Génie Logiciel

Diagramme de classes UML

Idir AIT SADOUNE
http://wdi.supelec.fr/aitsadoune/
idir.aitsadoune@centralesupelec.fr

Campus de Paris-Saclay - Gif sur Yvette - France

CentraleSupélec

1A CentraleSupelec - Cursus Ingénieurs Supelec - Séquence 4
Année universitaire 2017-2018

Plan du cours

1. La modélisation en informatique
2. Unified Modeling Language (UML)
3. Diagramme de classes UML
4. Les associations dans un diagramme de classes
5. La Généralisation/ Spécialisation en UML
6. Les Classes abstraites et les Interfaces en UML
7. UML vers Java

Plan de la présentation

1. La modélisation en informatique

2. Unified Modeling Language (UML)

3. Diagramme de classes UML

4. Les associations dans un diagramme de classes

5. La Généralisation/ Spécialisation en UML

6. Les Classes abstraites et les Interfaces en UML

7. UML vers Java

Qu'est ce qu'un modèle ?

Qu'est ce qu'un modèle ?

- Un **modèle** est une *représentation/simplification/abstraction* de la réalité

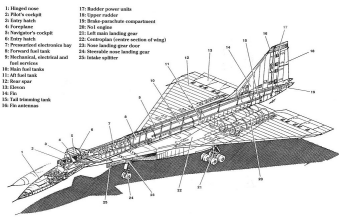
Qu'est ce qu'un modèle ?

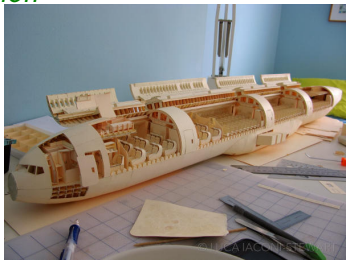
- Un **modèle** est une *représentation/simplification/abstraction* de la réalité
  - mieux comprendre le sujet (le problème) étudié

Qu'est ce qu'un modèle ?

- Un **modèle** est une *représentation/simplification/abstraction* de la réalité
  - mieux comprendre le sujet (le problème) étudié

Différents modèles d'un avion





Qu'est ce qu'un modèle en informatique ?

- Un **modèle** est une *représentation simplifiée d'un système*, établie dans un certain objectif. Il doit permettre de répondre à des questions que l'on se pose sur le système.

Qu'est ce qu'un modèle en informatique ?

- Un **modèle** est une *représentation simplifiée d'un système*, établie dans un certain objectif. Il doit permettre de répondre à des questions que l'on se pose sur le système.
  - mieux comprendre les systèmes complexes.

Qu'est ce qu'un modèle en informatique ?

- Un **modèle** est une *représentation simplifiée d'un système*, établie dans un certain objectif. Il doit permettre de répondre à des questions que l'on se pose sur le système.
  - mieux comprendre les systèmes complexes.
- Un **modèle** a pour objectif de *structurer les informations* (données) et les *activités* (traitements) d'un système.

Qu'est ce qu'un modèle en informatique ?

- Un **modèle** est une *représentation simplifiée d'un système*, établie dans un certain objectif. Il doit permettre de répondre à des questions que l'on se pose sur le système.
  - mieux comprendre les systèmes complexes.
- Un **modèle** a pour objectif de *structurer les informations* (données) et les *activités* (traitements) d'un système.
- L'**activité de modélisation** (Conception ou Spécification) consiste à *décrire de manière non ambiguë* le fonctionnement futur du système afin d'en faciliter la réalisation ;

Qu'est ce qu'un modèle en informatique ?

- Un **modèle** est une *représentation simplifiée d'un système*, établie dans un certain objectif. Il doit permettre de répondre à des questions que l'on se pose sur le système.
  - mieux comprendre les systèmes complexes.
- Un **modèle** a pour objectif de *structurer les informations* (données) et les *activités* (traitements) d'un système.
- L'**activité de modélisation** (Conception ou Spécification) consiste à *décrire de manière non ambiguë* le fonctionnement futur du système afin d'en faciliter la réalisation ;
  - *Spécification du système* : description des fonctionnalités du système ;

Qu'est ce qu'un modèle en informatique ?

- Un **modèle** est une *représentation simplifiée d'un système*, établie dans un certain objectif. Il doit permettre de répondre à des questions que l'on se pose sur le système.
  - mieux comprendre les systèmes complexes.
- Un **modèle** a pour objectif de *structurer les informations* (données) et les *activités* (traitements) d'un système.
- L'**activité de modélisation** (Conception ou Spécification) consiste à *décrire de manière non ambiguë* le fonctionnement futur du système afin d'en faciliter la réalisation ;
  - *Spécification du système* : description des fonctionnalités du système ;
  - *Conception de l'architecture* : description de la structure générale du système ;

Qu'est ce qu'un modèle en informatique ?

- Un **modèle** est une *représentation simplifiée d'un système*, établie dans un certain objectif. Il doit permettre de répondre à des questions que l'on se pose sur le système.
  - mieux comprendre les systèmes complexes.
- Un **modèle** a pour objectif de *structurer les informations* (données) et les *activités* (traitements) d'un système.
- L'**activité de modélisation** (Conception ou Spécification) consiste à *décrire de manière non ambiguë* le fonctionnement futur du système afin d'en faciliter la réalisation ;
  - *Spécification du système* : description des fonctionnalités du système ;
  - *Conception de l'architecture* : description de la structure générale du système ;
  - *Conception détaillée* : description des composants, des algorithmes et des structures de données ;

Les méthodes de conception

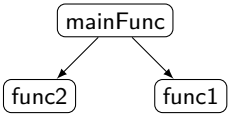
- **Une méthode de conception** est une démarche qui a pour objectif la *formalisation des différentes étapes du développement* d'un système afin de rendre ce développement plus fidèle aux besoins du client.

Les méthodes de conception

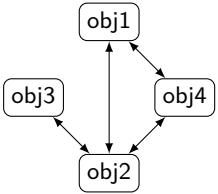
- **Une méthode de conception** est une démarche qui a pour objectif la *formalisation des différentes étapes du développement* d'un système afin de rendre ce développement plus fidèle aux besoins du client.
- **Une méthode de conception** fournit une *méthodologie* et des *notations standards* qui aident à concevoir des logiciels de qualité.

Les méthodes de conception

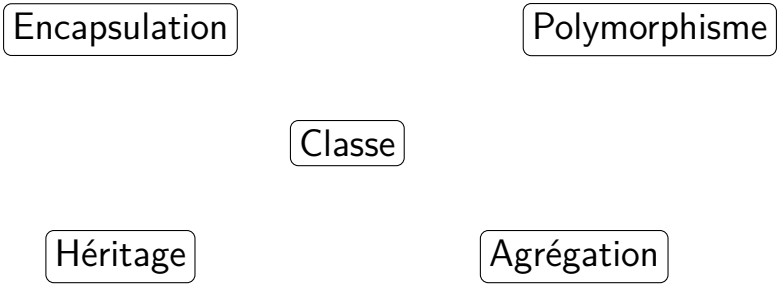
- Il existe différentes manières pour classer les méthodes de conception. Nous citons les approches **fonctionnelle** et **orientée objet** :
  - **la conception fonctionnelle (procédurale)** : un système est vu comme un ensemble hiérarchique de fonctions.



  - **la conception orientée objet** : un système est un ensemble d'objets en interaction.



Concepts importants de l'approche objet



Concepts importants de l'approche objet

Encapsulation

Polymorphisme

Classe

Héritage

Agrégation

UML est une méthode de conception orientée objet

Plan de la présentation

1. La modélisation en informatique
2. Unified Modeling Language (UML)
3. Diagramme de classes UML
4. Les associations dans un diagramme de classes
5. La Généralisation/ Spécialisation en UML
6. Les Classes abstraites et les Interfaces en UML
7. UML vers Java

Pourquoi UML ?

Historique

Pourquoi UML ?

Historique

- *1980 - 1994* : Apparition des langages orientés objets puis de nombreuses méthodes permettant de modéliser la conception.

Pourquoi UML ?

Historique

- *1980 - 1994* : Apparition des langages orientés objets puis de nombreuses méthodes permettant de modéliser la conception.
- *1995* : Proposition d'une méthode commune : Unified Method (UM).

Pourquoi UML ?

Historique

- *1980 - 1994* : Apparition des langages orientés objets puis de nombreuses méthodes permettant de modéliser la conception.
- *1995* : Proposition d'une méthode commune : Unified Method (UM).
- *1996* : La méthode est significativement améliorée et est maintenant appelée Unified Modeling Language (UML).

Pourquoi UML ?

Historique

- *1980 - 1994* : Apparition des langages orientés objets puis de nombreuses méthodes permettant de modéliser la conception.
- *1995* : Proposition d'une méthode commune : Unified Method (UM).
- *1996* : La méthode est significativement améliorée et est maintenant appelée Unified Modeling Language (UML).
- *1997* : UML est adopté par l'Object Management Group (OMG) comme standard (UML 1.0).

Qui est UML ?

- **UML** (*Unified Modeling Language*) est un langage (*une notation graphique et une sémantique*) de modélisation basé sur la *notion d'objets*.

Qui est UML ?

- **UML** (*Unified Modeling Language*) est un langage (*une notation graphique et une sémantique*) de modélisation basé sur la *notion d'objets*.
- **UML** est considéré comme un outil pour :

Qui est UML ?

- **UML** (*Unified Modeling Language*) est un langage (*une notation graphique et une sémantique*) de modélisation basé sur la *notion d'objets*.
- **UML** est considéré comme un outil pour :
  - présenter l'analyse d'un système ou les pré-requis le concernant ;

Qui est UML ?

- **UML** (*Unified Modeling Language*) est un langage (*une notation graphique et une sémantique*) de modélisation basé sur la *notion d'objets*.
- **UML** est considéré comme un outil pour :
  - présenter l'analyse d'un système ou les pré-requis le concernant ;
  - spécifier et concevoir des systèmes ;

Qui est UML ?

- **UML** (*Unified Modeling Language*) est un langage (*une notation graphique et une sémantique*) de modélisation basé sur la *notion d'objets*.
- **UML** est considéré comme un outil pour :
  - présenter l'analyse d'un système ou les pré-requis le concernant ;
  - spécifier et concevoir des systèmes ;
  - communiquer sur des processus logiciels ou d'entreprise ;

Qui est UML ?

- **UML** (*Unified Modeling Language*) est un langage (*une notation graphique et une sémantique*) de modélisation basé sur la *notion d'objets*.
- **UML** est considéré comme un outil pour :
  - présenter l'analyse d'un système ou les pré-requis le concernant ;
  - spécifier et concevoir des systèmes ;
  - communiquer sur des processus logiciels ou d'entreprise ;
  - documenter un système, un processus ou une organisation existants.

Qui est UML ?

- **UML** (*Unified Modeling Language*) est un langage (*une notation graphique et une sémantique*) de modélisation basé sur la *notion d'objets*.
- **UML** est considéré comme un outil pour :
  - présenter l'analyse d'un système ou les pré-requis le concernant ;
  - spécifier et concevoir des systèmes ;
  - communiquer sur des processus logiciels ou d'entreprise ;
  - documenter un système, un processus ou une organisation existants.
- **UML** se présente sous forme d'un *ensemble de diagrammes* qui peuvent être utilisés à différents niveaux d'un cycle de vie d'un logiciel.

Les diagrammes d'UML (1/2)

Diagrammes de structure ou diagrammes statiques

- Diagramme de classes (class diagram)
- Diagramme d'objets (object diagram)
- Diagramme de composants (component diagram)
- Diagramme de déploiement (deployment diagram)
- Diagramme des paquets (package diagram)
- Diagramme de structure composite (composite structure diagram)

Diagrammes de comportement

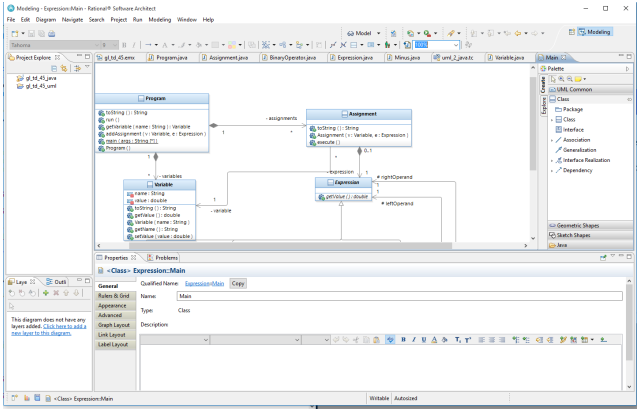
- Diagramme des cas d'utilisation (use-case diagram)
- Diagramme états-transitions (state machine diagram)
- Diagramme d'activité (activity diagram)

Les diagrammes d'UML (2/2)

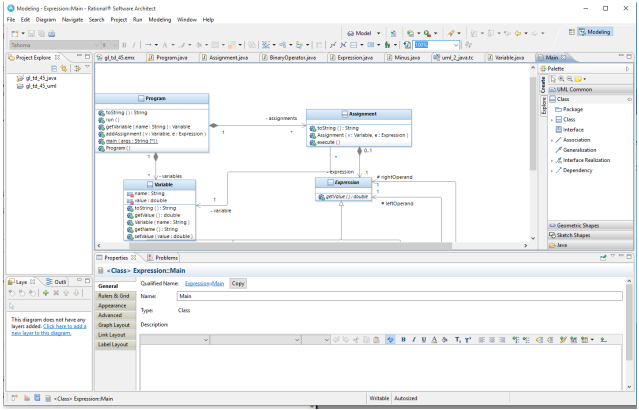
Diagrammes d'interaction ou diagrammes dynamiques

- Diagramme de séquence (sequence diagram)
- Diagramme de communication (collaboration) (communication diagram)
- Diagramme global d'interaction (interaction overview diagram)
- Diagramme de temps (timing diagram)

Outil : Rational Software Architect (RSA)

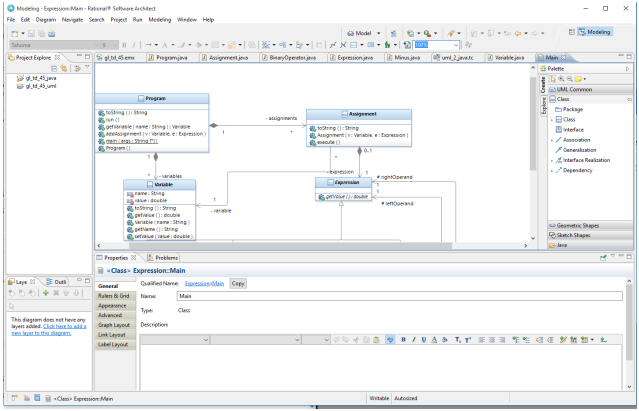


Outil : Rational Software Architect (RSA)



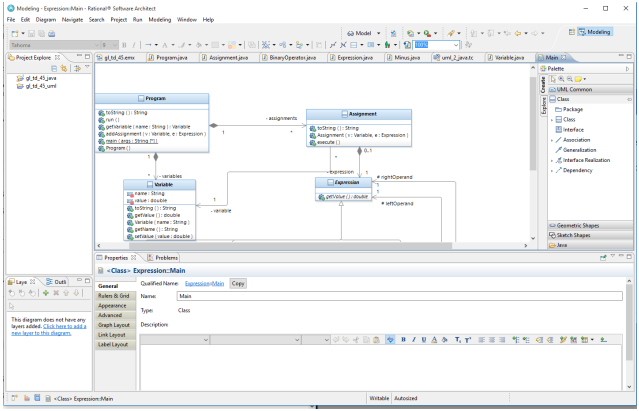
- **RSA** a été développé par **IBM** et il est basé sur l'environnement *Éclipse*.

Outil : Rational Software Architect (RSA)



- **RSA** a été développé par **IBM** et il est basé sur l'environnement *Éclipse*.
- Un outil complet de *conception*, de *modélisation* et de *développement* pour la distribution de bout en bout de logiciels.

Outil : Rational Software Architect (RSA)



- **RSA** a été développé par **IBM** et il est basé sur l'environnement *Éclipse*.
- Un outil complet de *conception*, de *modélisation* et de *développement* pour la distribution de bout en bout de logiciels. Il fait appel à **UML** pour la conception d'applications **Java**.

Outil : Rational Software Architect (RSA)

Les commandes principales de RSA

Outil : Rational Software Architect (RSA)

Les commandes principales de RSA

- création d'un nouveau projet **UML** (contient un modèle vide) :
  File > New > Project... > Modeling > UML Project > (Project Name) Next > (General:BlanckPackage) Finish

Outil : Rational Software Architect (RSA)

Les commandes principales de RSA

- création d'un nouveau projet **UML** (contient un modèle vide) :
  File > *New* > *Project...* > *Modeling* > *UML Project* > *(Project Name) Next* > *(General:BlanckPackage) Finish*
- création d'un nouveau modèle :
  (sélectionner le dossier models avec un clique droit)
  Create model > *(Standard template) Next* > *(General:BlanckPackage) Finish*

Outil : Rational Software Architect (RSA)

Les commandes principales de RSA

- création d'un nouveau projet **UML** (contient un modèle vide) :
  File > *New* > *Project...* > *Modeling* > *UML Project* > *(Project Name) Next* > *(General:BlanckPackage) Finish*
- création d'un nouveau modèle :
  (sélectionner le dossier models avec un clique droit)
  Create model > *(Standard template) Next* > *(General:BlanckPackage) Finish*
- création d'un nouveau diagramme **UML** :
  (sélectionner le modèle avec un clique droit)
  Add Diagram > *Class Diagram*

Plan de la présentation

1. La modélisation en informatique

2. Unified Modeling Language (UML)

3. Diagramme de classes UML

4. Les associations dans un diagramme de classes

5. La Généralisation/ Spécialisation en UML

6. Les Classes abstraites et les Interfaces en UML

7. UML vers Java

C'est quoi un diagramme de classes

Définition

- *Diagramme de classe* : décrit *les entités* (classes et interfaces) constituant le système à modéliser et *les relations statiques* entre celles-ci.

Diagramme de classes UML

Définition de la Classe en UML

Classe et Objet : définition

Définition de la Classe en UML

Classe et Objet : définition

- Une *classe* représente un *groupe d'objets* possédant des états et un comportement communs.

Diagramme de classes UML

Définition de la Classe en UML

Classe et Objet : définition

- Une *classe* représente un *groupe d'objets* possédant des états et un comportement communs.
- Une *classe* peut représenter un *concept concret* (voiture, facture ...) ou un *concept abstrait* (stratégie ...).

Définition de la Classe en UML

Classe et Objet : définition

- Une *classe* représente un *groupe d'objets* possédant des états et un comportement communs.
- Une *classe* peut représenter un *concept concret* (voiture, facture ...) ou un *concept abstrait* (stratégie ...).
- En **UML**, une *classe* représente un type de *classificateur*.

Définition de la Classe en UML

Classe et Objet : définition

- Une *classe* représente un *groupe d'objets* possédant des états et un comportement communs.
- Une *classe* peut représenter un *concept concret* (voiture, facture ...) ou un *concept abstrait* (stratégie ...).
- En **UML**, une *classe* représente un type de *classificateur*.

Exemple

Définition de la Classe en UML

Classe et Objet : définition

- Une *classe* représente un *groupe d'objets* possédant des états et un comportement communs.
- Une *classe* peut représenter un *concept concret* (voiture, facture ...) ou un *concept abstrait* (stratégie ...).
- En **UML**, une *classe* représente un type de *classificateur*.

Exemple

- *Une classe* : Automobile

Définition de la Classe en UML

Classe et Objet : définition

- Une *classe* représente un *groupe d'objets* possédant des états et un comportement communs.
- Une *classe* peut représenter un *concept concret* (voiture, facture ...) ou un *concept abstrait* (stratégie ...).
- En **UML**, une *classe* représente un type de *classificateur*.

Exemple

- *Une classe* : Automobile
- *Des objets* : Renault, Peugeot, Volkswagen, ...

Définition de la Classe en UML

En **UML**, une classe est définie par un cadre rectangulaire comportant 3 zones :

Définition de la Classe en UML

En **UML**, une classe est définie par un cadre rectangulaire comportant 3 zones :

1. Zone contenant le *nom de la classe*

Définition de la Classe en UML

En **UML**, une classe est définie par un cadre rectangulaire comportant 3 zones :

1. Zone contenant le *nom de la classe*
2. Zone contenant les *attributs de la classe*

Définition de la Classe en UML

En **UML**, une classe est définie par un cadre rectangulaire comportant 3 zones :

1. Zone contenant le *nom de la classe*
2. Zone contenant les *attributs de la classe*
3. Zone contenant les *opérations de la classe*

Définition de la Classe en UML

En **UML**, une classe est définie par un cadre rectangulaire comportant 3 zones :

1. Zone contenant le *nom de la classe*
2. Zone contenant les *attributs de la classe*
3. Zone contenant les *opérations de la classe*

Représentation d'une classe en UML

Définition de la Classe en UML

En **UML**, une classe est définie par un cadre rectangulaire comportant 3 zones :

1. Zone contenant le *nom de la classe*
2. Zone contenant les *attributs de la classe*
3. Zone contenant les *opérations de la classe*

Représentation d'une classe en UML

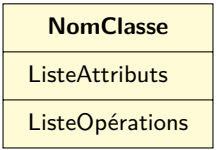


Diagramme de classes UML

Les éléments de la Classe en UML

Les attributs d'une classe UML

Les éléments de la Classe en UML

Les attributs d'une classe UML

- Les *caractéristiques* (propriétés) d'une classe **UML** sont représentées sous la forme d'*attributs*.

Les éléments de la Classe en UML

Les attributs d'une classe UML

- Les *caractéristiques* (propriétés) d'une classe **UML** sont représentées sous la forme d'*attributs*.
- Un *attribut* peut être représenté au moyen de deux notations différentes :

Les éléments de la Classe en UML

Les attributs d'une classe UML

- Les *caractéristiques* (propriétés) d'une classe **UML** sont représentées sous la forme d'*attributs*.
- Un *attribut* peut être représenté au moyen de deux notations différentes :
  1. *en ligne* pour des données primitives simples (entiers, réels, ...),

Les éléments de la Classe en UML

Les attributs d'une classe UML

- Les *caractéristiques* (propriétés) d'une classe **UML** sont représentées sous la forme d'*attributs*.
- Un *attribut* peut être représenté au moyen de deux notations différentes :
  1. *en ligne* pour des données primitives simples (entiers, réels, ...),
  2. *relation* pour exprimer des liens avec d'autres classes (*voir la section suivante*).

Diagramme de classes UML

Les éléments de la Classe en UML

Les attributs en ligne

Les éléments de la Classe en UML

Les attributs en ligne

- La forme générale d'un *attribut en ligne* dans une classe :

Les éléments de la Classe en UML

Les attributs en ligne

- La forme générale d'un *attribut en ligne* dans une classe :
  visibilité / nom : **type** *multiplicité* = *valeur-initiale* {**propriétés**}

Les éléments de la Classe en UML

Les attributs en ligne

- La forme générale d'un *attribut en ligne* dans une classe :
  visibilité / nom : **type** *multiplicité* = *valeur-initiale* {**propriétés**}

Visibilité (par défaut public)

Les éléments de la Classe en UML

Les attributs en ligne

- La forme générale d'un *attribut en ligne* dans une classe :
  visibilité / nom : **type** *multiplicité* = *valeur-initiale* {**propriétés**}

Visibilité (par défaut public)

+(*public*), −(*private*), #(*protected*), ~(*package*)

Les éléments de la Classe en UML

Les attributs en ligne

- La forme générale d'un *attribut en ligne* dans une classe :
  visibilité / nom : **type** *multiplicité* = *valeur-initiale* {**propriétés**}

Visibilité (par défaut public)

+(*public*), −(*private*), #(*protected*), ~(*package*)

Les attributs dérivés "/"

C'est un attribut qui peut être calculé à partir d'autres attributs de la classe

Les éléments de la Classe en UML

Les attributs en ligne

- La forme générale d'un *attribut en ligne* dans une classe :
  visibilité / nom : **type** *multiplicité* = *valeur-initiale* {**propriétés**}

Visibilité (par défaut public)

+(*public*), −(*private*), #(*protected*), ~(*package*)

Les attributs dérivés "/"

C'est un attribut qui peut être calculé à partir d'autres attributs de la classe

Multiplicité (par défaut 1)

Les éléments de la Classe en UML

Les attributs en ligne

- La forme générale d'un *attribut en ligne* dans une classe :
  visibilité / nom : **type** *multiplicité* = *valeur-initiale* {**propriétés**}

Visibilité (par défaut public)

+(*public*), −(*private*), #(*protected*), ~(*package*)

Les attributs dérivés "/"

C'est un attribut qui peut être calculé à partir d'autres attributs de la classe

Multiplicité (par défaut 1)

- [*val*] : il y a *val* fois cet attribut

Les éléments de la Classe en UML

Les attributs en ligne

- La forme générale d'un *attribut en ligne* dans une classe :
  visibilité / nom : **type** *multiplicité* = *valeur-initiale* {**propriétés**}

Visibilité (par défaut public)

+(*public*), −(*private*), #(*protected*), ~(*package*)

Les attributs dérivés "/"

C'est un attribut qui peut être calculé à partir d'autres attributs de la classe

Multiplicité (par défaut 1)

- [*val*] : il y a *val* fois cet attribut
- [*min*..*max*] : il y a entre *min* et *max* fois cet attribut

Les éléments de la Classe en UML

Les attributs en ligne

- La forme générale d'un *attribut en ligne* dans une classe :
  visibilité / nom : **type** *multiplicité* = *valeur-initiale* {**propriétés**}

Visibilité (par défaut public)

+(*public*), −(*private*), #(*protected*), ~(*package*)

Les attributs dérivés "/"

C'est un attribut qui peut être calculé à partir d'autres attributs de la classe

Multiplicité (par défaut 1)

- [*val*] : il y a *val* fois cet attribut
- [*min*..*max*] : il y a entre *min* et *max* fois cet attribut
- [*min*..*] : il y a au moins *min* fois cet attribut

Les éléments de la Classe en UML

Les attributs en ligne

- La forme générale d'un *attribut en ligne* dans une classe :
  visibilité / nom : **type** *multiplicité* = *valeur-initiale* {**propriétés**}

Visibilité (par défaut public)

+(*public*), −(*private*), #(*protected*), ~(*package*)

Les attributs dérivés "/"

C'est un attribut qui peut être calculé à partir d'autres attributs de la classe

Multiplicité (par défaut 1)

- [*val*] : il y a *val* fois cet attribut
- [*min*..*max*] : il y a entre *min* et *max* fois cet attribut
- [*min*..*] : il y a au moins *min* fois cet attribut
- [*] : il y a un nombre indéterminé d'occurrence de cet attribut

Diagramme de classes UML

Les éléments de la Classe en UML

Propriétés

Les éléments de la Classe en UML

Propriétés

- **readOnly** : valeur constante

Les éléments de la Classe en UML

Propriétés

- **readOnly** : valeur constante
- **static** : attribut partagé entre tous les objets (souligné dans le diagramme **UML**)

Les éléments de la Classe en UML

Propriétés

- **readOnly** : valeur constante
- **static** : attribut partagé entre tous les objets (souligné dans le diagramme **UML**)
- **unique** : si multiplicité > 1, valeurs distinctes

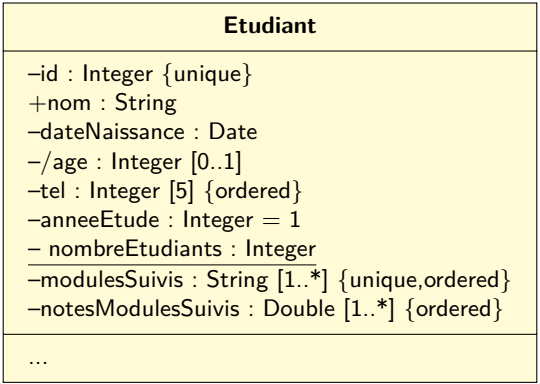
Les éléments de la Classe en UML

Propriétés

- **readOnly** : valeur constante
- **static** : attribut partagé entre tous les objets (souligné dans le diagramme **UML**)
- **unique** : si multiplicité > 1, valeurs distinctes
- **ordered** : si multiplicité > 1, valeurs ordonnées

Les éléments de la Classe en UML

Exemple



Les éléments de la Classe en UML

Les opérations d'une classe UML

Les éléments de la Classe en UML

Les opérations d'une classe UML

- Une *opération* permet d'invoquer une *fonctionnalité* offerte par une *classe*.

Les éléments de la Classe en UML

Les opérations d'une classe UML

- Une *opération* permet d'invoquer une *fonctionnalité* offerte par une *classe*.
- La forme générale d'une *opération* dans une classe :

Les éléments de la Classe en UML

Les opérations d'une classe UML

- Une *opération* permet d'invoquer une *fonctionnalité* offerte par une *classe*.
- La forme générale d'une *opération* dans une classe :
  visibilité nom (*paramètres*) : **type-retourné** {**propriétés**}

Les éléments de la Classe en UML

Les opérations d'une classe UML

- Une *opération* permet d'invoquer une *fonctionnalité* offerte par une *classe*.
- La forme générale d'une *opération* dans une classe :
  visibilité nom (*paramètres*) : **type-retourné** {**propriétés**}
- Un paramètre d'une opération est de la forme :

Les éléments de la Classe en UML

Les opérations d'une classe UML

- Une *opération* permet d'invoquer une *fonctionnalité* offerte par une *classe*.
- La forme générale d'une *opération* dans une classe :
  visibilité nom (*paramètres*) : **type-retourné** {**propriétés**}
- Un paramètre d'une opération est de la forme :
  direction nom : **type** *multiplicité* = *valeur-initiale* {**propriétés**}

Les éléments de la Classe en UML

Les opérations d'une classe UML

- Une *opération* permet d'invoquer une *fonctionnalité* offerte par une *classe*.
- La forme générale d'une *opération* dans une classe :

  visibilité nom (*paramètres*) : **type-retourné** {**propriétés**}
- Un paramètre d'une opération est de la forme :

  direction nom : **type** *multiplicité* = *valeur-initiale* {**propriétés**}
  - les directions sont **in** (entrée seule sans modification), **out** (sortie seule) et **inout** (entrée et sortie).

Les éléments de la Classe en UML

Les opérations d'une classe UML

- Une *opération* permet d'invoquer une *fonctionnalité* offerte par une *classe*.
- La forme générale d'une *opération* dans une classe :

  visibilité nom (*paramètres*) : **type-retourné** {**propriétés**}
- Un paramètre d'une opération est de la forme :

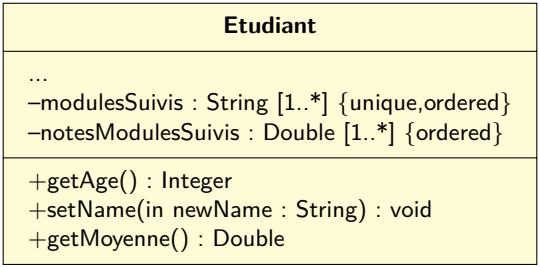
  direction nom : **type** *multiplicité* = *valeur-initiale* {**propriétés**}
  - les directions sont **in** (entrée seule sans modification), **out** (sortie seule) et **inout** (entrée et sortie).

Remarque

La **visibilité** et les **propriétés** ont les mêmes définitions et les mêmes utilisations que dans le cas des attributs.

Les éléments de la Classe en UML

Exemple



Plan de la présentation

1. La modélisation en informatique
2. Unified Modeling Language (UML)
3. Diagramme de classes UML
4. Les associations dans un diagramme de classes
5. La Généralisation/ Spécialisation en UML
6. Les Classes abstraites et les Interfaces en UML
7. UML vers Java

Les attributs d'une classe UML

- Les *caractéristiques* (propriétés) d'une classe **UML** sont représentées sous la forme d'*attributs*.
- Un *attribut* peut être représenté au moyen de deux notations différentes :
  1. *en ligne* pour des données primitives simples (entiers, réels, ...),
  2. *relation* pour exprimer des liens avec d'autres classes (*voir la section suivante*).

Les relations d'association

Les relations d'association

- Un système est constitué d'*un ensemble de classes* (d'objets) qui *interagissent* entre elles (des classes avec des relations entre elles).

Les relations d'association

- Un système est constitué d'*un ensemble de classes* (d'objets) qui *interagissent* entre elles (des classes avec des relations entre elles).
- Une relation d'*association* entre deux classes peut être interprétée par un *"... a un ..."*, *"... est propriétaire de ..."* ou *"... est composé de ..."*.

Les relations d'association

- Un système est constitué d'*un ensemble de classes* (d'objets) qui *interagissent* entre elles (des classes avec des relations entre elles).
- Une relation d'*association* entre deux classes peut être interprétée par un *"... a un ..."*, *"... est propriétaire de ..."* ou *"... est composé de ..."*.
- Une relation d'*association* doit être *stable* (elle dure dans le temps et elle est non ponctuelle).

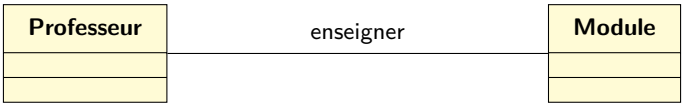
Les relations d'association

- Un système est constitué d'*un ensemble de classes* (d'objets) qui *interagissent* entre elles (des classes avec des relations entre elles).
- Une relation d'*association* entre deux classes peut être interprétée par un *"... a un ..."*, *"... est propriétaire de ..."* ou *"... est composé de ..."*.
- Une relation d'*association* doit être *stable* (elle dure dans le temps et elle est non ponctuelle).
- Il est possible de *nommer* une relation d'*association*.

Les relations d'association

- Un système est constitué d'*un ensemble de classes* (d'objets) qui *interagissent* entre elles (des classes avec des relations entre elles).
- Une relation d'*association* entre deux classes peut être interprétée par un *"... a un ..."*, *"... est propriétaire de ..."* ou *"... est composé de ..."*.
- Une relation d'*association* doit être *stable* (elle dure dans le temps et elle est non ponctuelle).
- Il est possible de *nommer* une relation d'*association*.

Exemple d'une relation d'association



Les relations d'association

La navigabilité dans une association

Les relations d'association

La navigabilité dans une association

- Une association est par défaut *bidirectionnelle*

Les relations d'association

La navigabilité dans une association

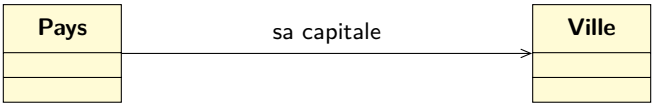
- Une association est par défaut *bidirectionnelle*
- Une association peut être *orientée* (sens de navigation)

Les relations d'association

La navigabilité dans une association

- Une association est par défaut *bidirectionnelle*
- Une association peut être *orientée* (sens de navigation)

Exemple



Les relations d'association

Les rôles dans une association

Les relations d'association

Les rôles dans une association

- En plus du nom, nous pouvons définir le *rôle* que joue chaque classe dans une *association*

Les relations d'association

Les rôles dans une association

- En plus du nom, nous pouvons définir le *rôle* que joue chaque classe dans une *association*
  - nom de l'extrémité d'une association

Les relations d'association

Les rôles dans une association

- En plus du nom, nous pouvons définir le *rôle* que joue chaque classe dans une *association*
  - nom de l'extrémité d'une association

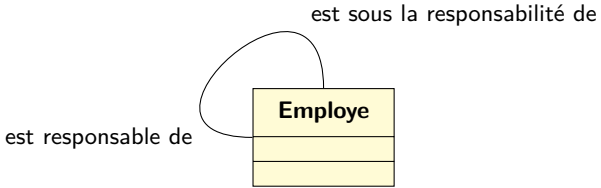
Exemple

Les relations d'association

D'autres exemples

Les relations d'association

D'autres exemples



Les relations d'association

Multiplicité dans une association

Les relations d'association

Multiplicité dans une association

- spécifie, dans une association, le *nombre d'instance* d'une classe qui sont liées à une instance d'une autre classe

Les relations d'association

Multiplicité dans une association

- spécifie, dans une association, le *nombre d'instance* d'une classe qui sont liées à une instance d'une autre classe
  - 1, 0..1, M..N, *, 0..*, 1..*

Les relations d'association

Multiplicité dans une association

- spécifie, dans une association, le *nombre d'instance* d'une classe qui sont liées à une instance d'une autre classe
  - 1, 0..1, M..N, *, 0..*, 1..*

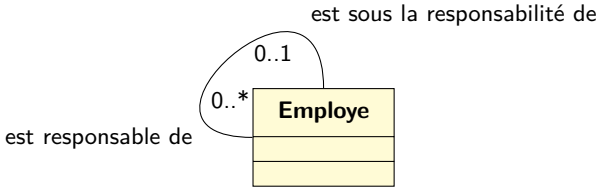
Exemple

Les relations d'association

Autres exemples

Les relations d'association

Autres exemples



Les relations d'association

Les associations n-aires

Les relations d'association

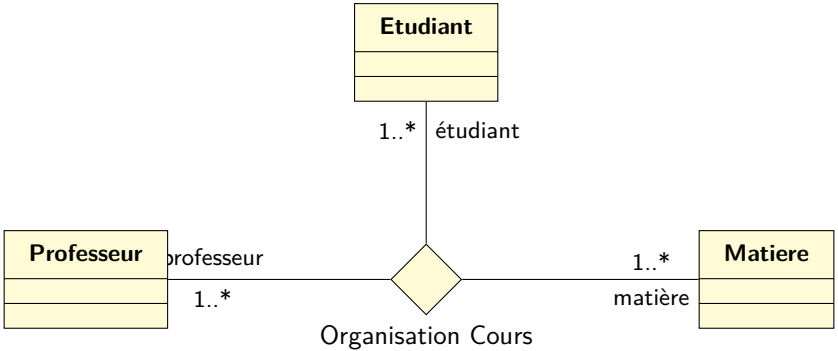
Les associations n-aires

- C'est une association qui relie *plus de deux classes*

Les relations d'association

Les associations n-aires

- C'est une association qui relie *plus de deux classes*



Les associations d'agrégation

Les associations d'agrégation

- Une *association d'agrégation* est une relation entre classes qui est *plus forte* qu'une relation d'association classique.

Les associations d'agrégation

- Une *association d'agrégation* est une relation entre classes qui est *plus forte* qu'une relation d'association classique.
- Une *association d'agrégation* peut être traduite par le verbe "*posséder*"

Les associations d'agrégation

- Une *association d'agrégation* est une relation entre classes qui est *plus forte* qu'une relation d'association classique.
- Une *association d'agrégation* peut être traduite par le verbe "*posséder*"
- Une *association d'agrégation* correspond généralement à une relation tout ou parties (*composé/composant*)

Les associations d'agrégation

- Une *association d'agrégation* est une relation entre classes qui est *plus forte* qu'une relation d'association classique.
- Une *association d'agrégation* peut être traduite par le verbe "*posséder*"
- Une *association d'agrégation* correspond généralement à une relation tout ou parties (*composé/composant*)
  - Il y a agrégation si le *composé* dépend de l'existence de ses *composants* pour avoir un sens.

Exemple d'une association d'agrégation

Les associations d'agrégation

- Une *association d'agrégation* est une relation entre classes qui est *plus forte* qu'une relation d'association classique.
- Une *association d'agrégation* peut être traduite par le verbe "*posséder*"
- Une *association d'agrégation* correspond généralement à une relation tout ou parties (*composé/composant*)
  - Il y a agrégation si le *composé* dépend de l'existence de ses *composants* pour avoir un sens.

Exemple d'une association d'agrégation

Les associations de composition

Les associations de composition

- L'*association de composition* est une association d'*agrégation* avec un *lien plus fort*

Les associations de composition

- L'*association de composition* est une association d'*agrégation* avec un *lien plus fort*
- L'*association de composition* peut être traduite par le verbe "*composer de*"

Les associations de composition

- L'*association de composition* est une association d'*agrégation* avec un *lien plus fort*
- L'*association de composition* peut être traduite par le verbe "*composer de*"
  - Si on détruit une instance du *composé*, on détruit tous *ses composants*

Les associations de composition

- L'*association de composition* est une association d'*agrégation* avec un *lien plus fort*
- L'*association de composition* peut être traduite par le verbe "*composer de*"
  - Si on détruit une instance du *composé*, on détruit tous *ses composants*

Exemple d'une association de composition

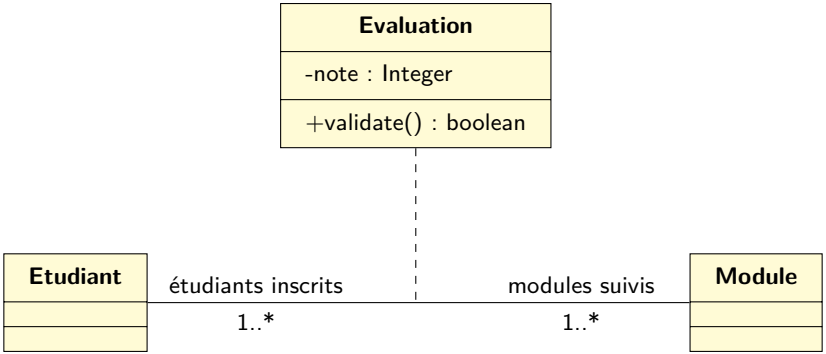
Les classes d'association

Les classes d'association

- C'est une association possédant un nom, des attributs et des méthodes

Les classes d'association

- C'est une association possédant un nom, des attributs et des méthodes



Plan de la présentation

1. La modélisation en informatique

2. Unified Modeling Language (UML)

3. Diagramme de classes UML

4. Les associations dans un diagramme de classes

5. La Généralisation/ Spécialisation en UML

6. Les Classes abstraites et les Interfaces en UML

7. UML vers Java

La Généralisation/ Spécialisation en UML

La Généralisation/ Spécialisation en UML

- Une relation de *généralisation* permet d'indiquer qu'une classe constitue un *cas plus général* d'une autre classe.

La Généralisation/ Spécialisation en UML

- Une relation de *généralisation* permet d'indiquer qu'une classe constitue un *cas plus général* d'une autre classe.
- Une relation de *généralisation* est utilisée pour *extraire des propriétés/opérations communes* à plusieurs classes afin de les regrouper dans une *super-classe*.

La Généralisation/ Spécialisation en UML

- Une relation de *généralisation* permet d'indiquer qu'une classe constitue un *cas plus général* d'une autre classe.
- Une relation de *généralisation* est utilisée pour *extraire des propriétés/opérations communes* à plusieurs classes afin de les regrouper dans une *super-classe*.
- A l'inverse, une relation de *spécialisation* permet de *décrire les spécificités* d'un cas particulier d'une classe dans *une sous-classe* (relation *"... est un ..."*).

La Généralisation/ Spécialisation en UML

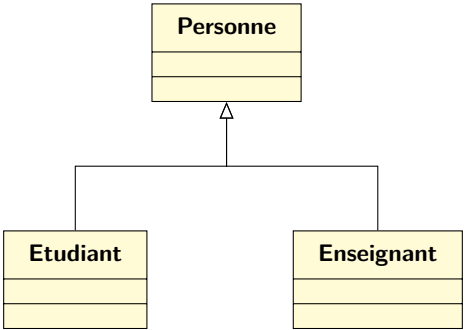
- Une relation de *généralisation* permet d'indiquer qu'une classe constitue un *cas plus général* d'une autre classe.
- Une relation de *généralisation* est utilisée pour *extraire des propriétés/opérations communes* à plusieurs classes afin de les regrouper dans une *super-classe*.
- A l'inverse, une relation de *spécialisation* permet de *décrire les spécificités* d'un cas particulier d'une classe dans *une sous-classe* (relation *"... est un ..."*).

Remarque

Une sous classe possède, en plus de ses propres propriétés/opérations, les propriétés/opérations de ses super-classes.

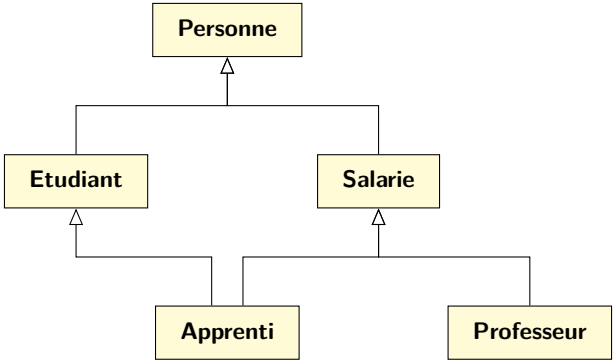
La Généralisation/ Spécialisation en UML

Exemple



La Généralisation/ Spécialisation en UML

Un autre exemple



Plan de la présentation

1. La modélisation en informatique

2. Unified Modeling Language (UML)

3. Diagramme de classes UML

4. Les associations dans un diagramme de classes

5. La Généralisation/ Spécialisation en UML

6. Les Classes abstraites et les Interfaces en UML

7. UML vers Java

Les Classes abstraites

Les Classes abstraites

- Une classe *abstraite* permet de *regrouper* des propriétés et des fonctionnalités communes à *différents types d'objets*.

Les Classes abstraites

- Une classe *abstraite* permet de *regrouper* des propriétés et des fonctionnalités communes à *différents types d'objets*.
  - Une classe *abstraite* est une classe qui *ne peut pas être instanciée*

Les Classes abstraites

- Une classe *abstraite* permet de *regrouper* des propriétés et des fonctionnalités communes à *différents types d'objets*.
  - Une classe *abstraite* est une classe qui *ne peut pas être instanciée*
- Une classe *abstraite* peut être une classe dont *toutes* les méthodes *n'ont pas été implémentées*.

Les Classes abstraites

- Une classe *abstraite* permet de *regrouper* des propriétés et des fonctionnalités communes à *différents types d'objets*.
  - Une classe *abstraite* est une classe qui *ne peut pas être instanciée*
- Une classe *abstraite* peut être une classe dont *toutes* les méthodes *n'ont pas été implémentées*.
  - Une classe possédant une *méthode abstraite* doit être déclarée comme une classe *abstraite*

Les Classes abstraites

L'héritage d'une classe abstraite

Les Classes abstraites

L'héritage d'une classe abstraite

- Une classe *qui hérite* d'une classe *abstraite* peut *implémenter* les *méthodes abstraites* de la classe mère (sauf si la classe *fille* est également une classe *abstraite*).

Les Classes abstraites

L'héritage d'une classe abstraite

- Une classe *qui hérite* d'une classe *abstraite* peut *implémenter* les *méthodes abstraites* de la classe mère (sauf si la classe *fille* est également une classe *abstraite*).
- Une classe *qui hérite* d'une classe *abstraite* peut *ré-implémenter* les méthodes déjà implémentées.

Les Classes abstraites

L'héritage d'une classe abstraite

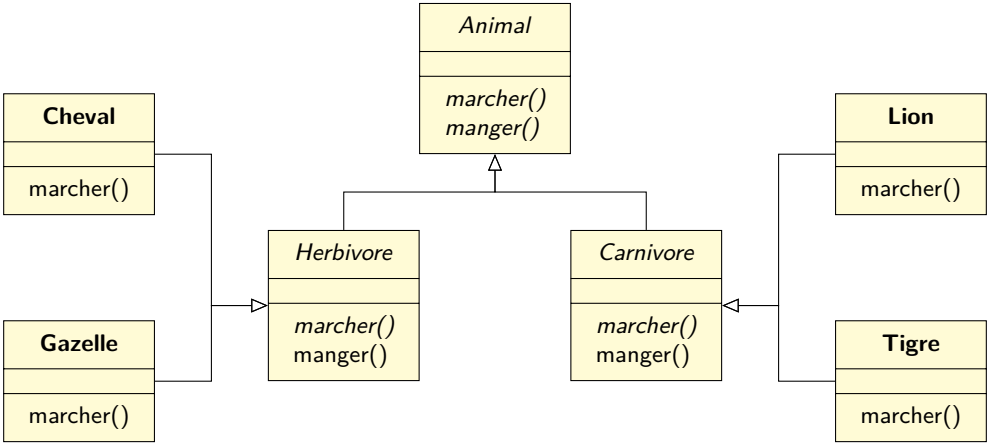
- Une classe *qui hérite* d'une classe *abstraite* peut *implémenter* les *méthodes abstraites* de la classe mère (sauf si la classe *fille* est également une classe *abstraite*).
- Une classe *qui hérite* d'une classe *abstraite* peut *ré-implémenter* les méthodes déjà implémentées.

Remarque

En **UML**, la classe/méthode abstraite est notée en *italique*

Les Classes abstraites

Exemple



Les Interfaces

Les Interfaces

- Une *interface* est un *classificateur* contenant des déclarations de *propriétés* et de *méthodes abstraites* assurant un *service cohérent*.

Les Interfaces

- Une *interface* est un *classificateur* contenant des déclarations de *propriétés* et de *méthodes abstraites* assurant un *service cohérent*.
  - Une *interface* est comme une classe *abstraite* dans laquelle *aucune méthode ne serait implémentée* (donc ne peut pas être instanciée).

Les Interfaces

- Une *interface* est un *classificateur* contenant des déclarations de *propriétés* et de *méthodes abstraites* assurant un *service cohérent*.
  - Une *interface* est comme une classe *abstraite* dans laquelle *aucune méthode ne serait implémentée* (donc ne peut pas être instanciée).
  - Une *interface* permet de définir *un ensemble de services* sans se préoccuper de leurs implémentations (méthodes abstraites).

Les Interfaces

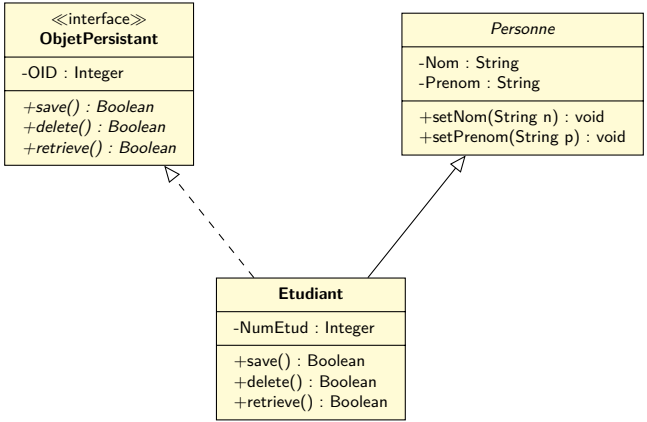
- Une *interface* est un *classificateur* contenant des déclarations de *propriétés* et de *méthodes abstraites* assurant un *service cohérent*.
  - Une *interface* est comme une classe *abstraite* dans laquelle *aucune méthode ne serait implémentée* (donc ne peut pas être instanciée).
  - Une *interface* permet de définir *un ensemble de services* sans se préoccuper de leurs implémentations (méthodes abstraites).
- Une *interface* représente un *contrat* que doit respecter chaque *classe* qui *implémente* cette *interface*.

Les Interfaces

- Une *interface* est un *classificateur* contenant des déclarations de *propriétés* et de *méthodes abstraites* assurant un *service cohérent*.
  - Une *interface* est comme une classe *abstraite* dans laquelle *aucune méthode ne serait implémentée* (donc ne peut pas être instanciée).
  - Une *interface* permet de définir *un ensemble de services* sans se préoccuper de leurs implémentations (méthodes abstraites).
- Une *interface* représente un *contrat* que doit respecter chaque *classe* qui *implémente* cette *interface*.
  - Une classe *implémentant* une interface doit obligatoirement *implémenter toutes les méthodes* déclarées dans l'interface, à moins qu'elle ne soit elle-même déclarée abstraite !

Les Interfaces

Exemple



Conclusion : Classe Abstraite vs Interface

Conclusion : Classe Abstraite vs Interface

- La notion d'*interface* est utilisée pour représenter des *propriétés transverses* de classes.

Conclusion : Classe Abstraite vs Interface

- La notion d'*interface* est utilisée pour représenter des *propriétés transverses* de classes.
- Là où une *classe abstraite* doit être étendue et *spécialisée*,

Conclusion : Classe Abstraite vs Interface

- La notion d'*interface* est utilisée pour représenter des *propriétés transverses* de classes.
- Là où une *classe abstraite* doit être étendue et *spécialisée*, une *interface* nous dit juste que telle classe *possède* telle *propriété* et *assure* tel *service*, indépendamment de ce qu'elle représente.

Plan de la présentation

1. La modélisation en informatique
2. Unified Modeling Language (UML)
3. Diagramme de classes UML
4. Les associations dans un diagramme de classes
5. La Généralisation/ Spécialisation en UML
6. Les Classes abstraites et les Interfaces en UML
7. UML vers Java

Les classes

Transcription des classes

Les classes

Transcription des classes

- A chaque *classe* **UML**, correspondra une *classe* **Java**.

Les classes

Transcription des classes

- A chaque *classe* **UML**, correspondra une *classe* **Java**.
- A chaque *attribut* d'une *classe* **UML**, correspondra un *attribut* d'une *classe* **Java**.

Les classes

Transcription des classes

- A chaque *classe* **UML**, correspondra une *classe* **Java**.
- A chaque *attribut* d'une *classe* **UML**, correspondra un *attribut* d'une *classe* **Java**.
  - cardinalité ≤ 1 : *TypeAttribut* nomAttribut ;

Les classes

Transcription des classes

- A chaque *classe* **UML**, correspondra une *classe* **Java**.
- A chaque *attribut* d'une *classe* **UML**, correspondra un *attribut* d'une *classe* **Java**.
  - cardinalité ≤ 1 : *TypeAttribut* nomAttribut ;
  - cardinalité > 1 : Collection$<$*TypeAttribut*$>$ nomAttribut ;

Les classes

Transcription des classes

- A chaque *classe* **UML**, correspondra une *classe* **Java**.
- A chaque *attribut* d'une *classe* **UML**, correspondra un *attribut* d'une *classe* **Java**.
  - cardinalité ≤ 1 : *TypeAttribut* nomAttribut ;
  - cardinalité > 1 : Collection$<$*TypeAttribut*$>$ nomAttribut ;
- A chaque *opération* d'une *classe* **UML**, correspondra une *méthode* d'une *classe* **Java**.

Les classes

Transcription des classes

- A chaque *classe* **UML**, correspondra une *classe* **Java**.
- A chaque *attribut* d'une *classe* **UML**, correspondra un *attribut* d'une *classe* **Java**.
  - cardinalité ≤ 1 : *TypeAttribut* nomAttribut ;
  - cardinalité > 1 : Collection$<$*TypeAttribut*$>$ nomAttribut ;
- A chaque *opération* d'une *classe* **UML**, correspondra une *méthode* d'une *classe* **Java**.
- A chaque *classe* ***UML*** *abstraite*, correspondra une *classe* ***Java*** *abstraite*.

Les classes

Transcription des classes

- A chaque *classe* **UML**, correspondra une *classe* **Java**.
- A chaque *attribut* d'une *classe* **UML**, correspondra un *attribut* d'une *classe* **Java**.
  - cardinalité ≤ 1 : *TypeAttribut* nomAttribut ;
  - cardinalité > 1 : Collection$<$*TypeAttribut*$>$ nomAttribut ;
- A chaque *opération* d'une *classe* **UML**, correspondra une *méthode* d'une *classe* **Java**.
- A chaque *classe* ***UML*** *abstraite*, correspondra une *classe* ***Java*** *abstraite*.

Remarque

Les *visibilités* des *attributs* et des *opérations* d'une *classe* **UML** seront les mêmes dans la *classe* **Java** obtenue.

Les associations

Transcription des associations

Les associations

Transcription des associations

- A chaque *association* **UML**, correspondra un *attribut* dans les *classes* **Java** participantes à l'*association*

Les associations

Transcription des associations

- A chaque *association* **UML**, correspondra un *attribut* dans les *classes* **Java** participantes à l'*association*
- Un *attribut* est ajouté uniquement si l'*association* est *navigable*

Les associations

Transcription des associations

- A chaque *association* **UML**, correspondra un *attribut* dans les *classes* **Java** participantes à l'*association*
- Un *attribut* est ajouté uniquement si l'*association* est *navigable*
- Le nom de l'*attribut* est le nom de l'extrémité *navigable*

Les associations

Transcription des associations

- A chaque *association* **UML**, correspondra un *attribut* dans les *classes* **Java** participantes à l'*association*
- Un *attribut* est ajouté uniquement si l'*association* est *navigable*
- Le nom de l'*attribut* est le nom de l'extrémité *navigable*
  - cardinalité ≤ 1 : *TypeAttribut* nomAttribut ;

Les associations

Transcription des associations

- A chaque *association* **UML**, correspondra un *attribut* dans les *classes* **Java** participantes à l'*association*
- Un *attribut* est ajouté uniquement si l'*association* est *navigable*
- Le nom de l'*attribut* est le nom de l'extrémité *navigable*
  - cardinalité ≤ 1 : *TypeAttribut* nomAttribut ;
  - cardinalité > 1 : Collection<*TypeAttribut*> nomAttribut ;

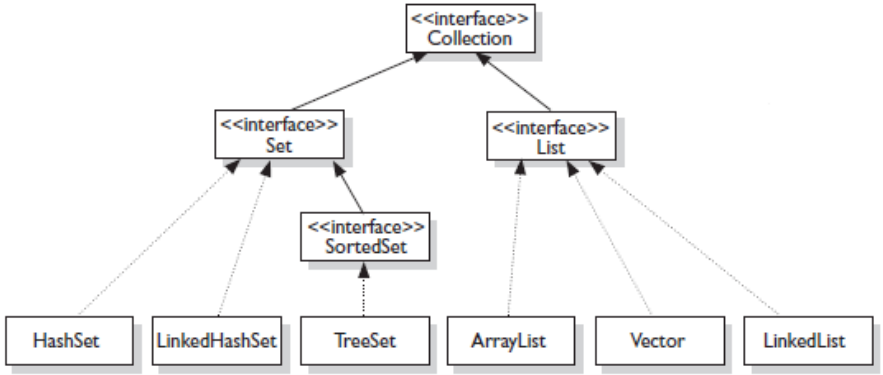
Les associations

Transcription des associations

- A chaque *association* **UML**, correspondra un *attribut* dans les *classes* **Java** participantes à l'*association*
- Un *attribut* est ajouté uniquement si l'*association* est *navigable*
- Le nom de l'*attribut* est le nom de l'extrémité *navigable*
  - cardinalité ≤ 1 : *TypeAttribut* nomAttribut ;
  - cardinalité > 1 : Collection$<$*TypeAttribut*$>$ nomAttribut ;
- La relation de *généralisation* en **UML** est traduite par une relation d'*héritage* en **Java**.

Les collections

Les collections en Java



Les collections

Les attributs de type Collection

Les collections

Les attributs de type Collection

Le type de la *Collection* affecté à un attribut d'une *classe* **Java**, dépend des *propriétés* de l'*attribut/association* déclarées en **UML**.

Les collections

Les attributs de type Collection

Le type de la *Collection* affecté à un attribut d'une *classe* **Java**, dépend des *propriétés* de l'*attribut/association* déclarées en **UML**.

- aucune propriété :
  - traduit en *Collection* (à configurer par l'outil).

Les collections

Les attributs de type Collection

Le type de la *Collection* affecté à un attribut d'une *classe* **Java**, dépend des *propriétés* de l'*attribut/association* déclarées en **UML**.

- aucune propriété :
  - traduit en *Collection* (à configurer par l'outil).
- possède la propriété *ordered* :
  - traduit en *List*.

Les collections

Les attributs de type Collection

Le type de la *Collection* affecté à un attribut d'une *classe* **Java**, dépend des *propriétés* de l'*attribut/association* déclarées en **UML**.

- aucune propriété :
  - traduit en *Collection* (à configurer par l'outil).
- possède la propriété *ordered* :
  - traduit en *List*.
- possède la propriété *unique* :
  - traduit en *Set*.

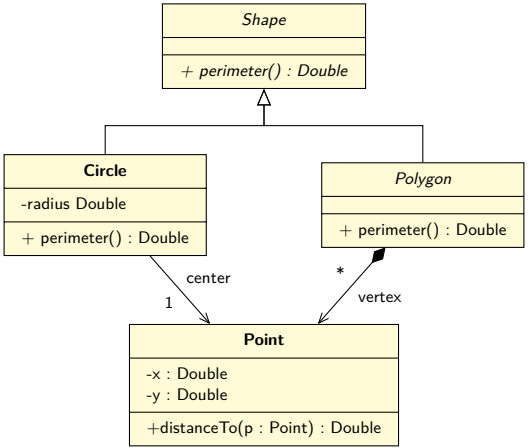
Les collections

Les attributs de type Collection

Le type de la *Collection* affecté à un attribut d'une *classe* **Java**, dépend des *propriétés* de l'*attribut/association* déclarées en **UML**.

- aucune propriété :
  - traduit en *Collection* (à configurer par l'outil).
- possède la propriété *ordered* :
  - traduit en *List*.
- possède la propriété *unique* :
  - traduit en *Set*.
- possède les propriétés *ordered* et *unique* :
  - traduit en *SortedSet*.

Exemple


Shape
+ perimeter() : Double
Circle
-radius Double
+ perimeter() : Double
Polygon
+ perimeter() : Double
center
1
*
vertex
Point
-x : Double
-y : Double
+distanceTo(p : Point) : Double

Exemple

```java
public class Point{
    private double x;
    private double y;

    public double distanceTo(Point autre){
        return 0.0;
    }
}
```

```java
public  abstract class Shape{
    public abstract double perimeter();
}
```

Exemple

```java
public class Circle extends Shape{
    private Point center ;
    private double radius ;

    public double perimeter(){
        return 0.0;
    }
}
```

```java
public abstract class Polygon extends Shape {
    private Collection<Point> vertex;

    public double perimeter(){
        return 0.0;
    }
}
```