Une composition est une agrégation plus forte impliquant que :
  - une partie ne peut appartenir qu'à un seul composite (agrégation non partagée) ;
  - la destruction du composite entraîne la destruction de toutes ses parties (le composite est responsable du cycle de vie des parties).

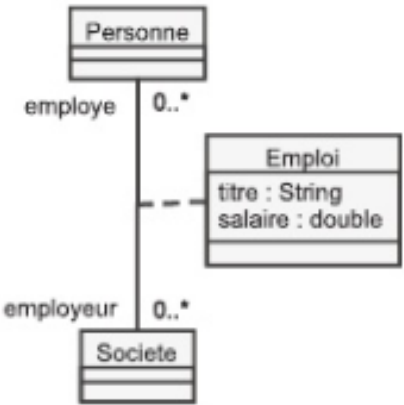
# AGRÉGATION ET COMPOSITION

- Les classes imbriquées peuvent permettre de traduire la composition.

UML	Java	C#
<pre>classDiagram     class Voiture {         -modele:String     }     class Moteur {         -puissance:int     }     Voiture "1" o-- "*" Moteur</pre>	<pre>public class Voiture {     private String modele;     private Moteur moteur; }</pre>	<pre>public class Voiture {     private string modele;     private Moteur moteur; }</pre>

# CLASSE D'ASSOCIATION

- Il s'agit d'une association promue au rang de classe. Elle possède tout à la fois les caractéristiques d'une association et d'une classe et peut donc porter des attributs.

UML	Java
	<pre>public class Emploi {     private String titre;     private double salaire;     private Personne employe;     private Societe employeur ;     ... }</pre>
	<b>C#</b> <pre>public class Emploi {     private string titre;     private double salaire;     private Personne employe;     private Societe employeur ;     ... }</pre>