université PARIS-SACLAY

FACULTÉ DES SCIENCES D'ORSAY

Introduction au Génie Logiciel

Constat

If the automobile had followed the same development cycle as the computer, a Rolls-Royce would today

- cost $100,
- get a million miles per gallon,
- and explode once a year, killing everyone inside.

Robert X. Cringely

Pourquoi de l'ingénierie logicielle ?

On veut avoir des raisons de faire confiance aux logiciels

Problèmes technologiques :

- Logiciels de plus en plus massifs et complexes
- Contraintes de fiabilité et de sécurité
- Interactions matériel/logiciel

Problèmes sociaux :

- Présence de clients aux demandes imprécises et fluctuantes, avec qui il faut communiquer
- Développement en (grandes) équipes
- Le cycle de vie des logiciels est long et inclut de la maintenance
- Contraintes légales, problèmes d'image..

Un logiciel peut être dangereux

Pour certains logiciels il y a des vies en jeu :

- dans les transports, la médecine, l'industrie, le nucléaire, les missiles...

De nombreux autres ont un fort pouvoir de nuisance :

- dans les communications, dans les transactions bancaires...

Évolution observée des logiciels :

- des composants plus nombreux
- risque d'accumulation de "petits" problèmes

Un logiciel peut être gros

Windows 7 :

- 1200 personnes (rien que pour les programmeurs)
- Coûts de développement : estimés à 5 milliards
- Revenus : estimés à 20 milliards
- Engagements légaux sur 15 à 20 ans
- Un précédent litige sur Windows Server 98 avait coûté 700 millions

Quelques fiascos mémorables

1962 Mariner 1 (Venus) : mauvais calcul d'une trajectoire, crash en vol.

1985 Therac-25 : radiothérapie surdosée. 5 morts.

1996 Ariane 5 : crash.

1997 .com : blocage de tous les noms de domaine.

1999 Mars Climate Orbiter : un satellite à 120 millions $ perdu pour une confusion entre unités.

2004 SNCF : système de réservation défaillant.

2004 Réseaux Bouygues et France Telecom inopérants.

2005 Régulateur de vitesse Renault Laguna.

2010 NPfIT : coût de 120 milliards £.

Procédures de validation (questions)

Comment justifier et contrôler le processus de développement ?

- Comment garantir la représentativité des tests ?
- Traiter toutes les combinaisons d'options ?
- Comment tester tous les facteurs extérieurs ?
- Qui peut autoriser la mise en service ? Quelle est sa responsabilité ?

Des ordres de grandeur : taille et coût

Taille des systèmes logiciels :

- Système d'exploitation : plusieurs millions (ou dizaines de millions) de lignes de code.
- Navette spatiale : plusieurs dizaines (ou centaines) de millions.

Coûts de développement :

- Codage : ~ 15-20 %
- Validation et vérification : ~ 40 %
- Spécification et conception : ~ 40 %

Des chiffres sur les défauts logiciels

Distribution des défauts :

Code	15%
Algorithmes	25%
Spéc./conception	60%

Distribution des coûts des défauts :

Code	8%
Algorithmes	9%
Spéc./conception	83%

Source : F. Fichot, Cours de conduite de projets, IFIP.,

Difficultés particulières du logiciel

Problèmes dépendants des applications

- Synchronisation des processus
- Nature et volume des données, problèmes algorithmiques
- Interactions matériel/logiciel
- Contraintes de temps-réel

Il est facile de modifier un programme

... mais beaucoup moins de prédire les conséquences.

- Un changement mineur peut être catastrophique.
- Difficile de connaître les probabilités des différentes situations.
- Comment mesurer la qualité ?

Ce qu'il nous faut

Des phases bien définies

- Analyse, Spécification, Conception, Validation

Des processus de développements "industriels"

- Transparents, Traçables => associé à des documents

Conduite de projet: Planification, organisation, détection des problèmes, réaction !
• Estimation des coûts et délais
• Planification (diagrammes PERT ou GANTT)
• Suivi du personnel, des échéances, et des délivrables à fournir au client
• Gestion des risques
• Gestion de la configuration (versions, dépendances)
• Gestion des documents associés aux diverses phases
• ...
et tout ça dans suivant des normes de qualité, des circuits de validation, etc

Phases de développement et documents

Phases de développement et documents

- Cahier des charges informel, besoin fonctionnel, non fonctionnel
- Modèle d'analyse (décrit les besoins opérationnels)
- Modèle de conception (décrit la mise en œuvre/implémentation)
- Code
- Protocoles de validation, documentation

Chaque phase produit et valide le document correspondant

Chaque phase contient des "activités" transverses :

- Production de documentation (manuels d'utilisateur, de référence, d'installation...)
- Validation / Vérification

Phases et activités, 1

Phase d'analyse des besoins (requirements analysis)

- **Étude du contexte de l'application, des contraintes de performance, d'ergonomie, de portabilité...**
- **Produit : cahier des charges (requirement specification)**

Phase d'analyse des spécifications (specification analysis)

- **Ce que le système doit faire (et pas comment le faire)**
- **Produit : modèle d'analyse (analysis model)**

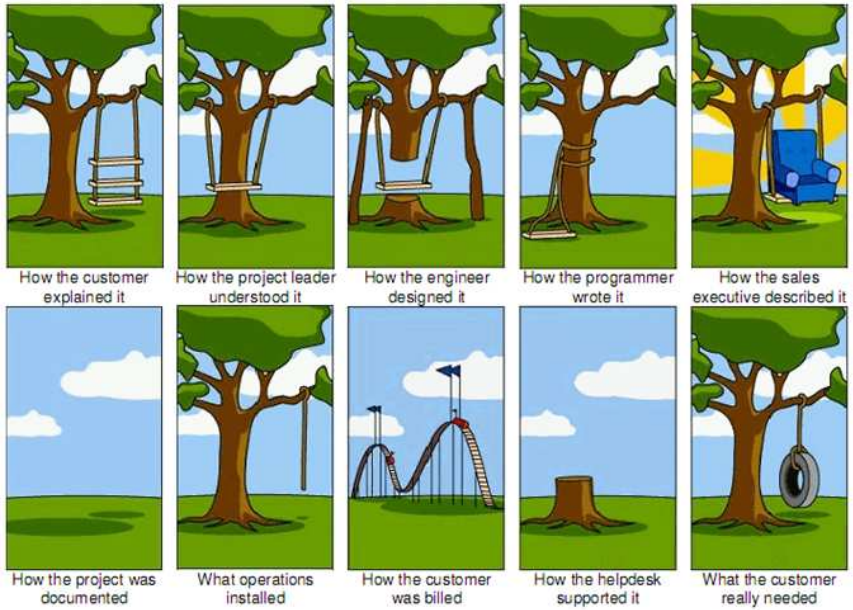
Phase conception architecturale (architectural design)

- **Décomposition en "composants"**
- **Conception de l'intégration et des tests d'intégration**

Phase conception détaillée (design)

- **Choix des algorithmes et des structures de données, définition des interfaces, conception des tests**
- **Produits : documents de conception, interfaces de code, fragments de code, pseudo-code, prototypes**

Ah bon, y faut spécifier ?



Phases et activités, 2

Phase de codage

- C'est maintenant !

Phases et activités, 3

Phase de test unitaire (unit test)

- Test indépendant de chaque méthode (comportement bas-niveau)

Phase d'intégration (integration test)

- Exécution des scénarios de tests conçus pendant l'analyse (comportement haut-niveau)

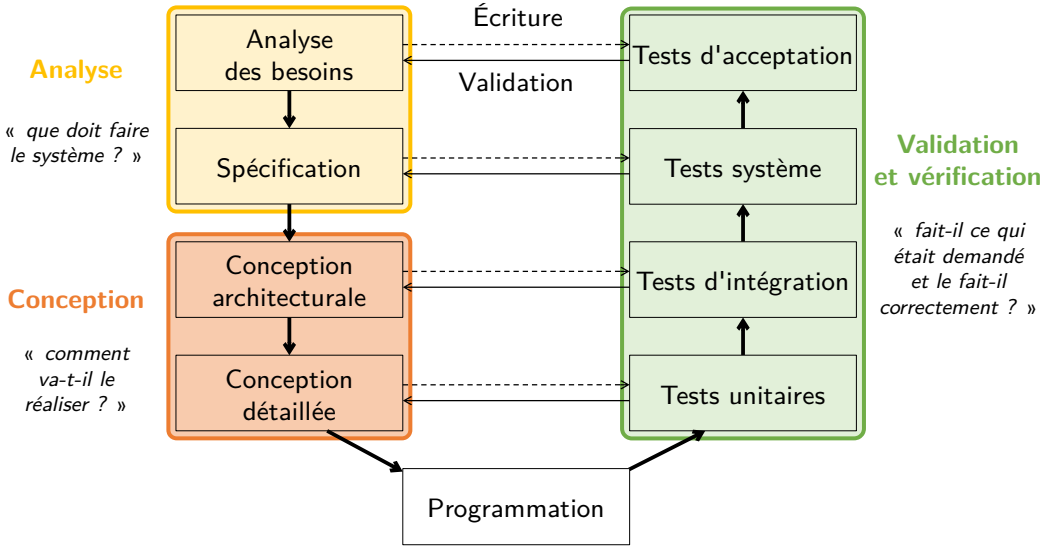
Phase de test du système (acceptance test, system test)

- Test global en conditions réelles, mesure de performances

Phase de déploiement (deployment), par le client

- Test, inspection des documents et des normes qualité

Résumé : Processus de développement en V



Au-delà des phases

La documentation !

- Cibles : utilisateurs, administrateurs, mainteneurs, fournisseurs...

Ne pas confondre deux activités complémentaires :

- Validation : vérifier qu'on construit le bon système

  (ce dont le client a besoin ; cohérence externe)
- Vérification : vérifier qu'on le construit bien

  (ce qui a été spécifié ; cohérence interne)

Assurer la traçabilité de chaque choix.

Réaliser l'archivage, la gestion des versions

- Garantir cohérence des documents avec les différentes versions.
- Garantir possibilité de reconstruire chaque version

Et après le déploiement

La maintenance : c'est compliqué et coûteux. Plusieurs aspects :

- Correction des bugs critiques
- Adaptation à de nouveaux OS, matériels, problèmes de perf.
- Évolution : intégration de nouvelles fonctionnalités

Pire : éventuellement maintenir plusieurs versions en parallèle !

Coûts de maintenance : 2 à 4 fois supérieurs au développement ?

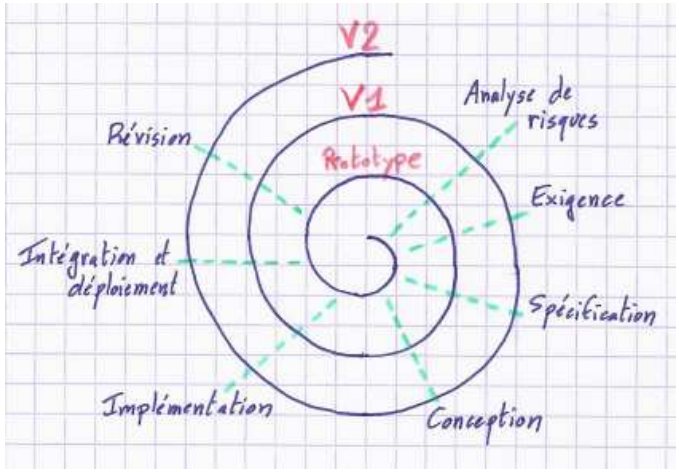
Besoin de tests de régression sur les nouvelles versions :

- Descriptions précises des tests (entrées, sorties, contexte)
- Besoin d'automatisation

Autres processus de développement : Modèle en spirale

- Intégration et validation progressive.
- Retours utilisateur rapides,
- Diminue les risques,
- Meilleure visibilité,

Methode « Agile »
Modéle itératif
Cycle court de production pour mieux intéragir avec le client



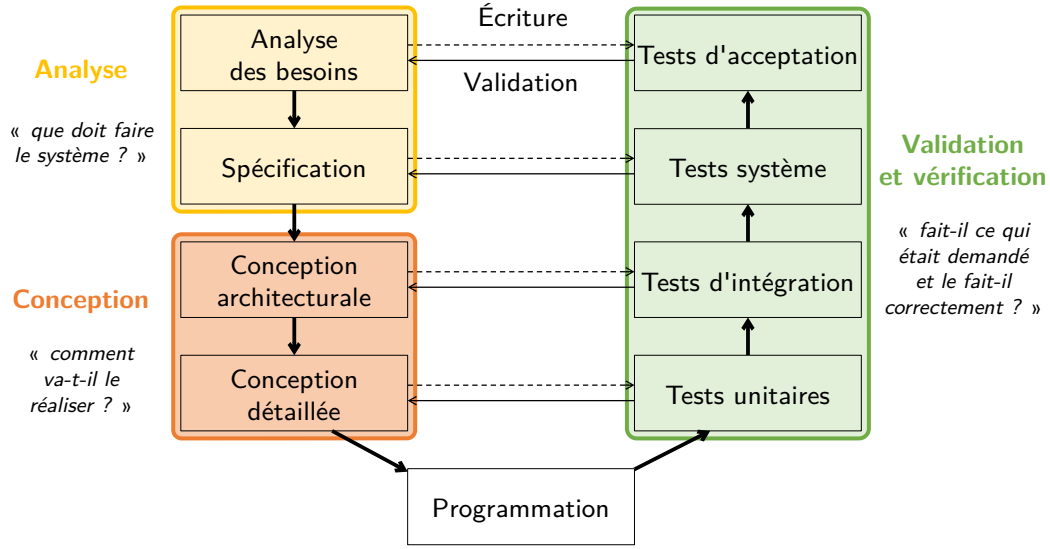
université PARIS-SACLAY

FACULTÉ DES SCIENCES D'ORSAY

UML

Diagrammes de cas d'utilisation

Processus de développement en V



Cas d'utilisation

Objectif : Comprendre les besoins du client pour rédiger le cahier des charges fonctionnel

Trois questions :

1. Définir les utilisations principales du système : à quoi sert-il ?
2. Définir l'environnement du système : qui va l'utiliser ou interagir avec lui ?
3. Définir les limites du système : où s'arrête sa responsabilité ?

Éléments de description :

- Diagramme de cas d'utilisation
- Description textuelle des cas d'utilisation
- Diagrammes de séquence des scénarios d'utilisation

Scénarios d'utilisation

Séquences d'étapes

- décrivant une interaction entre l'utilisateur et le système
- permettant à l'utilisateur de réaliser un objectif

Système : Site de vente en ligne

Scénario : Commander

Le client s'authentifie dans le système puis choisit une adresse et un mode de livraison. Le système indique le montant total de sa commande au client. Le client donne ses informations de paiement. La transaction est effectuée et le système en informe le client par e-mail.

Scénarios d'utilisation

Séquences d'étapes

- décrivant une interaction entre l'utilisateur et le système
- permettant à l'utilisateur de réaliser un objectif

Système : Site de vente en ligne

Scénario : Commander

Le client s'authentifie dans le système puis choisit une adresse et un mode de livraison. Le système indique le montant total de sa commande au client. Le client donne ses informations de paiement. La transaction n'est pas autorisée, le système invite le client à changer de mode de paiement. Le client modifie ses informations. La transaction est effectuée et le système en informe le client par e-mail.

Cas d'utilisation

- Ensemble de scénarios réalisant un objectif de l'utilisateur
- Fonctionnalités principales du système du point de vue extérieur

Acteur : Entité qui interagit avec le système

- Personne, chose, logiciel, extérieur au système décrit
- Représente un rôle (plusieurs rôles possibles pour une même entité)
- Identifié par le nom du rôle

Cas d'utilisation : Fonctionnalité visible de l'extérieur

- Action déclenchée par un acteur
- Identifié par une action (verbe à l'infinitif)

Vision du système centrée sur l'utilisateur

Diagramme de cas d'utilisation

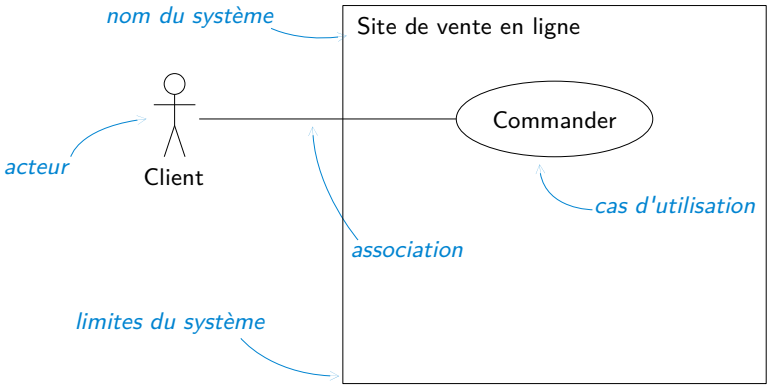
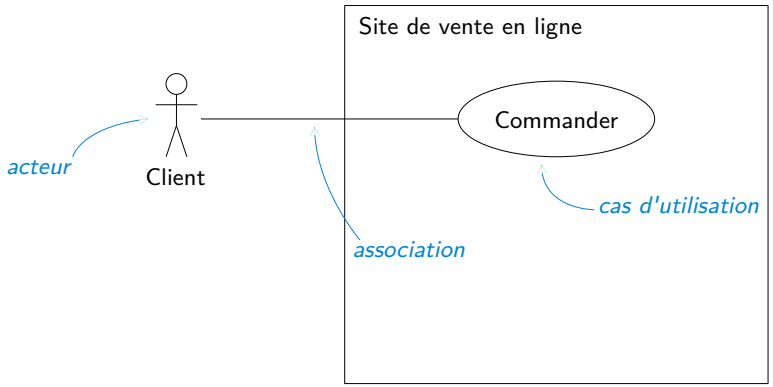


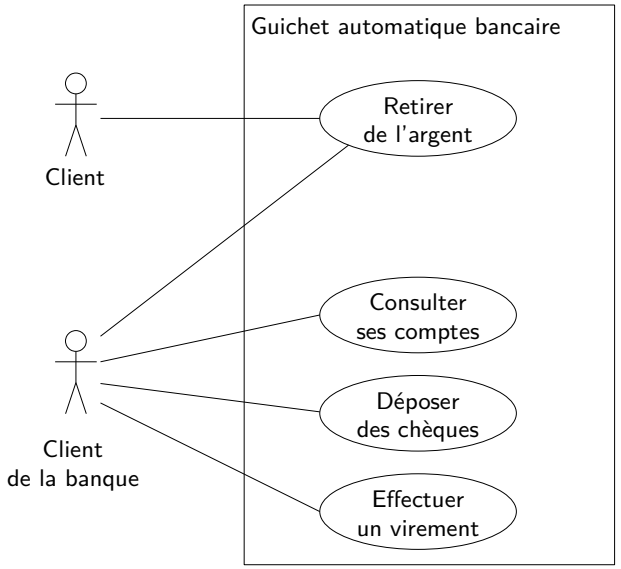
Diagramme de cas d'utilisation



Association :

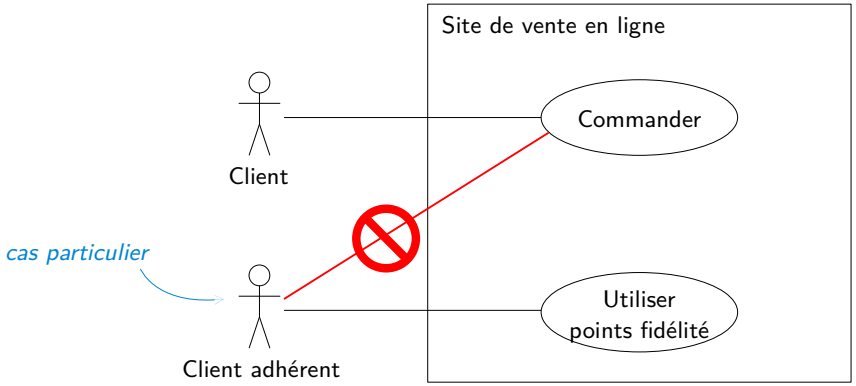
- Relation entre acteurs et cas d'utilisation
- Représente la possibilité pour l'acteur de déclencher le cas

Exemple



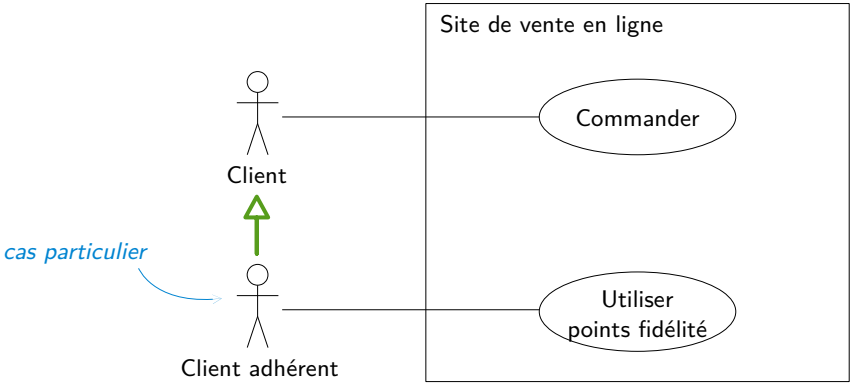
Conseil : Pas plus de 6 ou 8 cas

Généralisation de rôle



Situation : Y peut faire tout ce que fait X

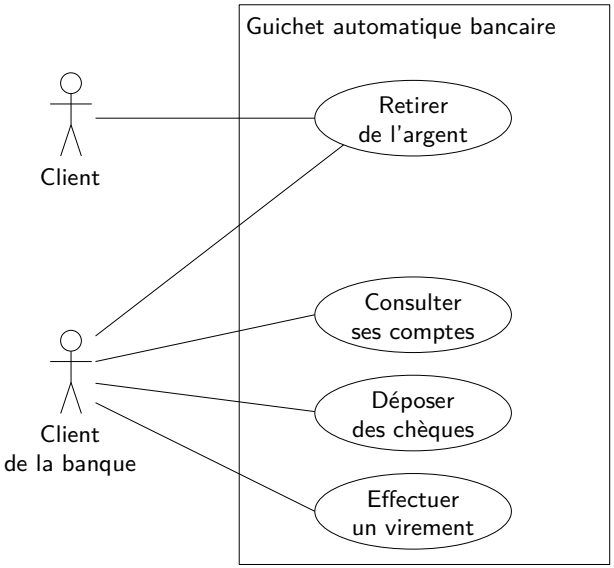
Généralisation de rôle



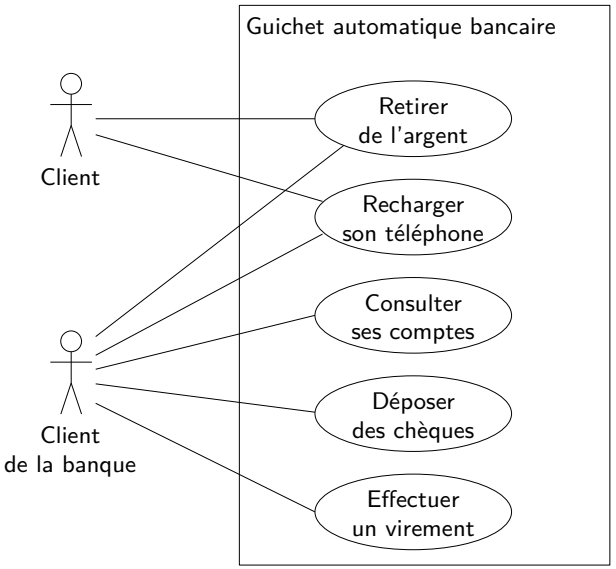
Situation : Y peut faire tout ce que fait X

Modélisation : Faire apparaître Y comme un cas particulier de X
(ou X généralisation de Y)

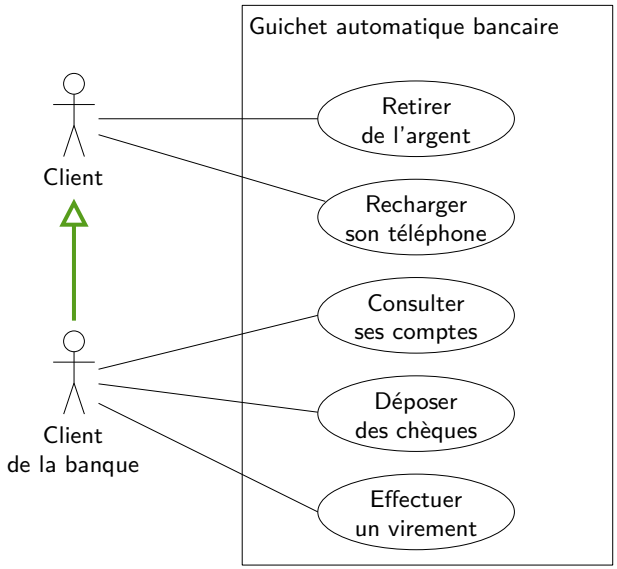
Exemple : généralisation de rôle



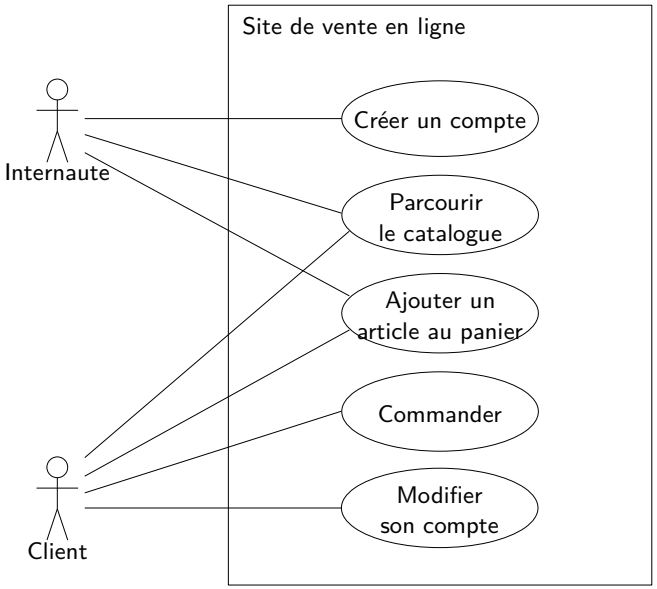
Exemple : généralisation de rôle



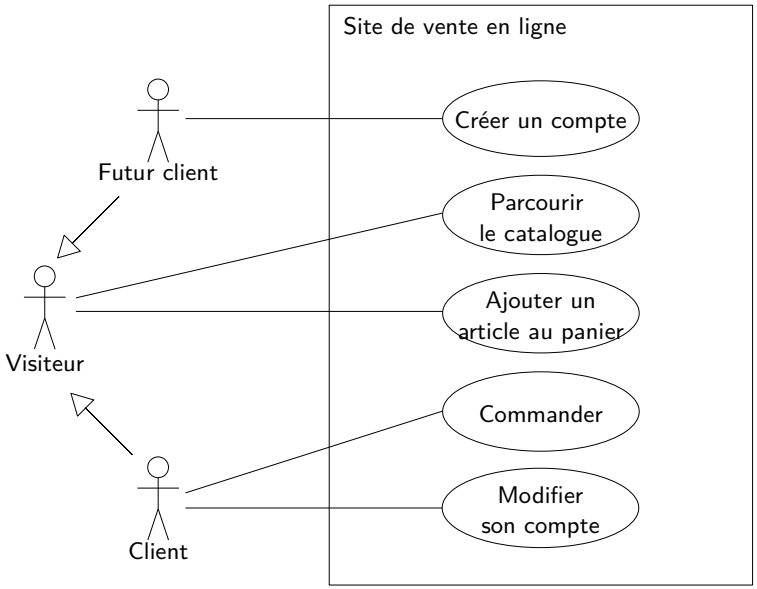
Exemple : généralisation de rôle



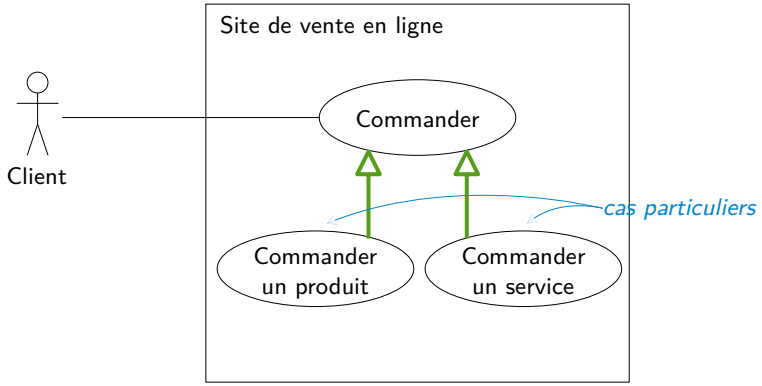
Autre exemple de généralisation de rôle



Autre exemple de généralisation de rôle

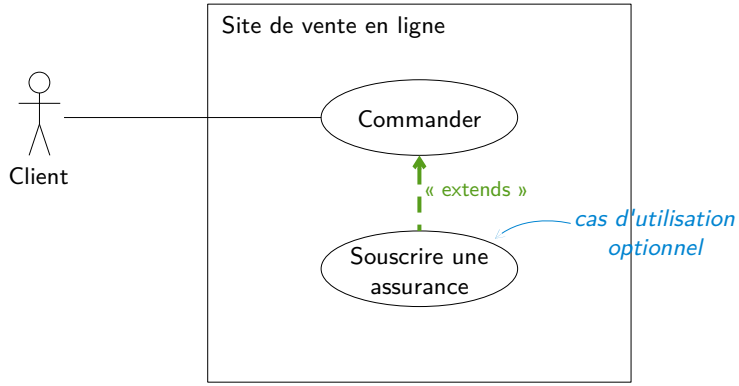


Relations entre cas d'utilisation



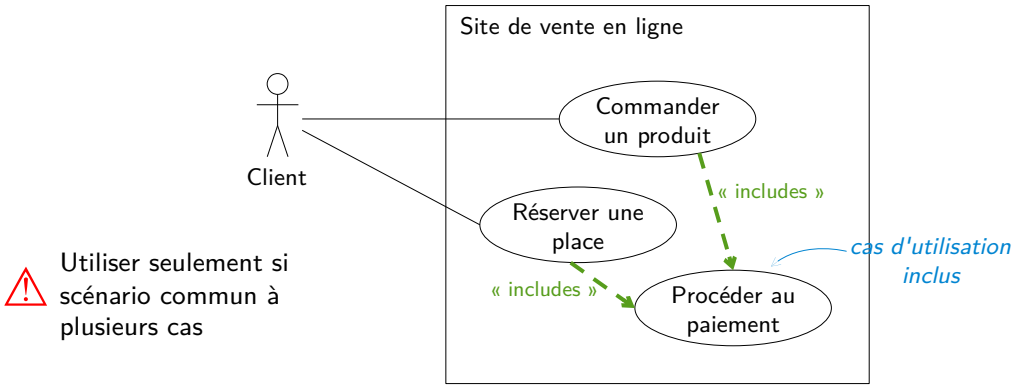
Généralisation : X est un cas particulier de Y
Tout ou partie du scénario de Y est spécifique à X

Relations entre cas d'utilisation



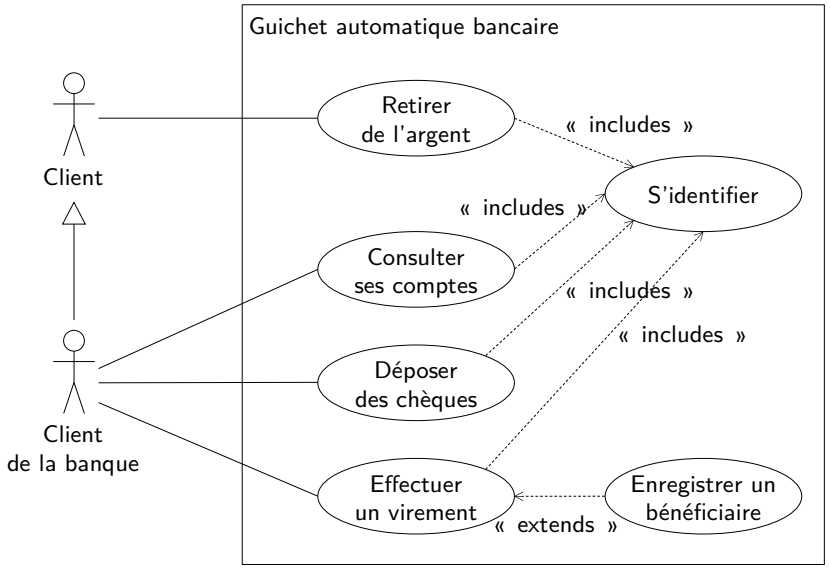
Extension : X « extends » Y
- Cas d'utilisation X peut être déclenché au cours du scénario de Y
- X est optionnel pour Y

Relations entre cas d'utilisation



Inclusion : X « includes » Y
- Scénario de Y inclus dans le scénario de X
- Cas d'utilisation Y déclenché au cours du scénario de X

Exemple : relations entre cas d'utilisation



Conseil : Relations entre cas seulement si nécessaires et pas trop lourds

université PARIS-SACLAY

FACULTÉ DES SCIENCES D'ORSAY

UML

Scénarios détaillés et diagrammes de séquence

Description textuelle des cas d'utilisation

Diagrammes de cas d'utilisation

- Utiles pour la discussion avec le client car intuitifs et concis
- Pas suffisants pour l'équipe de développement

Nécessité d'une description détaillée des scénarios représentés par chacun des cas :

- Description textuelle en langue naturelle structurée
- Explication du vocabulaire utilisé dans les diagrammes

Cas d'utilisation détaillé

Description textuelle d'un cas d'utilisation

- Nom du cas d'utilisation
- Brève description
- Acteurs
- Contexte
- Données en entrée et pré-conditions
- Données en sortie et post-conditions
- Scénario principal pour ce cas d'utilisation : étapes à suivre pour réaliser ce cas
- Variantes, cas d'erreur : déviations des étapes du scénario principal, scénarios alternatifs, scénarios d'erreur

Cas d'utilisation détaillé

Nom : Commander

Acteur : Client

Données d'entrée : Produits sélectionnés par le client

Le cas d'utilisation commence lorsque le client clique sur le bouton « Commander »

Scénario principal :

1. Le système demande au client de saisir son identifiant et son mot de passe
2. Le client saisit son identifiant et son mot de passe et valide
3. Le système demande au client de choisir son adresse de livraison parmi sa liste d'adresses ou d'en saisir une nouvelle
4. Le client choisit une adresse de livraison et valide
5. Le système demande au client de choisir un mode d'expédition parmi une liste prédéfinie (à préciser)
6. Le client choisit un mode d'expédition et valide

Cas d'utilisation détaillé (suite)

7. Le système affiche un récapitulatif de la commande, indique le montant total de la livraison et demande au client de choisir un mode de paiement parmi une liste prédéfinie (à préciser)

8. Le client choisit un mode de paiement et valide

9. Le système demande au client de saisir ses informations de paiement

10. Le client saisit ses informations de paiement et valide

11. Le système informe le client que la transaction s'est effectuée correctement et un e-mail récapitulatif de la commande est envoyé au client

Cas d'utilisation détaillé

Scénario d'erreur : Client inconnu

3a. Le client n'est pas connu du système. Le système affiche un message d'erreur. Retour à l'étape 1.

Scénario alternatif : Nouvelle adresse de livraison

4a. Le client saisit une nouvelle adresse de livraison et valide.

Le scénario reprend à l'étape 5

Scénario alternatif : Modifications des choix de livraison

8a. Le client demande à modifier son adresse de livraison.

Retour à l'étape 3.

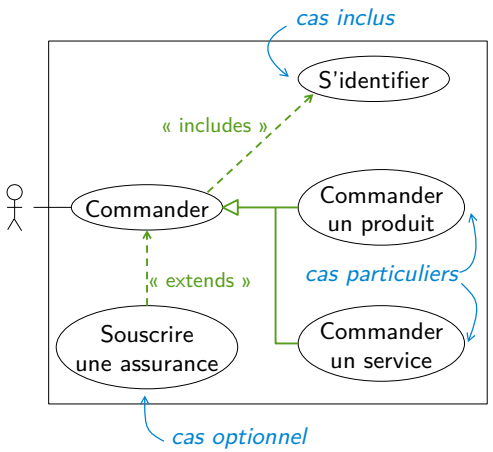
8b. Le client demande à modifier le mode de livraison.

Retour à l'étape 5.

Scénario d'erreur : Transaction impossible

11a. Le système informe le client que ses informations de paiement sont incorrectes. Retour à l'étape 9.

Exemple de liens entre diagramme et texte



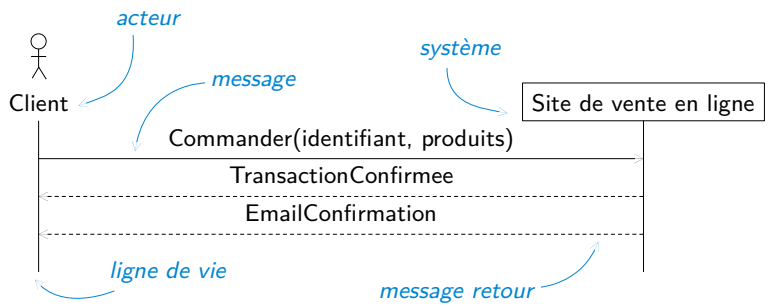
Commander
Acteur : Client
Le cas commence lorsque le client clique sur « Commander »
Scénario principal
1. Déclenchement du cas S'identifier
2. Fin du cas S'identifier
3. ...
4.1 L'objet de la commande est un produit. Déclenchement du cas Commander un produit.
4.2 L'objet de la commande est un service. Déclenchement du cas Commander un service.

Scénario alternatif
3a. Le client choisit de prendre une assurance.

Diagramme de séquence (analyse)

Représentation graphique de la chronologie des échanges de messages entre les acteurs et le système

- Temps représenté verticalement
- Échanges de messages représentés horizontalement

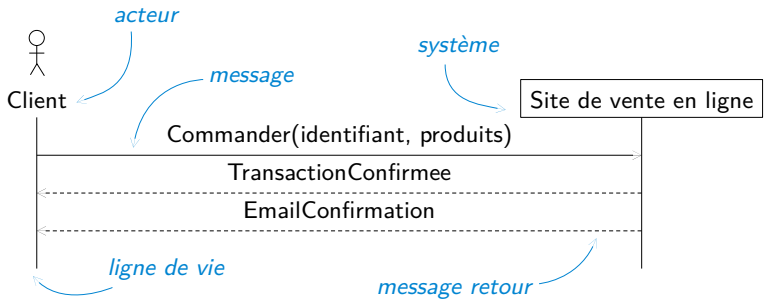


Cas d'utilisation

Diagramme de séquence (analyse)

Niveau analyse

- Messages informels (pas des appels de méthodes)
- Noms des messages liés aux cas d'utilisation
- Mise en avant des données utiles au scénario (arguments)

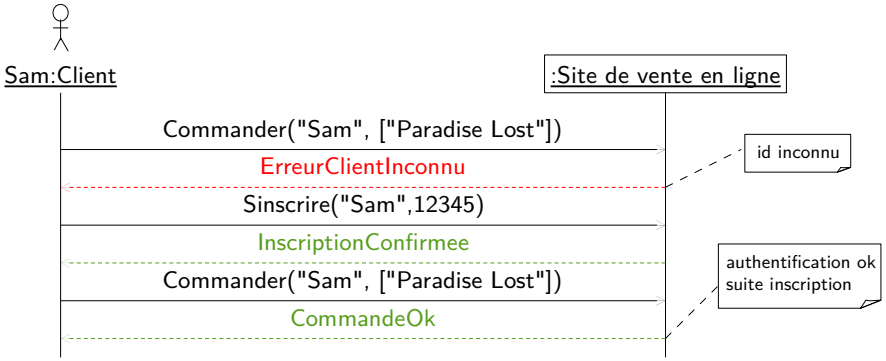


Cas d'utilisation

Scénario d'utilisation concret

Principe : Variables remplacées par des valeurs concrètes pour

- illustrer les différents scénarios d'un cas d'utilisation
- mettre en évidence les relations entre les différents cas
- construire des scénarios d'utilisation complexes pour le test

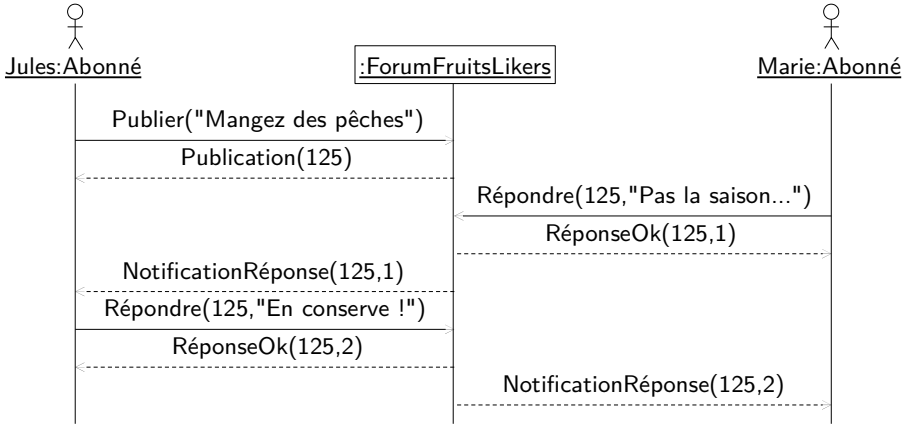


Mise en évidence de la nécessité d'être inscrit pour pouvoir commander

Scénario avec plusieurs acteurs

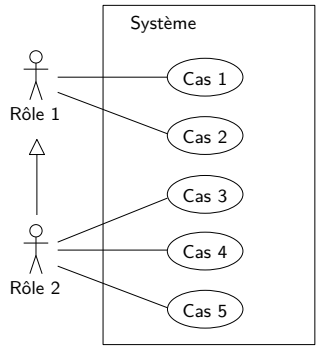
Scénario d'utilisation = interactions entre les acteurs et le système

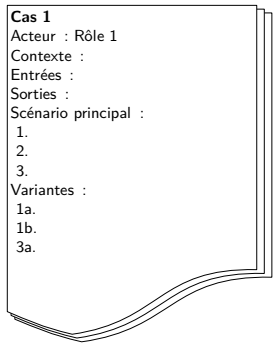
Pas de messages entre acteurs



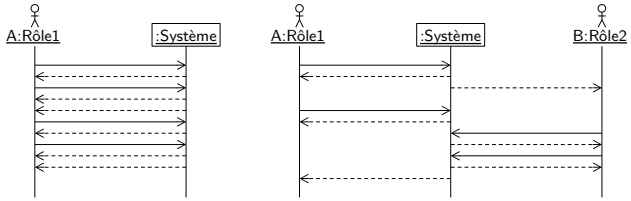
Spécification des cas d'utilisation

Diagrammes de cas d'utilisation + Description textuelle





+ Scénarios d'utilisation



université PARIS-SACLAY

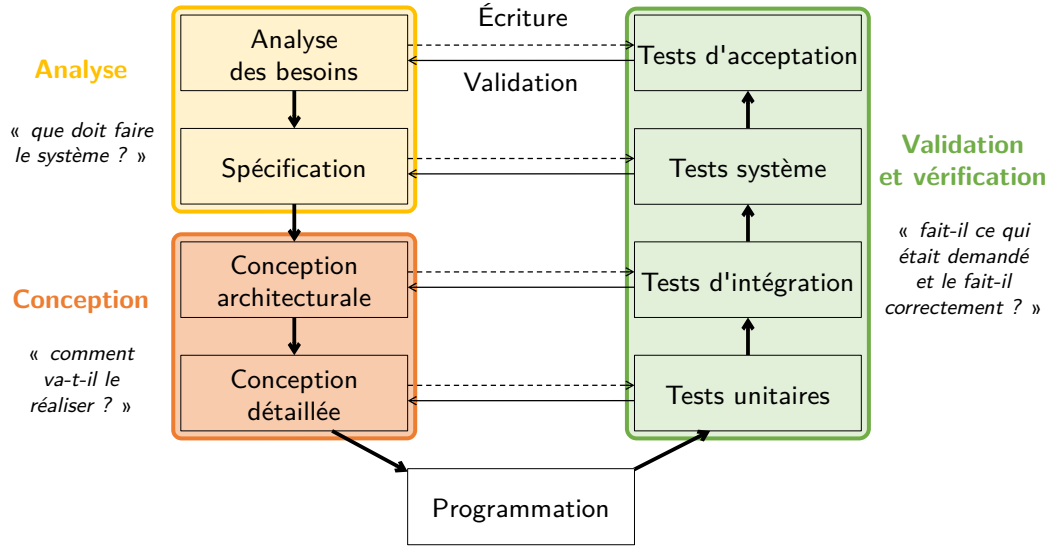
FACULTÉ DES SCIENCES D'ORSAY

UML

Diagrammes de classes
Diagrammes d'objets

Delphine Longuet
delphine.longuet@lri.fr

Processus de développement en V



Objets et classes

Conception orientée objet : Représentation du système comme un ensemble d'objets interagissant

Diagramme de classes

- Représentation de la structure interne du logiciel
- Utilisé surtout en conception mais peut être utilisé en analyse

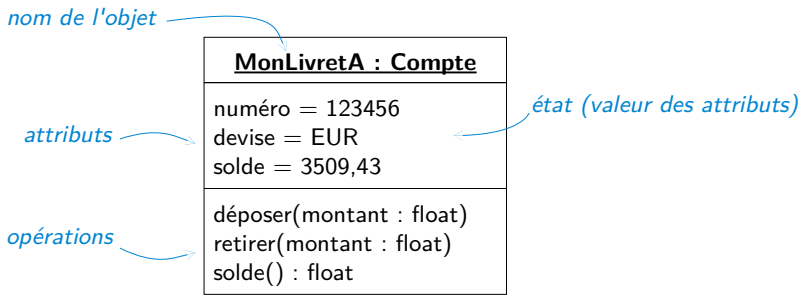
Diagramme d'objets

- Représentation de l'état du logiciel (objets + relations)
- Diagramme évoluant avec l'exécution du logiciel
  - création et suppression d'objets
  - modification de l'état des objets (valeurs des atributs)
  - modification des relations entre objets

Objets et classes

Objet

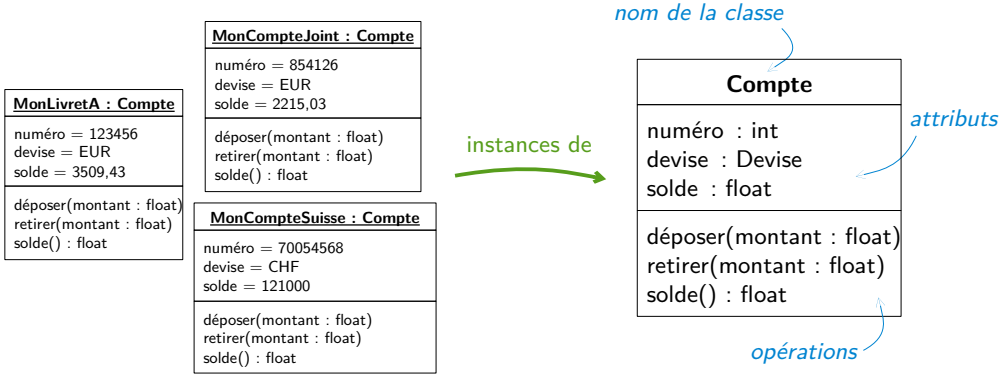
- Entité concrète ou abstraite du domaine d'application
- Décrit par : identité (adresse mémoire)
  + état (attributs)
  + comportement (opérations)



Objets et classes

Classe : Regroupement d'objets de même nature (mêmes attributs + mêmes opérations)

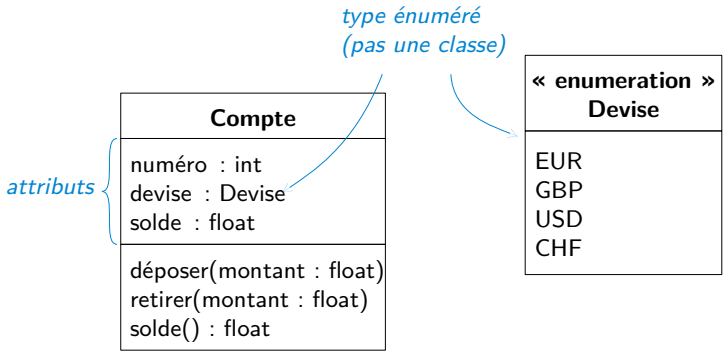
Objet = instance d'une classe



Classes

Attributs

- Caractéristique partagée par tous les objets de la classe
- Associe à chaque objet une valeur
- Type associé simple (int, bool...), primitif (Date) ou énuméré



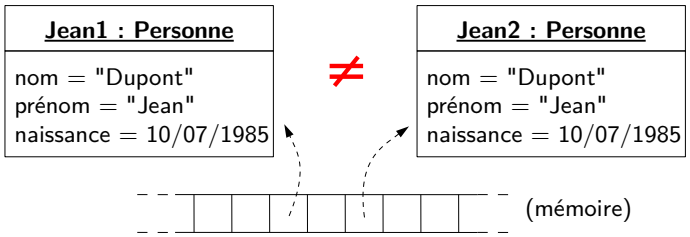
Classes

Attributs

- Caractéristique partagée par tous les objets de la classe
- Associe à chaque objet une valeur
- Type associé simple (int, bool...), primitif (Date) ou énuméré

Valeur des attributs : État de l'objet

- Objets différents (identités différentes) peuvent avoir mêmes attributs

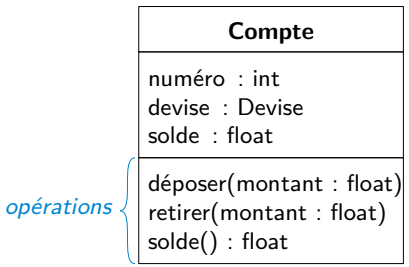


Classes

Opérations

- Service qui peut être demandé à tout objet de la classe
- Comportement commun à tous les objets de la classe

⚠ Ne pas confondre avec une méthode = implantation de l'opération



Exemple de la bibliothèque

On cherche à développer un système qui gère les emprunts et les retours dans une bibliothèque.

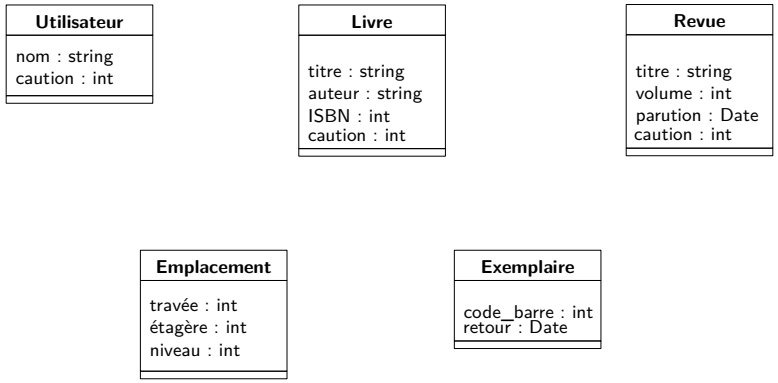
La bibliothèque gère des livres et des revues. Un livre est caractérisé par son titre, son auteur et son code ISBN. Un numéro de revue est caractérisé par le titre de la revue, un numéro de volume et sa date de parution. Chaque exemplaire d'une ressource est caractérisé par un code barre au sein de la bibliothèque.

Pour emprunter un ouvrage, un utilisateur doit être enregistré. Il s'enregistre auprès du bibliothécaire en donnant son nom et une caution. Chaque ouvrage a une caution. Un utilisateur ne peut emprunter un ouvrage que si la caution qui lui reste sur son compte est supérieure à la caution de l'ouvrage. La durée de l'emprunt est fixée à 15 jours.

On ne peut pas emprunter plus d'un exemplaire d'une même ressource, ni emprunter une nouvelle ressource si on est en retard pour rendre une ressource.

L'emplacement de stockage d'un ouvrage dans la bibliothèque est représenté par un numéro de travée, un numéro d'étagère dans la travée, et un niveau. Différentes ressources peuvent être rangées au même emplacement, mais tous les exemplaires d'une même ressource sont stockés au même endroit.

Exemple de la bibliothèque (1)

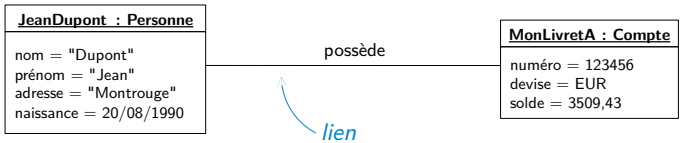


Note : si un exemplaire n'est pas emprunté, retour a la valeur *null*

Relations entre objets

Lien entre objets

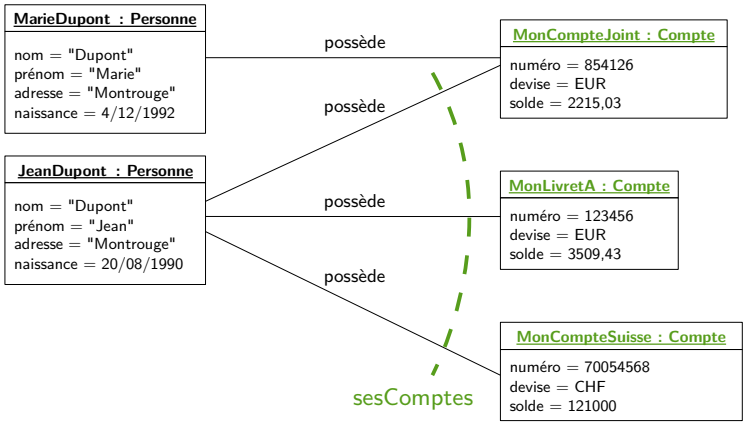
- relation binaire (en général)
- au plus un lien entre deux objets (pour une association)



Relations entre objets

Lien entre objets

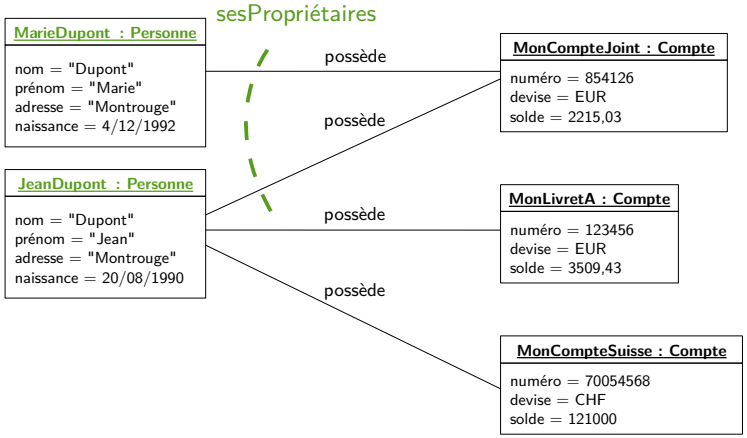
- relation binaire (en général)
- au plus un lien entre deux objets (pour une association)



Relations entre objets

Lien entre objets

- relation binaire (en général)
- au plus un lien entre deux objets (pour une association)

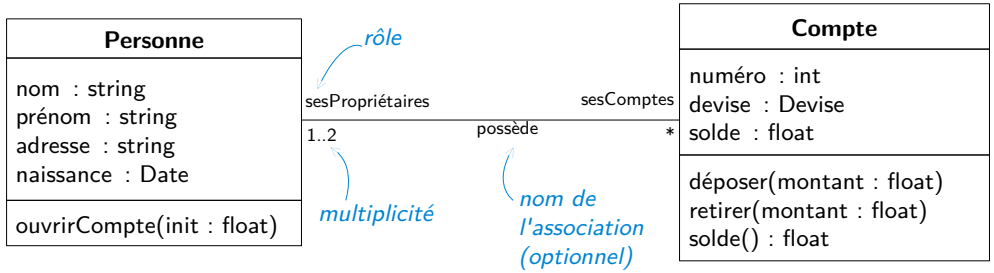


Relations entre classes

Association entre classes : Relation binaire (en général)

Rôle : Nomme l'extrémité d'une association, permet d'accéder aux objets liés par l'association à un objet donné

Multiplicité : Contraint le nombre d'objets liés par l'association

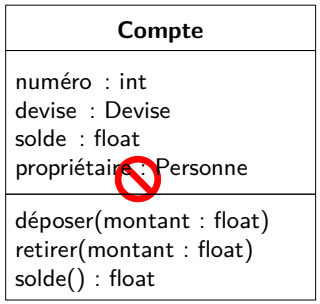


Lien = instance d'association

Attribut et association

Rappel : Types des attributs simple, primitif ou énuméré

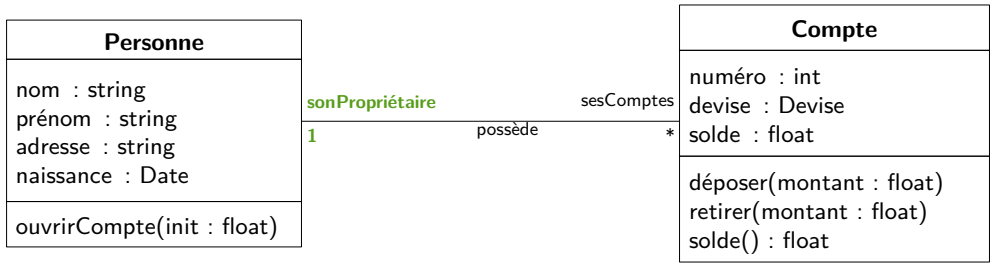
En particulier, pas d'attribut dont le type est une classe du diagramme



Attribut et association

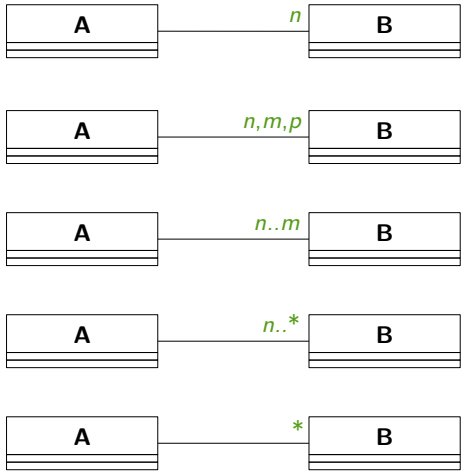
Rappel : Types des attributs simple, primitif ou énuméré

En particulier, pas d'attribut dont le type est une classe du diagramme Mais association vers cette classe



Multiplicités

Nombre d'objets de la classe B associés à un objet de la classe A



Exactement n

Exactement n ou m ou p

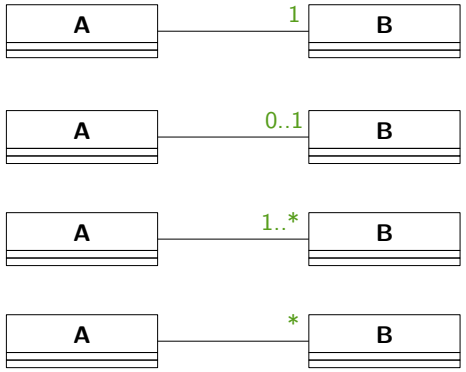
Entre n et m

Au moins n

Plusieurs (0 ou plus)

Multiplicités en pratique

Nombre d'objets de la classe B associés à un objet de la classe A



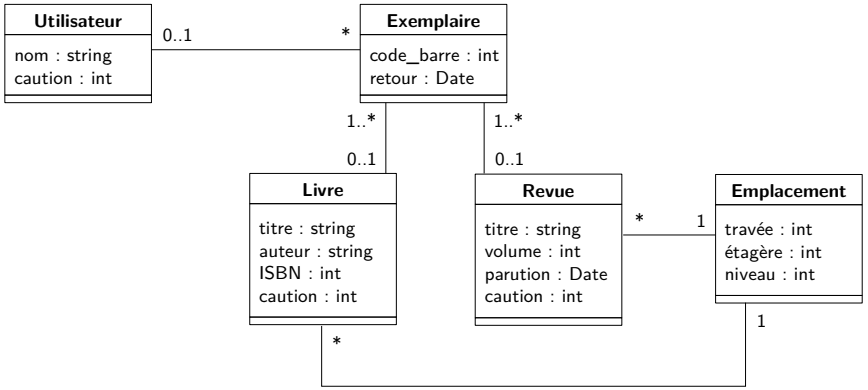
Exactement 1

Au plus 1 (0 ou 1)

Au moins 1 (jamais 0)

0 ou plus

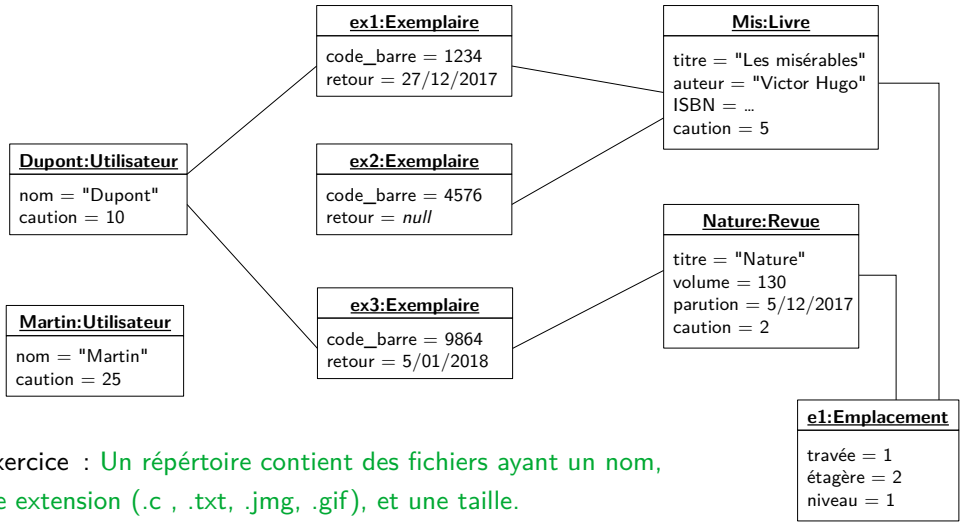
Exemple de la bibliothèque (2)



Notes : Si un exemplaire n'est pas emprunté, retour a la valeur *null*
Un exemplaire est un exemplaire d'un livre ou d'une revue

Exemple de la bibliothèque (2)

Exemple de diagramme d'objets



Exercice : Un répertoire contient des fichiers ayant un nom, une extension (.c , .txt, .jmg, .gif), et une taille.
Un répertoire a aussi un nom. Faire Diagramme classe + objets

université PARIS-SACLAY

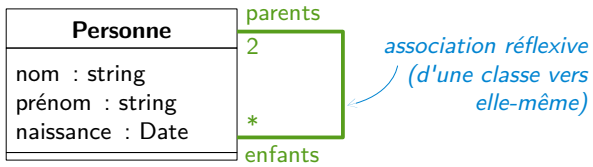
FACULTÉ DES SCIENCES D'ORSAY

UML

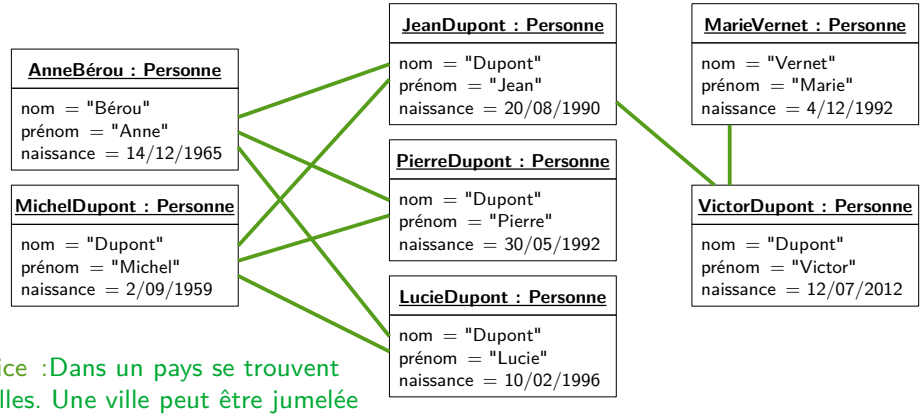
Associations particulières, héritage

Association réflexive

Diagramme de classes



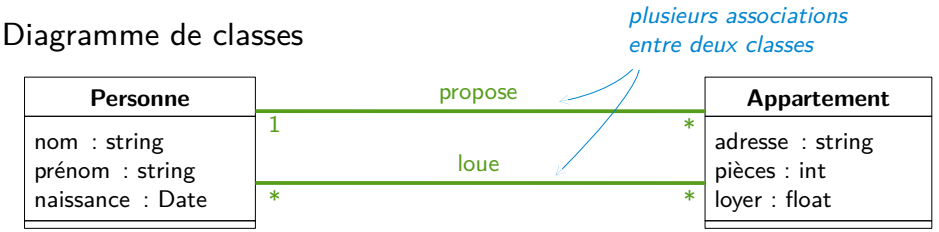
Exemple de diagramme d'objets



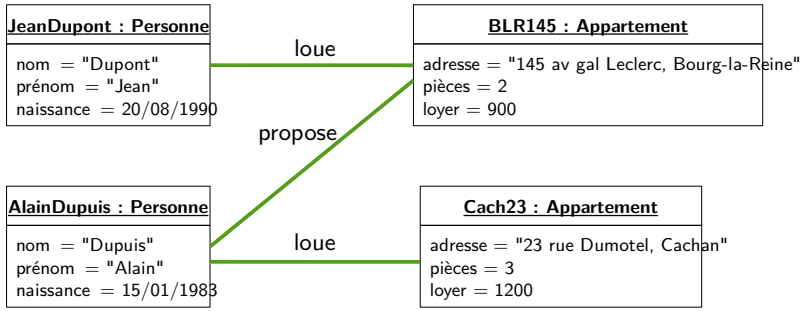
Exercice :Dans un pays se trouvent des villes. Une ville peut être jumelée à une ville d'un autre pays. Diagramme ?

Associations multiples

Diagramme de classes



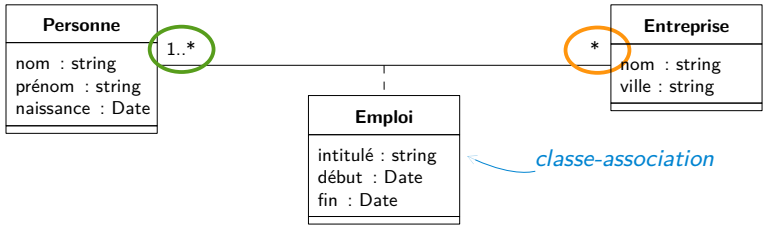
Exemple de diagramme d'objets



Exercice : on rajoute que chaque pays a une capitale

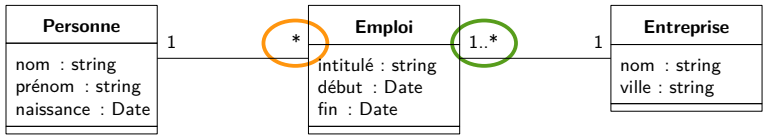
Classe-association

Permet de paramétrer une association entre deux classes par une classe



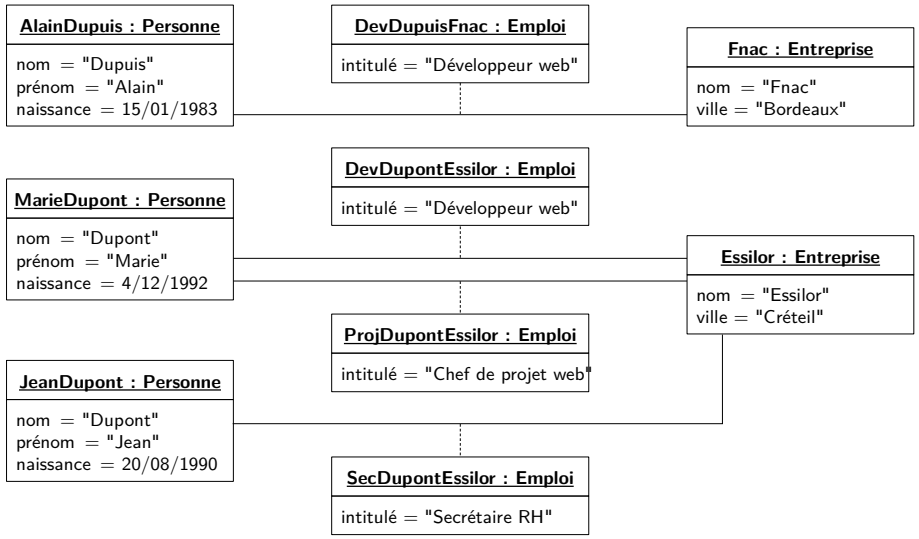
Instance unique de la classe-association pour chaque lien entre objets

Équivalence en termes de classes et d'associations :



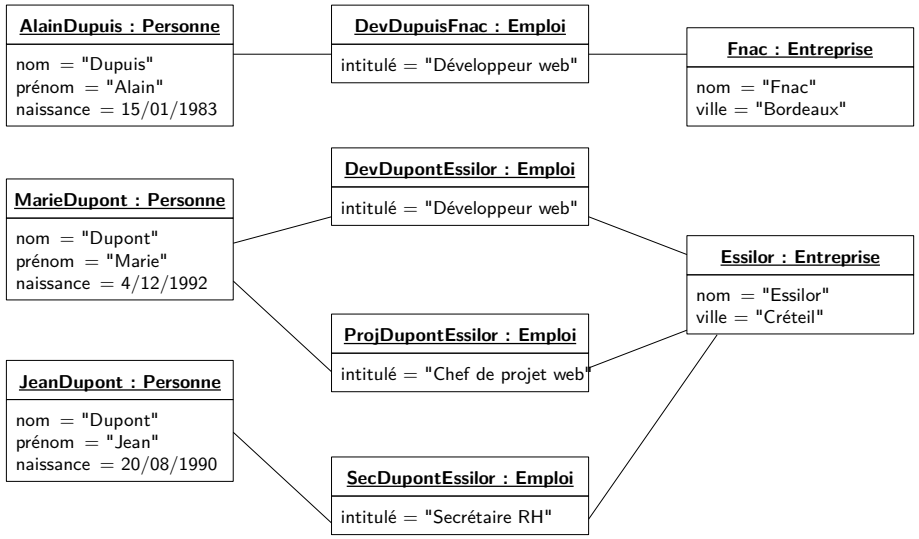
Classe-association

Exemple de diagramme d'objets



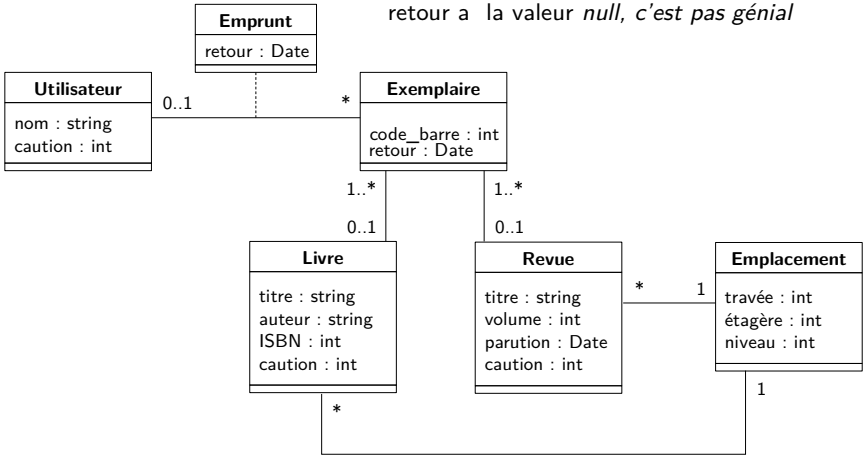
Classe-association

Diagramme d'objets équivalent



Exemple de la bibliothèque (3)

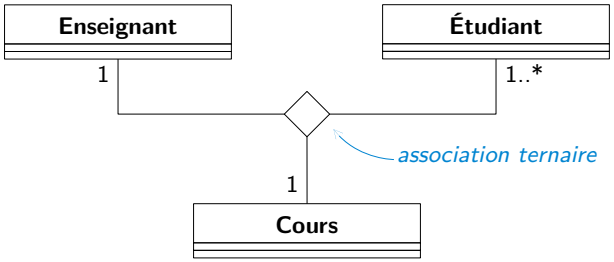
Si un exemplaire n'est pas emprunté, retour a la valeur *null, c'est pas génial*



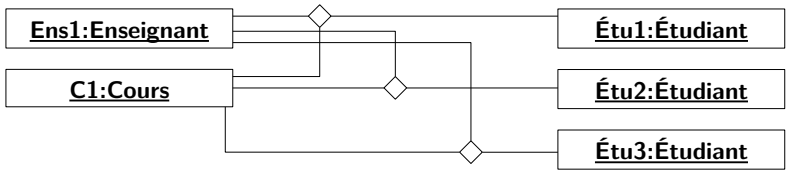
Exercice : Une commande de pizzas est composée d'une ou plusieurs pizzas proposées à la carte, chaque pizza pouvant être présente plusieurs fois dans la commande.

Association *n*-aire

Association reliant plus de deux classes

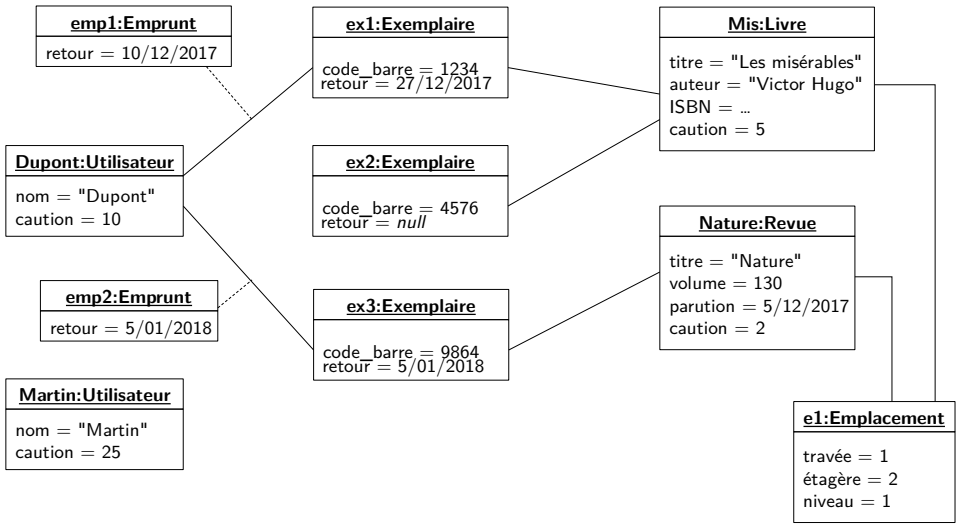


Instance d'une association *n*-aire = lien entre *n* objets



Exemple de la bibliothèque (3)

Exemple de diagramme d'objets

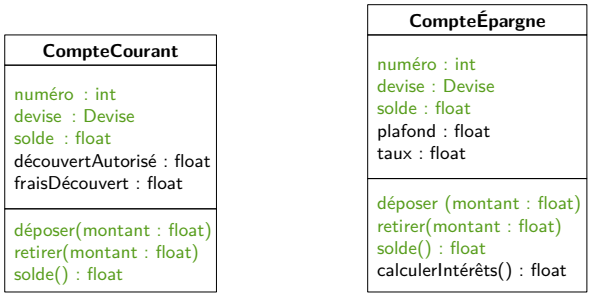


Hiérarchie de classes

Principe : Regrouper les classes partageant des attributs et des opérations et les organiser en arborescence

Spécialisation : raffinement d'une classe en une sous-classe

Généralisation : abstraction d'un ensemble de classes en super-classe

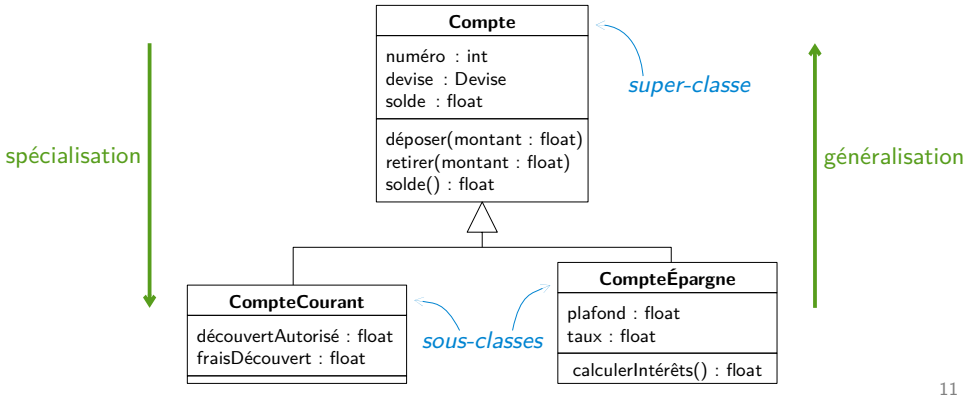


Hiérarchie de classes

Principe : Regrouper les classes partageant des attributs et des opérations et les organiser en arborescence

Spécialisation : raffinement d'une classe en une sous-classe

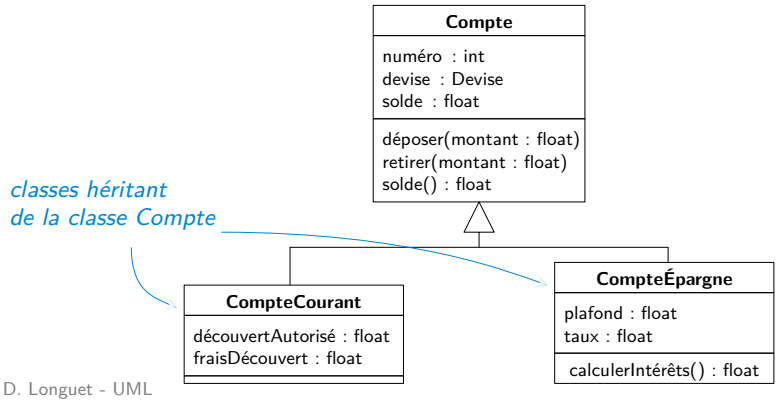
Généralisation : abstraction d'un ensemble de classes en super-classe



Hiérarchie de classes

Principe : Regrouper les classes partageant des attributs et des opérations et les organiser en arborescence

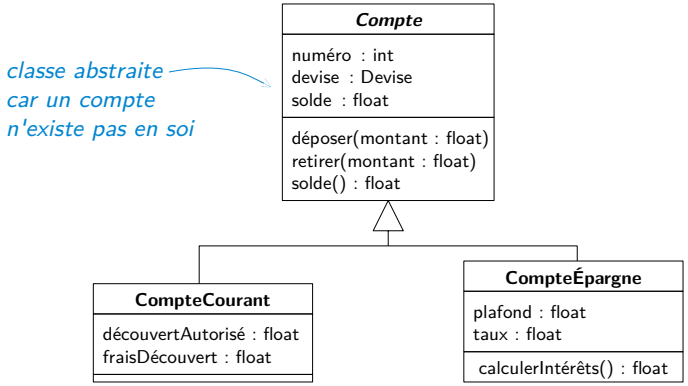
Héritage : Construction d'une classe à partir d'une classe plus haute dans la hiérarchie (partage des attributs, opérations, contraintes...)



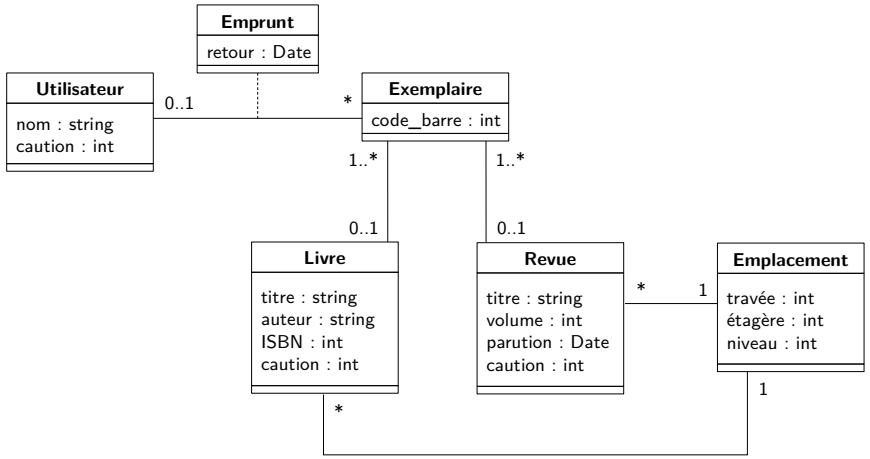
Classe abstraite

Classe sans instance, seulement une base pour classes héritées

Notation : nom de la classe en italique (ou stéréotype « abstract »)

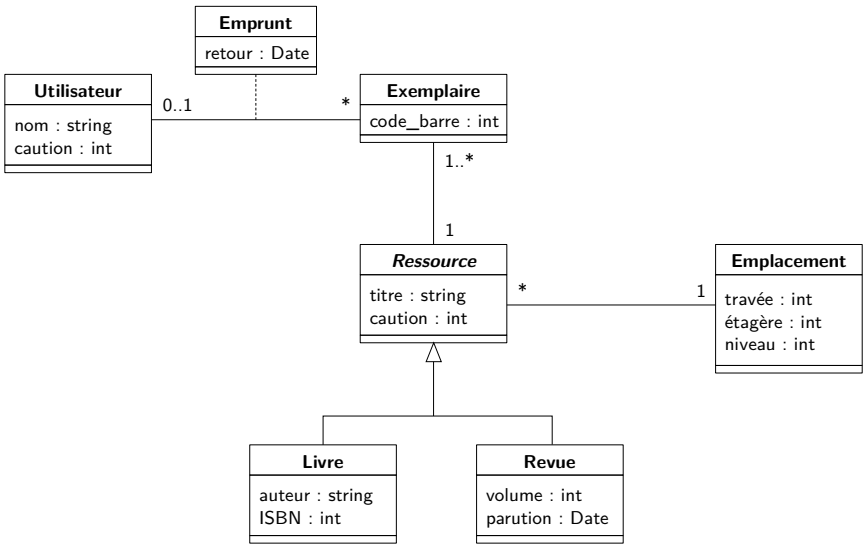


Exemple de la bibliothèque (4)



Note : Un exemplaire est un exemplaire d'un livre ou d'une revue

Exemple de la bibliothèque (4)

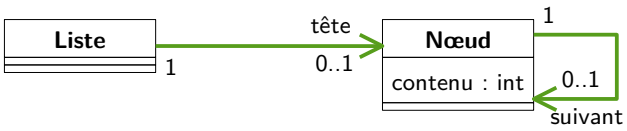


Navigabilité

Orientation d'une association

- Restreint l'accessibilité des objets
- Depuis un A, on a accès aux objets de B qui lui sont associés, mais pas l'inverse

Exemple (listes chaînées)



Par défaut, associations navigables dans les deux sens (pas de flèche)

Agrégation

Association particulière entre classes

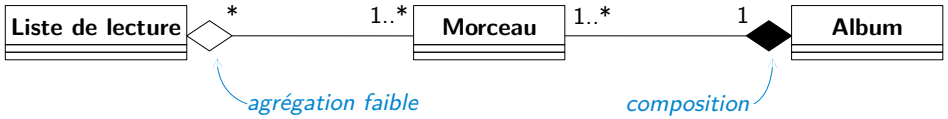
- Dissymétrique : une classe prédominante sur l'autre
- Relation de type composant-composite

Deux types d'agrégation

- Agrégation faible
- Composition

Exemple

Lecteur de contenu audio permettant de créer des listes de lecture



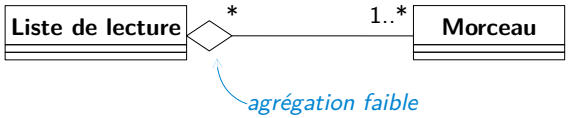
Agrégation faible

Agrégation par référence

- Le composite fait référence à ses composants
- La création ou destruction du composite est indépendante de la création ou destruction de ses composants
- Un objet peut faire partie de plusieurs composites à la fois

Exemple

- Une liste de lecture est composée d'un ensemble de morceaux
- Un morceau peut appartenir à plusieurs listes de lecture
- Supprimer la liste ne supprime pas les morceaux



Composition

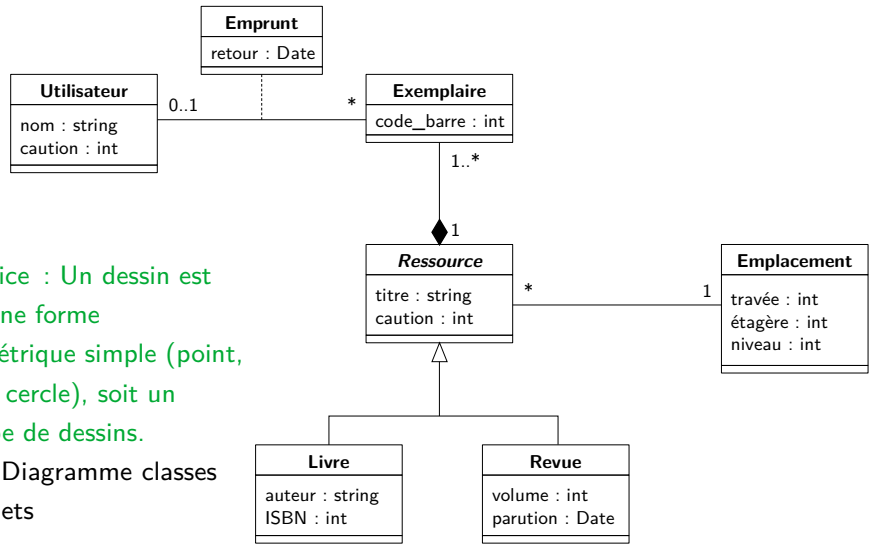
Agrégation par valeur

- Le composite contient ses composants
- La création ou destruction du composite entraîne la création ou destruction de ses composants
- Un objet ne fait partie que d'un composite à la fois

Exemple

- Un morceau n'appartient qu'à un album
- La suppression de l'album entraîne la suppression de tous ses morceaux

Exemple de la bibliothèque (5)



Exercice : Un dessin est soit une forme géométrique simple (point, ligne, cercle), soit un groupe de dessins.

Faire Diagramme classes + objets

université PARIS-SACLAY

FACULTÉ DES SCIENCES D'ORSAY

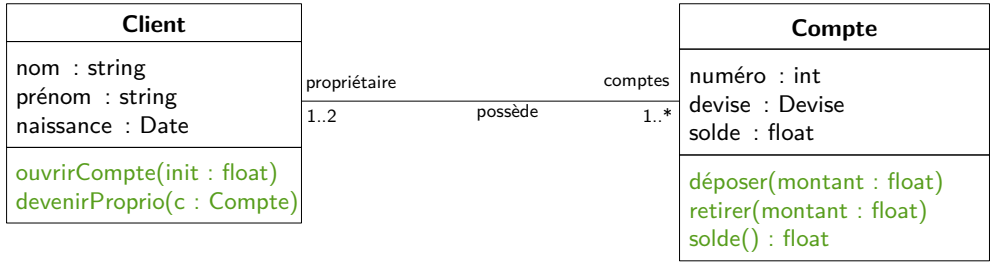
UML

Opérations

Opérations

Opération

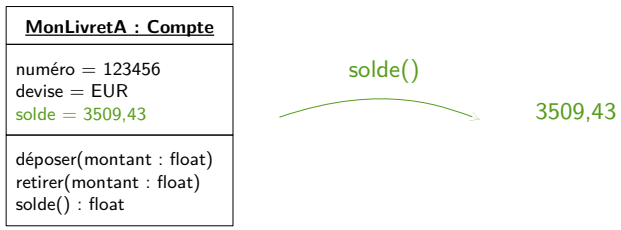
- Service qui peut être demandé à tout objet de la classe
- Comportement commun à tous les objets de la classe



Opérations

Effets possibles d'une opération

- Renvoyer le résultat d'un calcul



Opérations

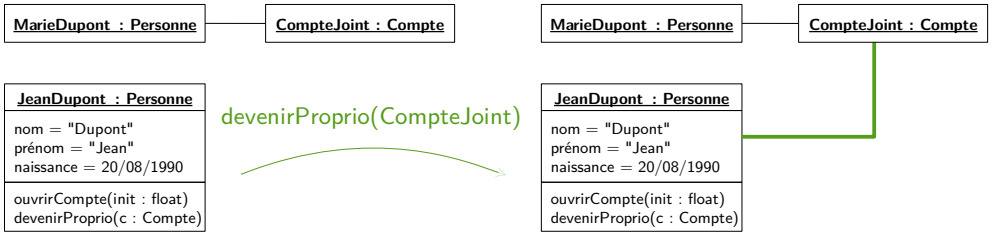
Effets possibles d'une opération

- Renvoyer le résultat d'un calcul
- Modifier l'état du système
  - modification de la valeur des attributs

Opérations

Effets possibles d'une opération

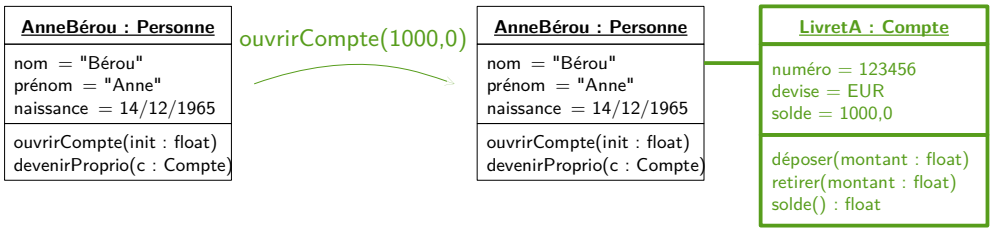
- Renvoyer le résultat d'un calcul
- Modifier l'état du système
  - modification de la valeur des attributs
  - ajout/suppressions de liens entre objets



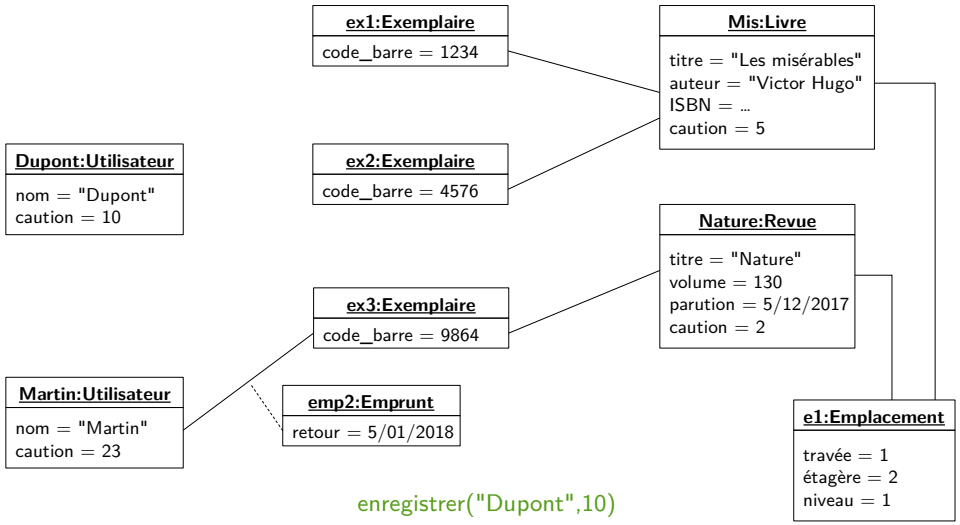
Opérations

Effets possibles d'une opération

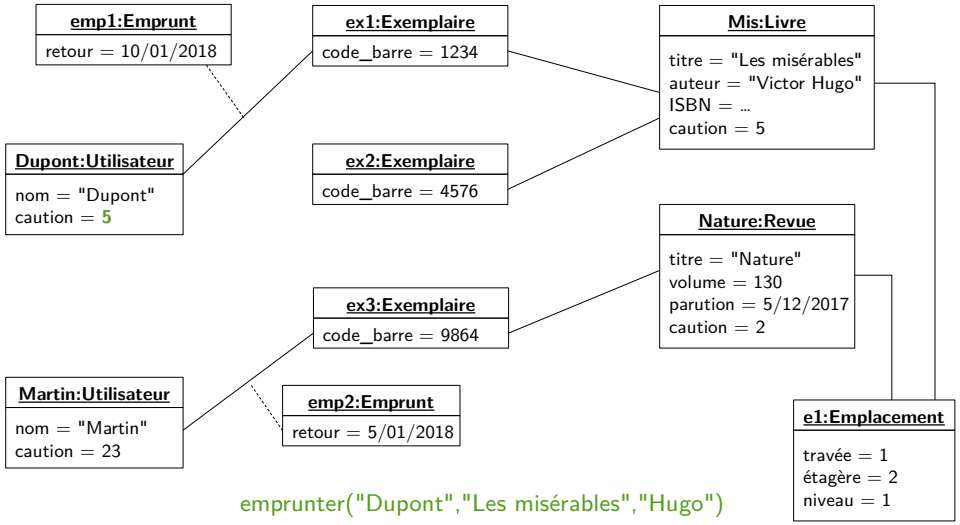
- Renvoyer le résultat d'un calcul
- Modifier l'état du système
  - modification de la valeur des attributs
  - ajout/suppressions de liens entre objets
  - création/destruction d'objets



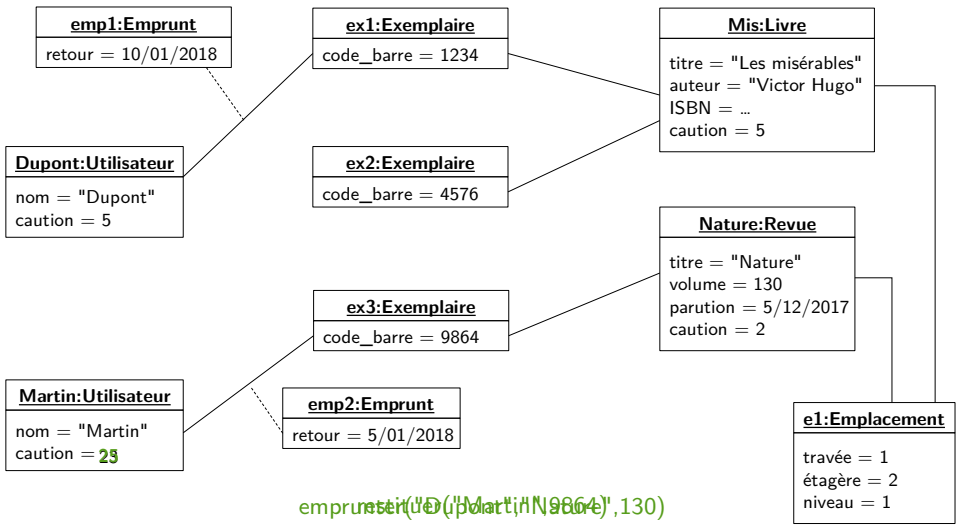
Exemple de la bibliothèque (6)



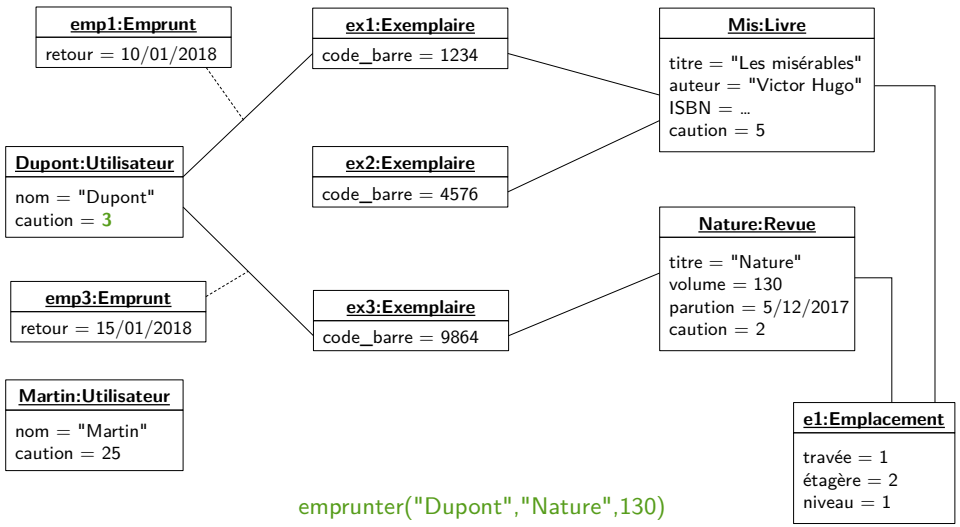
Exemple de la bibliothèque (6)



Exemple de la bibliothèque (6)



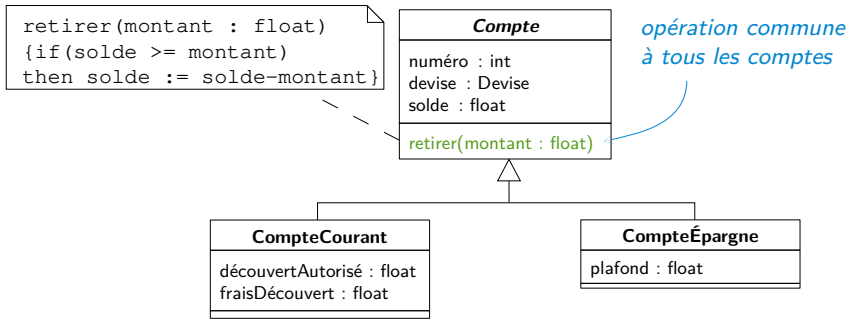
Exemple de la bibliothèque (6)



Héritage d'opération

Opération commune aux sous-classes :

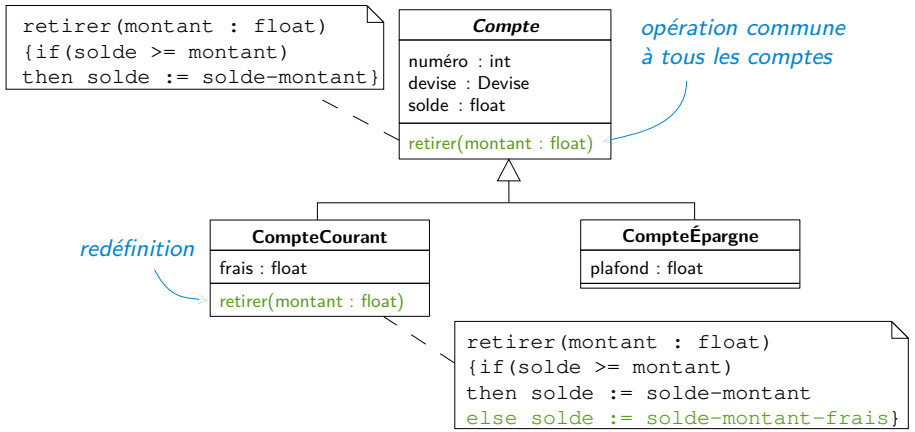
- Définition dans la super-classe



Redéfinition d'opération

Opération commune aux sous-classes :

- Définition dans la super-classe
- Possibilité de redéfinition locale de l'opération dans une sous-classe pour prendre en compte un cas particulier



Opération abstraite

Opération non définie pour une classe, car impossible de la définir pour tous les objets de la classe (opération non définie en italique)

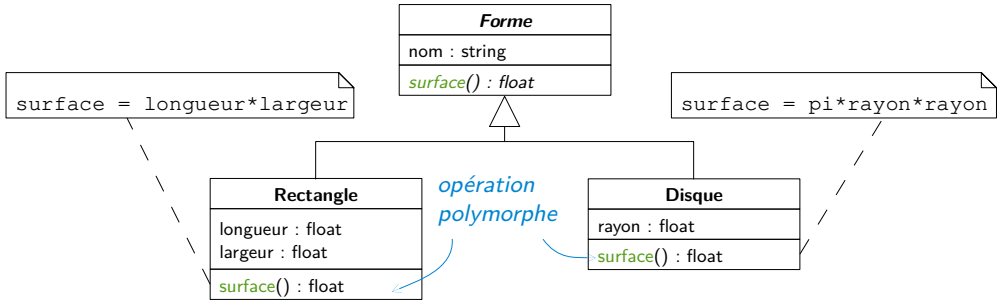
Opération abstraite implique classe abstraite

Exemple : On ne peut pas calculer la surface d'une forme sans savoir de quelle forme il s'agit

Polymorphisme

Contexte : Définition d'une opération abstraite dans les classes héritant d'une classe abstraite

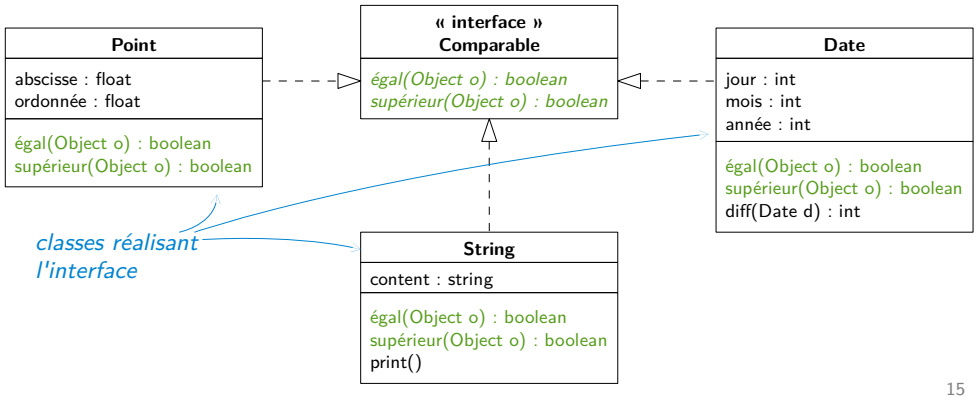
Opération polymorphe : Opération définie dans différentes sous-classes mais opération spécifique à la sous-classe



Interface

Liste d'opérations constituant un contrat à respecter par les classes réalisant l'interface

- Pas une classe, ne peut pas servir à créer des objets
- Toutes les opérations sont abstraites



université PARIS-SACLAY
FACULTÉ DES SCIENCES D'ORSAY

Slides Diagrammes de séquence (conception)

Objectif : Représenter les communications avec et au sein du logiciel

- Représentation temporelle des interactions entre les objets
- Chronologie des messages échangés entre les objets et avec les acteurs

En conception : Décrire la réalisation des cas d'utilisation sur le système représenté par le diagramme de classes

- Point de vue interne sur le fonctionnement du système
- Description au niveau de l'instance (état du système à un instant)
- Description de scénarios particuliers
- Représentation des échanges de messages
  - entre les acteurs et le système, entre les objets du système
  - de façon chronologique

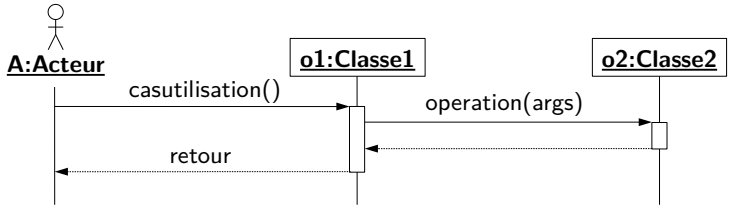
Diagrammes de séquence (conception)

Éléments du diagramme de séquence

- Acteurs
- Objets (instances)
- Messages (cas d'utilisation, appels d'opération)

Principes de base : Représentation graphique de la chronologie des échanges de messages avec le système ou au sein du système

- « Vie » de chaque entité représentée verticalement
- Échanges de messages représentés horizontalement



Utilisation en phase de conception

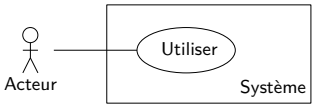


Diagramme de cas d'utilisation

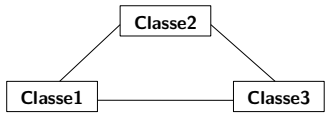


Diagramme de classes du système

Objectif : Description de la réalisation d'un cas d'utilisation sur le système décrit par le diagramme de classes

Problème : Communication entre les acteurs et le système vu comme un ensemble d'objets.

Utilisation en phase de conception

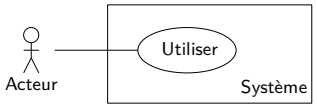


Diagramme de cas d'utilisation

Diagramme de classes du système

Communication entre acteurs et système via une interface (texte, web, physique...)

Utilisation en phase de conception

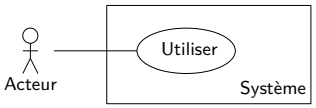


Diagramme de cas d'utilisation

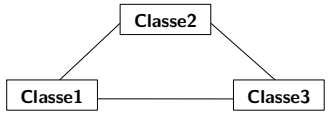
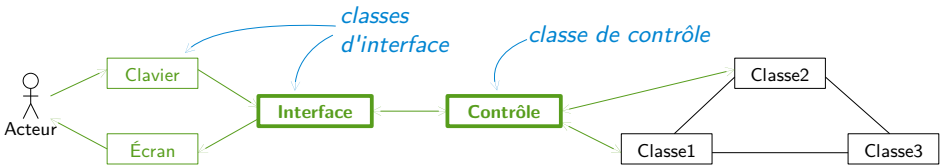


Diagramme de classes du système



Solution : Création de classes de contrôle et de classes d'interface qui :

- gèrent les interactions avec les acteurs
- encapsulent le résultat des opérations

Utilisation en phase de conception

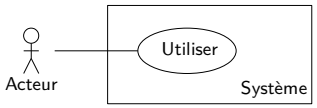


Diagramme de cas d'utilisation

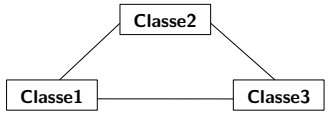


Diagramme de classes

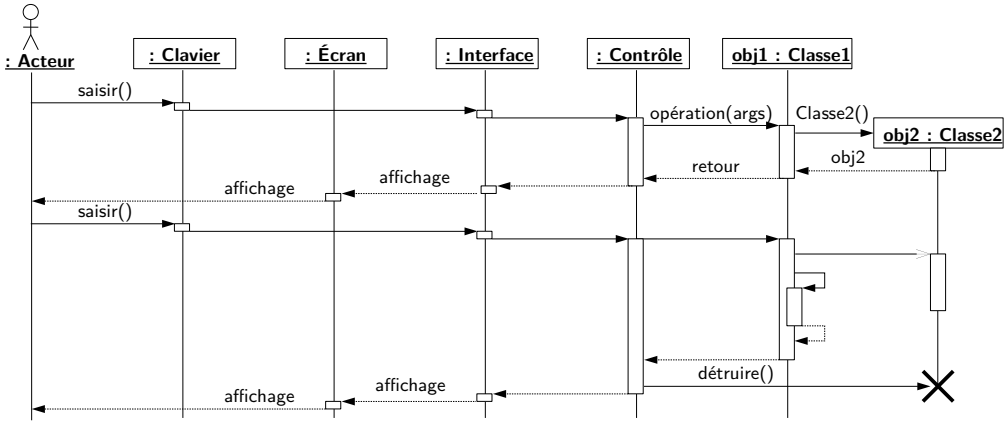
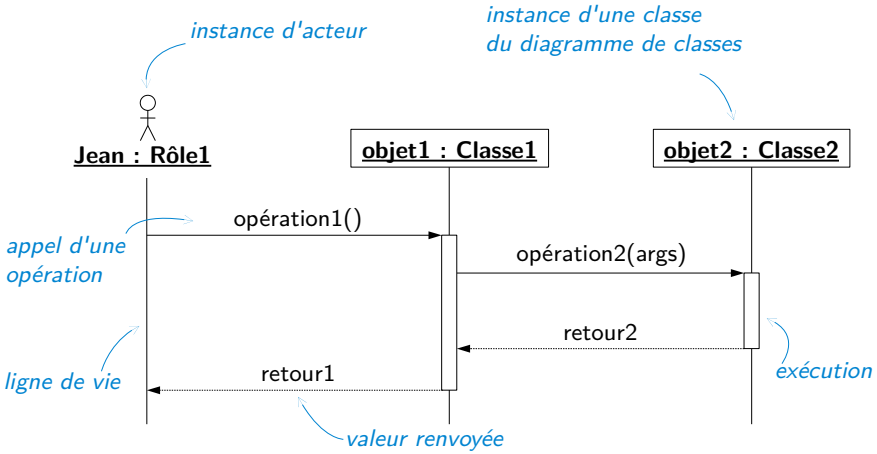


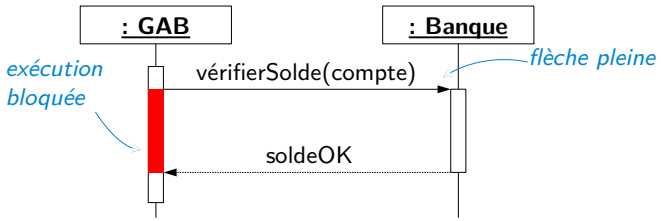
Diagramme de séquence du cas d'utilisation Utiliser

Éléments de base

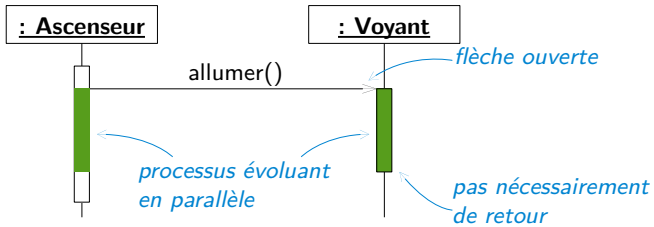


Types de messages

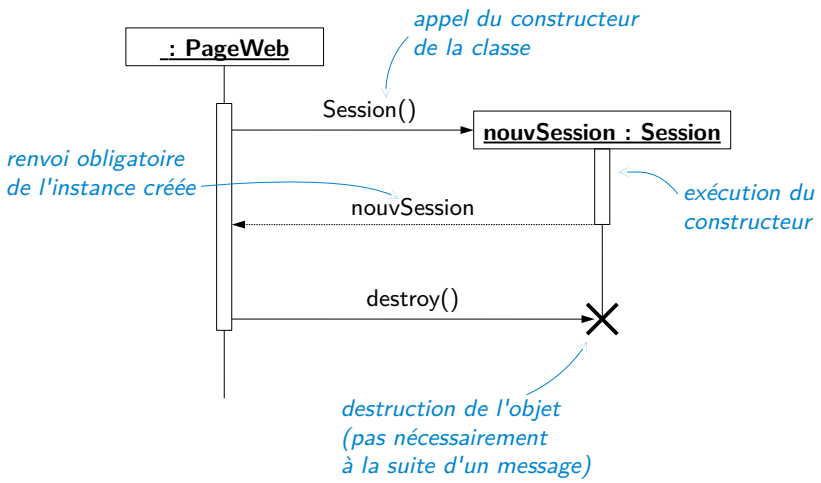
Message synchrone : Émetteur bloqué en attente du retour



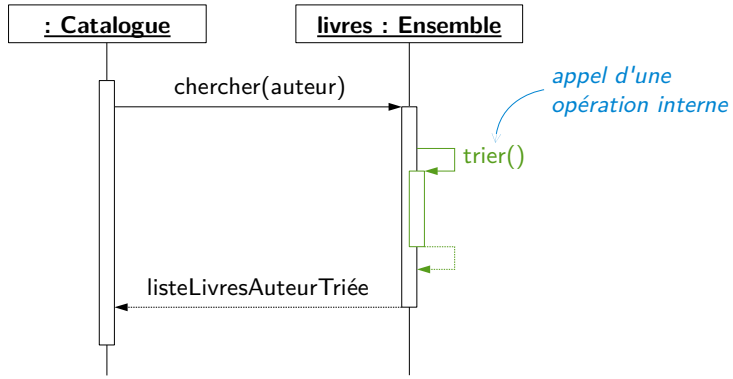
Message asynchrone : Émetteur non bloqué, continue son exécution



Création et destruction d'objet



Message réflexif

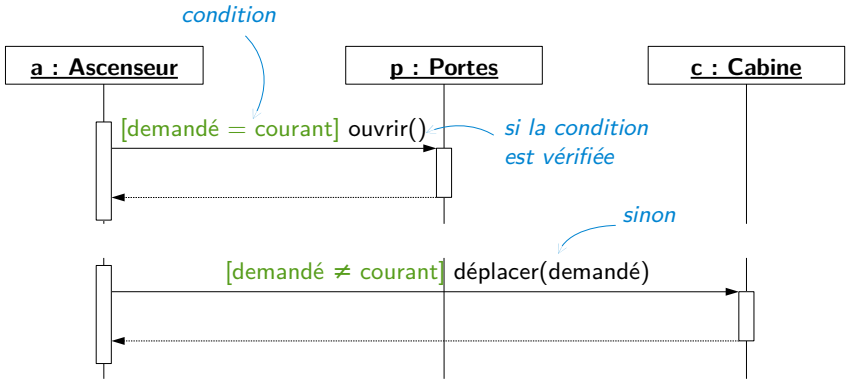


Alternative

Principe : Condition à l'envoi d'un message

Notation :

- Deux diagrammes

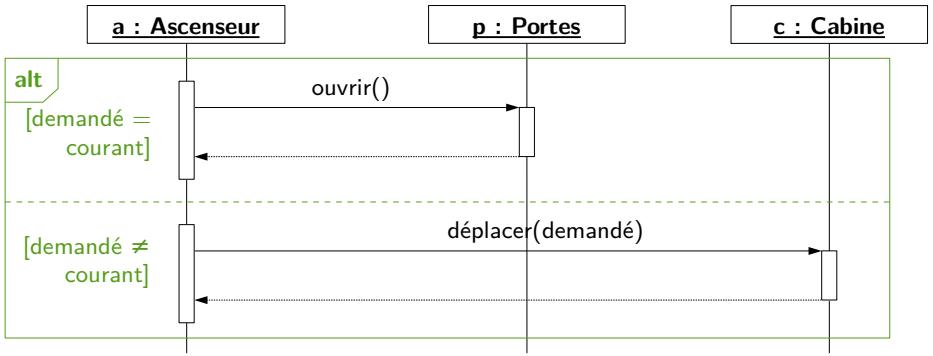


Alternative

Principe : Condition à l'envoi d'un message

Notation :

- Deux diagrammes
- Bloc d'alternative **alt**

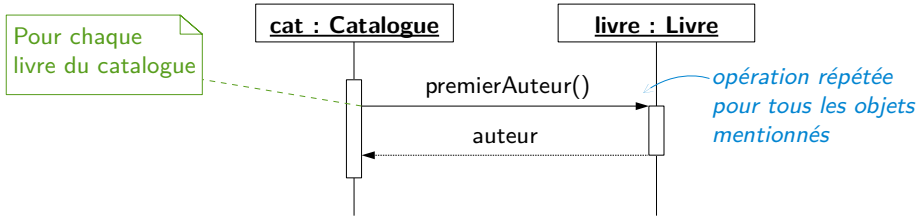


Boucle

Principe : Répéter un enchaînement de messages

Notation :

- Note

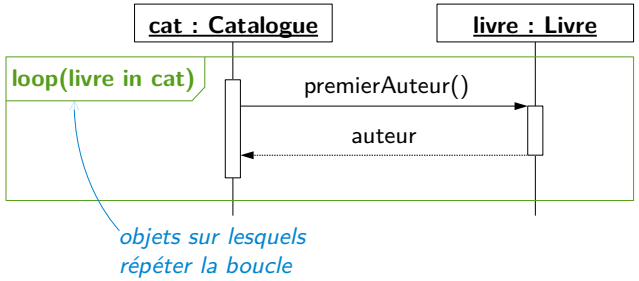


Boucle

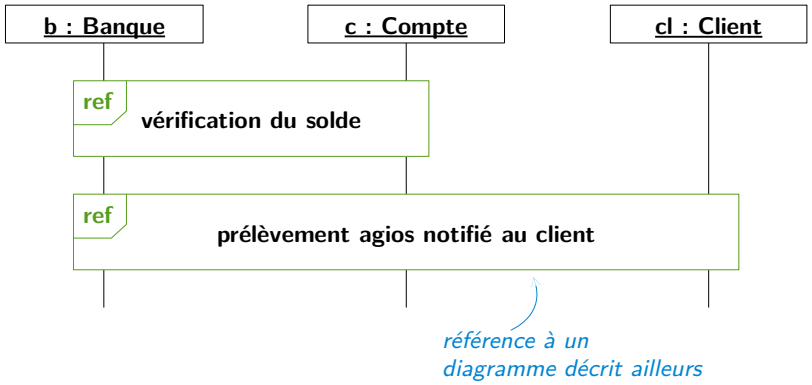
Principe : Répéter un enchaînement de messages

Notation :

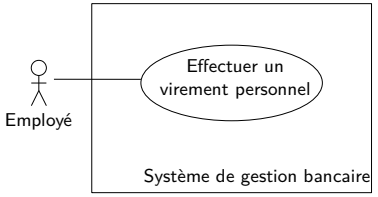
- Note
- Bloc de boucle **loop**

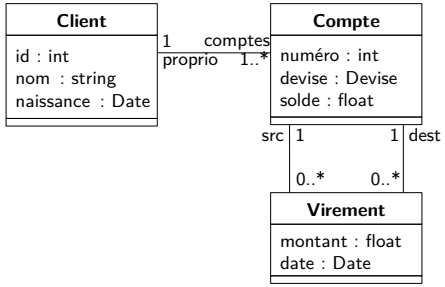


Référence à un autre diagramme

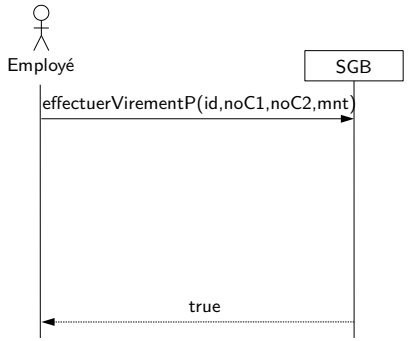


Exemple - Analyse

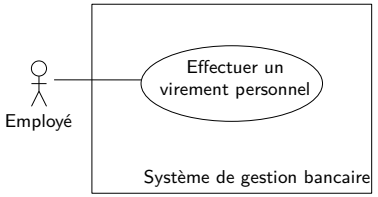


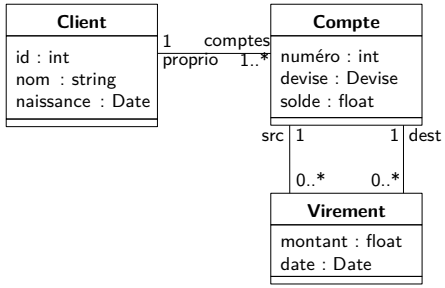


Cas d'utilisation
Effectuer un virement personnel
(niveau analyse)

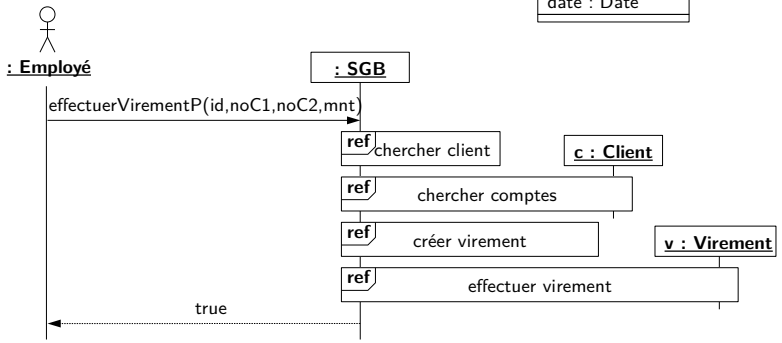


Exemple - Conception

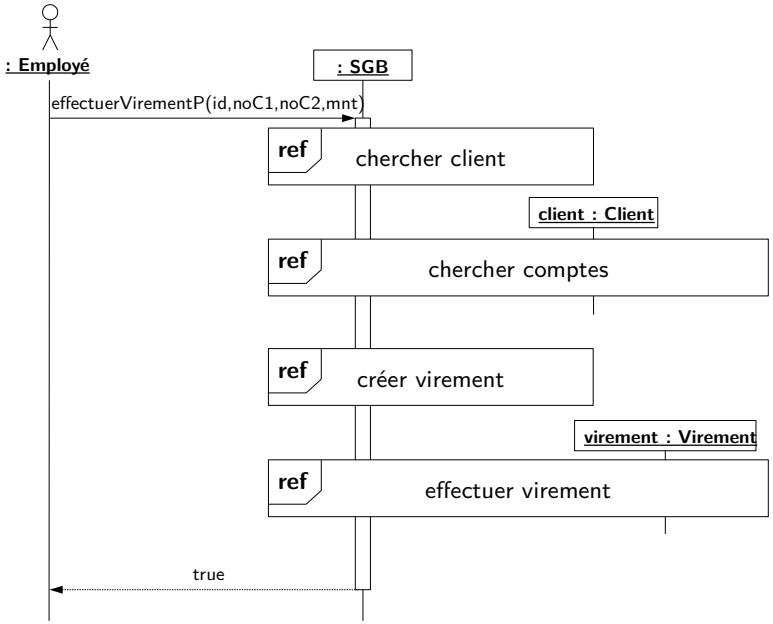




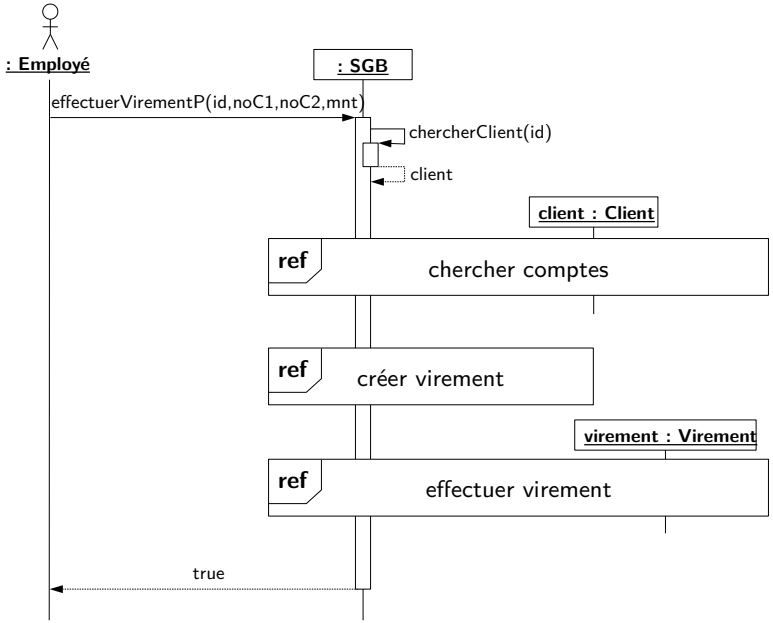
Réalisation du cas d'utilisation *Effectuer un virement personnel* (niveau conception)



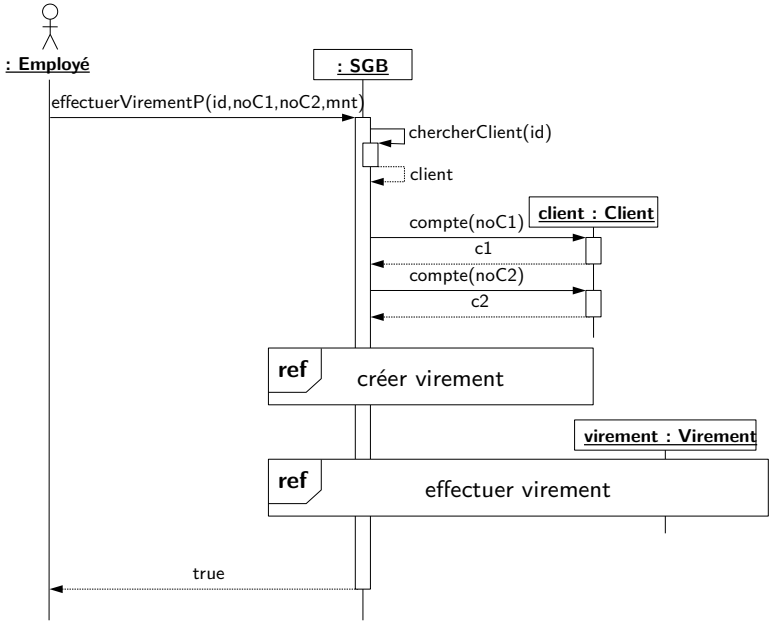
Exemple - Conception



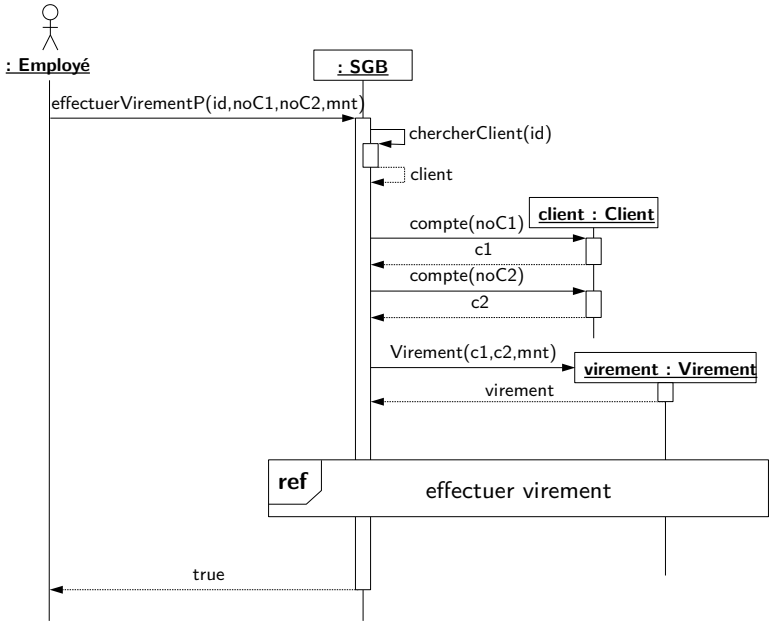
Exemple - Conception



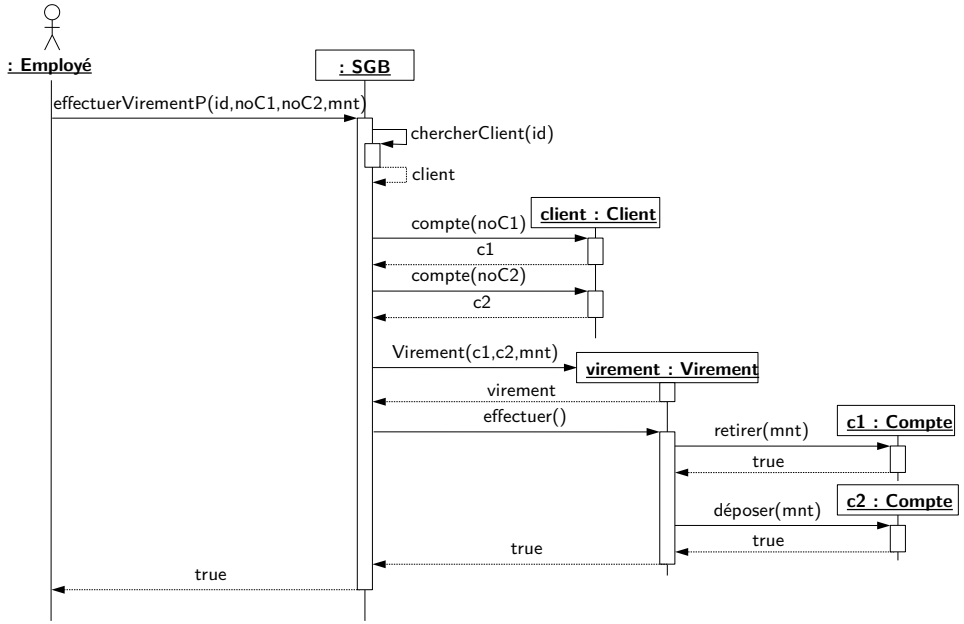
Exemple - Conception



Exemple - Conception

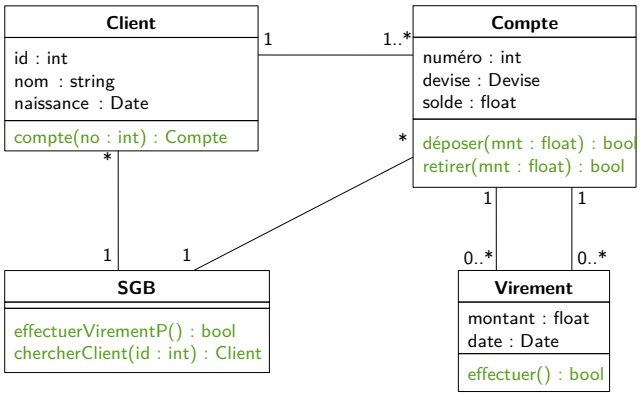


Exemple - Conception



Exemple - Conception

Diagramme de classes complété avec les classes techniques et les opérations nécessaires



Quelques règles

Messages entre acteurs et interface

- « Fausses » opérations liées au cas d'utilisation (même nom)
- Arguments (saisis) et valeurs de retour (affichées) simples : texte, nombre

Messages au sein du système

- Opérations du diagramme de classes
- Si message de **objA : ClasseA** vers **objB : ClasseB**, alors opération du message dans ClasseB

Constrainte/Invariant

Diagramme de classes représente la structure du système en termes d'objets et de relations entre ces objets

Ne permet pas de représenter des contraintes/invariants :

- Valeurs autorisées des attributs
- Conditions sur les associations
- Relations entre les attributs ou entre les associations

Expression des contraintes/invariants liées au diagramme :

- Notes dans le diagramme
- Texte accompagnant le diagramme
- OCL (Object Constraint Language) : langage de contraintes formel associé à UML

Contraintes, invariants

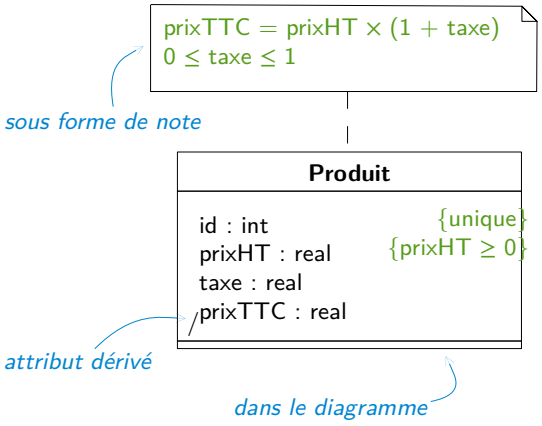
- Propriétés portant sur les éléments du modèle
- Doivent être vérifiées à tout instant
- En général, restriction sur les diagrammes d'objets possibles à partir du diagramme de classes
- Héritage des contraintes de la super-classe vers les sous-classes

Objectif : contraindre l'implémentation pour qu'elle réponde au cahier des charges

Contraintes présentes dans le diagramme :

- Type des attributs
- Multiplicités des associations

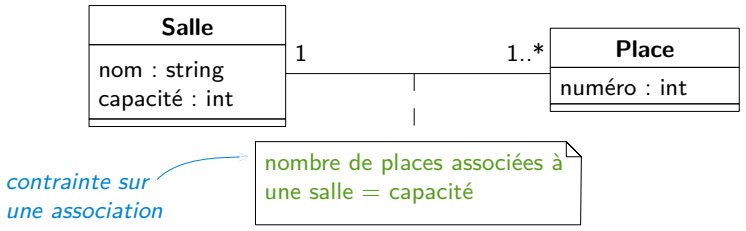
Contraintes sur les attributs

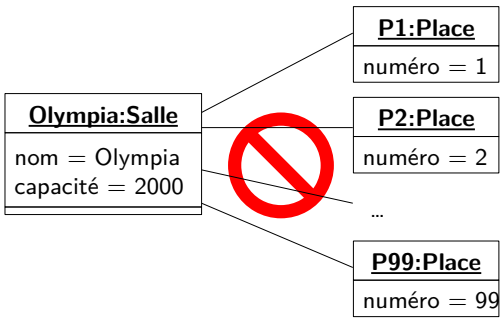


Contraintes sur la classe Produit :
- L'identifiant est unique
- Le prix hors taxe est toujours positif ou nul
- La taxe est comprise entre 0 et 1
- Le prix TTC est égal au prixHT augmenté de la valeur de la taxe appliquée au prixHT

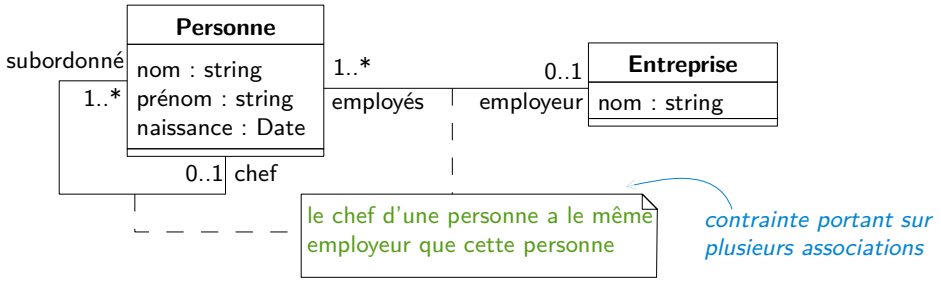
dans un document annexe

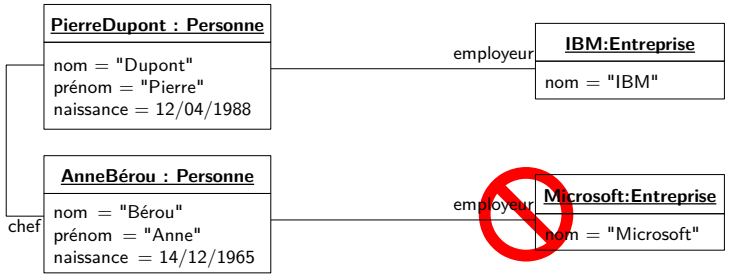
Contraintes sur les associations



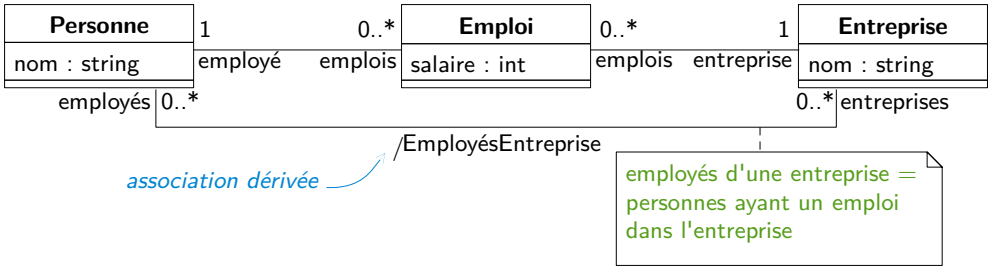


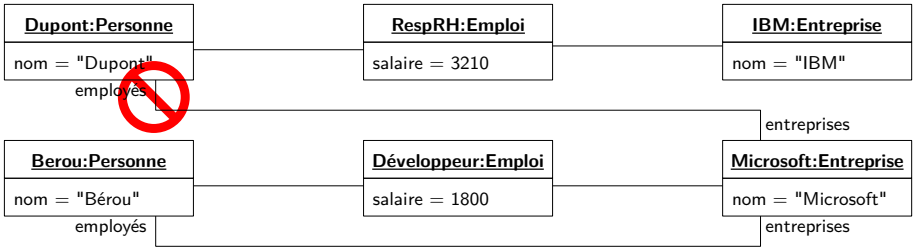
Contraintes sur les associations



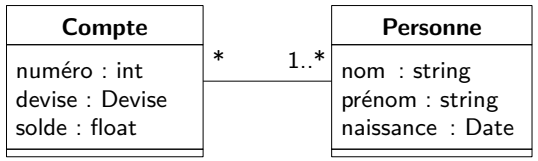


Contraintes sur les associations

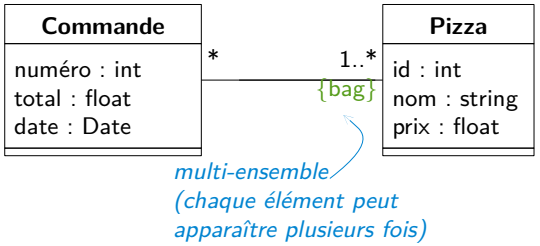


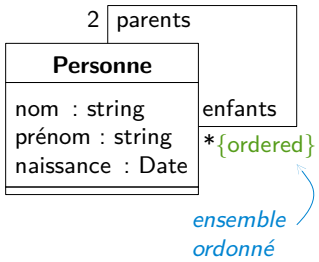


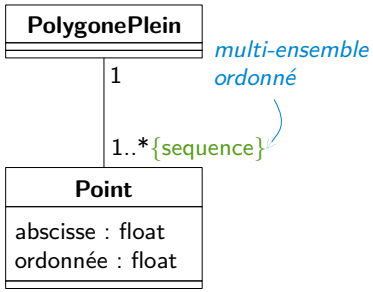
Contraintes associées à la multiplicité



Par défaut : ensemble non ordonné

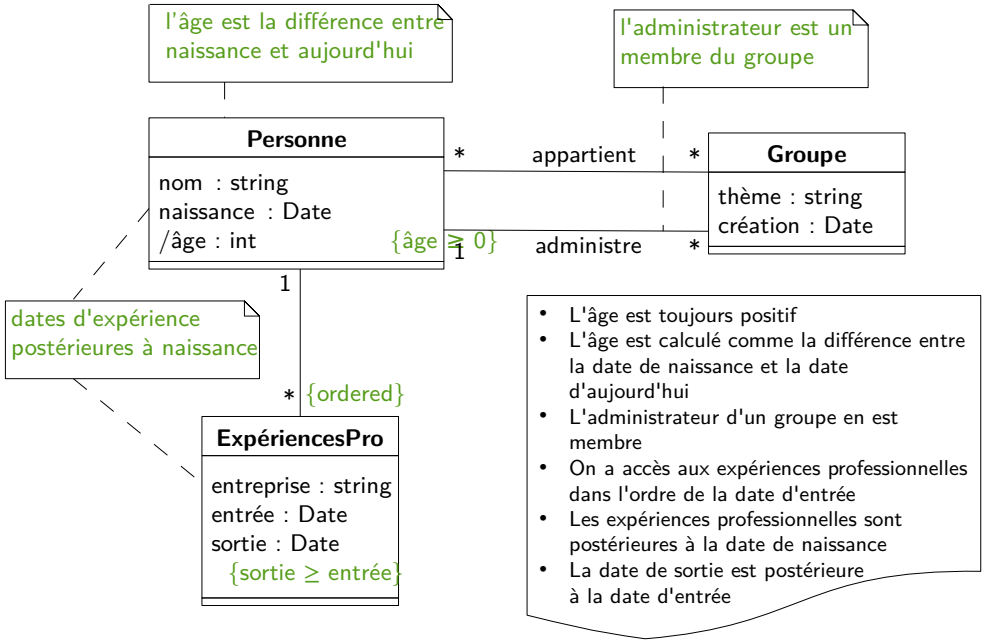






Exercice Une ligne de métro relie un ensemble de stations selon un ordre. Une station peut apparaître sur plusieurs lignes mais ne peut pas être présente plusieurs fois sur la même ligne. Diagramme Classe et Objet.

Contraintes, invariants

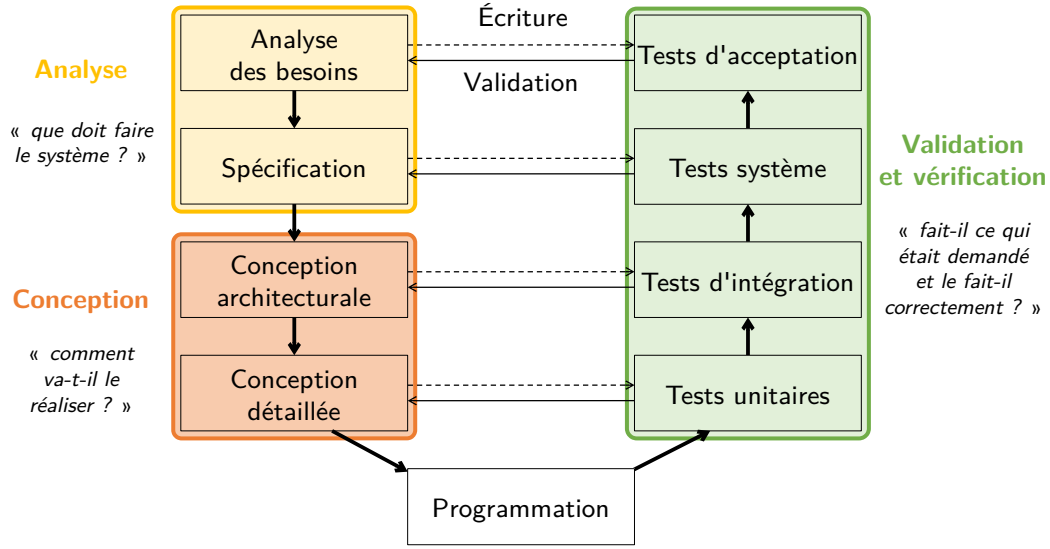


université PARIS-SACLAY

FACULTÉ DES SCIENCES D'ORSAY

Conception

Processus de développement en V



Programmation structurée

Le langage machine offrait une liberté totale :
-Grâce aux "jump", toute instruction était un point d'entréepotentiel.
-Un code pouvez se modifier lui-même.

Mais trop de liberté tue la liberté !
-Certes, elle autorise des optimisations.
-Mais la correction des programmes devient trop difficile à vérifier.
-Bien concevoir un logiciel => pouvoir raisonner sur ce logiciel.

Réduire la liberté pour mieux contrôler La programmation structurée a mis en avant les concepts suivants :

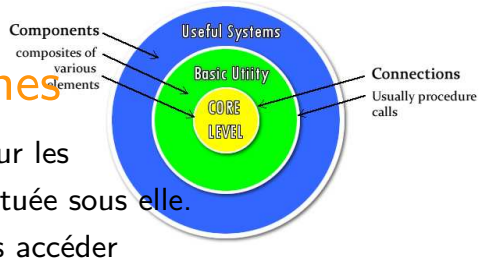
- 1 Les structures de contrôle restreintes
- 2 Organisation en couches
- 3 Programmer par raffinements successifs
- 4 Les types abstraits de données
- 5 L'architecture d'un logiciel
- 6 Les patrons de conceptions
- ....

1-Les structures de contrôle restreintes

À l'aide d'un goto, on peut tout faire. . . mais le flot de contrôle est alors arbitrairement complexe. « goto considered harmfull »(Dijkstra)
Pour raisonner sur un programme, il y a trois outils :

- L'abstraction→ Appels de fonctions
- L'induction→ Fonctions récursives, itération
- Le raisonnement par cas → Types algébriques, Pattern matching « Use recursive procedures for recursively-defined data structures. » (Kernighan)

2-Organisation en couches



Une couche de haut niveau s'appuie sur les primitives fournies par la couche(s) située sous elle.
Une couche de bas-niveau ne peut pas accéder aux primitives des couches de niveau plus élevé qu'elle

3-Conception par raffinements successifs

Chaque raffinement correspond à un choix de conception.
Espace de conceptions = domaine de recherche
Choix de conceptions = arbre de décision

Exemple du tri: on raffine données et algorithmes en parallèle.
1 Fait-on des hypothèses sur la distribution des entrées ?
2 Veut-on un algorithme parallèle ou séquentiel ?
3 Les données a trier arrivent une par une (tri par insertion)

Notion plus simple de raffinements :
Chaque section de votre projet complexifie votre programme
-pas de joueurs /un seul joueur /plusieurs joueurs
-rajouter des actions de joueurs
-rajouter des rôles (types de joueurs avec des pouvoirs spécifique)
-raffiner les régles du jeu

4-Types abstraits

-Un type abstrait défini une classe de composants totalement caractérisés par un ensemble d'opérations (notion abordée en S2).
-L'utilisateur du type abstrait n'a pas connaissance de la représentation concrète des données.
-Le maintien des invariants est circonscrit aux définitions des opérations.

Exemple relatif au projets
Faire un type abstrait pour manipuler une pile de cartes
On peut en faire un type générique, stack <T>
Opérations :
dépiler : stack <T> → stack<T>,T
empiler : stack <T>,T → stack<T>
is_empty : stack <T> → boolean
Mélanger : stack <T> → stack<T>

5- Conception architecturale

"Architecturer" n'est pas "Programmer (conception détaillée)"
Programmation=Implémenter / Déboguer / Prouver un algorithme.
Architecturer (i.e., modulariser)
-Minimiser les interdépendances ;
-Ordonner / Classifier / Organiser les composants ;
-Rendre explicite les interactions.
DeRemer, Kron *Programming-in-the large versus programming-in-the-small.*

6- Les patrons de conceptions architecturale

Cacher -la représentation des données (c.f. ADT)
-les fonctions internes d'un composant
Les interfaces doivent être en nombre restreint, petites, lister seul les services dont l'évolution est peu probable

6bis - Les patrons de conceptions détaillés

voir cours

Commentaires et documentation

Ajouter du texte au code source qui n'est pas compilé
-pour aider à la compréhension du code
-pour le reprendre soi-même plus tard, ou pour d'autres développeurs.
-explique comment utiliser la classe / la méthode,
-expliquer la suite : amélioration, problème à corriger, etc. ;

Les annotations avec @, exemple = @Override pour forcer la redéfinition

Documentation = annotation Javadoc @param @return pour spécifier
-chaque paramètre d'une méthode
-chaque attribut d'une classe,
-pour ensuite génerer automatiquement une documentation html

```
/**
* @param  url  an absolute URL giving the base location of the image
* @param  name the location of the image, relative to the url argument
* @return      the image at the specified URL
*/
public Image getImage(URL url, String name) {...}
```