PO3T - Cours 8

Modélisation avec UML

Sébastien Combéfis, Quentin Lurkin

Unified Modelling Language (UML)

- Language de modélisation utilisé en ingénierie logicielle

  Notation unifiée pour visualiser le design d'un système

- Standard de l'Object Management Group (OMG) en 1997

  Également adopté comme un standard ISO en 2005

- Très utilisé pour le design de programme en orientée objet

  Mais pourrait être utilisé pour le design de base de données

Modélisation

- Permet de visualiser le plan architectural d'un système

  À l'aide d'un ensemble de diagrammes

- Distinction entre le modèle du système et les diagrammes

  Diagramme = représentation graphique partielle d'un système

- Deux vues d'un système
  - **Statique** décrit la structure du système
  - **Dynamique** décrit le comportement dynamique du système

Diagrammes prédéfinis en UML 2

- Quatorze diagrammes prédéfinis divisés en trois catégories

Structurel	Classe	*(Class diagram)*
	Objet	*(Object diagram)*
	Composant	*(Component diagram)*
	Déploiement	*(Deployment diagram)*
	Package	*(Package diagram)*
	Structure composite	*(Composite structure diagram)*
	Profil	*(Profile diagram)*
Comportemental	Cas d'utilisation	*(Use case diagram)*
	États-transitions	*(State machine diagram)*
	Activité	*(Activity diagram)*
Interaction	Séquence	*(Sequence diagram)*
	Communication	*(Communication diagram)*
	Aperçu de l'interaction	*(Interaction overview diagram)*
	Temps	*(Timing diagram)*

Diagramme structurel

Diagramme de classe

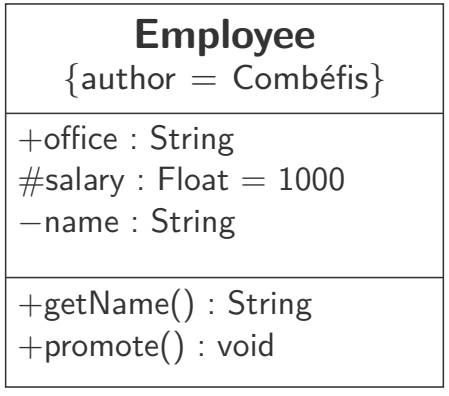
- Capturer les relations statiques dans le programme

  Comment les éléments du programme sont mis ensembles

- Plusieurs types de relation
  - Stockage de références vers d'autres classes
  - Appartenance de classes
  - ...
- Capture de la *« structure physique »* d'un système

Représenter une classe

- Classe modélisée par un diagramme de classe
- Définition composée de trois compartiments
  - **Nom** pour identifier la classe
  - **Attributs** avec visibilité et valeur par défaut (optionnel)
  - **Opérations** avec visibilité

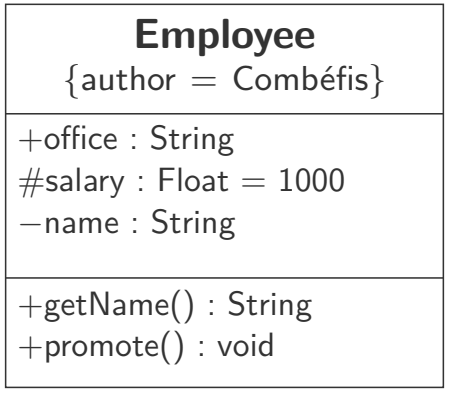


Représenter une classe

- Possibilité d'ajouter des valeurs tagguées

  Rattachées à un élément qui définit leur portée

- Quatre niveau de visibilité
  - $+$ pour public
  - $-$ pour private
  - $\#$ pour protected
  - $\sim$ pour package



Attribut

- Un attribut est un type primitif simple ou un objet

  Peut être une relation vers d'autres objets complexes

- Deux formes possibles pour un attribut
  - Attribut déclaré directement dans la classe
  - Relation entre classes

- Propriétés de multiplicité, d'unicité et d'ordre

  Définissable pour les deux formes d'attribut

visibilité nom : type [multiplicité] = défaut {propriétés}

Visibilité

- Quatre niveaux de visibilité
  - Visibilité **publique** (public) rend l'élément visible de partout
  - Visibilité **privée** (private) restreint la visibilité à la classe
  - Visibilité **protégée** (protected) utilisée avec l'héritage

    Publique pour les sous-classes et privée ailleurs
  - Visibilité **package** rend l'élément visible dans le package
- Applicable pour les attributs et les pour les opérations

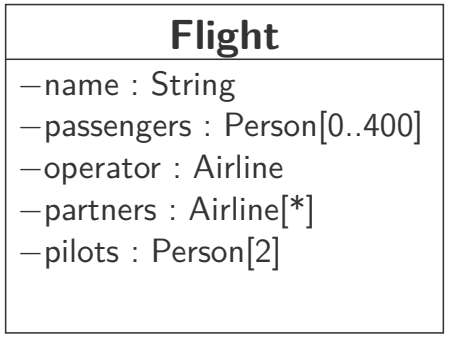
Multiplicité

- La multiplicité définit le nombre d'instances de l'attribut

  Pour chaque instance de la classe principale créée

- Simple entier, un liste d'entier, un intervalle d'entiers

  *Multiplicité par défaut est de 1, valeur non bornée avec **



Propriété des attributs

- Imposer un stockage séquentiel des éléments avec `{ordered}`

  Pour les attributs multiples uniquement, non ordonné par défaut

- Autoriser les doublons avec `{notunique}`

  Pour les attributs multiples uniquement, `{unique}` par défaut

- Figer la valeur d'un attribut avec `{readOnly}`

  La valeur de l'attribut ne peut être changée une fois initialisée

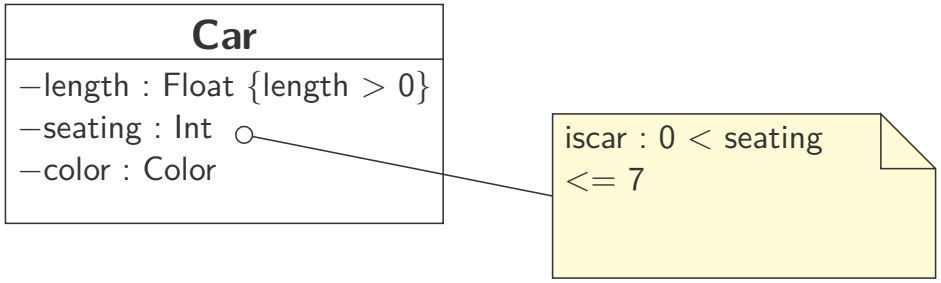
Contrainte des attributs

- Ajout de restriction sur les éléments avec des contraintes

  Expression booléenne derrière l'élément ou comme note

- On peut ajouter une étiquette à la contrainte

  Qu'on place devant l'expression booléenne, suivie de :



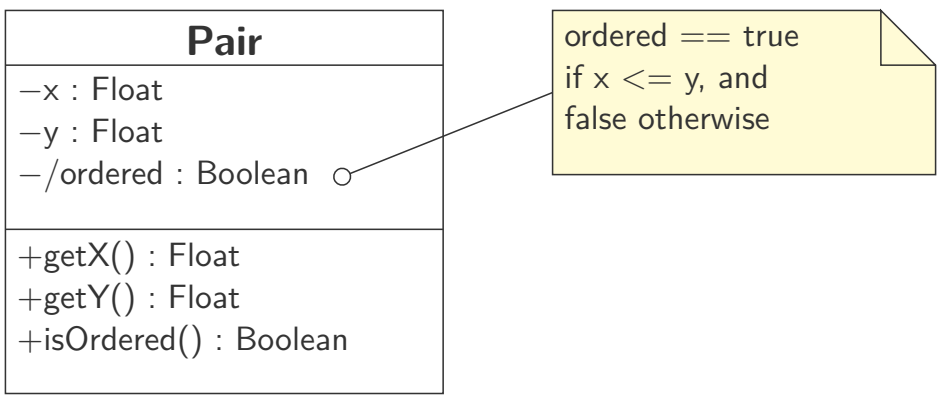
Attribut dérivé

- Attribut dérivé ne doit pas être dans l'implémentation

  Il peut être calculé à partir d'autres

- Peut néanmoins être présent pour raison d'efficacité

- Signalé en préfixant le nom avec un slash (/)



Relation

- Représentation d'un attribut par une relation

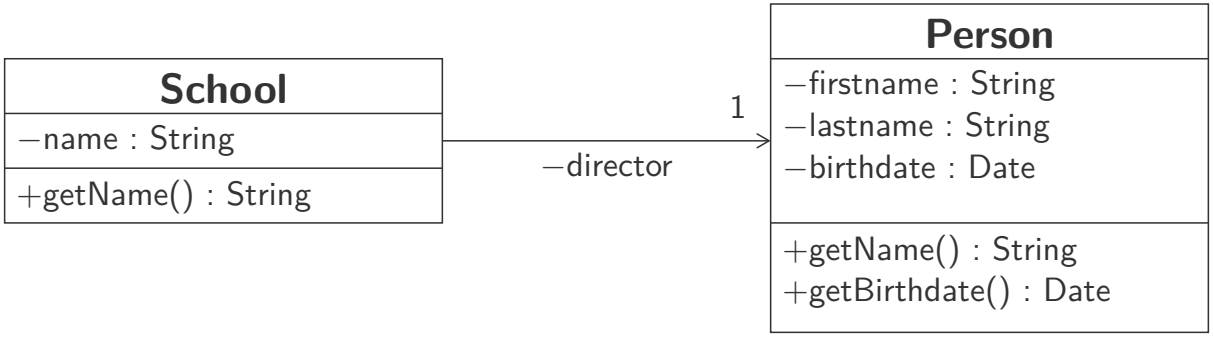
  Illustre comment l'attribut est en lien avec la classe

- Une classe qui contient l'attribut et une qui le représente

  Lien entre les classes, avec le nom de l'attribut

- Mêmes informations que pour attribut déclaré dans la classe

  Mais placées à différents endroits



Opération

- Une opération permet d'invoquer un comportement

  Le comportement agit sur une instance de la classe

- UML fait la distinction entre opération et méthode
  - **Opération** spécifie comment invoquer un comportement
  - **Méthode** implémente l'opération

visibilité nom (paramètres) : type {propriétés}

Paramètre

- Un paramètre d'une opération se définit comme un attribut

  Nom, type, multiplicité, valeur par défaut et propriétés

- La direction décrit comment un paramètre est utilisé
  - `in` : passé à l'opération
  - `inout` : passé par l'appelant, modifiable par l'opération
  - `out` : modifié par l'opération
  - `return` : passé par l'appelant, renvoyé par l'opération

direction nom : type [multiplicité] = défaut {propriétés}

Contrainte des opérations

- Association de contraintes sur les opérations

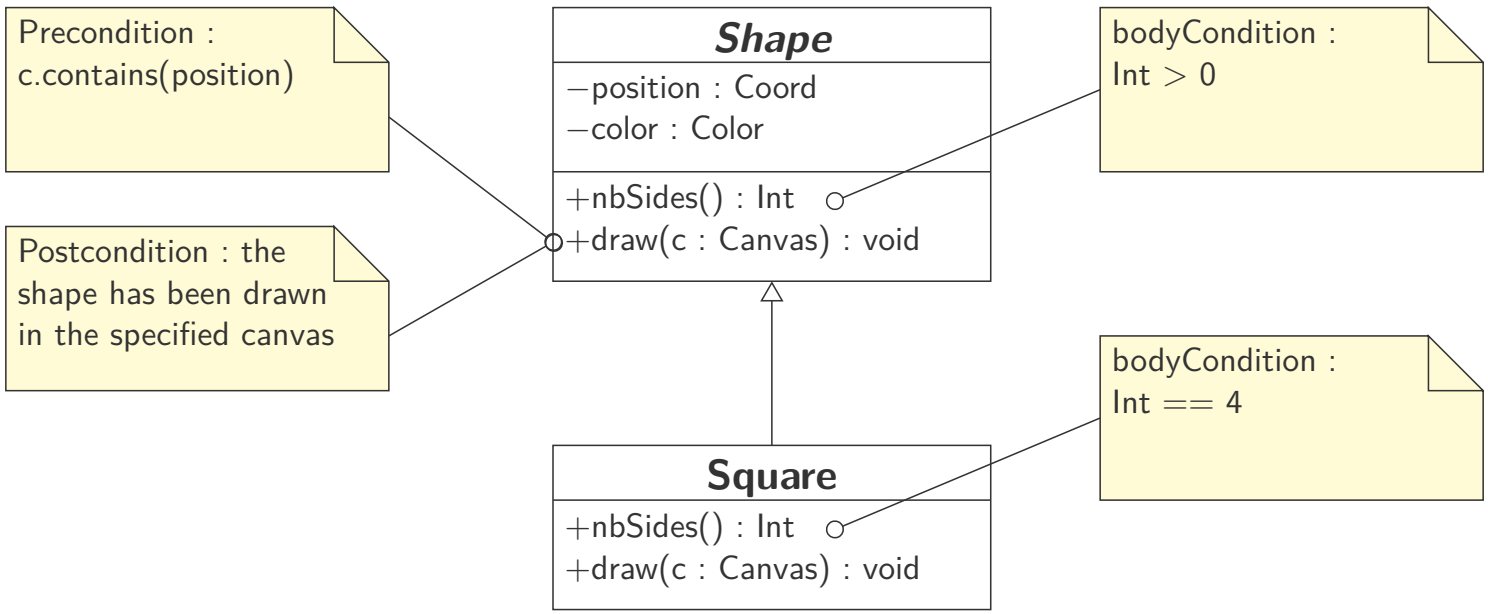
  Contrat que l'implémentation de l'opération doit respecter

- Définition des contraintes comme pour les attributs

- Quelques contraintes spéciales (étiquettes prédéfinies)

  - **Precondition** doivent être satisfaite avant l'appel
  - **Postcondition** seront satisfaites après l'appel
  - **bodyCondition** contraint la valeur de retour

Contrainte des opérations

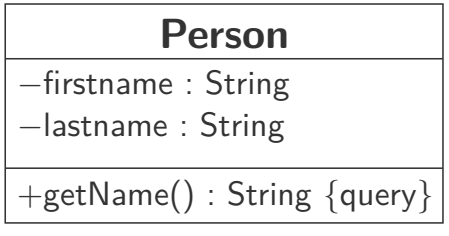


Effet de bord

- Indiquer que l'objet n'est pas modifié avec `{query}`

  L'implémentation ne peut pas modifier l'état de l'objet

- L'appel de l'opération ne provoque aucun effet de bord



Classe abstraite

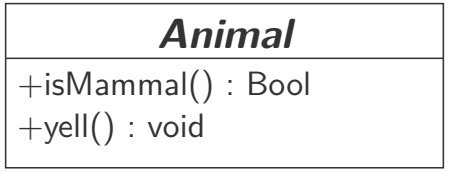
- Classe abstraite contient des opérations sans implémentation

  Peut ne contenir aucune opération

- Impossible de créer une instance d'une classe abstraite

  Étendue par des classes concrètes

- Nom de la classe écrit en italique dans le diagramme UML

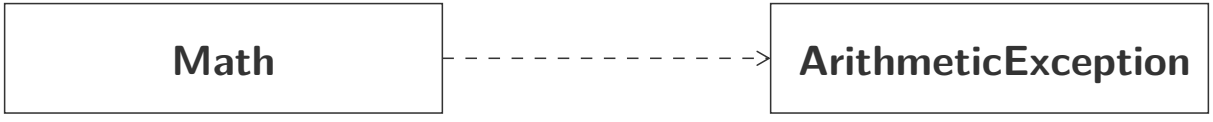


Relation de dépendance

- Dépendance lorsqu'une classe utilise une autre (relation *uses*)

  La relation entre les classes n'est pas durable

- Quelques exemples
  - Une opération reçoit un paramètre de type d'une autre classe
  - Le type de retour d'une opération est d'une autre classe
  - ...



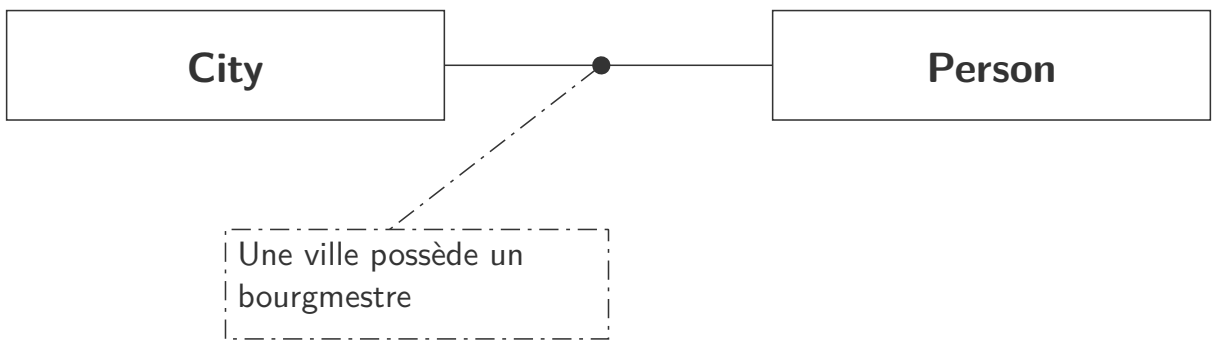
Relation d'association

- Association lorsqu'une classe retient une autre (relation *has-a*)

  La relation entre les classes est durable

- Les vies des objets ne sont pas dépendantes

  Un objet peut être détruit sans que l'autre ne le soit



Propriété des associations

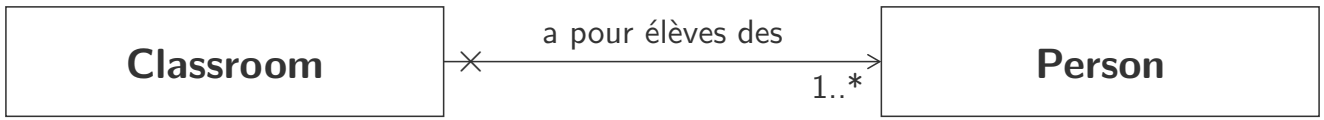
- On indique la navigabilité avec une flèche sur l'association

  Et on peut ajouter une croix pour indiquer la non navigabilité

- Le nom d'une association donne son contexte

- Utilisation multiplicité lorsque l'association est un attribut

  Par défaut, la multiplicité est de 1

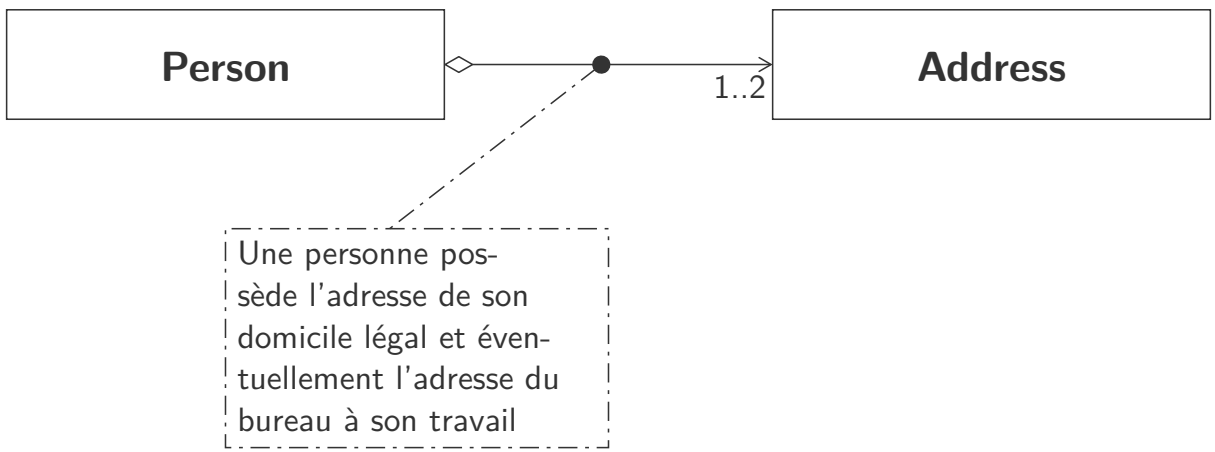


Relation d'aggrégation

- Représente une association forte (relation *owns-a*)

  Il y a appartenance entre deux classes

- On peut indiquer la navigabilité et la multiplicité



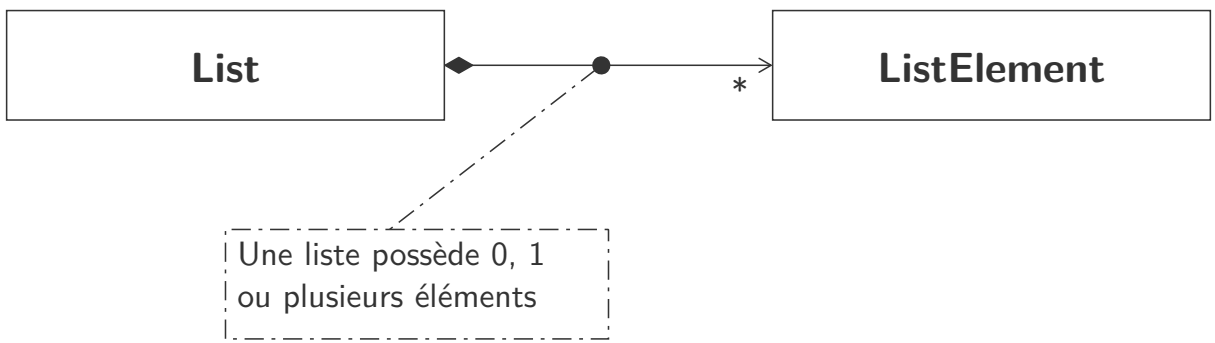
Relation de composition

- Représente une association très forte (relation *is-part-of*)

  Une classe est composée à partir d'autres classes

- Les vies des objets composés sont liées à celle du principal

  Les objets composés disparaissent en même temps que le principal

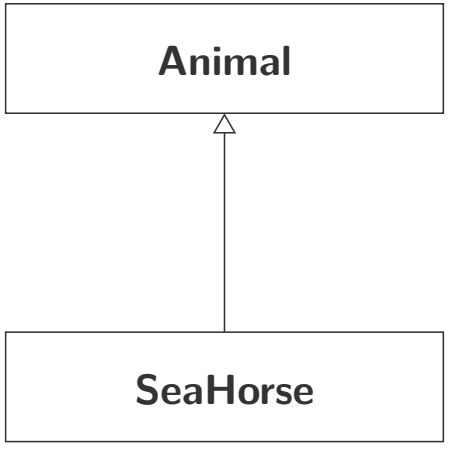


Relation de généralisation

- Une classe généralise une autre plus spécifique (relation *is-a*)

  Une classe est construite en dérivant d'une autre

- Pas de multiplicité ni de nom sur les relations de généralisation

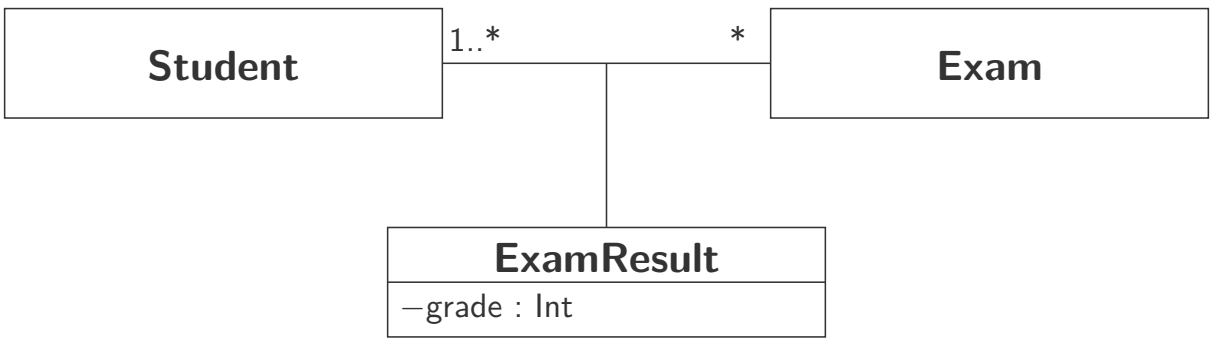


Classe d'association

- Une classe d'association représente une association complexe

  Possède un nom et des attributs

- Résultera typiquement en trois classes lors de l'implémentation



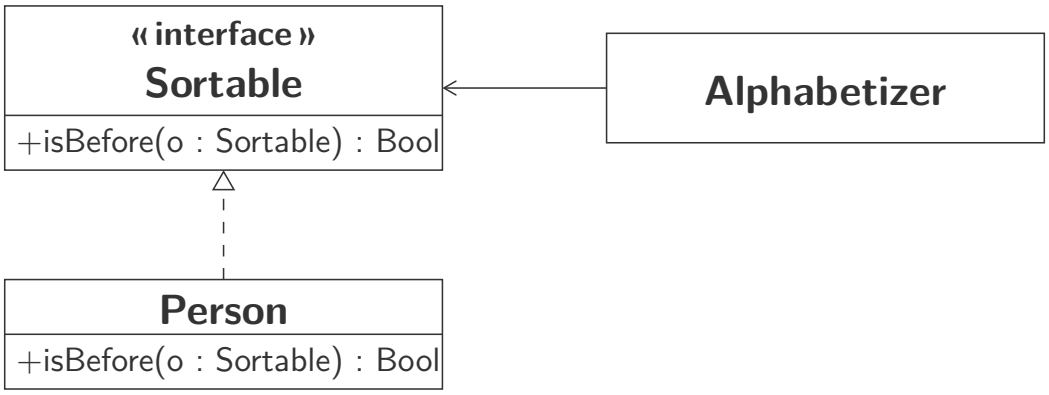
Interface

- Une interface a des déclarations de propriétés et méthodes

  Représente un contrat qu'une implémentation doit respecter

- Deux représentations graphique différentes

  Comme une relation ou notation « ball-and-socket »



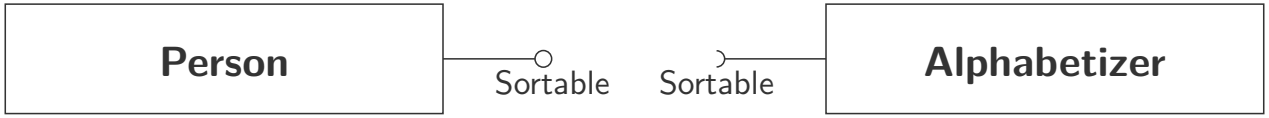
Interface

- Moins de détails sur l'interface

  Focus sur la relation entre les classes

- L'interface est indiquée sous la *« ball »*

  Et la classe qui dépend de l'interface avec un « socket »



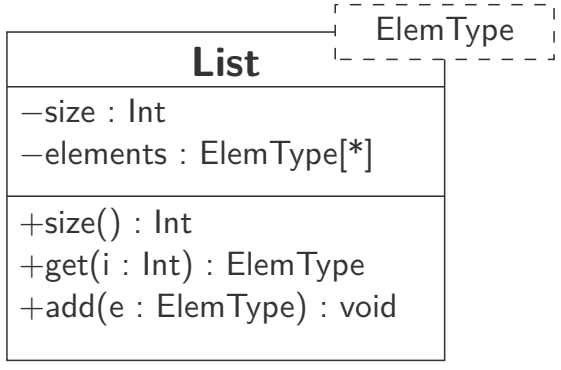
Template

- Classe paramétrisée reçoit des types en paramètre

  Permet d'indiquer un type de classe avec lequel la classe interagit

- Plusieurs templates à séparer avec des virgules

- On peut ajouter une restriction comme `ElemType : Sortable`



Binding

- Binding à faire en spécifiant un type concret pour le template

  Lorsque la classe est utilisée comme attribut
  ou dans une relation (composition, généralisation...)

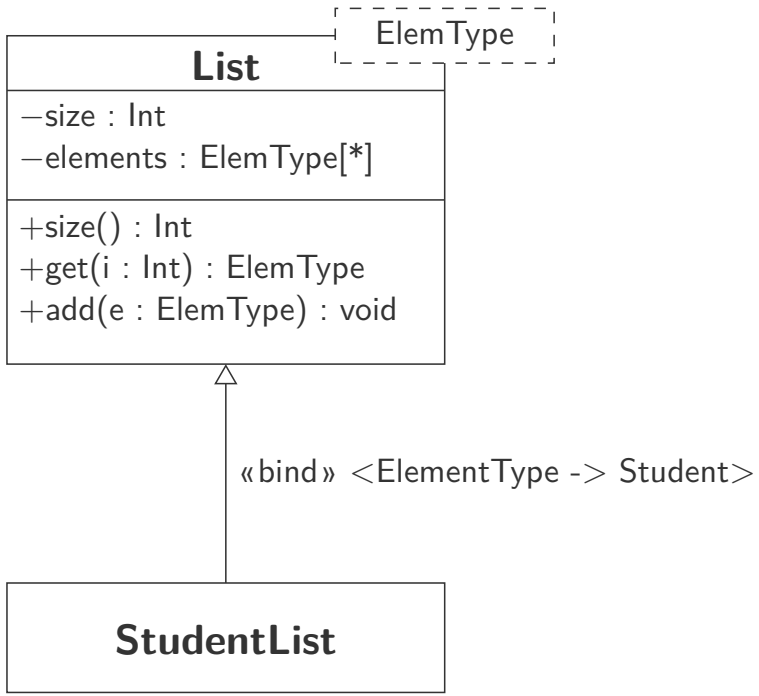


Diagramme comportemental

Diagramme d'activité

- Capturer l'exécution et le comportement d'un système

  Permet de modéliser n'importe quel comportement

- Représente le comportement suite aux appels de méthodes

  Lorsqu'on modélise un logiciel

- Deux éléments de base : activité et action

  Une activité est un comportement composé d'actions

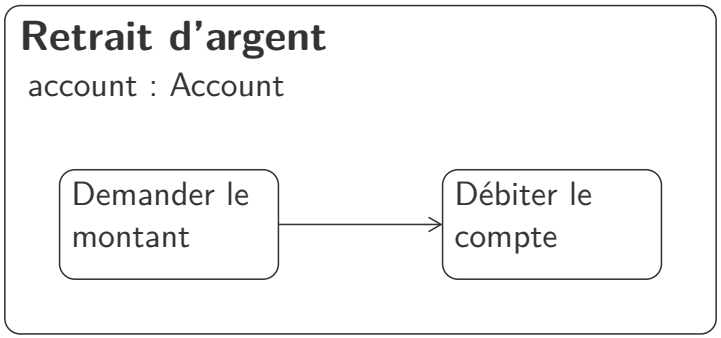
Activité et action

- Activité décomposée en actions, étapes élémentaires

  Une action ne peut pas être décomposée

- Une activité possède un nom et des paramètres éventuels

  Représentation avec un rectangle aux coins arrondis



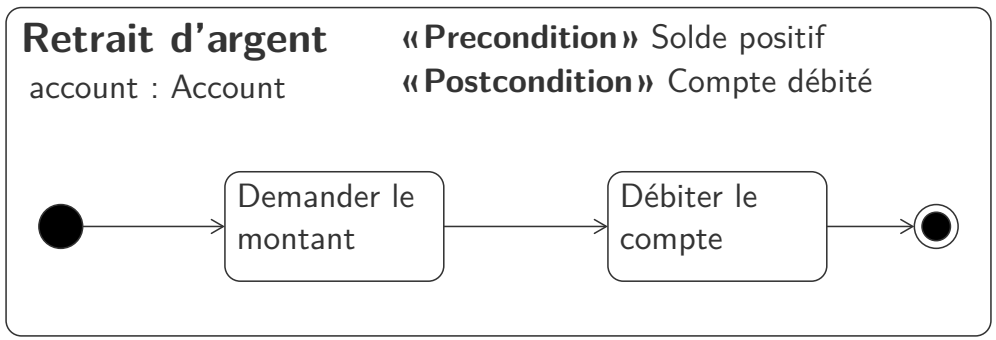
Début et fin de l'activité

- Nœuds spéciaux pour marquer le début et la fin d'une activité

  Point noir pour le début et point noir entouré pour la fin

- On peut attacher des pré et postconditions à une activité

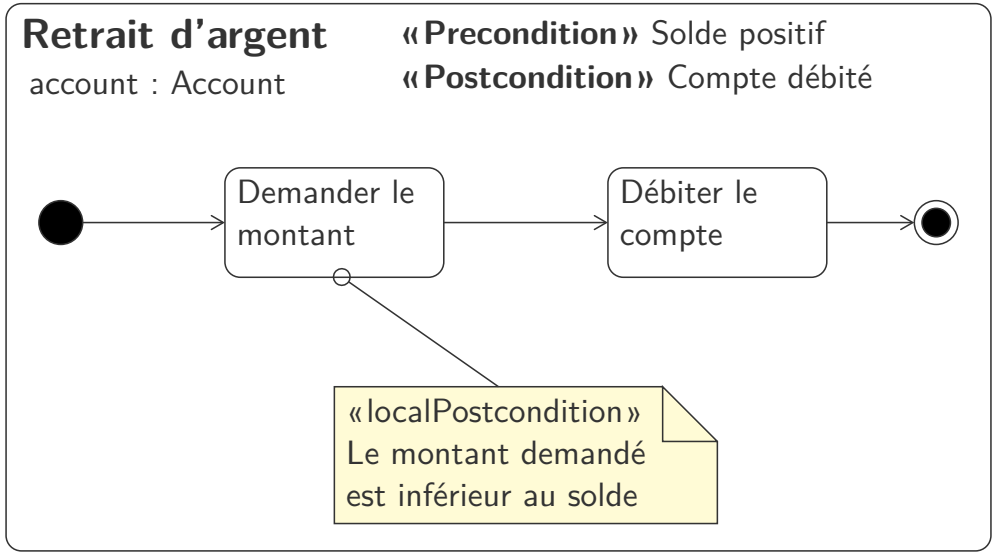
  À indiquer en haut dans la boite de l'activité



Pré et postconditions locales

- Pré et postconditions locales sur les actions

  Ajout de notes sur les actions pour ces conditions



Arête d'activité

- Description du flux d'exécution au sein d'une activité

  À l'aide d'arêtes reliant les actions

- Les arêtes indiquent les flux de contrôle et de donnée

  Les arêtes sont dirigées et peuvent être nommées

- Exécution concurrente des actions non liées par des arêtes

  Possibilité d'exécution parallèle sur une machine multiprocesseurs

Nœud d'activité

- Trois différents types de nœuds
  - **Paramètre** d'entrée et de sortie pour l'activité
  - **Objet** représentant des données complexes
  - **Contrôle** pour diriger le flux de contrôle
- Les différents nœuds sont reliés par des arêtes d'activité

  Par défaut une simple arête dirigée

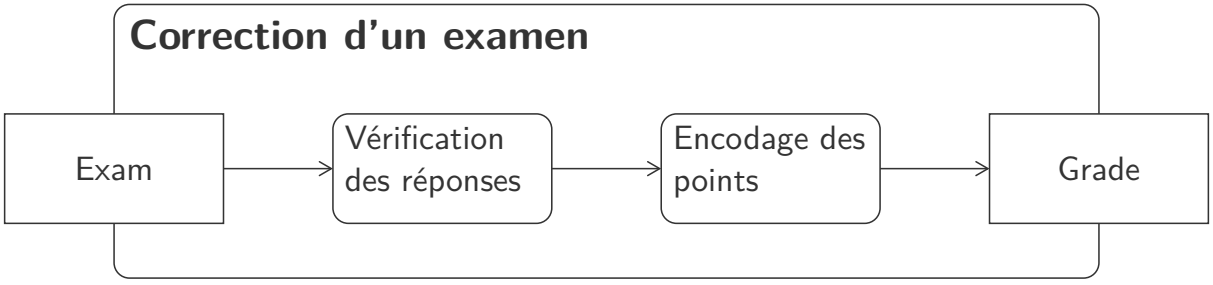
Nœud de paramètre

- Une activité reçoit des paramètres et produit des résultats

  Nœuds de paramètre placés au bord de l'activité

- Nœud de paramètre représenté par un rectangle

  Nom ou description des paramètres dans le rectangle



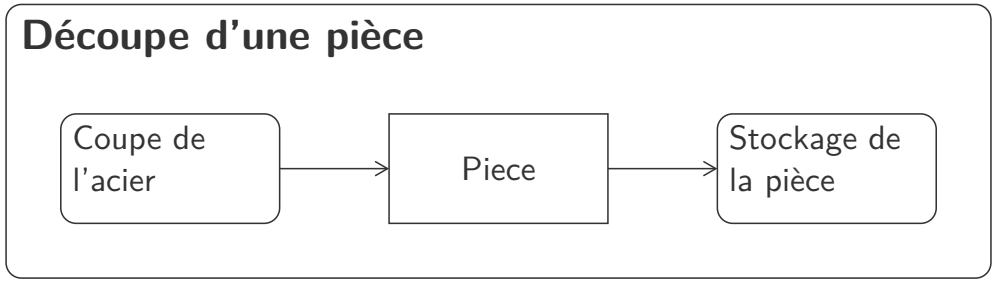
Nœud objet

- Données qui circulent dans l'activité représentée par des objets

  La donnée est transférée entre deux actions

- Représente une instance d'une classe transférée entre actions

  Production d'un objet par une action et réception par une autre



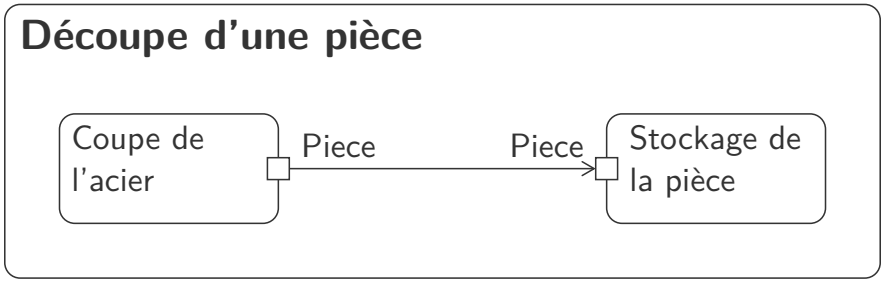
Nœud objet

- Notation compacte à l'aide de pins pour les entrées/sorties

  On attache le rectangle de l'objet sur l'action

- Les pins sont reliés par une arête pour indiquer le transfert

  Production d'un objet par une action et réception par une autre

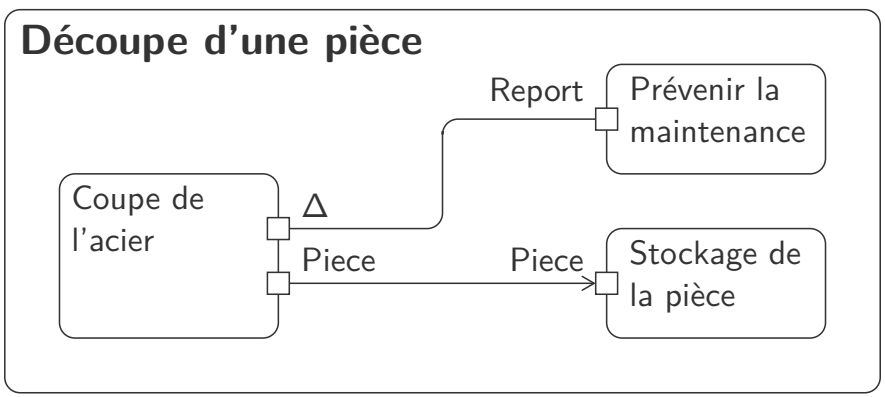


Pin d'exception

- Pin d'exception indique une condition d'erreur

  Ajout d'un triangle près de la pin

- Permet plusieurs chemins possibles en sortant d'une action



Nœud de contrôle

- Contrôle du flux d'exécution à l'aide de nœuds spéciaux

  Prise de décision, concurrence et synchronisation

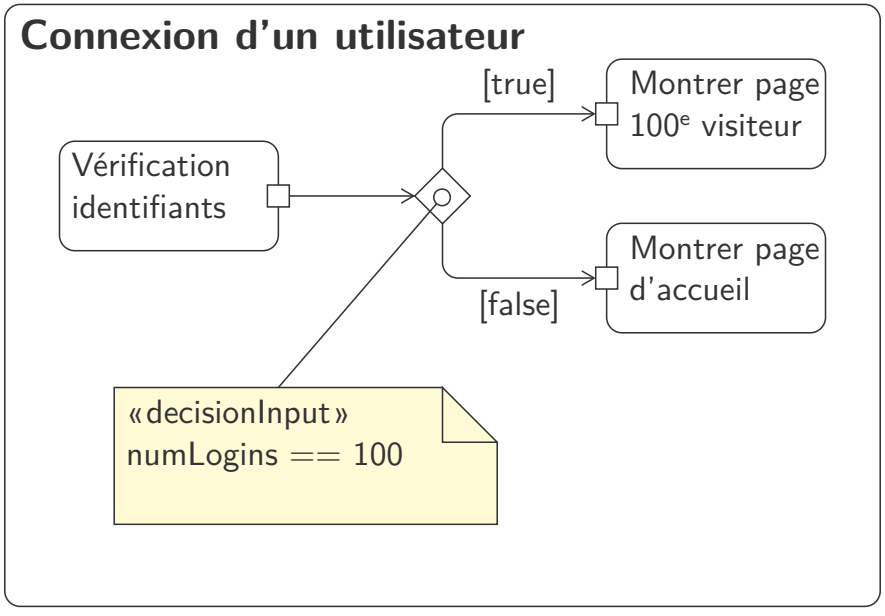
- Plusieurs types de nœud de contrôle
  - **Début** indique le point d'entrée de l'activité
  - **Décision** permet plusieurs chemins selon une condition
  - **Fork** pour lancer des actions concurrentes
  - **Final** indique le point de sortie de l'activité

Nœud de décision

- Choix d'un chemin de sortie en fonction d'une condition

  Gardes sur les arêtes et éventuellement entrée de condition

- Nœud représenté par un diamant et gardes entre crochets

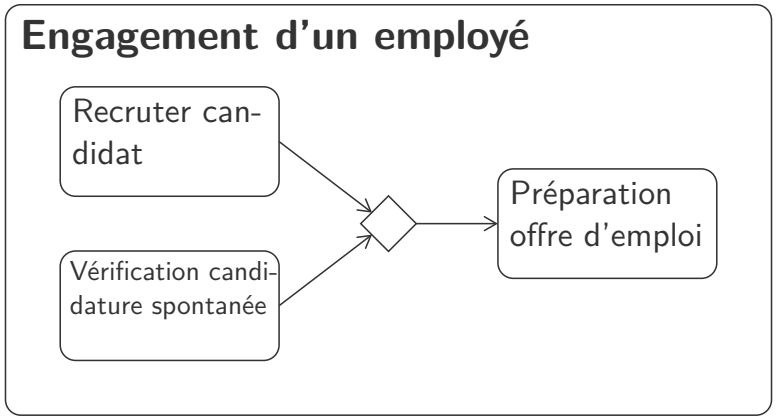


Nœud de fusion

- Fusion de plusieurs actions vers une autre

  Rend disponible à un nœuds les informations de plusieurs autres

- Il ne s'agit pas d'une synchronisation d'actions



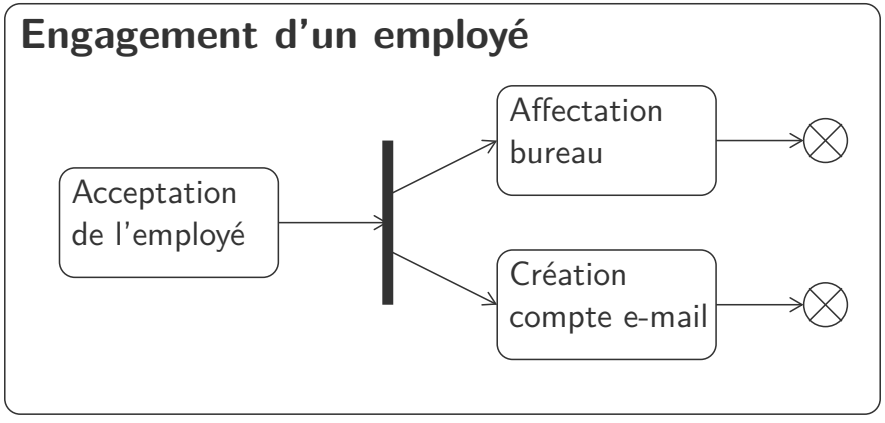
Nœud de fork

- Exécution concurrente de plusieurs actions

  Les données sont dupliquées pour toutes les arêtes sortantes

- Une arête entrante et plusieurs sortantes

  Chaque branche se finit avec un nœud de fin de branche



Nœud de jointure

- Jointure de plusieurs actions avant de poursuivre une autre

  Attend que les actions concurrentes soient toutes terminées

- Plusieurs arêtes entrantes et une arête sortante

  Permet de rassembler des branches en une seule