UVS
UNIVERSITE VIRTUELLE DU SENEGAL

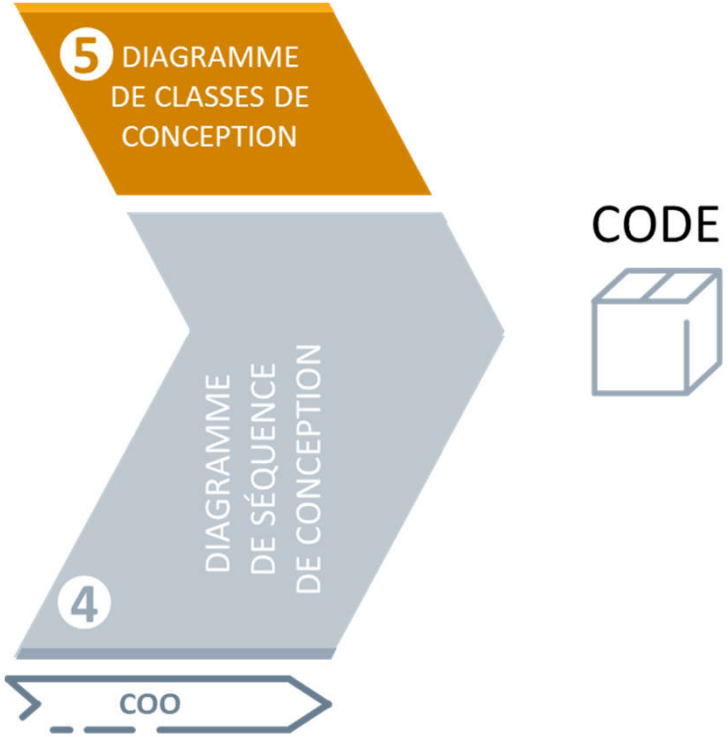
Analyse et Conception Orientée Objet

Conception Orientée Objet

Plan

- Conception Orientée Objet (COO)
- **Étape 1**: Diagramme de sequence de conception
- **Étape 2**: Diagramme de classes de conception
- Passage de la COO à la Programmation Orientée Objet

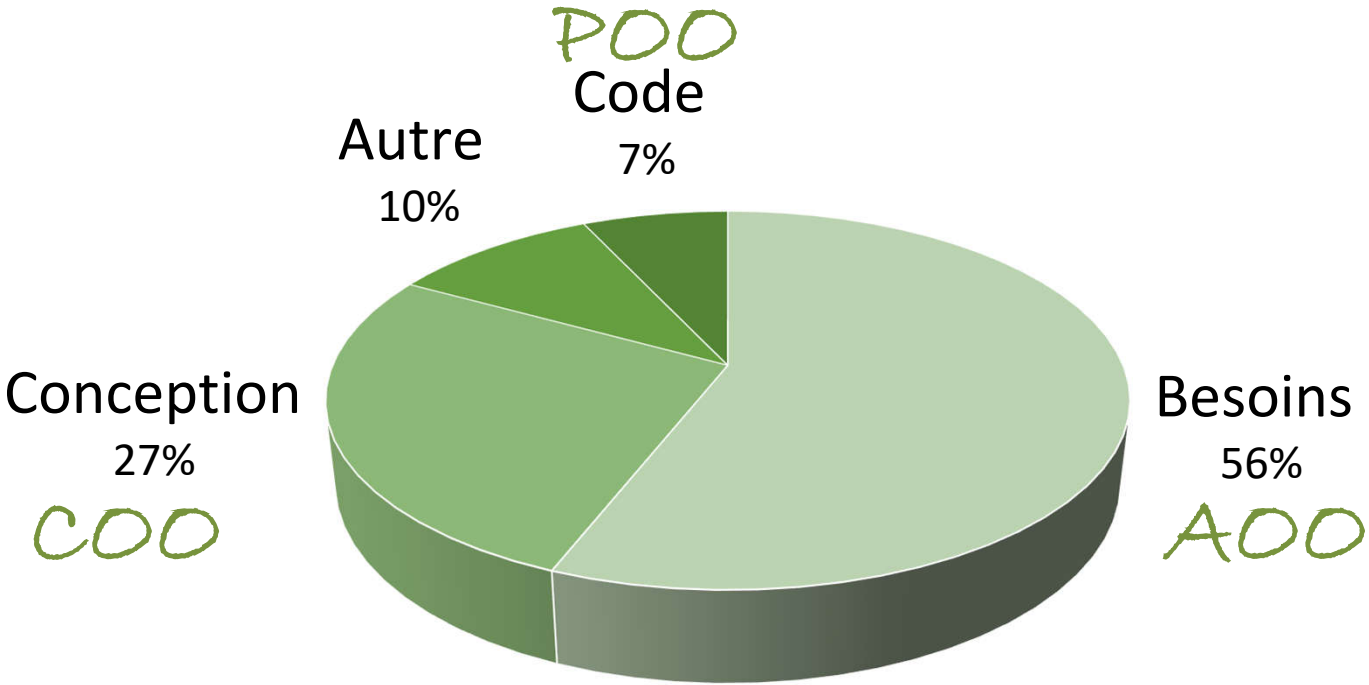
Conception Orientée Objet



Plan

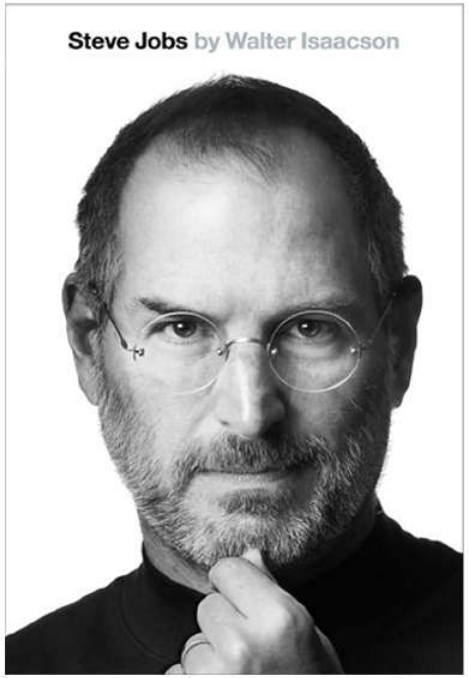
- Conception Orientée Objet (COO)
- Étape 1: Diagramme de sequence de conception
- Étape 2: Diagramme de classes de conception
- Passage de la COO à la Programmation Orientée Objet

Source des erreurs



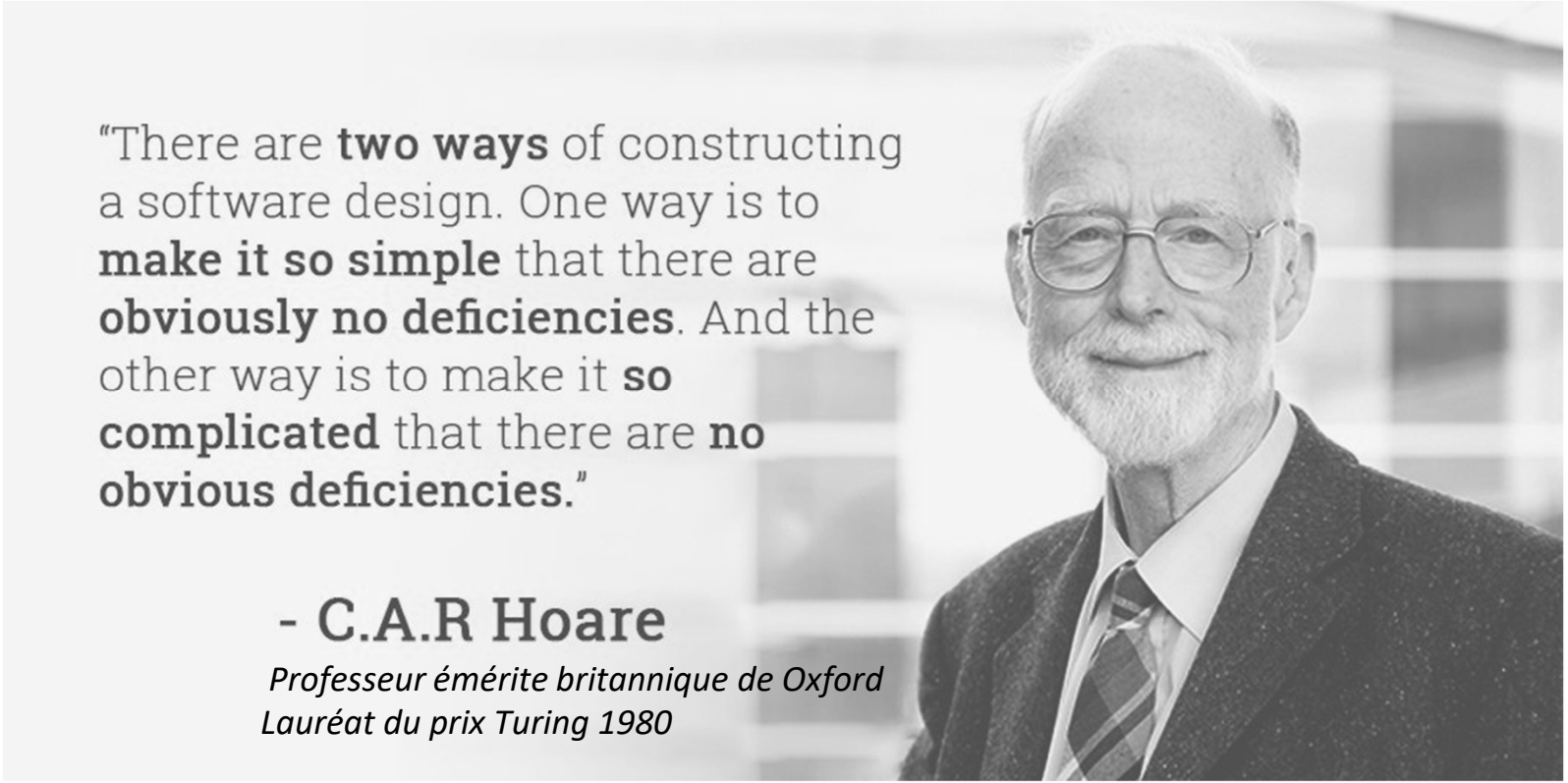
Source: Martin & Leffinel

Conception



DESIGN IS NOT JUST WHAT IT LOOKS LIKE. DESIGN IS HOW IT WORKS.

Conception



Credit: Programming Widsom

La seule et unique constante de la COO

Crédit: http://www.africansuccess.org

Un programme bien conçu

Un programme est «**bien conçu**» s'il permet de :

- Absorber les changements avec un minimum d'effort.
- Implémenter les nouvelles fonctionnalités sans toucher aux anciennes.
- Modifier les fonctionnalités existantes en modifiant localement le code.

Objectifs de la COO

- Affecter les responsabilités aux objets en utilisant des principes réutilisables.
- Réaliser des diagrammes de séquence illustrant les interactions entre objets.
- Concevoir des classes faiblement couplées et cohérentes.
- Créer des diagrammes de classes de conception.
- Traduire des modèles de conception dans un langage de programmation orienté objet.

UML Conception

Dans le cadre de ce cours lors de la phase de conception nous utiliserons deux (2) types de diagrammes :

- Diagramme de séquence
- Diagramme de classes

Il est possible d'utiliser d'autres diagrammes UML lors de la phase de conception par contre il faut être en mesure de le justifier.

20% de la notation UML est utilisée 80% du temps

Approche Conception Orientée Objet (COO)

A partir de l'Analyse Orientée Objet (AOO)

1. Créer des diagrammes de séquence de conception.

2. Créer des diagrammes de classes de conception.

Plan

- Conception Orientée Objet (COO)
- **Étape 1**: Diagramme de séquence de conception
- Étape 2: Diagramme de classes de conception
- Passage de la COO à la Programmation Orientée Objet

Prioriser et planifier les cas d'utilisation

Les Use Cases peuvent être classés en fonction de leur importance

- **Primaires** - utilisations importantes, courantes du système
  - «Il nous faut cela absolument ! »
- **Secondaires** - utilisations inhabituelles, rares du système
  - « Nous n'avons pas besoin de cela dans la première version... »

Prioriser les cas d'utilisation

Il faut traiter les cas d'utilisation les plus « importants » en premier :

- ceux représentant un processus clé du domaine;
- ceux ayant le plus d'impact sur l'augmentation des revenus et la réduction des coûts;
- ceux ayant le plus d'impact sur l'architecture;
- ceux contenant des fonctions complexes, risquées ou avec des contraintes temporelles;

Diagramme de cas d'utilisation

Chaque cas d'utilisation et chaque interface est une unité de travail.

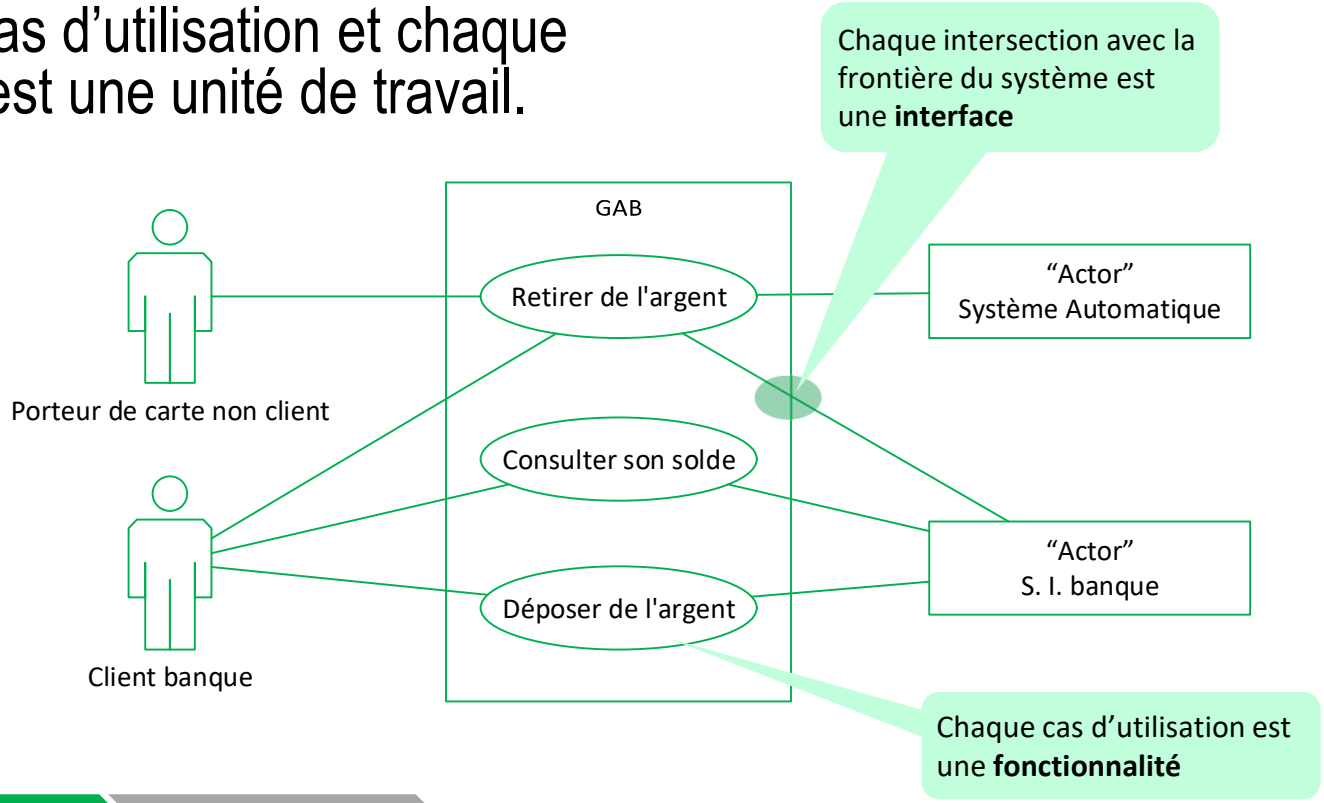
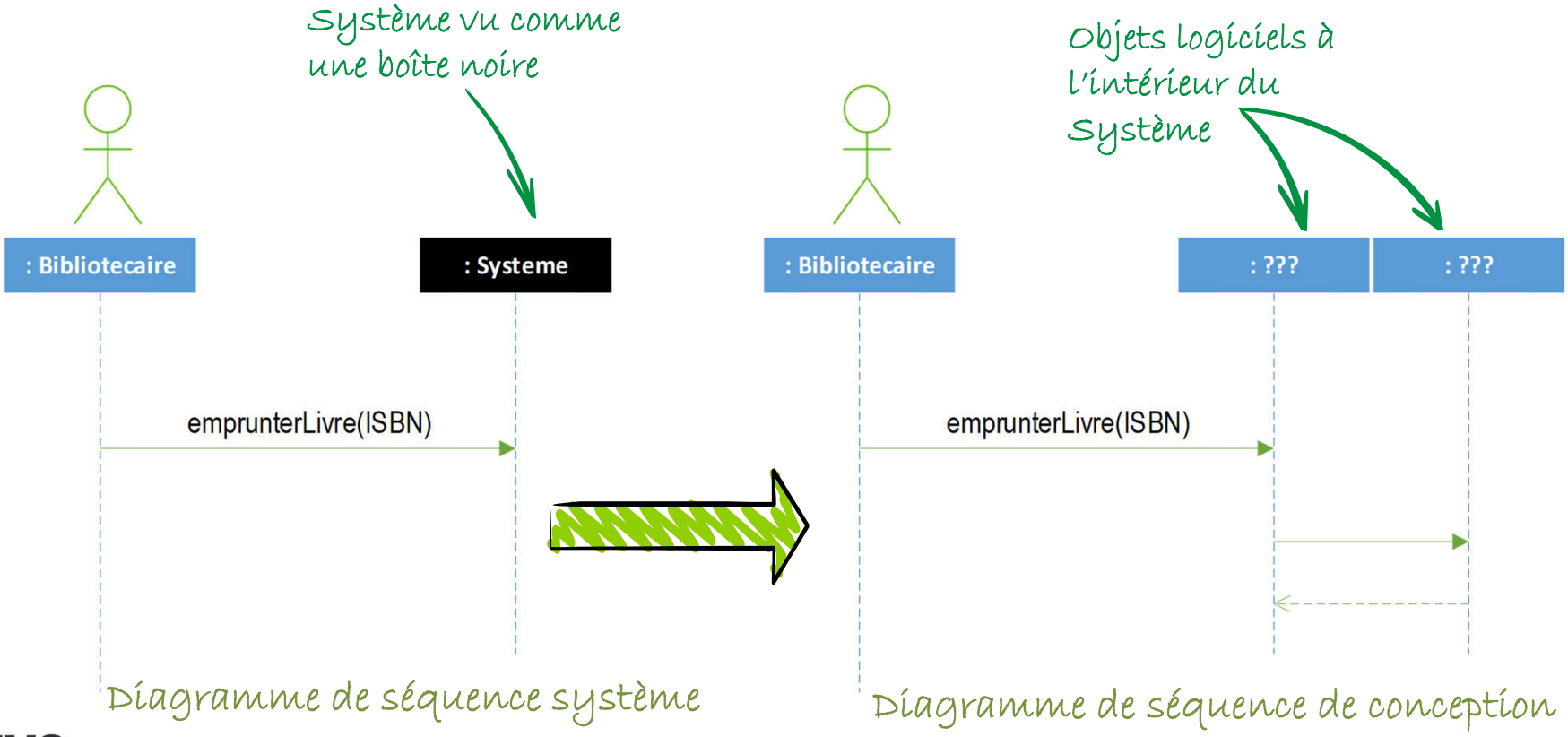


Diagramme de séquence de conception

À partir du partir du diagramme de séquence système et du diagramme de classes d'analyse, il s'agit de:

- choisir comment les objets logiciels vont interagir entre eux pour réaliser telle ou telle opération;
- remplacer le système vu comme une boîte noire (du point de vue de l'analyse) par des objets logiciels interopérants (du point de vue de la conception).

Passage de l'Analyse OO à la conception OO



Plan

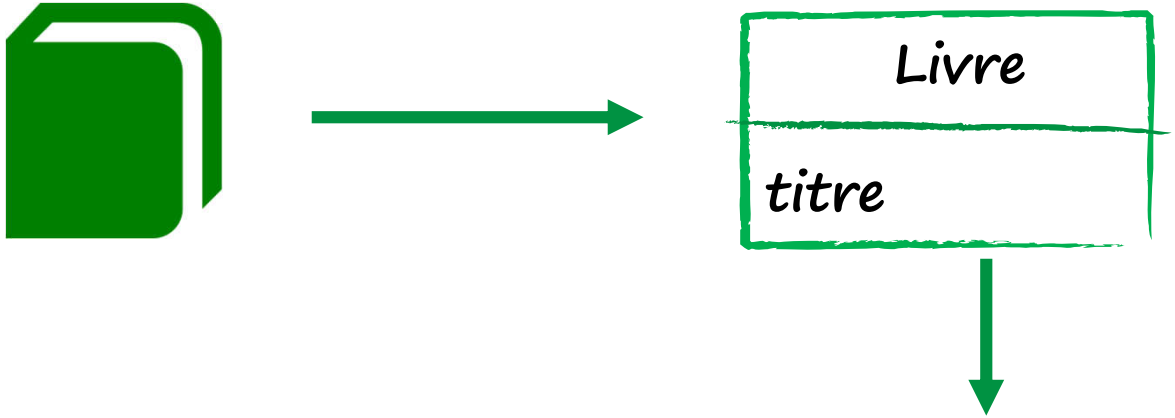
- Conception Orientée Objet (COO)
- Étape 1: Diagramme de séquence de conception
- **Étape 2**: Diagramme de classes de conception
- Passage de la COO à la Programmation Orientée Objet

Diagramme de classes de conception

Le diagramme de classes est le point central dans un développement orienté objet.

En conception, le diagramme de classes représente la structure d'un code orienté objet ou, à un niveau de détail plus important, les modules du langage de développement.

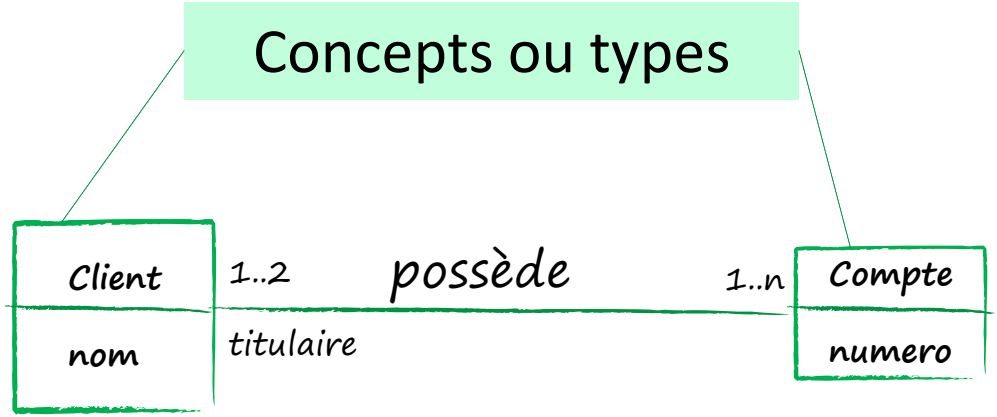
Concepts de modélisation




```
public class Livre {
        private  String titre;
        public Chapitre getChapitre(int){…}
        ….
}
```

Concepts et types

- Dans le jargon objet un concept est également appelé un type.



Classe et Instance

Une classe est:

- une définition - un modèle qui spécifie les propriétés des instances.
- un créateur - une usine pour créer des objets logiciels d'un certain type.

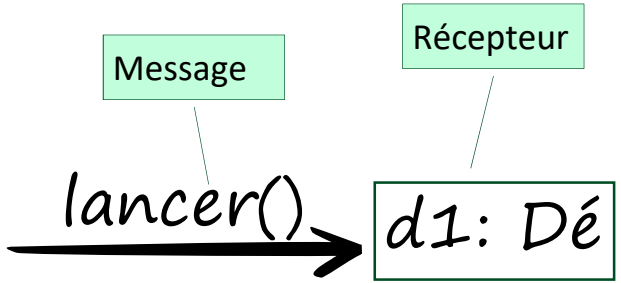
Une instance:

- est créée par une classe.
- occupe de l'espace dans la mémoire d'un ordinateur.
- conserve les valeurs des attributs.
- connaît sa classe

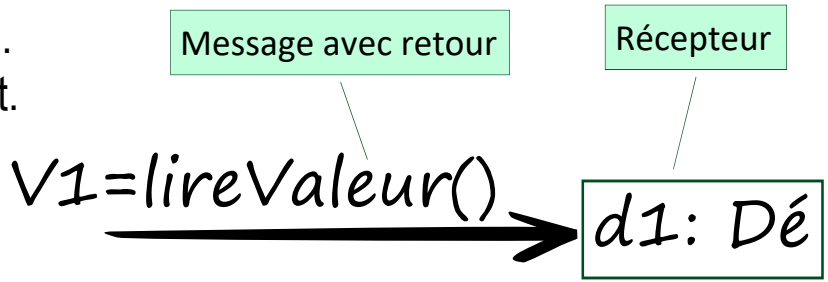
nomObjet : NomClasse

Messages et opérations

- **Message**
  - un signal vers un objet (récepteur) pour invoquer une opération;
  - peut retourner une valeur;
  - semblable à un appel à une fonction, mais dirigé vers un objet.

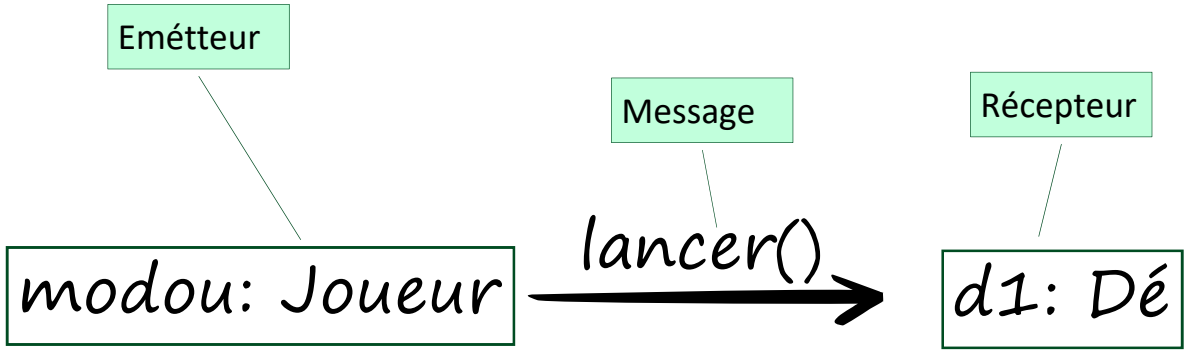


- **Opération**
  - un comportement d'un objet, invoqué par un message.
  - semblable à une fonction, mais appartenant à un objet.



Une interaction entre objects

- Les objets collaborent entre eux pour accomplir des taches.
- Un objet est l'émetteur, l'autre est le récepteur du message.



Contenance – association

- Les objets peuvent contenir d'autres objets.
- La complexité est hiérarchisée.
- Les objets sont inter-connectés.

Encapsulation

- **L'encapsulation fait référence à l'idée de dissimulation d'information ou de masquage des détails**

- **Un objet encapsule ses données.**
  - Les attributs dans un objet sont privés
  - L'environnement extérieur ne peut pas y accéder ou les modifier

- **Les opérations sont (typiquement) publiques; l'environnement extérieur peut envoyer des messages pour les invoquer**

Héritage

- **L'héritage :**
  - Est utile quand des classes peuvent être organisées selon une hiérarchie de type “généralisation/spécialisation”.
  - Signifie qu’une sous classe acquiert toutes les définitions d’attributs et d’opérations de sa super-classe.

- **Une sous classe peut redéfinir une opération héritée.**

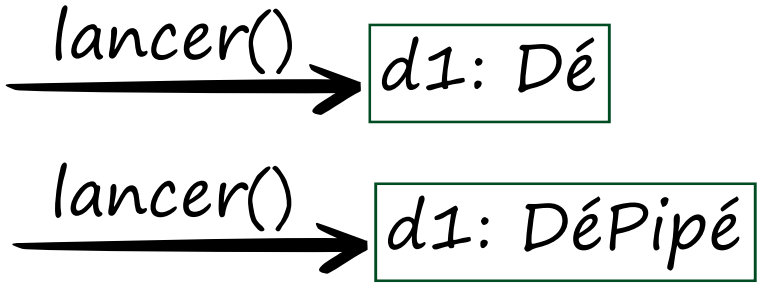
Classes abstraites et concrètes

Une classe abstraite :

- N’est pas instanciable.
- Regroupe des définitions communes aux sous-classes concrètes.

Polymorphisme

- **Le polymorphisme signifie que :**
  - Le même message peut être interprété de différentes façons, selon la classe du récepteur.
  - Une opération peut être définie par le même nom dans différentes classes.
- **Comment pouvez vous savoir quelle opération sera exécutée quand vous envoyez un message ?**



Catégories de concepts (1/2)

Créer une liste de concepts candidats à partir des catégories suivantes:

Catégorie de concept	Exemple
Objets tangibles ou physiques	Livre
Lieux	Bibliothèque
Transactions	Prêt, Vente
Unités de transaction	LigneDeVente
Rôles	Bibliothécaire, Client
Spécifications ou descriptions	DescriptionDeProduit

Catégories de concepts (2/2)

Catégorie de concept	Exemple
Evénements planifiés	RendezVous, Vol
Conteneurs	Bibliothèque, Catalogue, Etagère
Eléments dans un conteneur	Livre
Systèmes externes	SystèmedInformationDesEtudiants
Rôles	Bibliothécaire, Client
Organisations	Bibliothèque, Département

Les associations

- Sont des relations dignes d'intérêt entre des concepts.
- Sont bidirectionnelles.
- Doivent avoir un nom significatif;
  - De la forme: <Groupe verbal>.
- Quand les mettre en évidence?
  - Lorsque la relation a besoin d'être conservée.

Identifier les associations (1/2)

Catégorie d'association	Exemple
A fait partie de B	Phrase - Paragraphe
A est un article de B	Ligne de commande - Commande
A est contenu dans B	Livre - Bibliothèque
A est un membre de B	Bibliothècaire - Bibliothèque
A est une sous unité organisationnelle de B	Service - Entreprise
A est une règle liée à B	Politique de prêt – Document
A est enregistré dans B	Livre – Catalogue Prêt - Bibliothèque

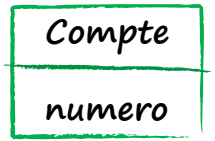
Identifier les associations (2/2)

Catégorie d'association	Exemple
A utilise ou dirige B	Client - Bibliothèque
A est lié à une transaction de B	Client - Prêt de document
A communique avec B	Client - Bibliothècaire
A est une transaction liée à une autre transaction B	Prêt – Retour Case d'échiquier— Case suivante
A vient après B	Case d'échiquier— Case suivante
A est lié à B par une transaction	Client - Bibliothécaire Mari - Femme

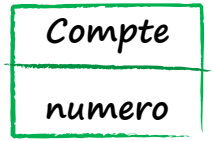
Attributs

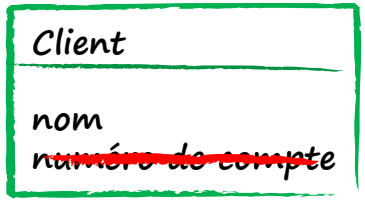
- **Seulement des attributs simples**
  - Exemples : un nombre, une chaîne de caractères, un booléen, une date, une heure
- **Pas de « clés étrangères » !**

Incorrect!



Incorrect!



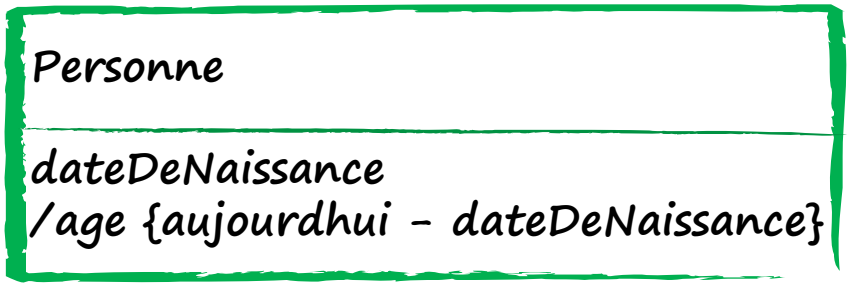


Correct!

Attribut dérivé

- **Un attribut est dérivé si sa valeur est déductible d'autres informations du modèle**
  - soit d'autres attributs du même objet
  - soit d'éléments externes à la classe

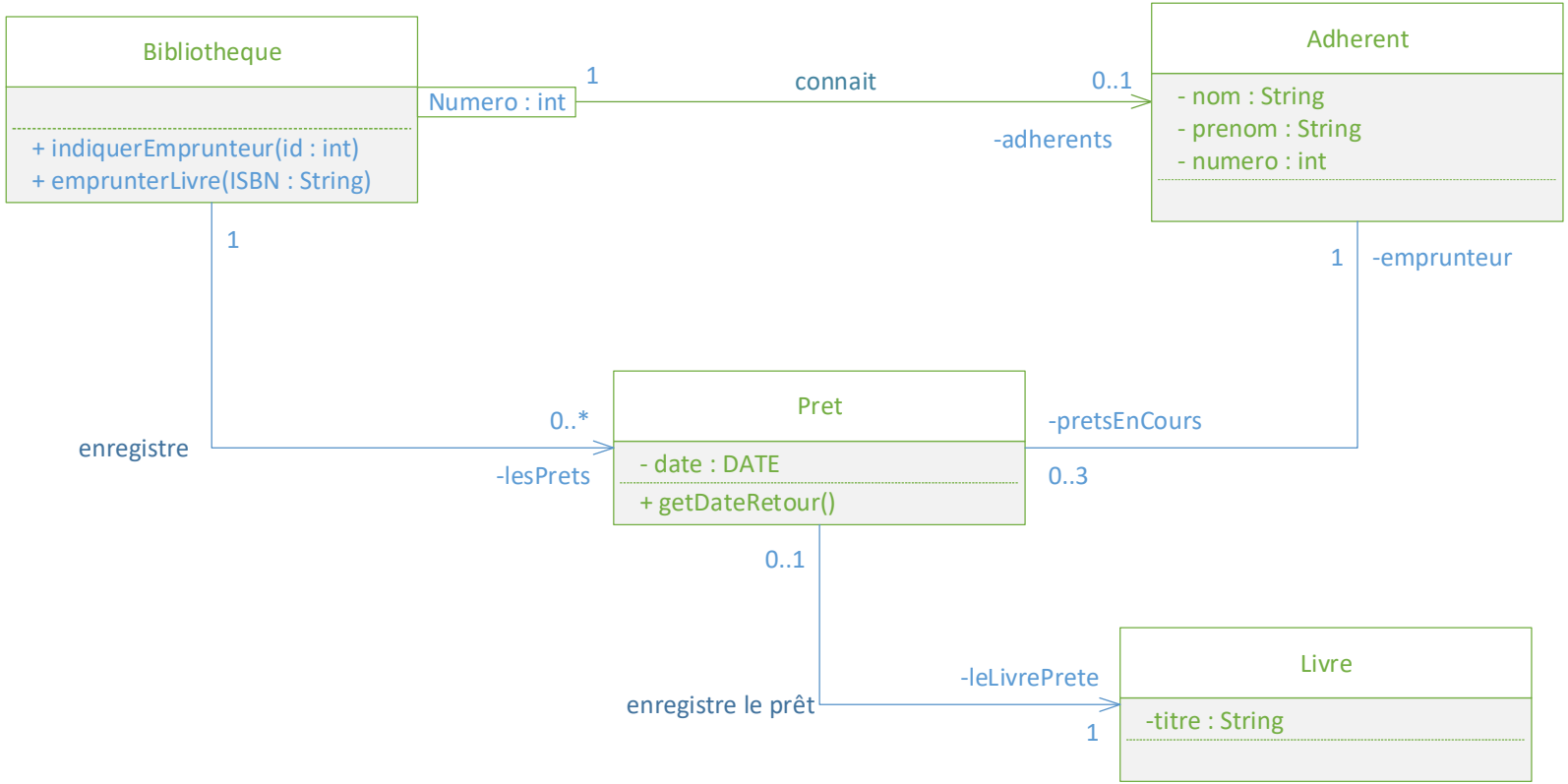
- **Les attributs dérivés sont préfixés par un «/»**
  - Une contrainte décrivant le calcul de l'attribut dérivé peut être indiquée entre { }



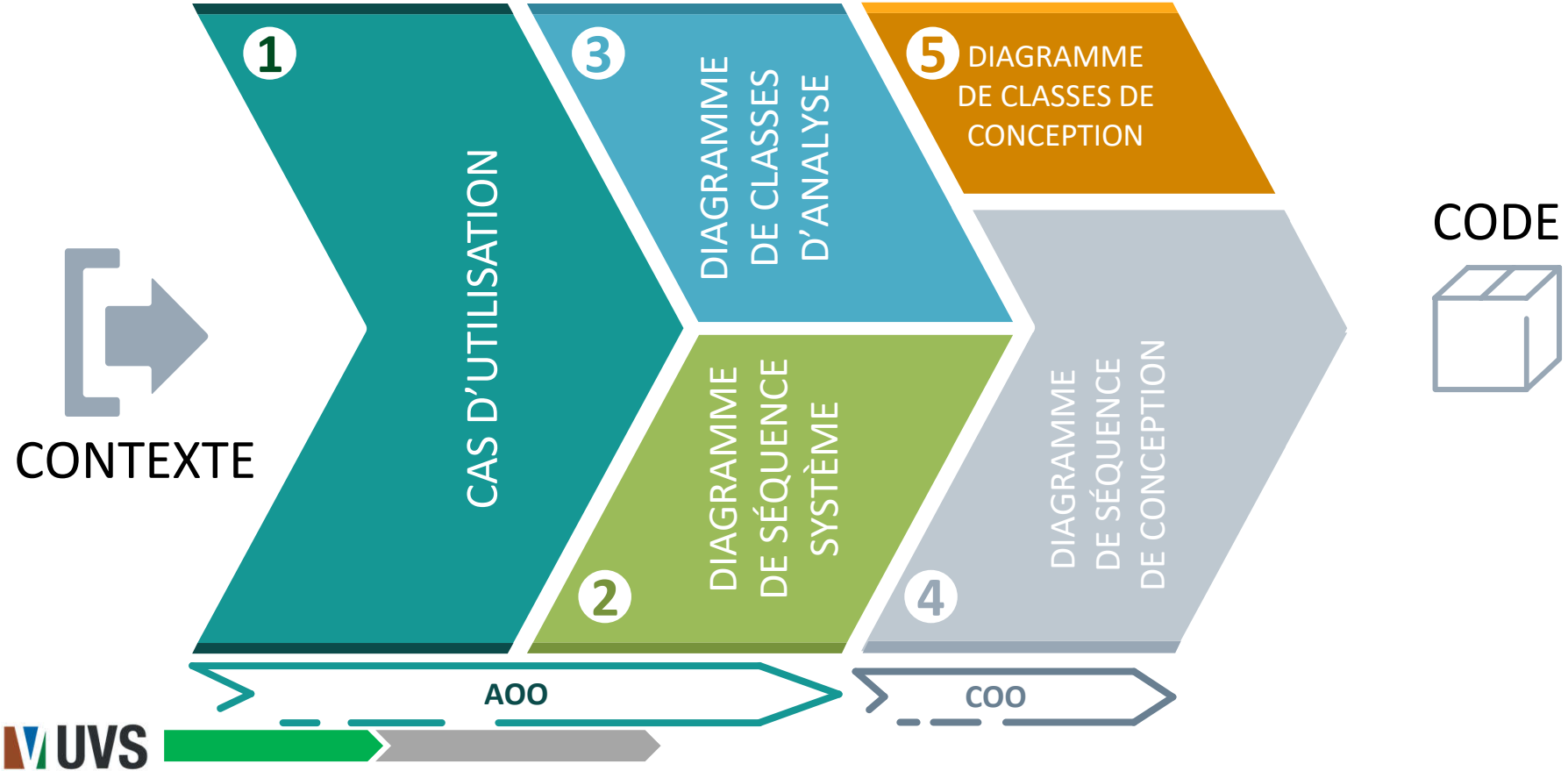
Créer des diagrammes de classes de conception

- **Les diagrammes de classes de conception permettent de modéliser les définitions des classes de programmation**
  - Leurs opérations (ou méthodes)
  - La visibilité des objets
  - Les attributs simples

Diagramme de classes de conception du S.I. d'une bibliothèque



Récapulatif



Plan

- Conception Orientée Objet (COO)
- Étape 1: Diagramme de séquence de conception
- Étape 2: Diagramme de classes de conception
- Passage de la COO à la Programmation Orientée Objet

Passage de la conception au Code

- Ceci n'est pas un cours de programmation mais la production de code reste cependant la finalité principale de notre travail précédent.

- Le code dans un langage de programmation orienté-objet peut être généré à partir des livrables de la phase de conception tels que :
  - Les diagrammes de séquence de conception.
  - Les diagrammes de classes de conception.

Passage de la COO à la POO

- les diagrammes de classes permettent de décrire le squelette du code, à savoir toutes les déclarations.

- les diagrammes de séquence de conception permettent d'écrire le corps des méthodes, en particulier la séquence d'appels de méthodes sur les objets qui interagissent.

- Commencer avec la classe la moins connectée (une qui ne soit pas reliée aux autres classes) et continuer en direction des classes les plus connectées.

- Implémenter et tester chaque classe de façon approfondie avant de poursuivre.

Approche du passage de la COO à la POO

- la classe UML devient une classe Java ou C# ;
- les attributs UML deviennent des variables d'instances Java ou C# ;
- les méthodes qui permettent l'accès en lecture (get) et en écriture (set) aux attributs, pour respecter le principe d'encapsulation, sont implicites (en C#, on utilisera la notion de property) ;

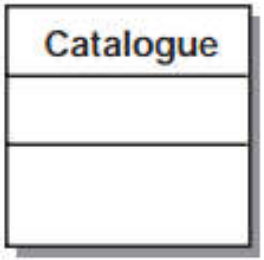
Approche du passage de la COO à la POO (suite)

- les opérations UML deviennent des méthodes Java ou C# ;
- les rôles navigables produisent des variables d'instances, tout comme les attributs, mais avec un type utilisateur au lieu d'un type simple ;
- le constructeur par défaut est implicite.

Correspondances UML – Java – C#

Classe

UML	Java

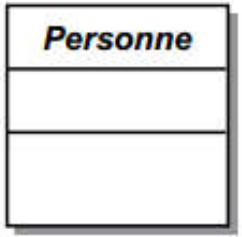
	`public class Catalogue {`
	`  ...`
	`}`

	C#

	`public class Catalogue {`
	`  ...`
	`}`

Classe abstraite

UML	Java

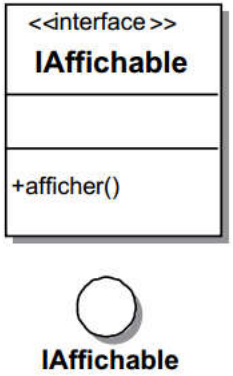
	`public abstract class Personne {`
	`  ...`
	`}`

	C#

	`public abstract class Personne {`
	`  ...`
	`}`

Interface

UML	Java

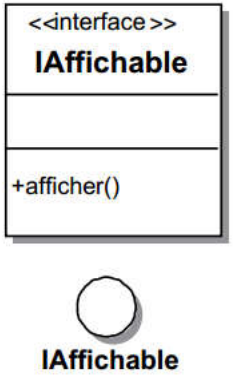
	`public interface IAffichable {`
	`  public void afficher();`
	`}`

	C#

	`public interface IAffichable {`
	`  void Afficher();`
	`}`

Package

UML	Java

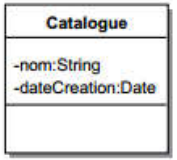
	`public interface IAffichable {`
	`  public void afficher();`
	`}`

	C#

	`public interface IAffichable {`
	`  void Afficher();`
	`}`

Attribut

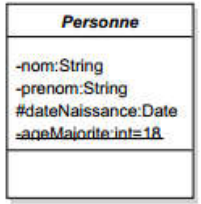
UML	Java

	`import java.util.Date;`
	
	`public class Catalogue {`
	`  private String nom;`
	`  private Date dateCreation;`
	`  …`
	`}`

	C#

	`using System;`
	
	`public class Catalogue {`
	`  private string nom;`
	`  private DateTime dateCreation;`
	`  …`
	`}`

UML	Java

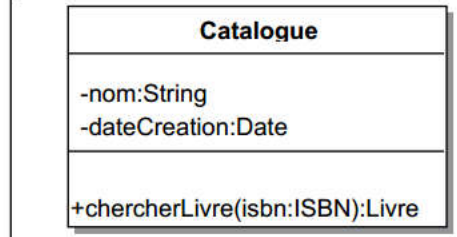
	`abstract public class Personne {`
	`  private String nom;`
	`  private String prenom;`
	`  protected Date dateNaissance;`
	`  private static int ageMajorite = 18;`
	`}`

	C#

	`abstract public class Personne {`
	`  private string nom;`
	`  private string prenom;`
	`  protected DateTime dateNaissance;`
	`  private static int ageMajorite = 18;`
	`}`

Opération (1/2)

UML



Java

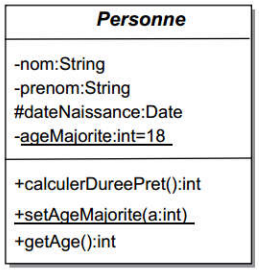
```
public class Catalogue {
    private String nom;
    private Date dateCreation;
    public Livre chercherLivre(ISBN isbn) {
        …
    }
    …
}
```

C#

```
public class Catalogue {
    private string nom;
    private DateTime dateCreation;
    public Livre ChercherLivre(ISBN isbn) {
        …
    }
    …
}
```

Opération (2/2)

UML



Java

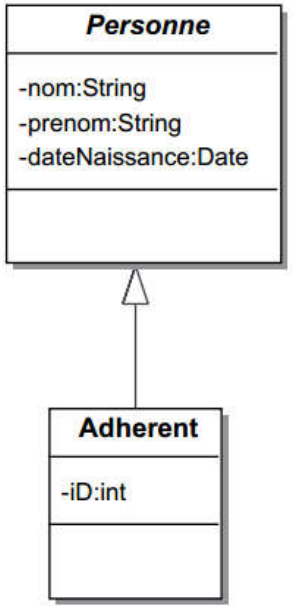
```
abstract public class Personne {
    private String nom;
    private String prenom;
    protected Date dateNaissance;
    private static int ageMajorite = 18;
    public abstract int calculerDureePret();
    public static void setAgeMajorite(int aMaj) {
    …
    }
    public int getAge() {
    …
    }
```

C#

```
abstract public class Personne {
    private string nom;
    private string prenom;
    protected DateTime dateNaissance;
    private static int ageMajorite = 18;
    public abstract int CalculerDureePret();
    public static void SetAgeMajorite(int aMaj) {
    …
    }
    public int GetAge() {
    …
    }
}
```

Généralisation

UML	Java

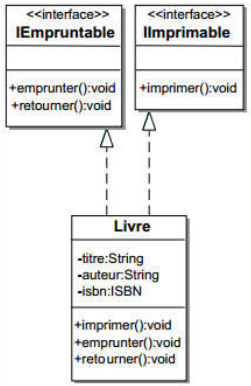
	`public class Adherent extends Personne {`
	`  private int iD;`
	`}`

	C#

	`public class Adherent : Personne {`
	`  private int iD;`
	`}`

Réalisation

UML



Java

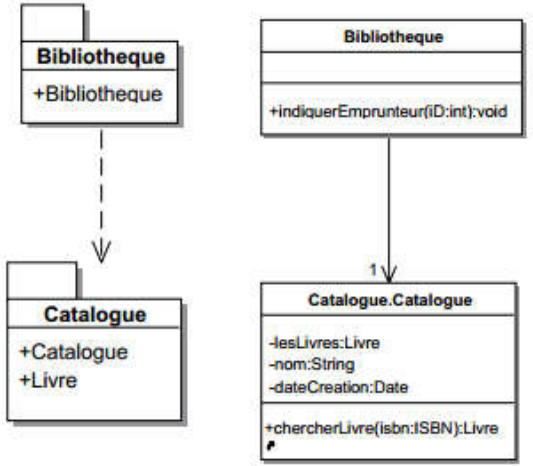
```
public class Livre implements
 IImprimable, IEmpruntable {
    private String titre;
    private String auteur;
    private ISBN isbn;
    public void imprimer(){
    …
 }
    public void emprunter(){
    …
 }
    public void retourner(){
    …
 }
}
```

C#

```
public class Livre :
 IImprimable, IEmpruntable {
    private string titre;
    private string auteur;
    private ISBN isbn;
    public void Imprimer(){
    …
 }
    public void Emprunter(){
    …
 }
    public void Retourner(){
    …
 }
}
```

Dépendance

UML



Java

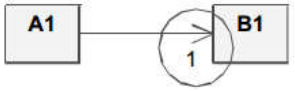
```
package bibliotheque;
import catalogue;
public class Bibliotheque {
    private Catalogue leCatalogue;
    …
    }
}
```

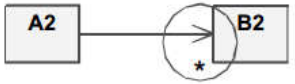
C#

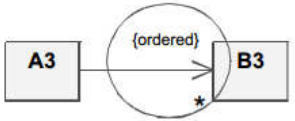
```
namespace Bibliotheque
{
  using Catalogue;
  public class Bibliotheque {
    private Catalogue leCatalogue;
    …
    }
  }
}
```

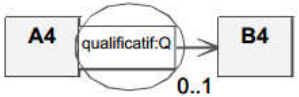
Association

UML	Java	C#

	public class A1 {
	  private B1 leB1;
	  …
	}
		public class A1 {
		  private B1 leB1;
		  …
		}

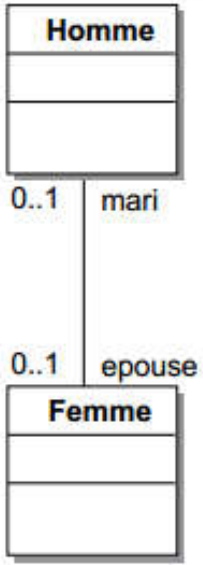
	public class A2 {
	  private B2 lesB2[];
	  …
	}
		public class A2 {
		  private B2[] lesB2;
		  …
		}

	public class A3 {
	  private List<B3> lesB3
	  = new ArrayList<B3>();
	  …
	}
		public class A3 {
		  private IList lesB3
		  = new List<B3>();
		  …
		}

	public class A4 {
	  private Map<Q,B4> lesB4
	  = new HashMap<Q,B4>();
	  …
	}
		public class A4 {
		  private IDictionary
		  <Q,B4> lesB4
		  = new Dictionary
		  <Q,B4>();
		  …
		}

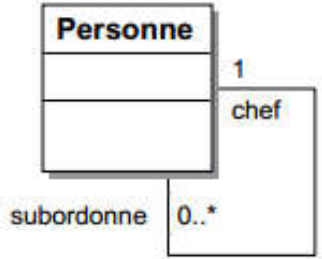
Association bidirectionnelle

UML	Java	C#

	`public class Homme {`
	`  private Femme epouse;`
	`  …`
	`}`
	
	`public class Femme {`
	`  private Homme mari;`
	`  …`
	`}`
		`public class Homme {`
		`  private Femme epouse;`
		`  …`
		`}`
		
		`public class Femme {`
		`  private Homme mari;`
		`  …`
		`}`

Association reflexive

UML



Java

```
public class Personne {
    private Personne subordonne[];
    private Personne chef ;
    …
}
```

C#

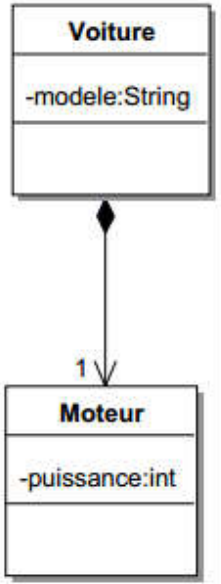
```
public class Personne {
    private Personne[] subordonne;
    private Personne chef ;
    …
}
```

Aggrégation et composition

Une composition est une agrégation plus forte impliquant que :

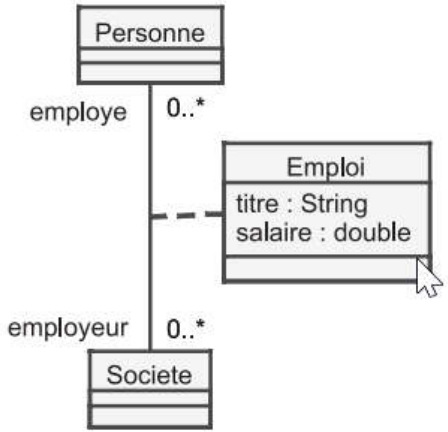
- une partie ne peut appartenir qu'à un seul composite (agrégation non partagée) ;
- la destruction du composite entraîne la destruction de toutes ses parties (le composite est responsable du cycle de vie des parties).

UML	Java	C#

	public class Voiture {
	  private String modele;
	  private Moteur moteur;
	  private static class Moteur {
	  private int puissance;
	  }
	  …
	}
		public class Voiture {
		  private string modele;
		  private Moteur moteur;
		  private class Moteur {
		  private int puissance;
		  }
		  …
		}

Classe d'association

UML



Java

```
public class Emploi {
    private String titre;
    private double salaire;
    private Personne employe;
    private Societe employeur ;
    …
}
```

C#

```
public class Emploi {
    private string titre;
    private double salaire;
    private Personne employe;
    private Societe employeur ;
    …
}
```

Ressources

1. StarUML (http://staruml.io/)


END