POO

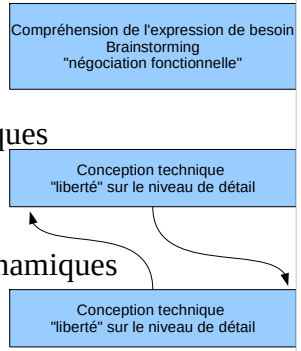
Rudiments d'UML

UML

- Unified Modeling Language
- Langage uniformisé pour la spécification de modèles objets
- Graphiques standardisés
- Modèle systémique abstrait : modèle UML

Diagrammes UML

- Diagrammes comportementaux
  - Diagramme des cas d'utilisation
  - ...
- Diagrammes structurels ou statiques
  - Diagramme de classes
  - Diagramme des paquetages
  - ...
- Diagrammes d'interaction ou dynamiques
  - Diagramme de séquence
  - ...

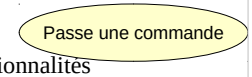


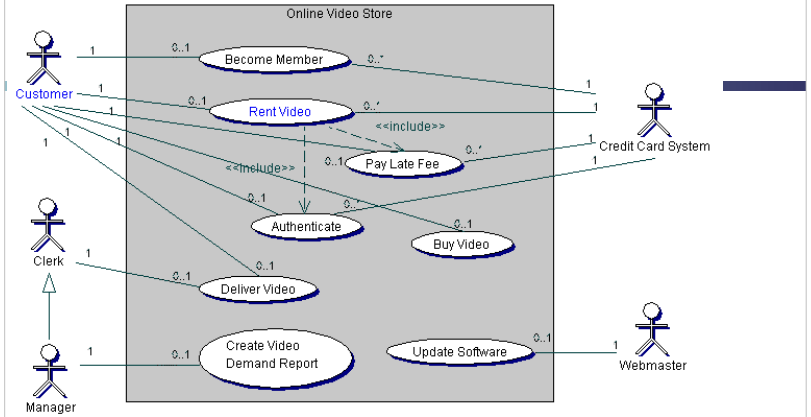
Au commencement

- le but d'un projet est de satisfaire le besoin !
  - Expression des besoins client
    - Cahier des charges fonctionnel
  - (Agile) User story
    - Who, What, Why
    - "In order to <receive benefit> as a <role>, I want <goal/desire>"
    - En tant que TE, je dois saisir mon RV en ligne pour que la cellule d'alternance soit avertie de ma visite et des éventuels problèmes.
- capturer les besoins principaux des utilisateurs.
- ne pas chercher l'exhaustivité, mais clarifier, filtrer et organiser les besoins !

Use Case Diagram

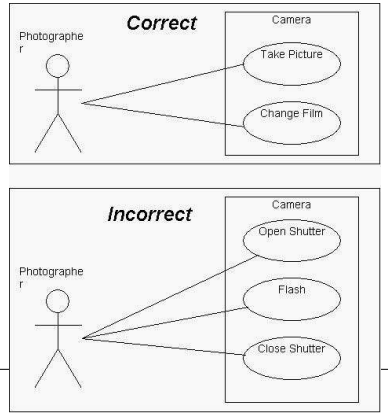
- Vision globale des interactions
  - interactions des utilisateurs (acteurs) avec le système
  - acteur = acteur physique ou système externe
- Contient :
  - Les acteurs
  - Les "Use Case"
    - objectifs du système, services rendus, fonctionnalités
  - Peut indiquer la nature des interactions
- Diagramme de haut niveau, vue d'ensemble
  - Ne peut donner une séquence précise d'actions





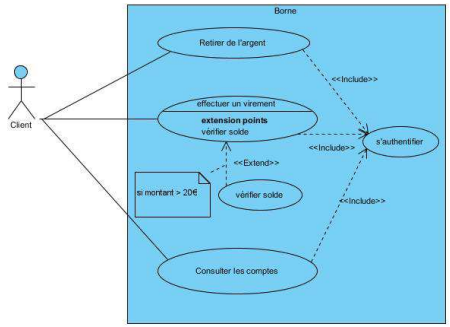
Use Case Diagram

- Identifier les Use Case (les *ovales*) ?



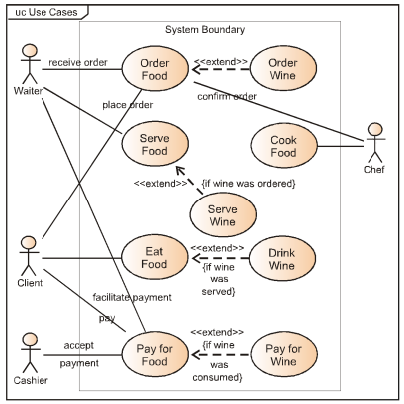
Use Case Diagram

- <<include>> (ou <<uses>>) et <<extend>>



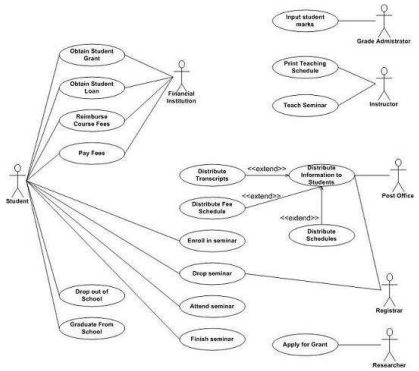
Use Case Diagram

- wikipedia



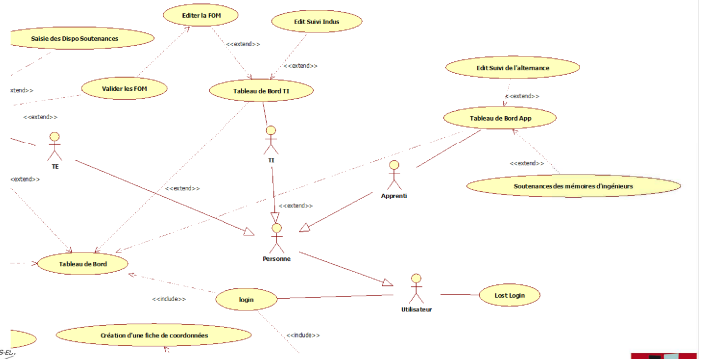
Use Case Diagram

- Un restaurant

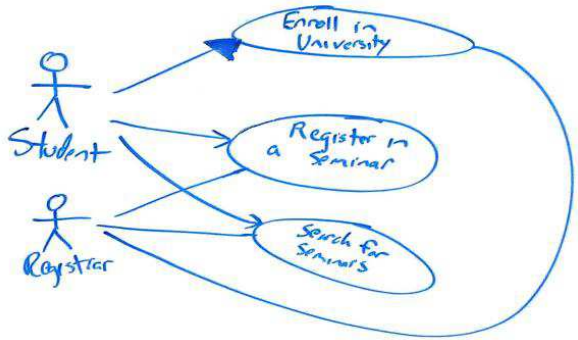


Use Case Diagram

- Site ei2000 (extrait 2009, MOA)



Use Case Diagram

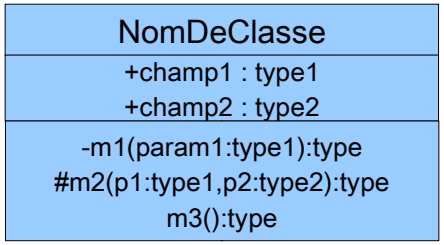


Use Case Diagram

- Attention : ne pas confondre avec UN cas d'utilisation
  - Décrivant l'ensemble des interactions pour UN scénario d'utilisation
- Utilisé pour discuter avec la MOA
- Complété par la description textuelle des scénarios d'utilisation

Diagramme de classes

- Chaque classe est représentée par un rectangle avec
  - son nom
  - ses champs
  - ses méthodes
  - types optionnels avec notation à la pascal



+ signifie "public"
- signifie "private"
signifie "protected"
~ signifie "package"

Diagramme de classes

- Chaque interface est représentée par un rectangle surmonté d' « interface » avec
  - son nom
  - ses méthodes
  - types optionnels avec notation à la pascal

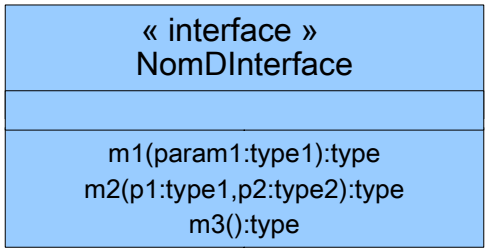
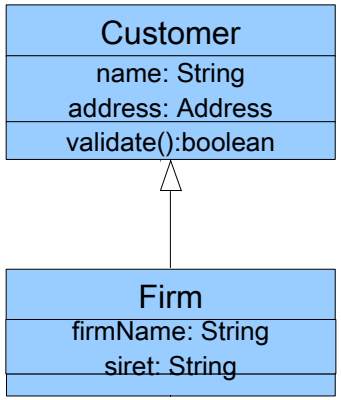


Diagramme de classes

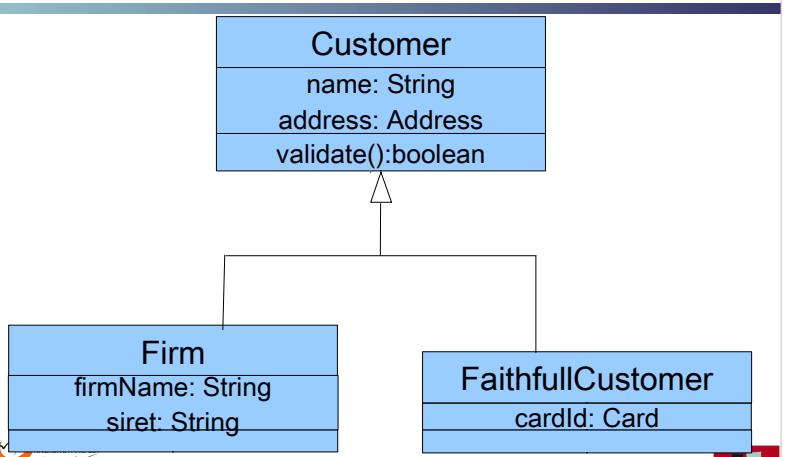
- On indique les relations entre classes et interfaces
  - généralisation (héritage)
  - réalisation (implémentation)
  - Association
    - simple ou bidirectionnelle
    - agrégation
    - composition
  - dépendance

Généralisation (héritage)

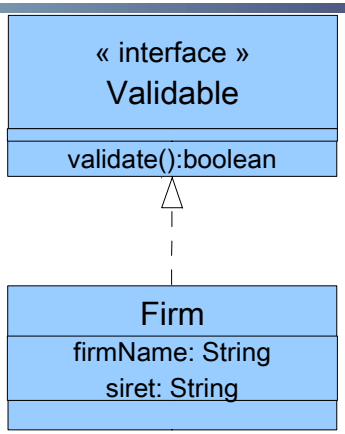
- On ne recopie pas les membres hérités



Généralisation

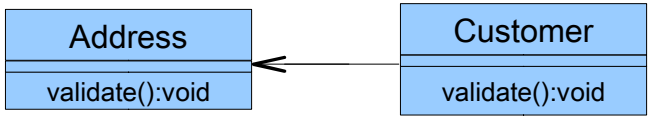


Réalisation (implémentation) :



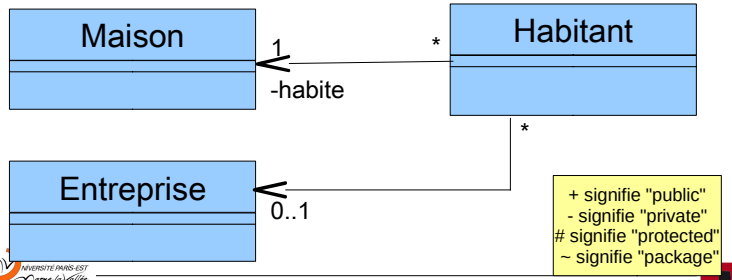
Association

- Une association signifie qu'une classe utilise une ou plusieurs instances d'une autre
- Si l'association est dans les deux sens, on ne met pas de flèche

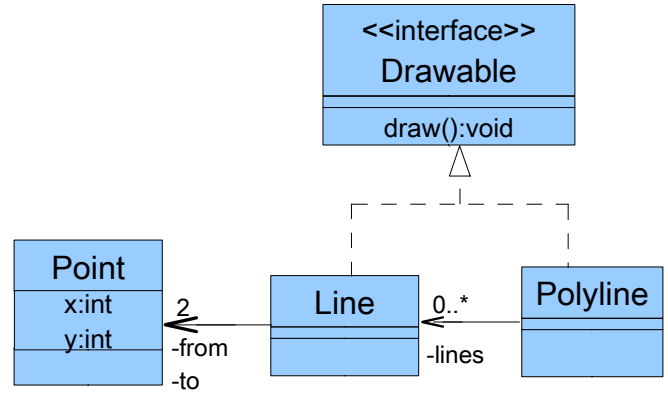


Association

- Information complémentaire :
  - cardinalité (indiqué à l'extrémité du lien)
  - nom du champ
  - visibilité

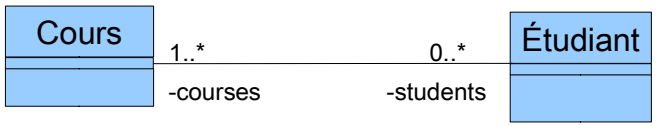


Association



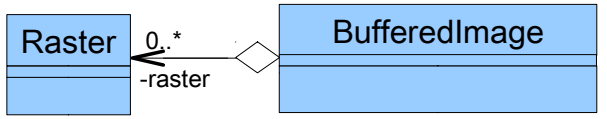
Association

- Un cours est suivi par des étudiants (peut-être aucun) et les connaît
- Un étudiant suit des cours et les connaît



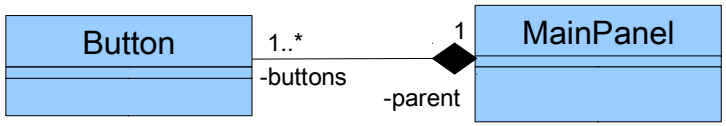
Agrégation

- L'agrégation est un cas particulier d'association :
  - Relation non symétrique, lien de subordination
  - Couplage fort
  - Relation « non vitale » les cycles de vie peuvent être indépendant



Composition

- La composition est un cas particulier d'agrégation
  - Agrégation forte
  - Cycles de vie liés
  - À un instant *t*, une instance de composant ne peut être liée qu'à un seul composé.



Composition

- En Java, si la référence sur l'instance crée quitte l'objet (getter), la composition est impossible
- En C, en théorie la composition peut être implémentée avec une structure qui contient l'autre structure (et non un pointeur)

Dépendance

- Signifie un « dépendance » de n'importe quel ordre
- On peut indiquer la nature de la dépendance

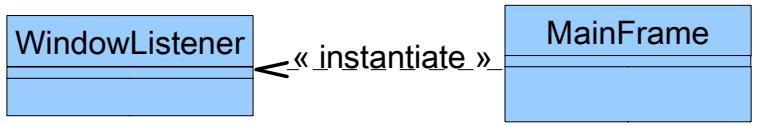


Diagramme de packages

- Simple diagramme où chaque paquetage est dans une boîte entre lesquelles on met un flèche pour dire que l'un est client de l'autre

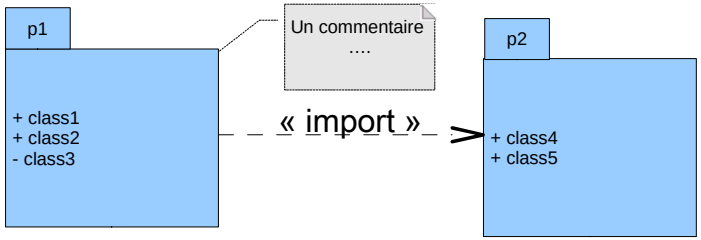


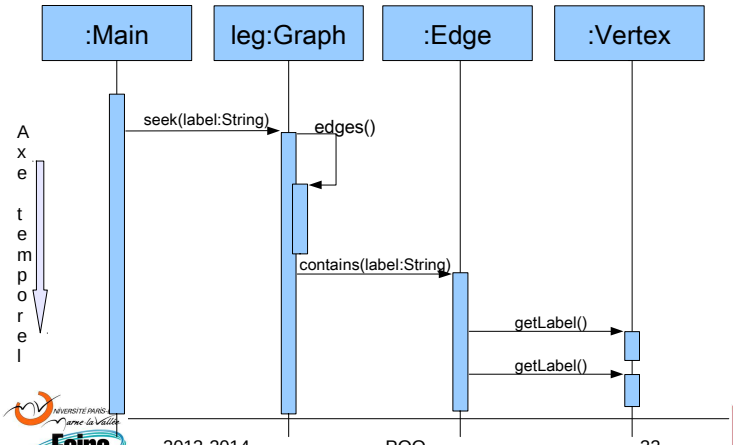
Diagramme de packages

- 2 axes
  - Cohérence
    - Nature
      - regroupement sémantique
      - finalité du service rendu
    - Evolution
      - Isoler ce qui varie
      - Niveau de stabilité
  - Indépendance
    - Faible couplage
    - Minimiser les interaction

Diagramme de séquence

- Indique ce qui se passe pendant une tâche particulière du programme
- Ce diagramme met l'accent sur les interactions entre objets
- Il permet aussi de prouver qu'une tâche cliente est possible
- En réalisant ce genre de diagrammes, on tombe souvent sur de nouvelles responsabilités.

Diagramme de séquence

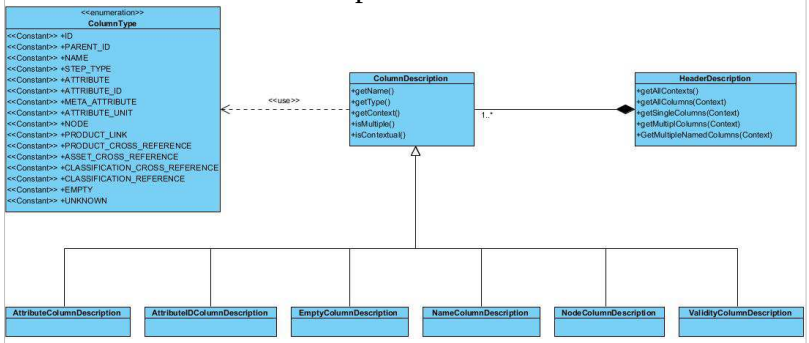


Logiciels UML (gratuits)

- Violet (java)
  - Très simple, prise en main en 5 minutes
  - Limité
- ArgoUML
  - Pas mal, après avoir été figé, a été repris par Collabnet
- BoUML
  - Pas mal ; version gratuite figée ; version payante
- StarUML
  - Pas mal ; n'évolue plus 2005
- Poseidon for UML, Community edition
- Visual paradigm for UML, Community edition

Exercice : Schéma → Code

- Ecrire le code correspondant



Exercice : Code → Schéma

```
public class D extends B {
  private int j;
  public void execute();
}
```

```
public abstract class B implements I {
  private int i;
}
```

```
public class A implements I {
  private B b;
  String attr;
  public A(B b) { this.b = b;}
  public void execute();
}
```

```
public class C implements I {
  private List<I> list;
  public void setList(List<I> l) ;
  public List<I> getList();
}
```

```
public interface I {
  public void execute();
}
```


Exercice : Code → Schéma