La modélisation structurelle avec les diagrammes de classes

ACDA – CPOO (M3105)

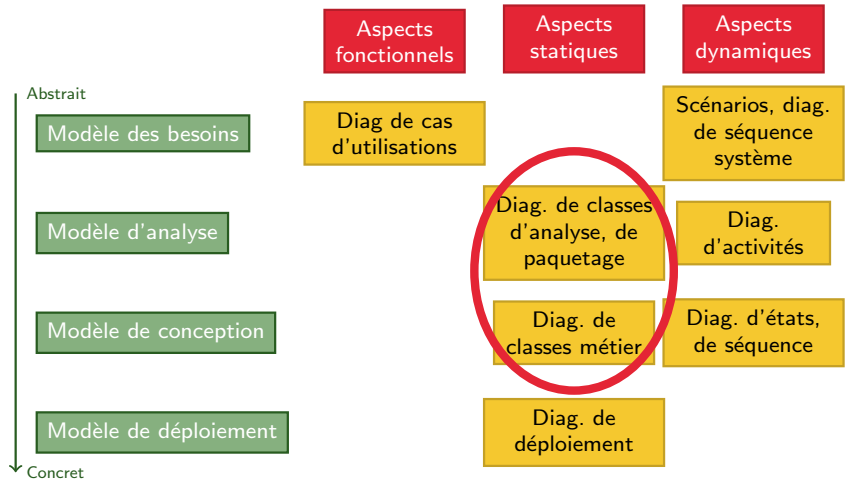
Mathieu Sassolas

IUT de Sénart Fontainebleau
Département Informatique

Année 2015-2016
Cours 3

UPEC
UNIVERSITÉ PARIS-EST CRÉTEIL
IUT SENART FONTAINEBLEAU

Plan de la séance — En image



Plan de la séance

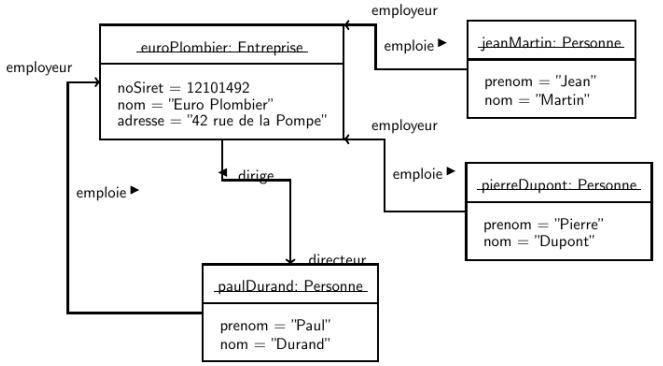
Plan de la séance

Les objets
Selon Wikipédia (fr)

- Une entité (une chose) définie dans un espace à trois dimensions, soit naturelle, soit fabriquée par l'homme (un artefact ou un produit de fabrication industrielle), qui a une fonction précise, désignable par une étiquette verbale (un nom). [...]
- Par ailleurs, certains objets sont incorporels (càd. qu'ils ne sont pas 'objet' des sens) : créations de l'esprit, idéalités, concepts, fantaisies, fictions, constructions mathématiques, classes ou catégories, définitions universelles, but poursuivi, et cetera. Ces objets manquent de concrétude, mais sont, pourtant, les uns réels, les autres irréels.[...]
- En programmation informatique un objet est un conteneur logiciel qui contient les informations et les mécanismes en rapport avec un objet concret ou abstrait. La programmation orientée objet est un style d'écriture de programme informatique basé sur des métaphores d'entités dont les caractéristiques sont manipulées ou simulées par informatique.

Les objets
Dans la programmation objet

- Représentation logique d'un conteneur.
- Une abstraction d'un état de mémoire du programme.
- Une abstraction des pointeurs internes à cet état de mémoire.
- Représenté par des boîtes reliées entre elles :



Les classes

- Un manière de générer des objets ayant tous la même forme : un « moule » pour nos « boîtes ».
- Une classe est donc la donnée des attributs (avec leur type) dont seront dotés les objets instance de cette classe.
- Ici « la forme suit la fonction » *("form follows function")*, donc des objets de même classe auront la même fonction : classe $\sim$ type.
- Un type sans fonctions l'utilisant est peu utile.
  - Il faut donner des fonctions qui utilisent les objets.
  - Certaines fonctions agissent sur l'objet lui même : les opérations.
  - Il peut bien sûr exister des fonctions prenant de tels objets en arguments.

Visibilité

- Détermine ce que l'on peut voir (pour les attributs) ou utiliser (pour les opérations) depuis l'extérieur de la classe.
- Exemple : lorsqu'un objet est passé en argument d'une fonction, on est à l'extérieur ; lorsqu'on est dans le code d'une opération de cette même classe, on est à l'intérieur.
- Les différentes visibilités :
  - $-$ Privé : visible uniquement depuis l'intérieur.
  - $+$ Public : visible depuis l'extérieur.
  - $\#$ Protégé : visible depuis l'intérieur et depuis l'intérieur d'objets de classe héritant.
  - $\sim$ Paquetage : visible seulement depuis les autres objets définis dans le paquetage.

En règle générale

Les attributs seront privés (à la rigueur protégés), les opérations seront publiques.

Éléments de classe

Propriété « statique » *(static)*

L'élément est relatif à la classe et non aux objets instance de cette classe.

- Concerne les attributs comme les opérations.
- Pour les attributs, utile pour spécifier des constantes.
- Pour les opérations, utile pour spécifier des opérations qui sont logiquement dans cette classe mais qui n'agissent pas sur des objets : sous-routines, comparaisons d'objets de cette classe...
- Un élément static apparaît souligné.

Ne pas confondre avec les clefs primaires utilisées pour la modélisation de bases de données.

Autres propriétés des attributs

Dérivé Qui peut être calculé.
- Concerne uniquement les attributs.
- Indiqué par un symbole /
- Utile si l'attribut est important, utilisé souvent, calculable mais à grand coût...
- Exemple : `-/ âge` est calculable depuis `- dateNaissance`.

Cardinalité Les attributs peuvent avoir une multiplicité.
- Exemple : `- autresPrenoms[0..3]`.
- Plus d'infos sur les cardinalités lorsqu'on parlera des associations.

Valeur par défaut Indiquée par `= valeur`.

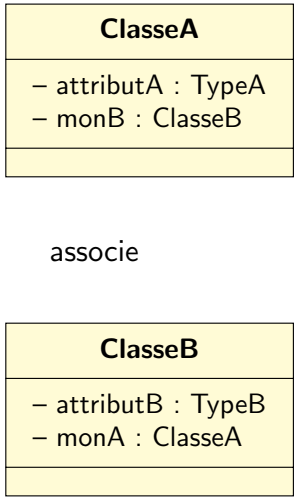
Type énumération Type (anonyme) qui a un nombre fini de valeur possibles : `- jour: {'Lundi',... }`.

Représentation graphique des classes

Classe
– attribut1 : Type1 = valeur # attribut2[0..1] : Type2 –/ attribut3 : Type3 – attribut4 : Type4 ~ attribut5 : {val1,val2}
+ opération(arg : TypeArg) + routine(arg : Classe) : Type

Plan de la séance

Association simple

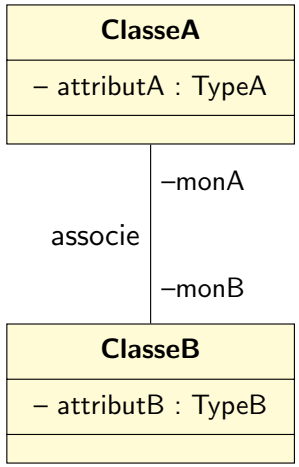


- Lorsqu'un attribut a pour type une classe présente sur le diagramme, on explicite cette relation par une association.
- On peut nommer l'association pour expliciter la nature de la relation.
- Une association fournit donc ces attributs implicites dont le nom est fourni par les rôles.

⇝ Les rôles ont une visibilité.

IUT SENART FONTAINEBLEAU
Classes
M. Sassolas
M3105
Cours 3
Programme
Classes & objets
Associations
Héritage
Dépendance
Interfaces
Analyse *vs* conception
Paquetages

Association simple

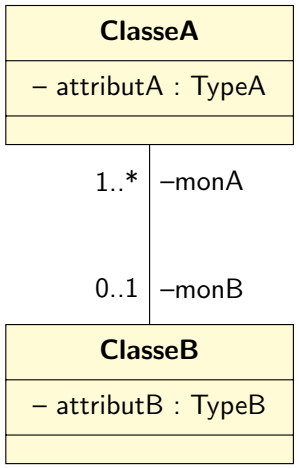


- Lorsqu'un attribut a pour type une classe présente sur le diagramme, on explicite cette relation par une association.
- On peut nommer l'association pour expliciter la nature de la relation.
- Une association fournit donc ces attributs implicites dont le nom est fourni par les rôles.

⇝ Les rôles ont une visibilité.

IUT SENART FONTAINEBLEAU
Classes
M. Sassolas
M3105
Cours 3
Programme
Classes & objets
Associations
Héritage
Dépendance
Interfaces
Analyse *vs* conception
Paquetages

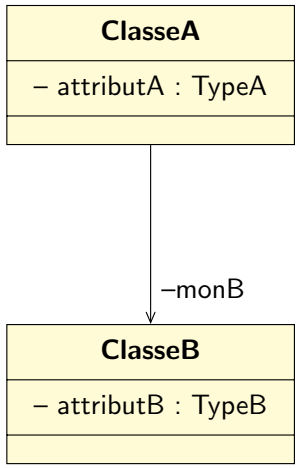
Cardinalités



- Indique le nombre d'objets qui peuvent être ainsi associés.
- Syntaxe : intervalle $x..y$, point x, arbitrairement $*$.
- Les plus utilisés :
  - 1 Unique (par défaut).
  - 0..1 Optionnel.
  - 1..* Au moins un.
  - * Autant que l'on veut (potentiellement 0).
- Les rôles s'entendent comme l'ensemble des objets associés.

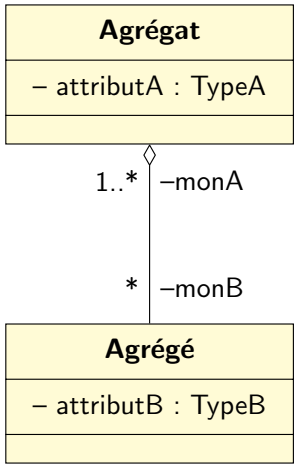
IUT SENART FONTAINEBLEAU
Classes
M. Sassolas
M3105
Cours 3
Programme
Classes & objets
Associations
Héritage
Dépendance
Interfaces
Analyse *vs* conception
Paquetages

Navigabilité



- Indique que l'accès ne se fait pas dans les deux sens.
- On remarque qu'il n'y a donc plus de rôle monA.
- Par défaut : la navigabilité est bidirectionnelle.

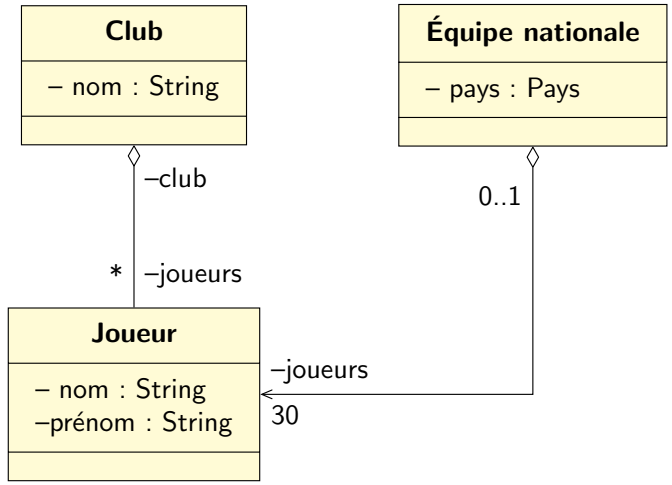
Agrégation



- Un raccourci pour la notion d'ensemble.
- L'agrégat est formée des objets agrégés.
- Un objet peut appartenir à plusieurs agrégats.
- Détruire un agrégat ne détruit pas les agrégés.

Agrégation — Exemple

Le diagramme de classes



Agrégation — Exemple

Le diagramme objets

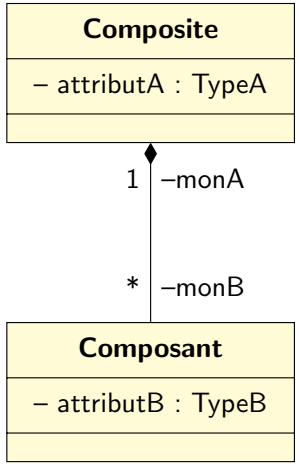
stadeToulousain: **Club**
nom = « Stade Toulousain » joueurs = {dusautoir, tolofua, . . . }

dusautoir: **Joueur**
nom = « Dusautoir » prénom = « Thierry » club = stadeToulousain

xvFrance: **Équipe nationale**
pays = france joueurs = {dusautoir, kayser, . . . }

- Détruire une équipe ne détruit pas un joueur.
- Un joueur peut appartenir à plusieurs équipes.

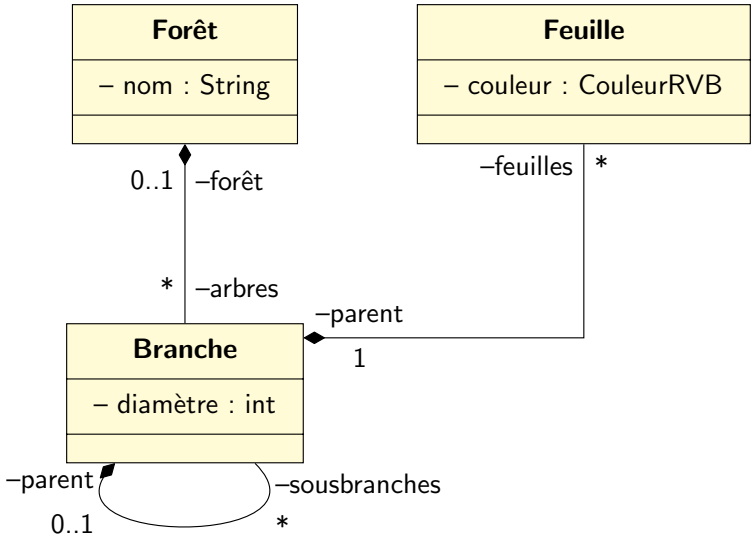
Composition



- Un raccourci pour la notion de partie.
- Le composite est formée des objets composants.
- Un **objet** ne peut appartenir qu'à un seul composite : cardinalité 1 ou 0..1 seulement.
- Détruire un composite signifie détruire ses composants

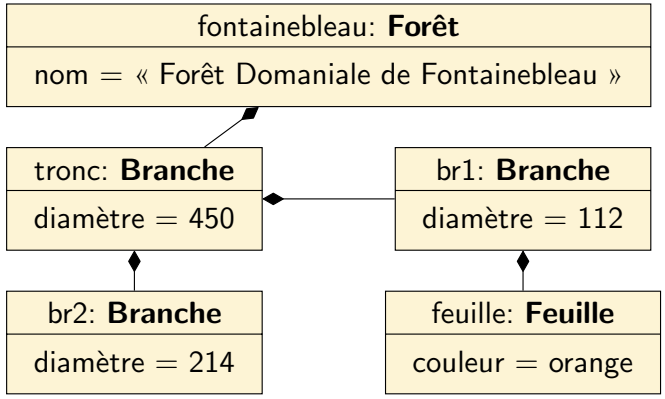
Composition — Exemple

Le diagramme de classes



Composition — Exemple

Le diagramme objets



- Chaque branche a un seul parent.
- Si l'on abat le tronc, toutes les sous-branches meurent. Si l'on abat une forêt, tous les arbres sont coupés.

Mais si on code ?

Cardinalités, agrégations et compositions dans le code

- Des cardinalités * impliquent souvent que l'attribut est un ensemble : collection, liste, tableau...

↪ Il faut souvent avoir les accesseurs *ad-hoc* pour modifier ces ensembles :

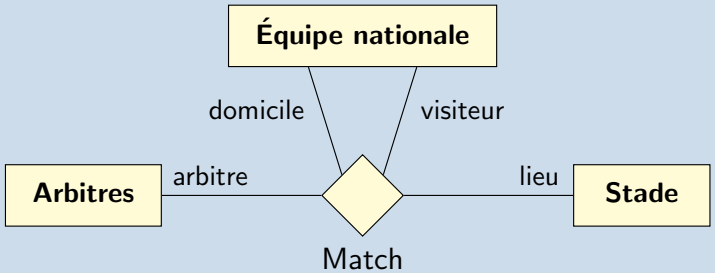
- il est utile de nommer le get adéquatement : getListe...
- insérer, supprimer au lieu du set habituel.

⇝ C'est souvent le cas avec des agrégations ou compositions.

- Il n'y a pas vraiment moyen de contrôler la différence entre agrégation et composition dans le code : le code effectué à la destruction du composite doit se charger de détruire le composant (si on veut être propre, autrement il y a les *garbage-collectors*...).

Associations *n*-aires

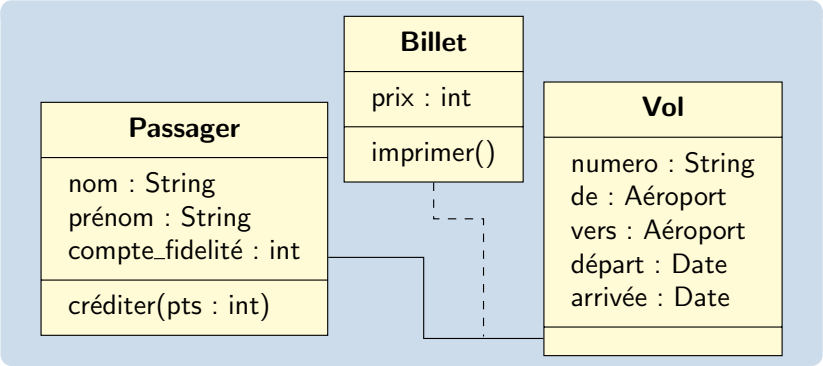
- Dans le cas de diagrammes de classe d'analyse, on peut envisager des relations *n*-aires.
- Au sens mathématique : n-uplet d'objets instances des classes liées : $(x_1, \ldots, x_n) \in X_1 \times \cdots \times X_n$.



Les associations *n*-aires ne sont pas obligatoires, et peuvent souvent être remplacées par une classe.

Classes d'association

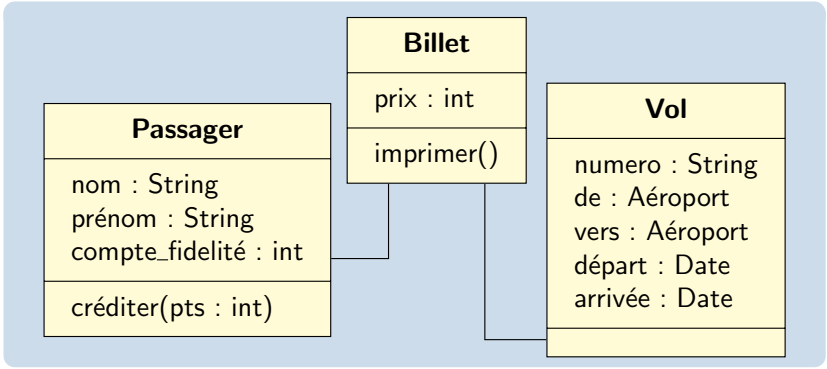
- Lorsque l'association entre deux classes dispose elle même d'attributs (voire d'opérations).
- Les instances de cette classe n'existent que lorsque l'association existe.
- Modélisent les informations relatives à l'interaction.



Mais si on code ?

Associations *n*-aires et classes d'associations dans le code

- Les classes d'association ou les associations *n*-aires n'ont pas de pendant dans les langages objets.
- On replace l'association *n*-aire par une classe sans autre attribut que ses associations.
- On remplace l'association disposant d'une classe par deux associations vers la classe d'association :

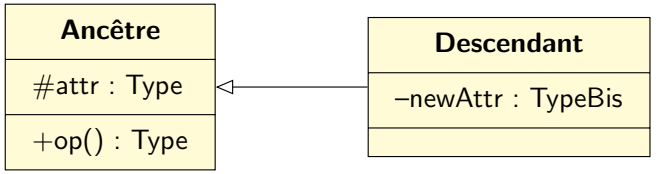


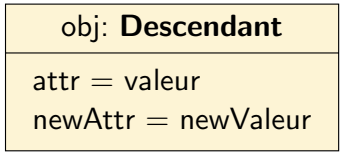
Plan de la séance

Héritage

Pourquoi ?

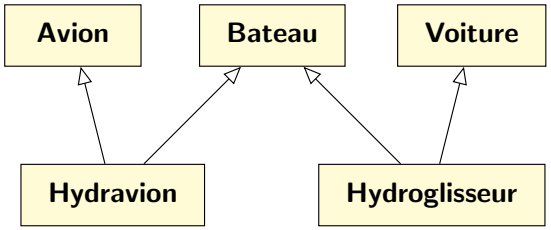
Factoriser pour en écrire le moins possible. 😄





Remarques sur l'héritage

- On hérite de tous les attributs même implicites ⇝ associations.
- On hérite des opérations mais elle peuvent être redéfinies *(override)* ⇝ polymorphisme : l'opération de l'ancêtre prend plusieurs forme dans les objets hérités.
- On peut hériter d'une classe héritant. . .
- On peut hériter de plusieurs classes à la fois :



Soyez prudents, cela ne fonctionne pas toujours dans le code.

Classes et opérations abstraites

Une classe abstraite n'est jamais instanciée.

- Le but est uniquement d'en hériter.
- Le nom de la classe apparaît en italique.
- À la main on indique le stéréotype <<abstract>>.

ClasseAbs

Une opération abstraite doit être redéfinie dans les classes héritant.

- Le nom de l'opération apparaît en italique.
- À la main on indique le stéréotype <<abstract>>.

Classe
+opAbs()

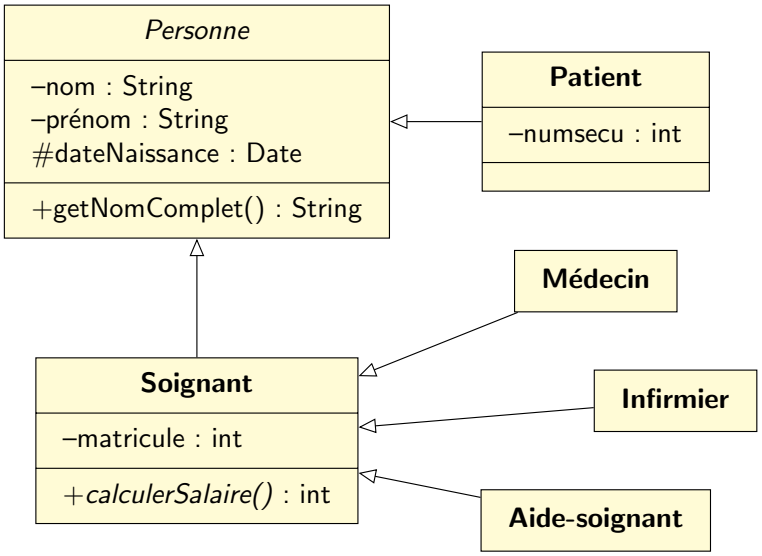
Attention au vocabulaire

Pour l'implémentation (et l'APL. . .)

Chaque langage de programmation utilise un vocabulaire différent.

- En Java :
  - Une opération abstraite reste une opération abstraite.
  - Une classe dont une opération est abstraite est `abstract`.
  - Une classe `abstract` de Java pourrait être instanciée.
  - On peut simuler une classe abstraite sans attributs par une interface ⇝ plus de détails sur l'interface plus tard.
- En C++ :
  - Une opération abstraite est appelée virtuelle pure.
  - Une classe dont une opération est virtuelle pure est abstraite.
  - Une classe abstraite est appelée une interface.

Exemple d'héritage



Retour sur la visibilité « protégé »

- Les attributs (et opérations) protégés sont visibles par suite d'héritage : `dateNaissance` est visible dans la classe `Médecin`.
- Lorsqu'une classe a vocation à être héritée, il faut se poser la question du privé *vs* protégé.
  - Si un accesseur a vocation à être redéfini, choisir protégé.
  - Sinon, choisir privé, la classe héritant utilisera si besoin les accesseurs définis dans la classe mère.

Plan de la séance

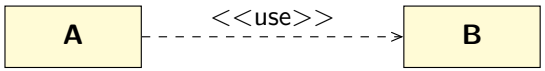
Dépendances

- Pour modéliser des liens logiques entre les classes.
- Une classe `A` dépend d'une classe `B` si le bon fonctionnement de `A` nécessite l'existence (et le bon fonctionnement) de `B`.
- Crée des dépendances dans le code :
  - il faut coder la classe `B` avant ;
  - une modification de `B` peut affecter `A` (maintenance).
- Plusieurs cas de dépendance sont possibles :

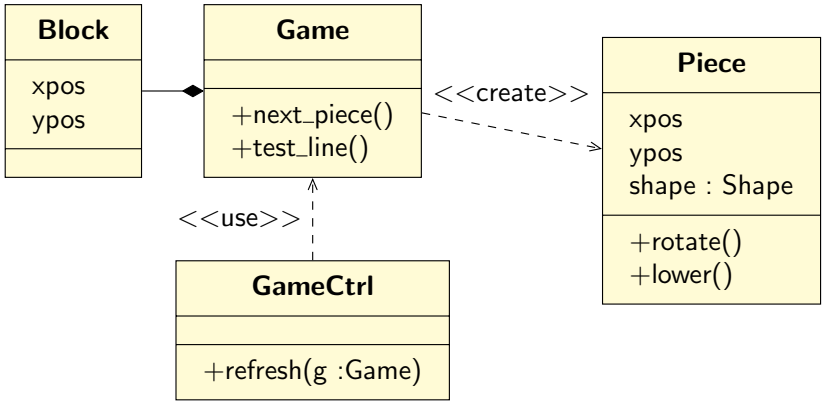
  use : utilisation : un objet de classe `B` est utilisé en argument d'une opération de `A` ;

  create : création/instanciation : la classe `A` va créer des objets de `B` ;

  call : appel : la classe `A` appelle une opération de `B`.



Exemple de dépendances



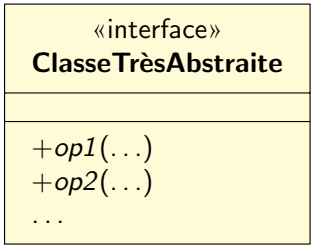
Dépendances implicites

L'association, l'héritage... ont des dépendance implicites : pas besoin de les indiquer !

Plan de la séance

Interfaces

- Un cas particulier de classes abstraites :
- sans aucun attribut ;
- dont toutes les opérations sont abstraites.
- On utilise le stéréotype <<interface>>.



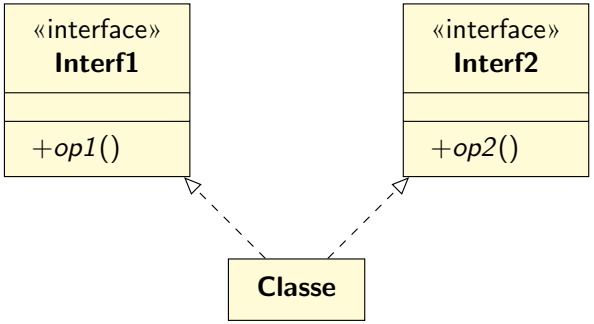
Philosophie

Les classes qui en héritent doivent implémenter un comportement.

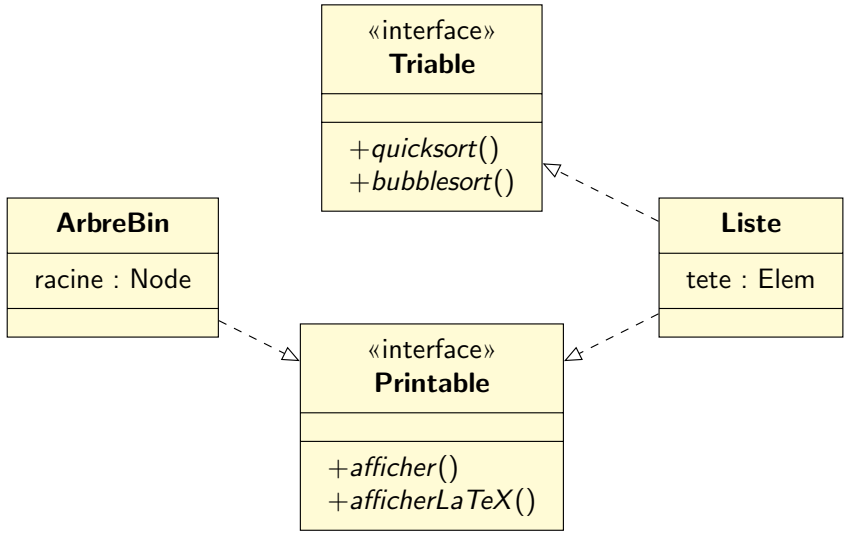
Réalisation

Héritage d'interfaces

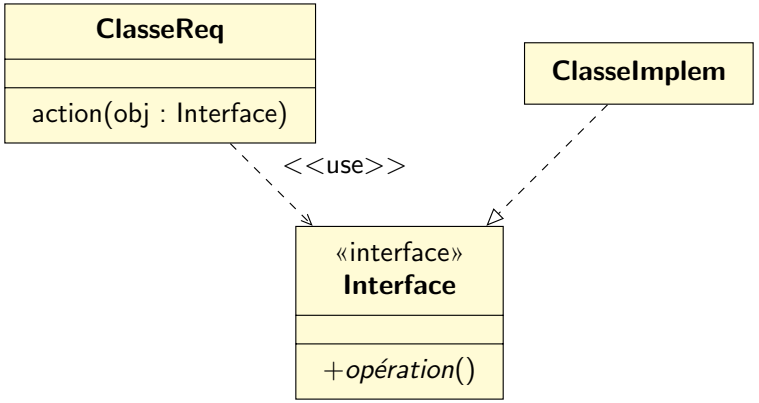
- Ce cas particulier est nommé réalisation.
- Notation : flèche d'héritage pointillée.
- Il évite l'héritage multiple.



Réalisation — Exemple



Utilisation d'interfaces

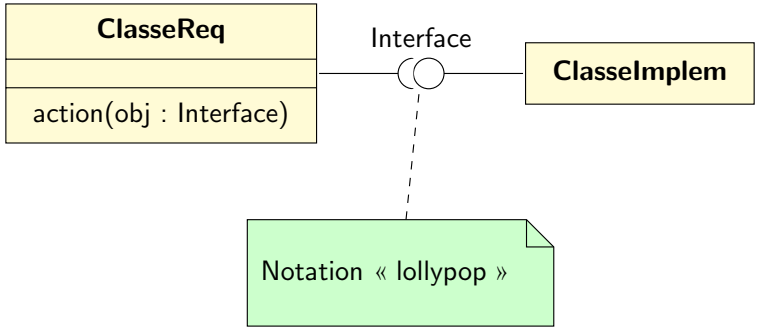


La classe `ClasseImplem` fournit la capacité de réaliser l'`opération` dont `ClasseReq` a besoin.

IUT SENART FONTAINEBLEAU

Utilisation d'interfaces

Classes
M. Sassolas
M3105
Cours 3
Programme
Classes & objets
Associations
Héritage
Dépendance
Interfaces
Analyse *vs* conception
Paquetages



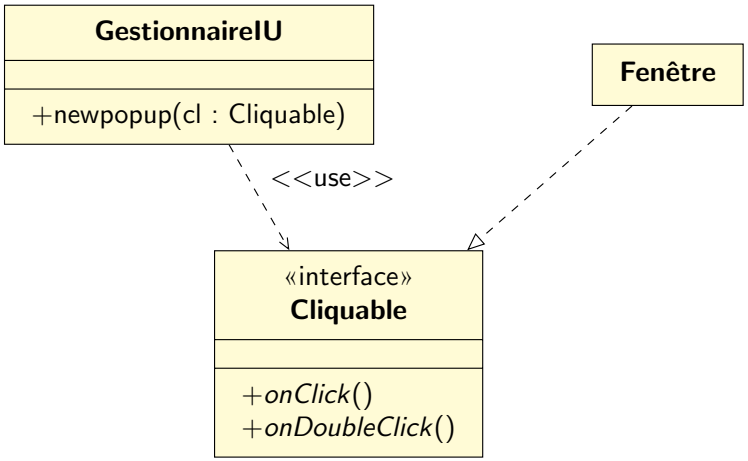
La classe `ClasseImplem` fournit la capacité de réaliser l'`opération` dont `ClasseReq` a besoin.

40 / 50

IUT SENART FONTAINEBLEAU

Exemple d'interfaces

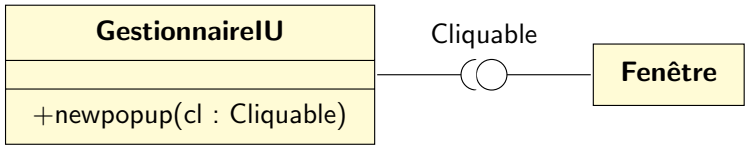
Classes
M. Sassolas
M3105
Cours 3
Programme
Classes & objets
Associations
Héritage
Dépendance
Interfaces
Analyse *vs* conception
Paquetages



IUT SENART FONTAINEBLEAU

Exemple d'interfaces

Classes
M. Sassolas
M3105
Cours 3
Programme
Classes & objets
Associations
Héritage
Dépendance
Interfaces
Analyse *vs* conception
Paquetages



41 / 50

Quand utiliser les interfaces ?

- Les interfaces découpent l'application selon des besoins et les fonctionnalités.
- Permet une forme de programmation par contrats.
- Quand l'héritage concerne les comportements et non les données.
- Permet de factoriser des classes qui n'ont pas forcément de lien entre elles.

Plan de la séance

Que modélise-t-on ?

- Le diagramme de classes permet de modéliser la structure à différents niveaux de détail, à différentes étapes de la modélisation.
- Les diagrammes d'analyse et de conception sont très différents.
- Un diagramme de classes (sans opérations) peut aussi modéliser une base de données (cf ACSI IS1).

Diagramme de classes d'analyse

- Identifier les classes.
  - Un acteur du diagramme de cas d'utilisation n'a pas nécessairement de classe le représentant.
  - De même, le *Système* du DCU n'a jamais de classe le représentant.
- Ces attributs mérite-t-il d'être rassemblés dans une classe ?
  - S'ils représentent un ensemble cohérent ($\rightsquigarrow$ pas de difficulté à trouver un nom à cette classe).
  - S'ils sont utilisés ailleurs, en particulier par des associations qui leur sont propres.
  - Si certaines opérations ne concernent que ceux-ci.
- Pas besoin de préciser les types.
- On peut utiliser des associations n-aires, des classes d'associations.
- On limite l'héritage aux classes logiquement liées (quelques attributs en commun ne font pas toujours un héritage).

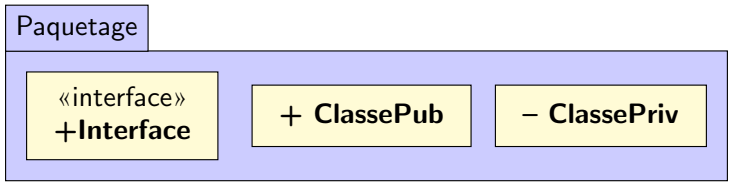
Diagramme de classes de conception

- Partir du diagramme de classe d'analyse : il donne (presque) les classes du modèle et la structure d'une éventuelle base de données.
- Préciser les types.
- Ajouter les classes qui ne sont pas métier : vues, contrôleurs...
- Ne pas hésiter à diviser le diagramme en fonction des différentes parties du système.
- Ensuite factoriser par héritage et interfaces.

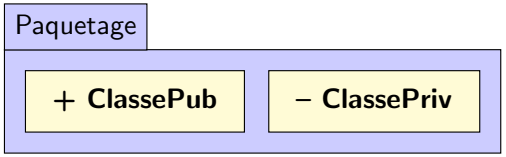
Plan de la séance

Diviser le système

- Séparer le système en parties logiques (presque) indépendantes.
- Exemple : une partie gère la base de donnée, une partie gère l'interface graphique, une partie gère le cœur de métier...
- Permet d'avoir une vue plus haut niveau des diagramme de classes.
- Pratique pour lister les interfaces fournies par un paquetage.



Visibilités



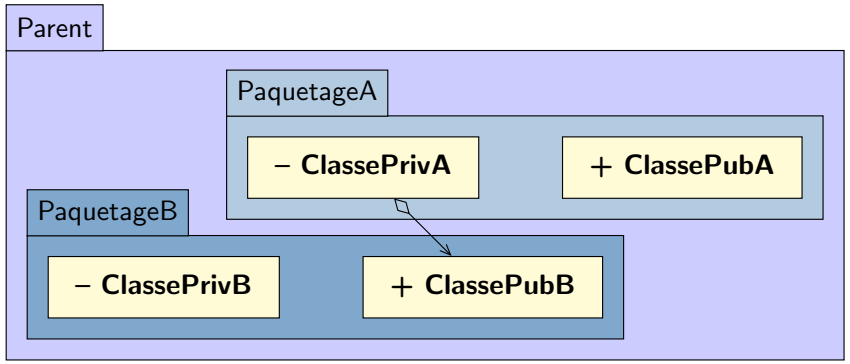
- Les classes au sein d'un paquetage peuvent être...
  - \+ publiques : visibles depuis l'extérieur du paquetage (défaut).
  - – privées : invisibles depuis l'extérieur du paquetage.

↪ « Invisible » signifie que l'on ne peut pas l'instancier.

- Rappel : les attributs/opérations peuvent avoir la visibilité paquetage (∼) :
  - L'attribut est publique depuis les autres classes du paquetage.
  - L'attribut est privé depuis l'extérieur du paquetage

⇝ Cette visibilité n'a de sens que si la classe elle-même est publique.

Hiérarchie, lien entre paquetages



- On n'interdit pas les associations entre classes de différents paquetages.
- On peut mettre des paquetages dans d'autres pour hiérarchiser le découpage.
- (On peut relier les paquetages d'autres manières ; nous n'en parlerons pas.)