Bases de la conception orientée objet

Diagramme de Cas d'Usage
Diagramme de Classe (initiation)

Luc Dartois

luc.dartois@u-pec.fr

Objectifs

Objectif d'aujourd'hui : Initiation à la modélisation

Modélisation : Diagramme de Cas d'Usage

- Comprendre un système décrit par un DCU,
- Identifier des besoins dans un texte,
- Modéliser ces besoins par un DCU.

Objectifs

Objectif d'aujourd'hui : Initiation à la modélisation

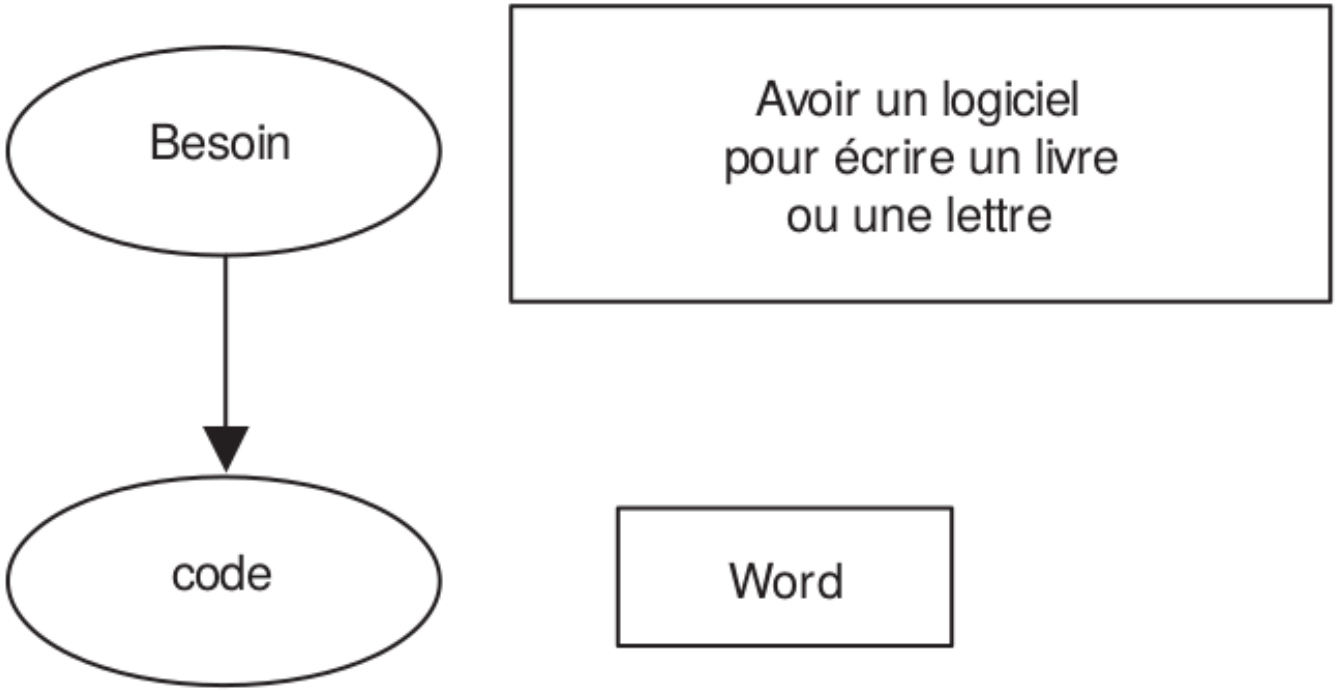
Modélisation : Diagramme de Cas d'Usage

- Comprendre un système décrit par un DCU,
- Identifier des besoins dans un texte,
- Modéliser ces besoins par un DCU.

Structure : Diagramme de Classe

- Lire et Transformer un Diagramme de Classe en squelette Java,
- Représenter une classe dans un Diagramme de Classe,
- Réprésenter dans un Diagramme Objet les objets d'une classe,
- Modéliser un besoin de structure par une classe (initiation).

Motivations



Développer un logiciel en théorie

Motivations

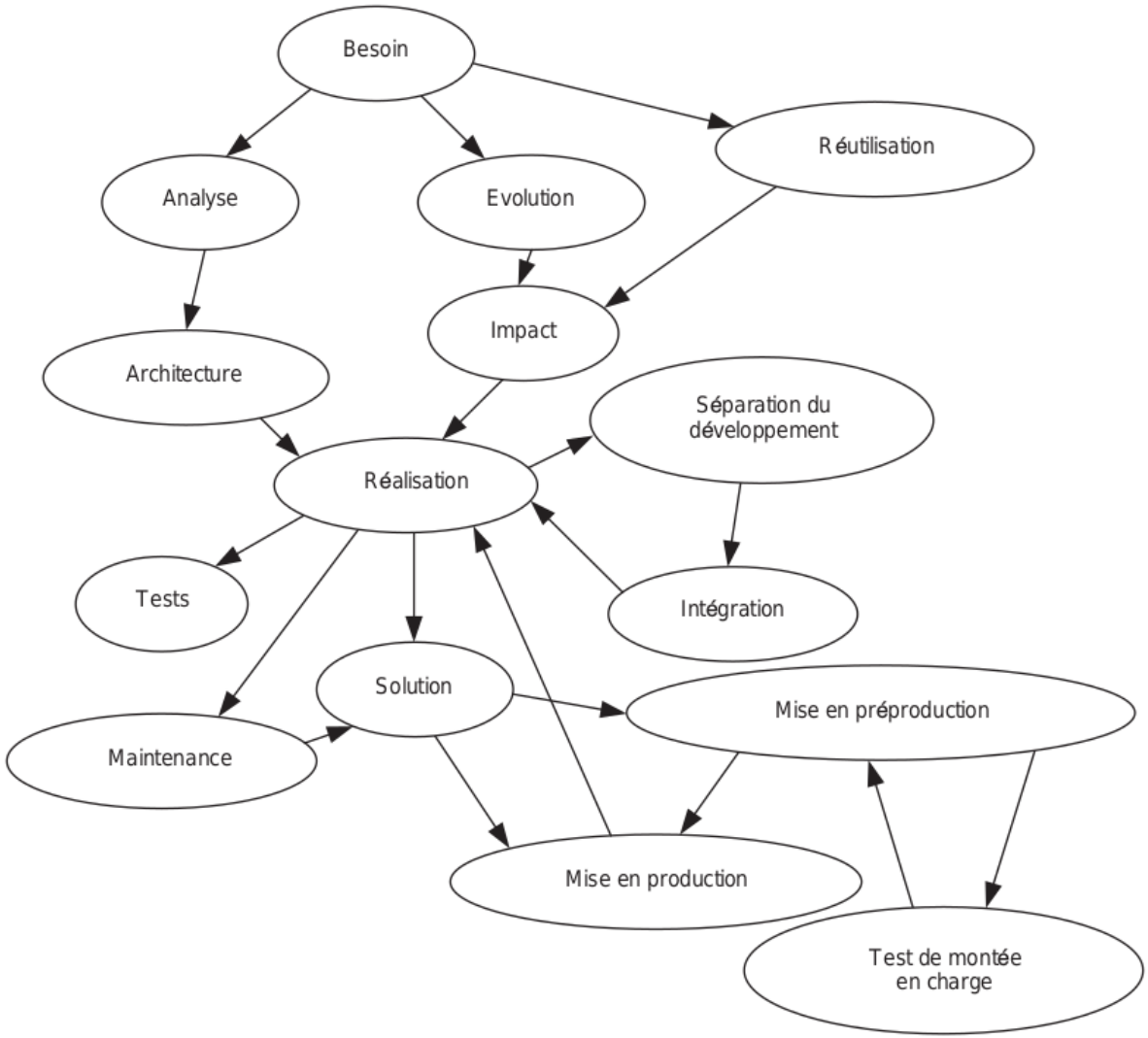
What the Client wanted


How the Client explained it


How the project manager understood it

Déjà des problèmes !!

Motivations



Développer un logiciel en (presque) pratique

Une seule solution : la modélisation !

Le but d'un langage de modélisation tel qu'UML (Unifier Modeling Language) est de :

- Faciliter une communication précise entre les agents de développement
  - → Avant : pour que tout le monde soit d'accord sur ce qu'on fait !
  - → Pendant et après : pour assurer des retours sur le développement effectué.

Une seule solution : la modélisation !

Le but d'un langage de modélisation tel qu'UML (Unifier Modeling Language) est de :

- Faciliter une communication précise entre les agents de développement
  - → Avant : pour que tout le monde soit d'accord sur ce qu'on fait !
  - → Pendant et après : pour assurer des retours sur le développement effectué.
- Formaliser chaque étape du processus de développement
  - → pour s'assurer qu'on fait bien ce qui a été décidé !
  - → pour faciliter la relecture du code pour des ajouts et modifications ultérieures.

Une seule solution : la modélisation !

Le but d'un langage de modélisation tel qu'UML (Unifier Modeling Language) est de :

- Faciliter une communication précise entre les agents de développement
  - → Avant : pour que tout le monde soit d'accord sur ce qu'on fait !
  - → Pendant et après : pour assurer des retours sur le développement effectué.
- Formaliser chaque étape du processus de développement
  - → pour s'assurer qu'on fait bien ce qui a été décidé !
  - → pour faciliter la relecture du code pour des ajouts et modifications ultérieures.

Comment ?

→ Avec des dessins !

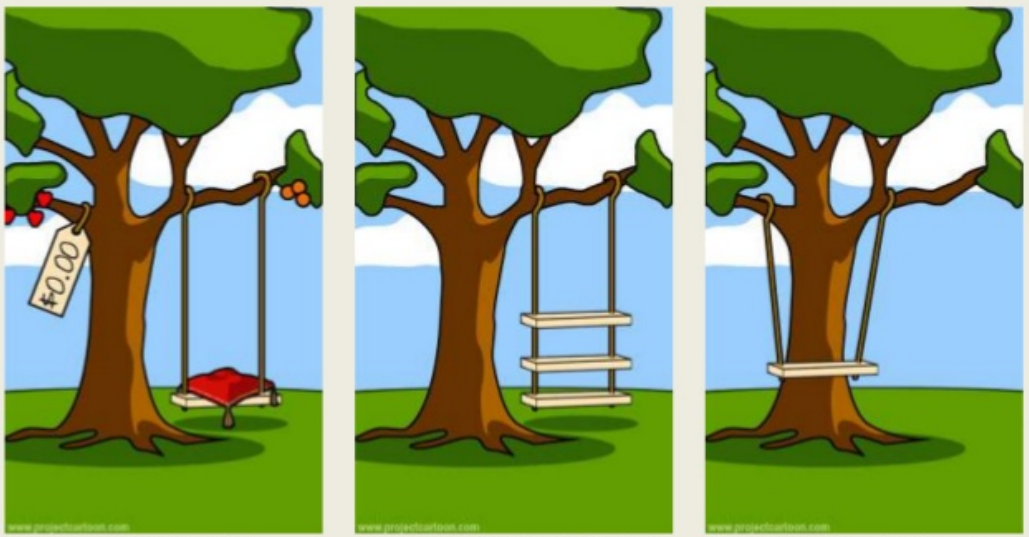
Une seule solution : la modélisation !

What the Client wanted


How the Client explained it


How the project manager understood it

Une seule solution : la modélisation !



What the Client wanted

How the Client explained it

How the project manager understood it

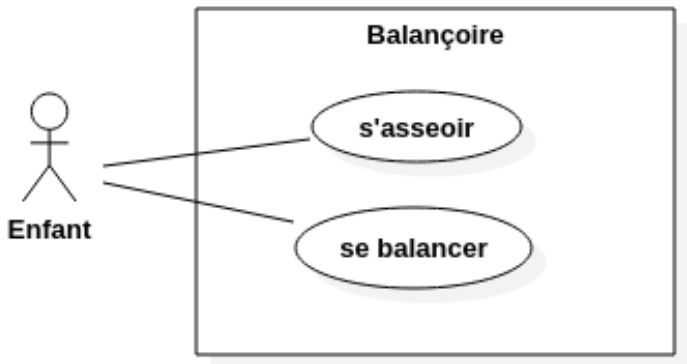
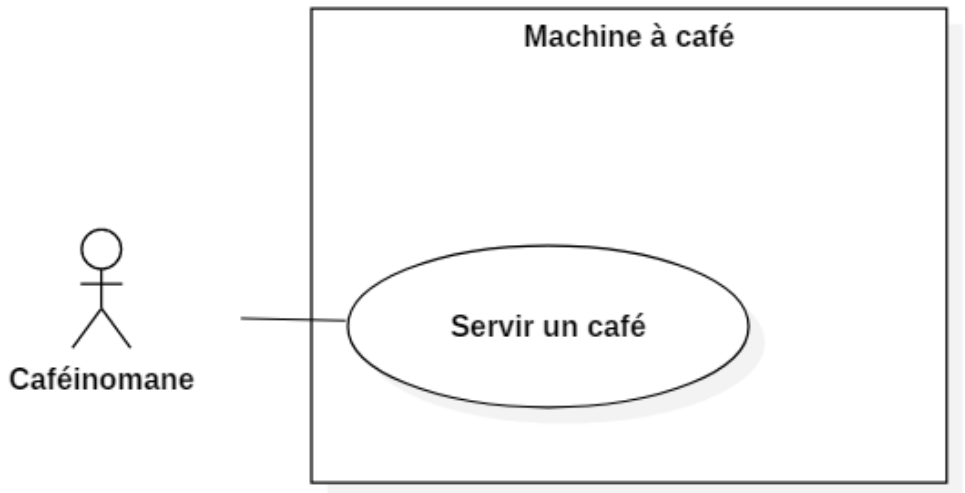


Diagramme de cas d'usage (DCU)



Un DCU sert à modéliser simplement

- un *système* : la machine à café
- un ou plusieurs *cas d'utilisation* : servir un café
- un ou plusieurs *acteurs* : le(s) adeptes de café

DCU : Pourquoi ?

Un diagramme de cas d'usage sert à :

- Communiquer, notamment avec des non informaticiens
- Décrire l'utilisation d'un système spécifique
- Déterminer des besoins

DCU : Pourquoi ?

Un diagramme de cas d'usage sert à :

- Communiquer, notamment avec des non informaticiens
- Décrire l'utilisation d'un système spécifique
- Déterminer des besoins

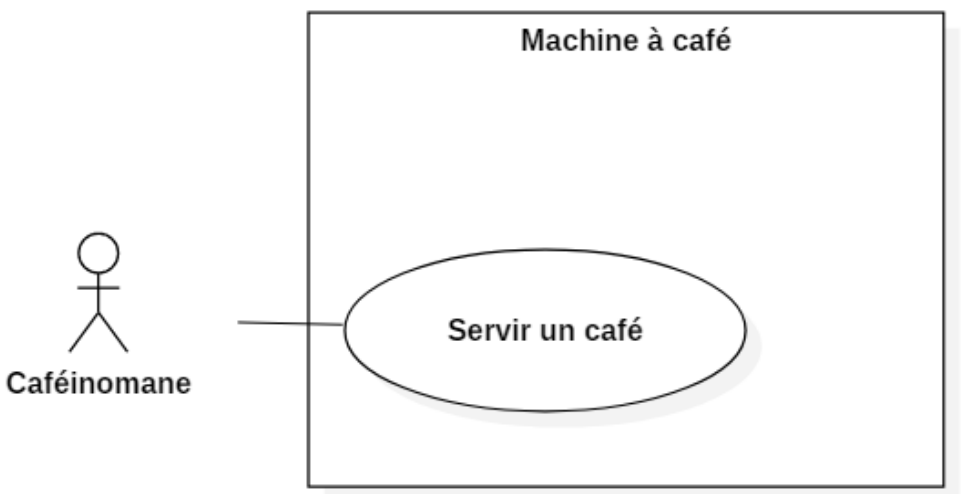
Un diagramme de cas d'usage doit :

- Être centré sur un système restreint (càd pas trop gros)
- Décrire les *interactions* entre ce système et l'extérieur (actions/réactions)
- Clarifier et structurer les besoins utilisateurs

Le système

Un diagramme de cas d'usage est centré sur un *système*, ici la Machine à café.

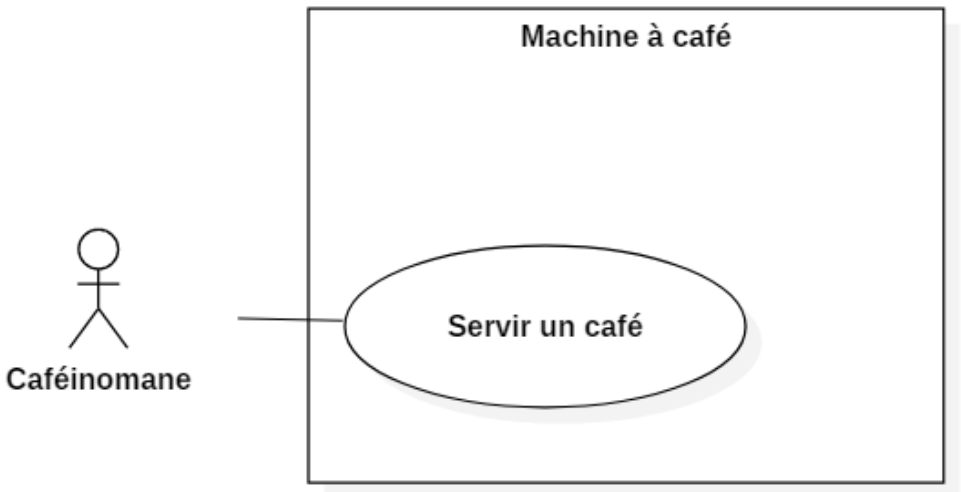
- Il est *unique* et *central* dans un DCU,
- On représente *ses* interactions avec l'extérieur.



Les acteurs

Les acteurs sont les entités qui *interagissent* avec le système ! Ce sont :

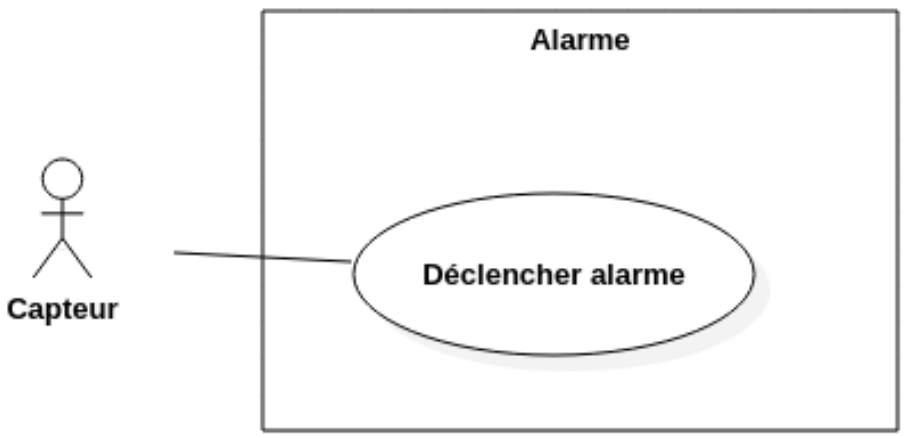
- Des êtres humains (utilisateurs, admin,...)



Les acteurs

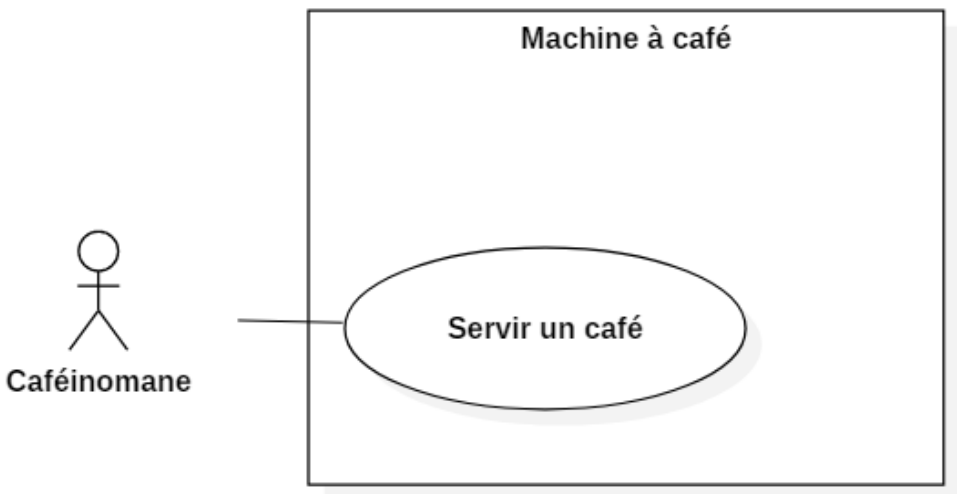
Les acteurs sont les entités qui *interagissent* avec le système ! Ce sont :

- Des êtres humains (utilisateurs, admin,...)
- D'autres systèmes (capteurs, imprimantes,...)



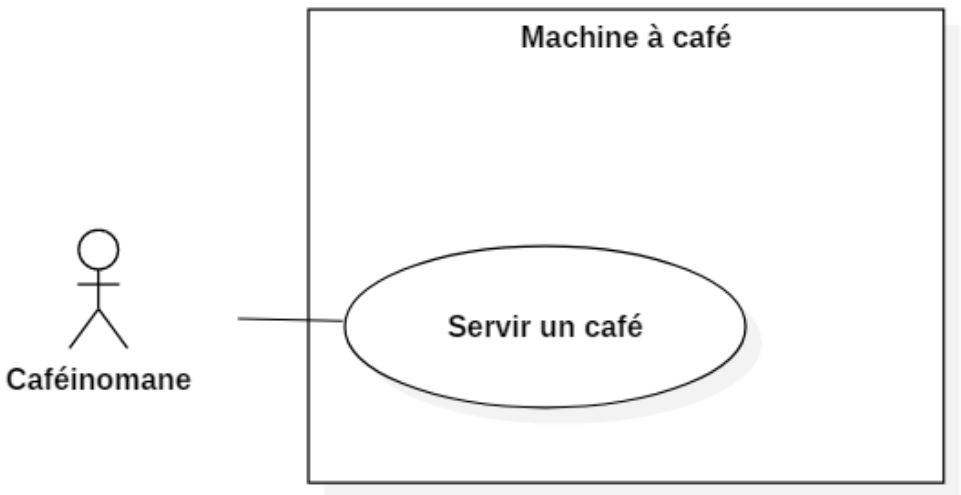
Les acteurs

Un acteur est relié à un ou plusieurs cas d'usage via une relation d'*association*, symbolisé par un trait.



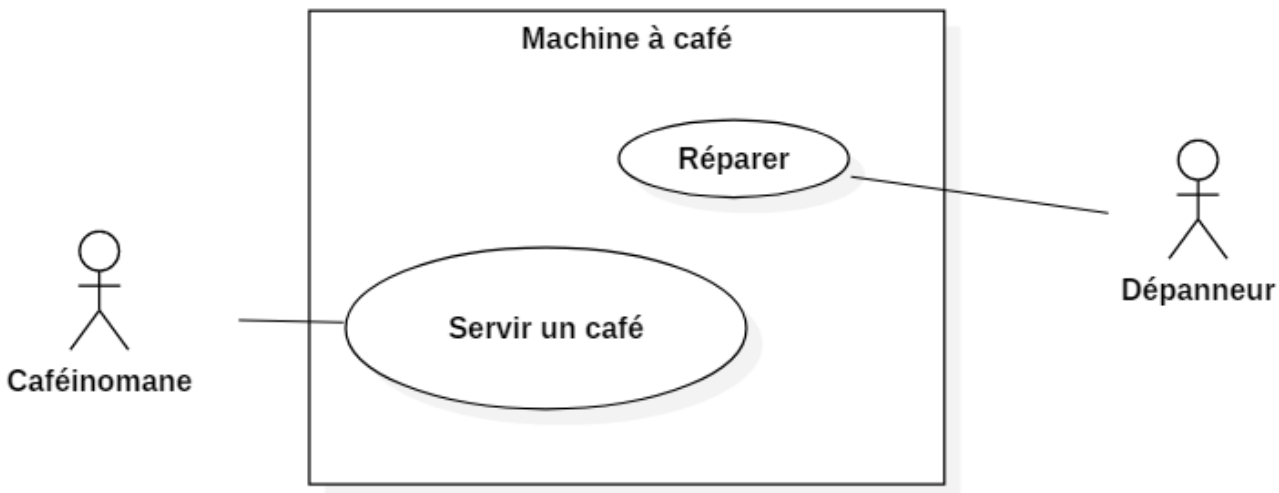
Les acteurs

Un acteur ne représente pas **une** personne mais un **rôle** dans l'interaction avec le système.
Plusieurs personnes peuvent être le même acteur...



Les acteurs

Un acteur ne représente pas **une** personne mais un **rôle** dans l'interaction avec le système.
Plusieurs personnes peuvent être le même acteur...
..et une personne peut être plusieurs acteurs



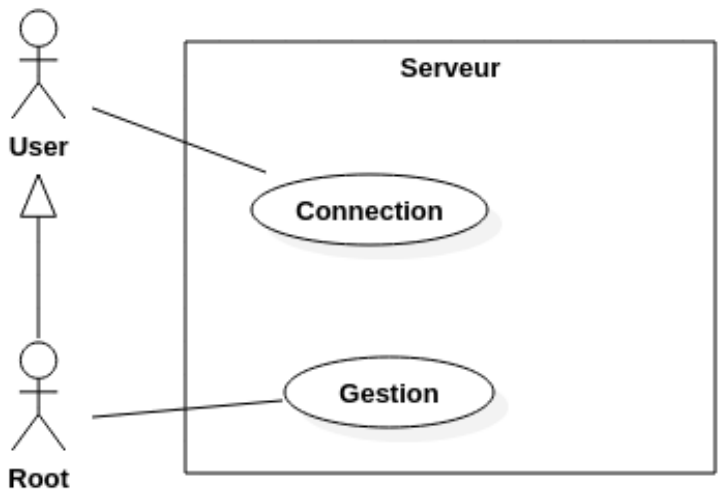
Les acteurs

Les acteurs peuvent être reliés entre eux par une *relation de généralisation* (ou *héritage*).

L'acteur enfant peut utiliser les cas d'usage du parent.

Les acteurs

Les acteurs peuvent être reliés entre eux par une *relation de généralisation* (ou *héritage*).



Les cas d'usage

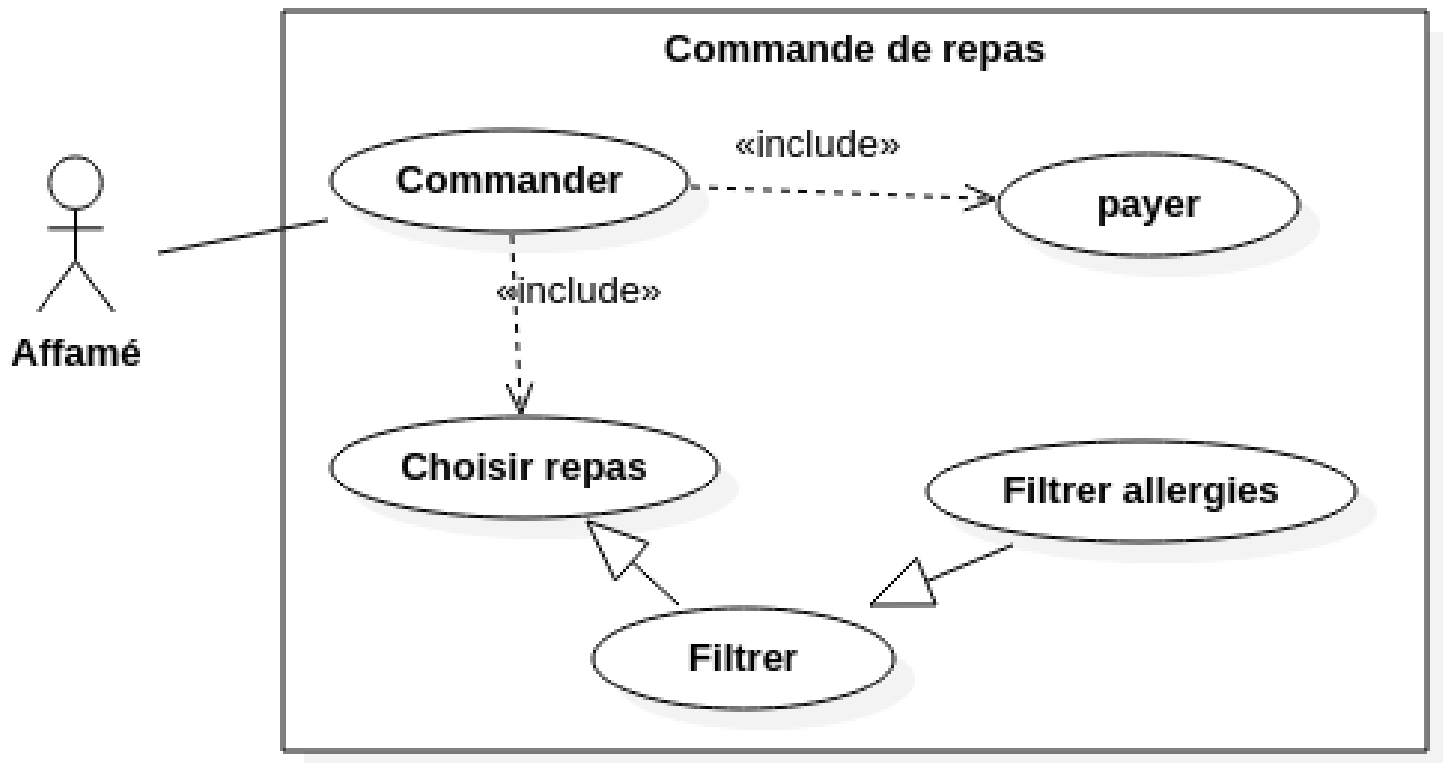
Les cas d'usage servent à modéliser :

- une fonctionnalité du système
- un échange entre le système et un acteur (envoi d'un message, appui d'un bouton, etc...)

Les cas d'usage

Les cas d'usage peuvent être reliés selon des relations de :

- **Généralisation :** La destination est une généralisation de la source.
  - Permet de spécifier des cas particulier (des extensions)
    - Filtrer → Choisir Repas
  - Sert à factoriser des fonctionnalités similaires.
    - Filtrer allergies, type de cuisine, etc...

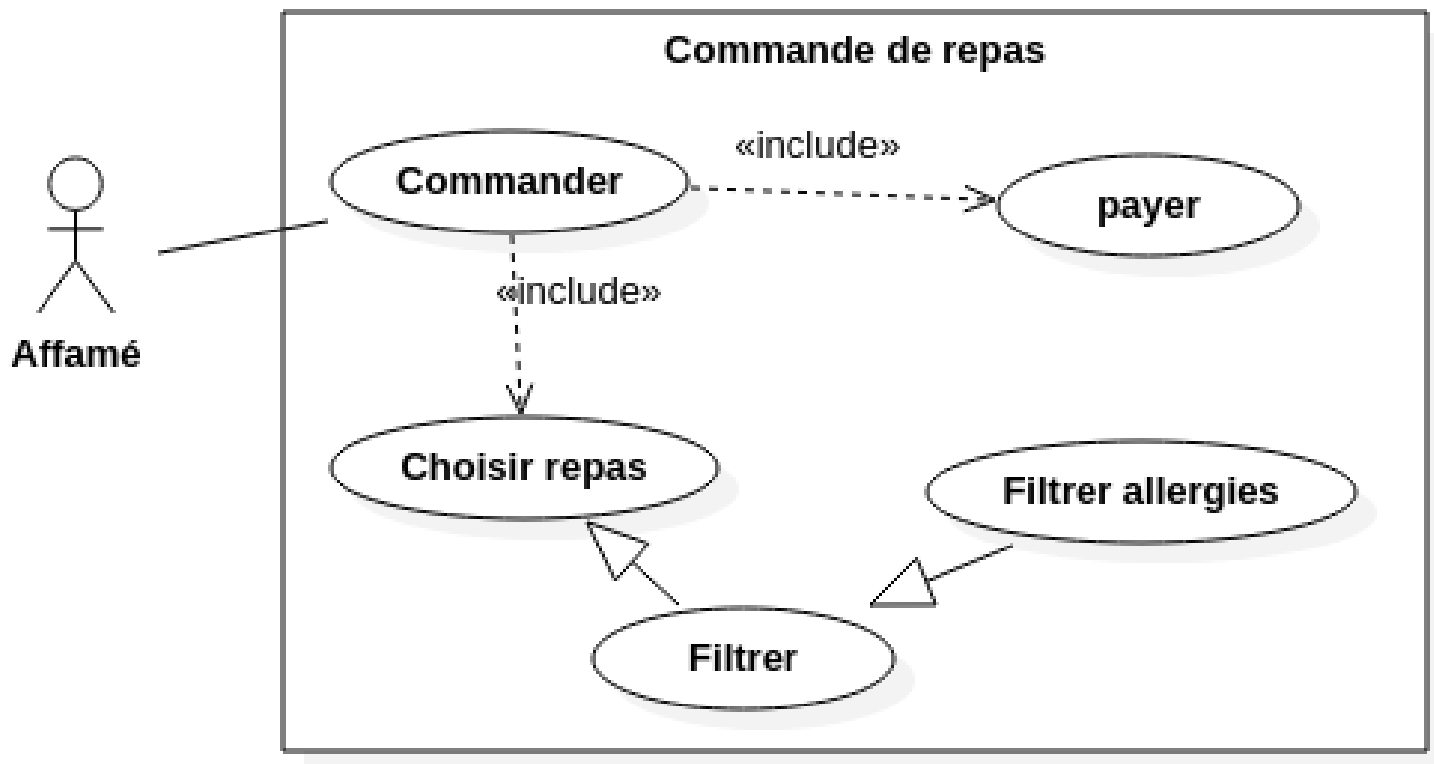


Les cas d'usage

Les cas d'usage peuvent être reliés selon des relations de :

- **Inclusion :** Faire la source *inclut* de faire la destination

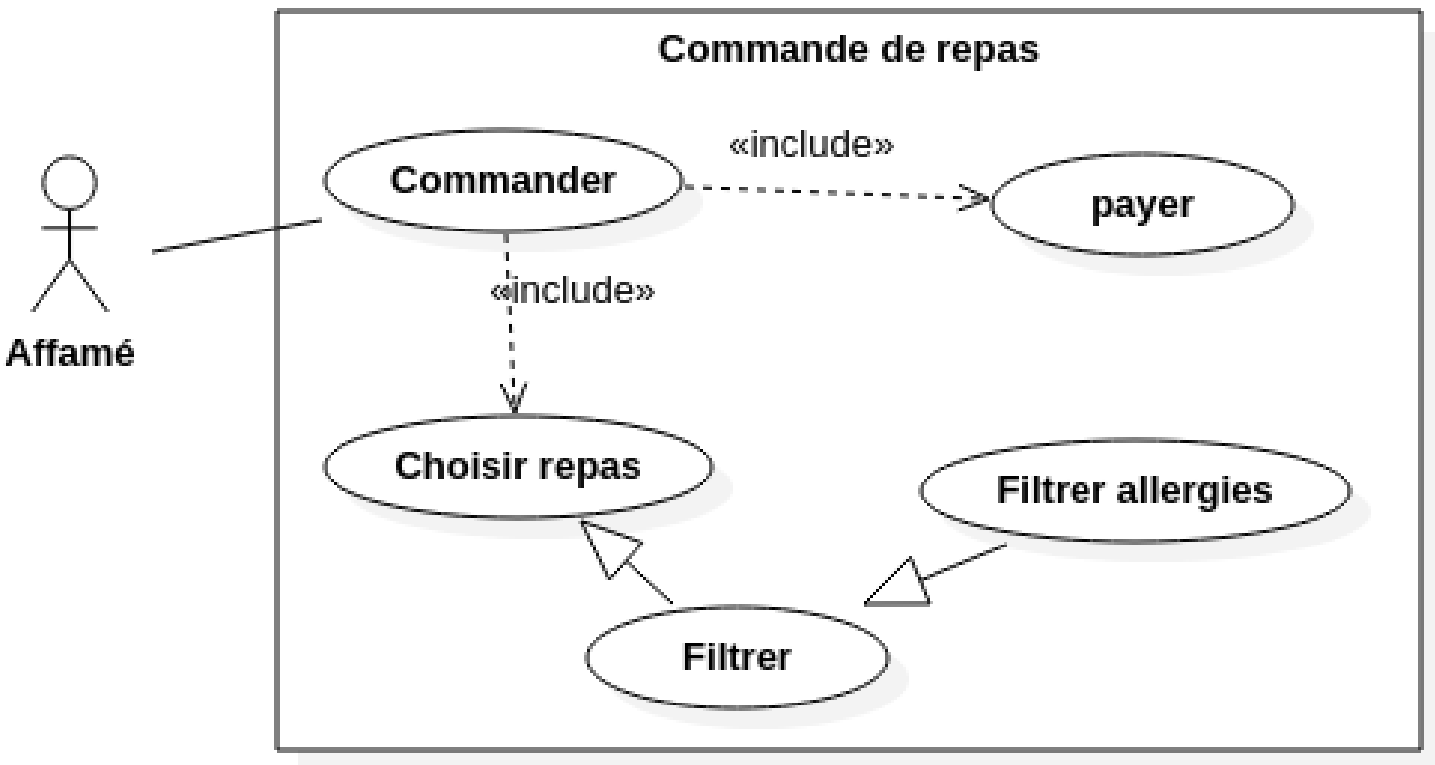
```
public Boolean Commander(...){
 ...
 payer(..);
 ...
 }
```



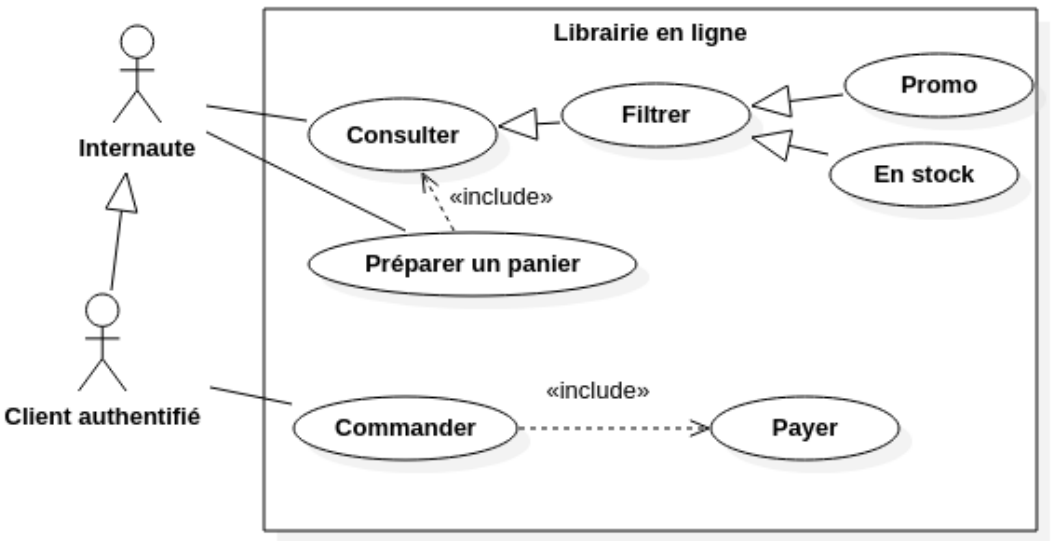
Les cas d'usage

Les cas d'usage peuvent être reliés selon des relations de :

- **Inclusion :** Faire la source *inclut* de faire la destination
  À faire : Modéliser un dépendance fonctionnelle : Commander va appeler la fonction payer.
  À ne pas faire : Modéliser une séquence temporelle : après Commander, il y aura une livraison.

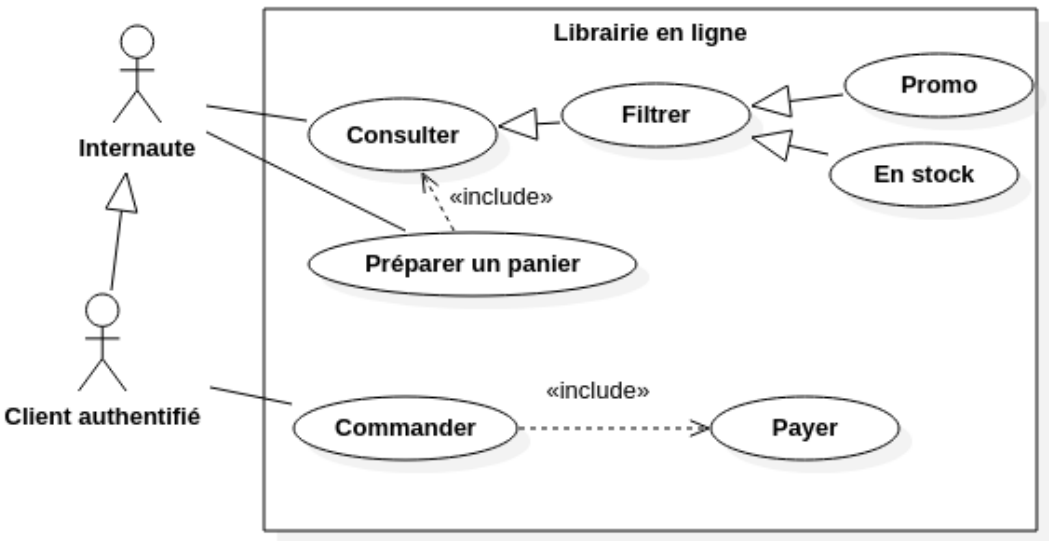


Exemple



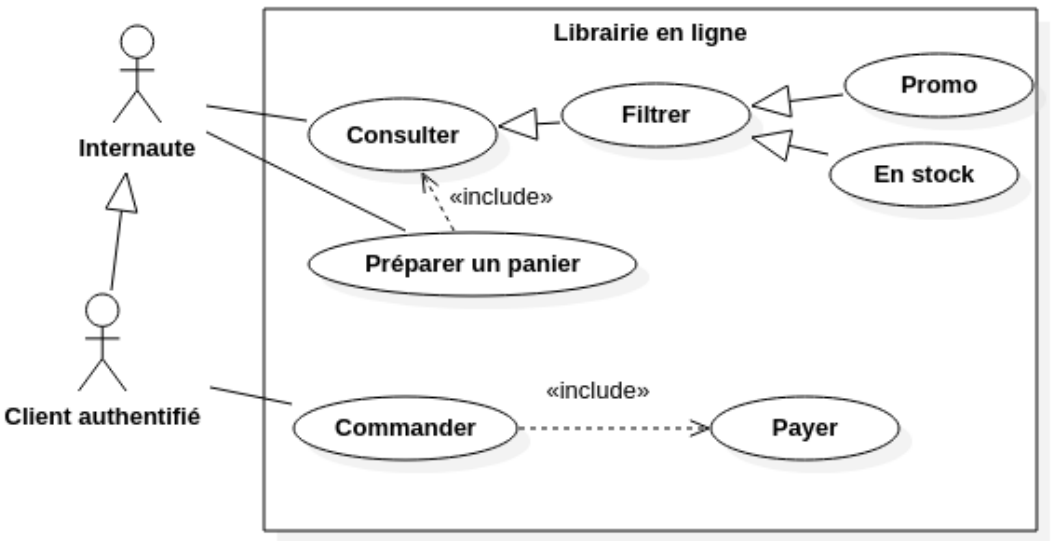
- Quel est le système ?

Exemple



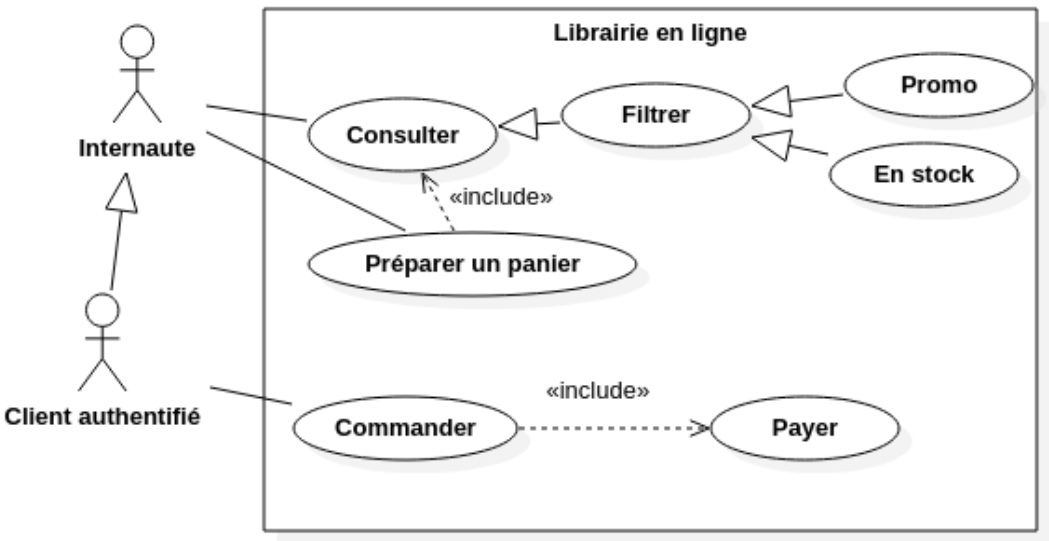
- Quel est le système ?
- Que représentent les acteurs ?

Exemple



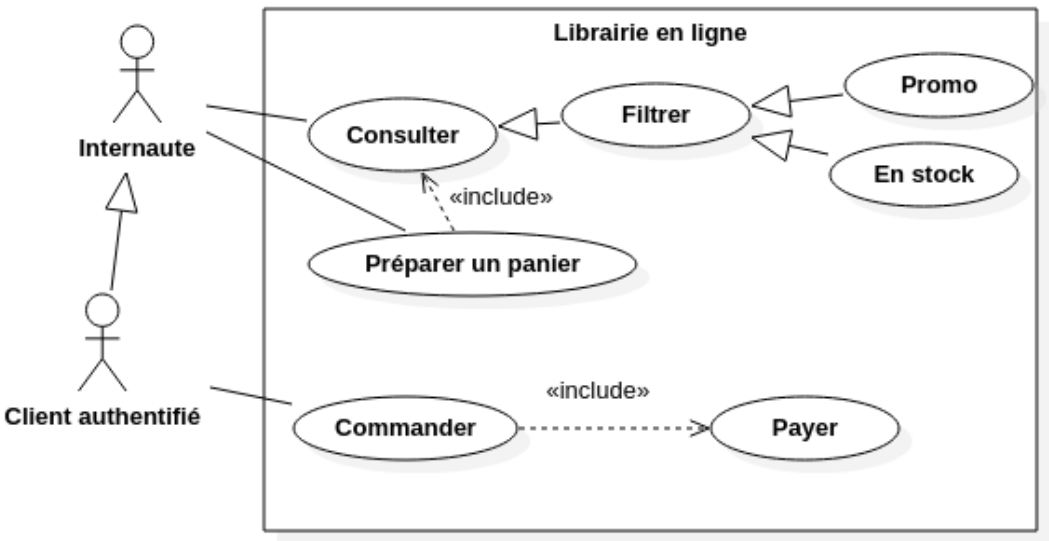
- Quel est le système ?
- Que représentent les acteurs ? Et leurs relations ?

Exemple



- Quel est le système ?
- Que représentent les acteurs ? Et leurs relations ?
- Quels sont les cas d'usage ?

Exemple



- Quel est le système ?
- Que représentent les acteurs ? Et leurs relations ?
- Quels sont les cas d'usage ?
- Que modélisent leurs relations ?

Un *bon* diagramme de cas d'usage

À faire

- Prendre un système simple,
- Rester concis (si plus de 10 cas d'usage, couper le diagramme),
- Limiter le nombre d'interactions avec les acteurs en factorisant les usages similaires

Un *bon* diagramme de cas d'usage

À faire

- Prendre un système simple,
- Rester concis (si plus de 10 cas d'usage, couper le diagramme),
- Limiter le nombre d'interactions avec les acteurs en factorisant les usages similaires

À ne pas faire

- Vouloir à tout prix mettre en relations les acteurs (ex: tout dépanneur peut être utilisateur mais il n'y a pas de dépendance de rôle)
- Utiliser la relation d'inclusion pour décrire une suite d'action (on aura les diagramme de séquence pour ça)
- Décrire la machinerie interne du système, et utiliser des cas d'usage sans interaction avec l'extérieur

Étude de cas

On veut créer un logiciel embarqué pour gérer une machine à café. Évidement, on veut pouvoir utiliser la machine pour commander un café, qui coûte $50c$. On veut également avoir la possibilité d'ajouter du sucre et/ou du lait. Enfin, un dépanneur doit pouvoir ouvrir la machine pour raisons de maintenance.

Étude de cas

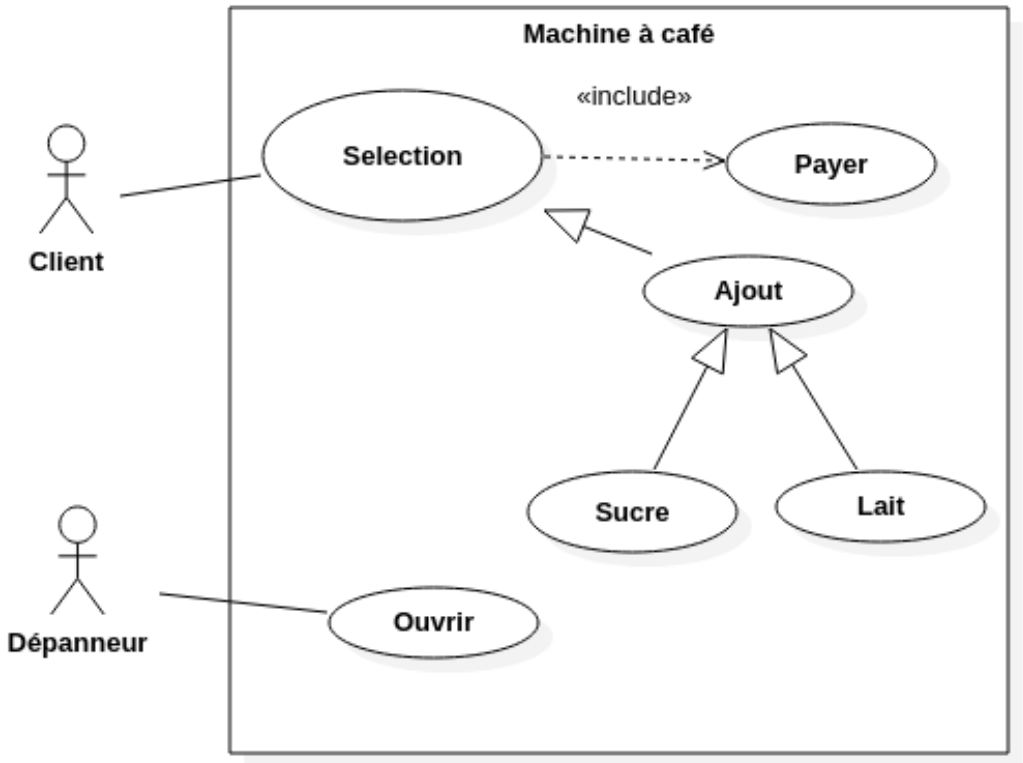
On veut créer un logiciel embarqué pour gérer une machine à café. Évidement, on veut pouvoir utiliser la machine pour commander un café, qui coûte $50c$. On veut également avoir la possibilité d'ajouter du sucre et/ou du lait. Enfin, un dépanneur doit pouvoir ouvrir la machine pour raisons de maintenance.

Pour modéliser ce système, on se pose les questions suivantes :

- Quel est le système ?
- Qui sont les acteurs ? Leurs relations ?
- Quels sont les cas d'usage ?
- Et leurs relations ?

Étude de cas

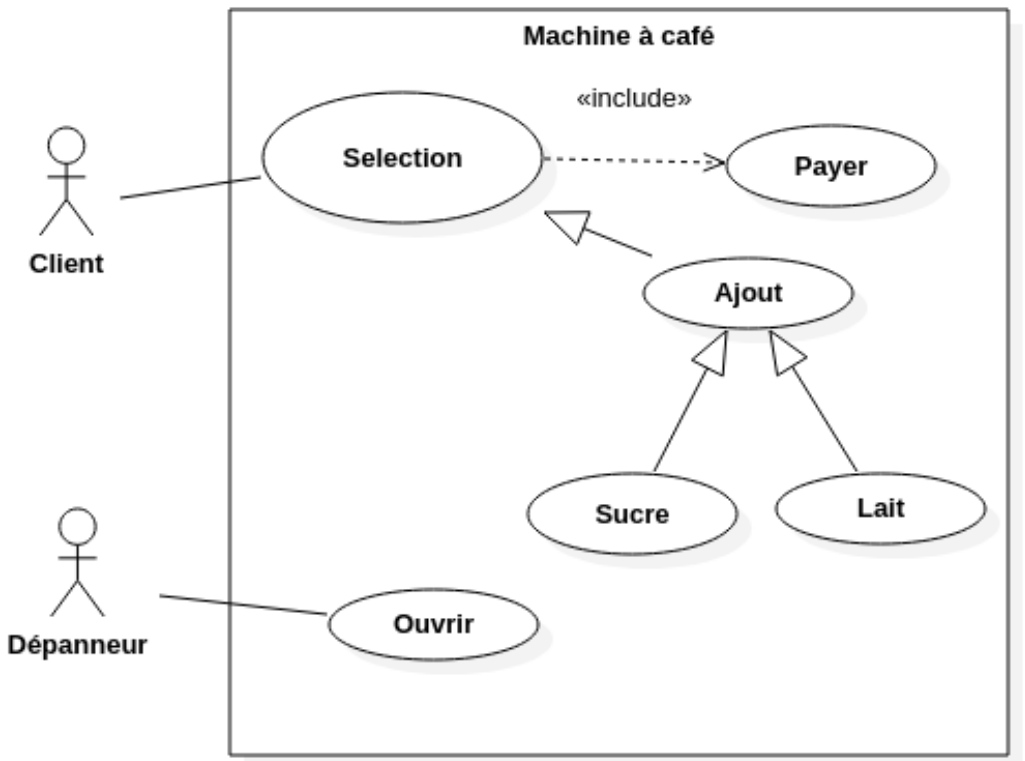
Exemple de réponse :



Il n'y a pas qu'une seule bonne réponse, mais la façon de répondre va influer sur le développement futur !

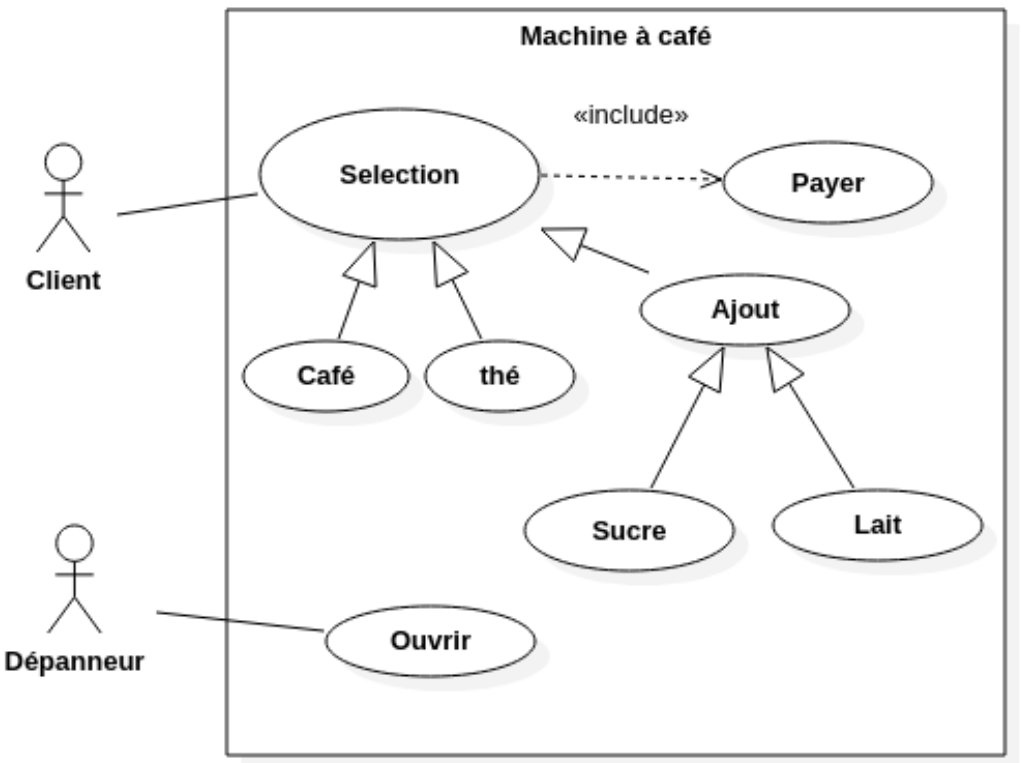
Étude de cas

Exemple de réponse :

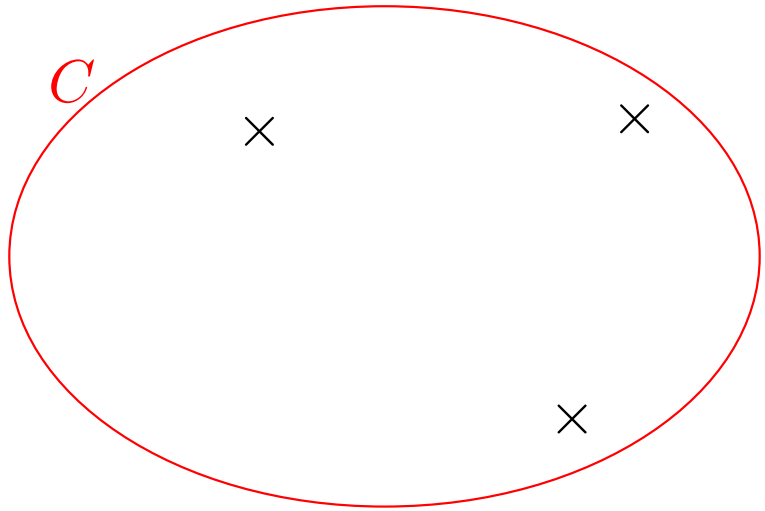


Et si on veut pouvoir choisir entre thé et café ?

Étude de cas

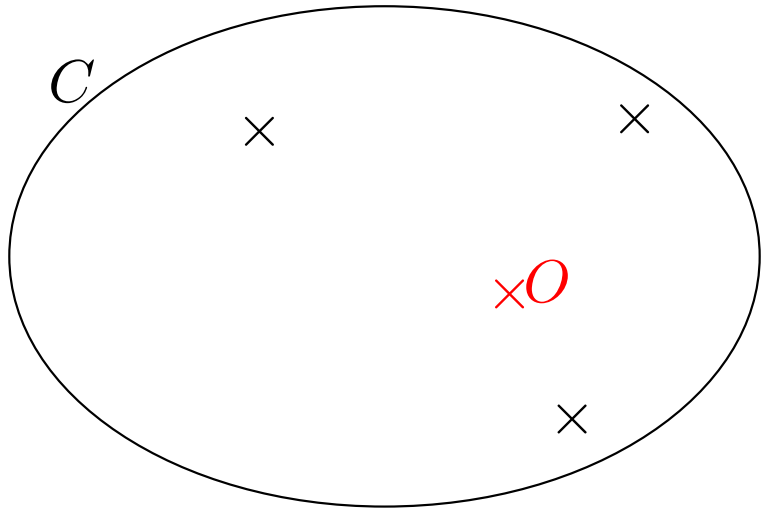

Machine à café
«include»
Selection
Payer
Client
Ajout
Café
thé
Sucre
Lait
Dépanneur
Ouvrir

Qu'est-ce qu'une classe ?



Une *classe* décrit un ensemble d'*objets*. Elle est définit par une *propriété* que tous ses objets vérifient.
Example : La classe Point est composée de 2 entiers x et y.

Qu'est-ce qu'une classe ? Et un objet ?

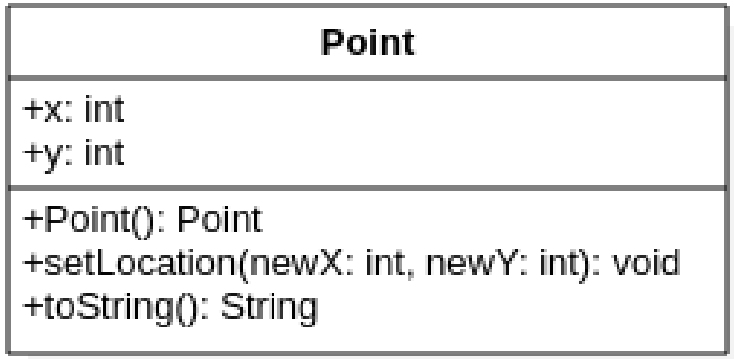


Une *classe* décrit un ensemble d'*objets*. Elle est définit par une *propriété* que tous ses objets vérifient.
Example : La classe Point est composée de 2 entiers x et y.

Par opposition, un *objet* est un élément de l'ensemble décrit par la classe.
Example : Le point p:(x=3,y=4) est un objet de classe Point.

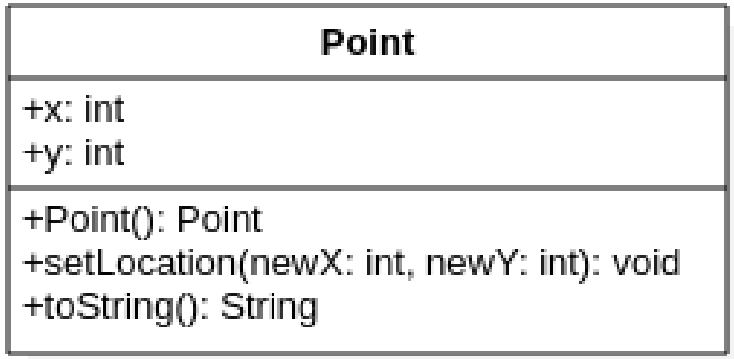
Example

Classe en UML



Example

Classe en UML

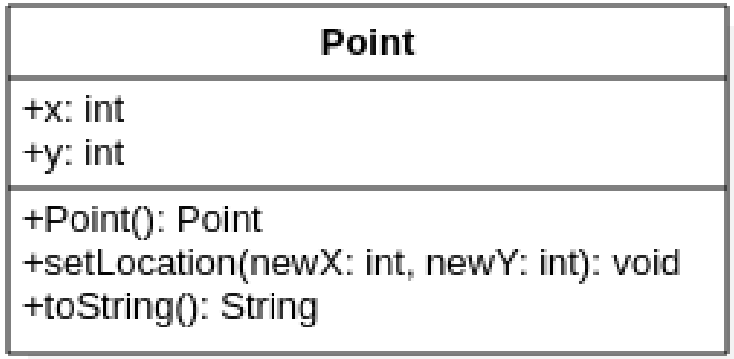


Codage en Java

```
public Class Point {
int x;
int y;
public Point() {...}
public void setLocation(int NewX,int
NewY){
this.x=NewX;
this.y=NewY;}
}
```

Example

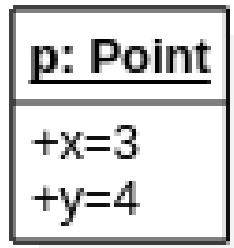
Classe en UML



Codage en Java

```
public Class Point {
int x;
int y;
public Point() {...}
public void setLocation(int NewX,int
NewY){
this.x=NewX;
this.y=NewY;}
}
```

Objet en UML

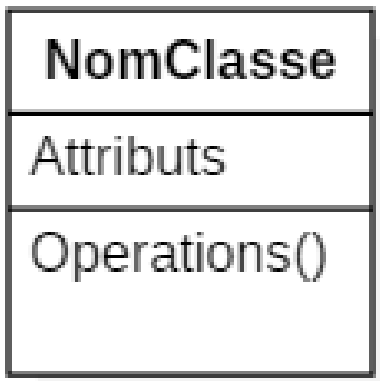


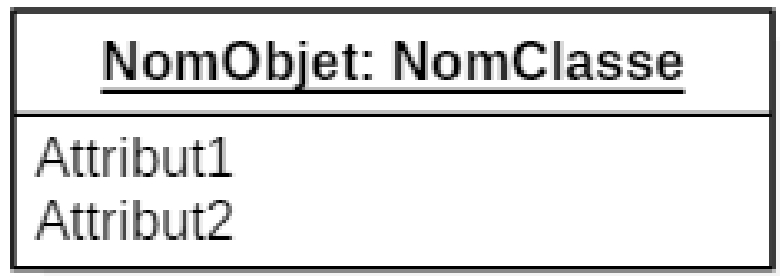
Codage en Java

```
public static void main
(String[] args){
Point p=new Point();
p.setLocation(3,4);
System.out.println(p);
}
```

Représenter une classe

Dans un diagramme de classe, une classe et un objet sont représentés par :

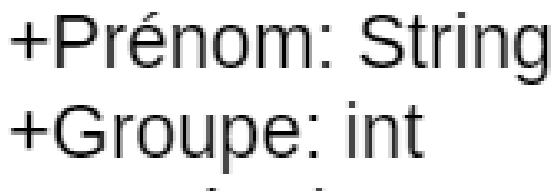




Point formel : le texte **en gras** ou sous-ligné fait partie de la nomenclature.

Les attributs

Les attributs sont les paramètres de l'objet. Ils peuvent être vus comme les variables de l'objet. Elles servent à le caractériser.

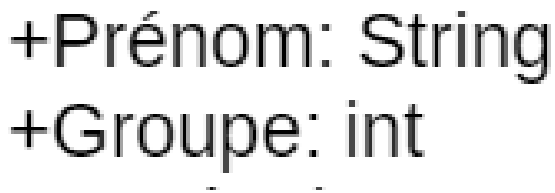


La notation générale pour une classe est :

$$visibilite\ nom :\ type = valeur\ initiale$$

Les attributs

Les attributs sont les paramètres de l'objet. Ils peuvent être vus comme les variables de l'objet. Elles servent à le caractériser.



La notation générale pour une classe est :

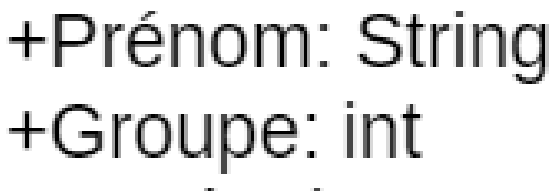
$$visibilite\ nom :\ type = valeur\ initiale$$

Le nom

Il s'agit du nom du paramètre/variable, et donc comment y accéder, principalement dans les opérations.

Les attributs

Les attributs sont les paramètres de l'objet. Ils peuvent être vus comme les variables de l'objet. Elles servent à le caractériser.



La notation générale pour une classe est :

$$visibilite\ nom :\ type = valeur\ initiale$$

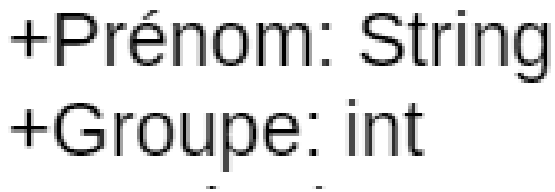
Le type

Il peut s'agir :

- d'un type primitif : int, char, boolean, etc...
- une autre classe

Les attributs

Les attributs sont les paramètres de l'objet. Ils peuvent être vus comme les variables de l'objet. Elles servent à le caractériser.



La notation générale pour une classe est :

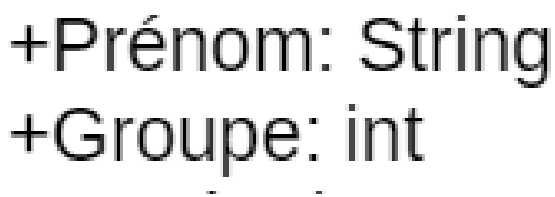
$$visibilite\ nom :\ type = valeur\ initiale$$

La valeur initiale

Paramètre optionnel, à préciser quand il s'applique.
Ex : `Age : int = 0`

Les attributs

Les attributs sont les paramètres de l'objet. Ils peuvent être vus comme les variables de l'objet. Elles servent à le caractériser.



La notation générale pour une classe est :

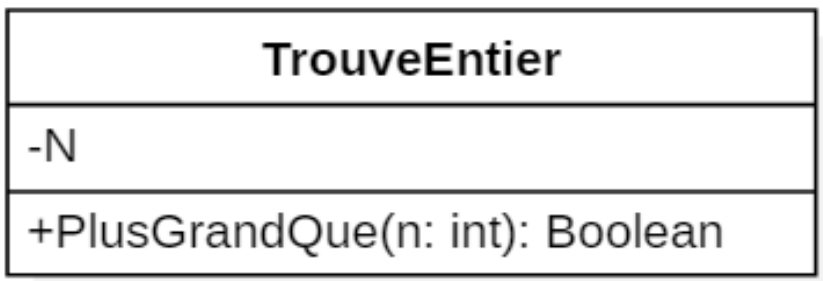
$$visibilite\ nom :\ type = valeur\ initiale$$

Pour un diagramme *objet*, la notation est :

$$nom\ =\ valeur$$

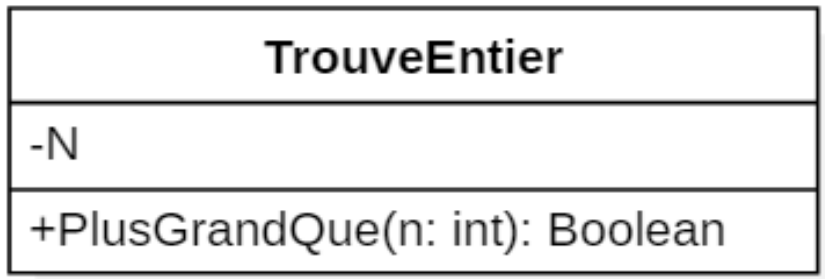
Les opérations

Les opérations sont les interactions de l'objet avec l'exterieur (ou lui-même). Il s'agit de fonctions (méthodes en java) ayant une spécification précise.



Les opérations

Les opérations sont les interactions de l'objet avec l'exterieur (ou lui-même). Il s'agit de fonctions (méthodes en java) ayant une spécification précise.



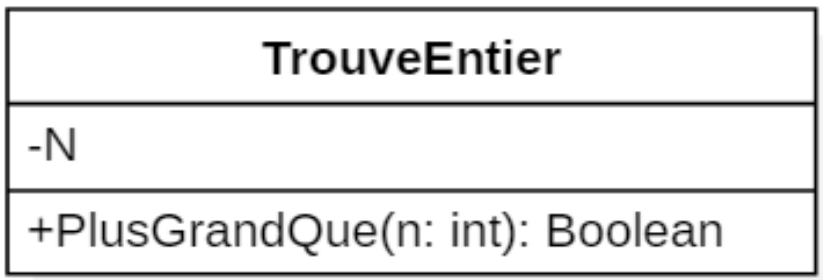
La notation générique est :

$$visibilite\ nom(arg1 : type1, ...) : typeRet$$

- la visibilité est la même qu'avec les attributs
- $arg1$ est le nom du premier argument, et $type1$ son type (primitif ou classe)
- $typeRet$ est le type du retour de la méthode.

Les opérations

Les opérations sont les interactions de l'objet avec l'exterieur (ou lui-même). Il s'agit de fonctions (méthodes en java) ayant une spécification précise.

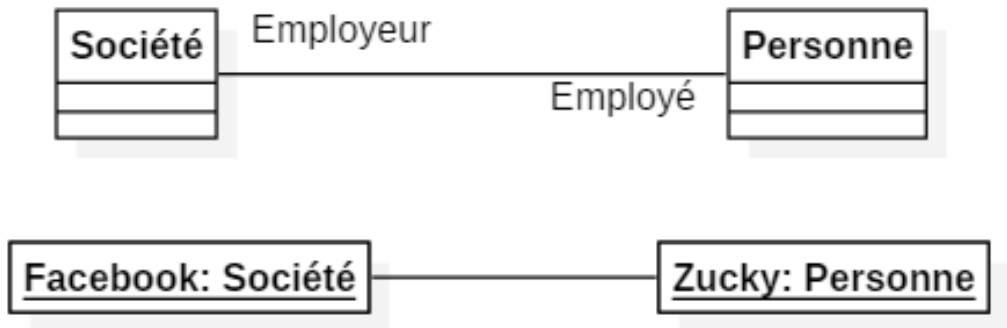


En java, cela se traduit par :

```
public Boolean PlusGrandQue(int n) {
    return (this.N>n);
}
```

Associations

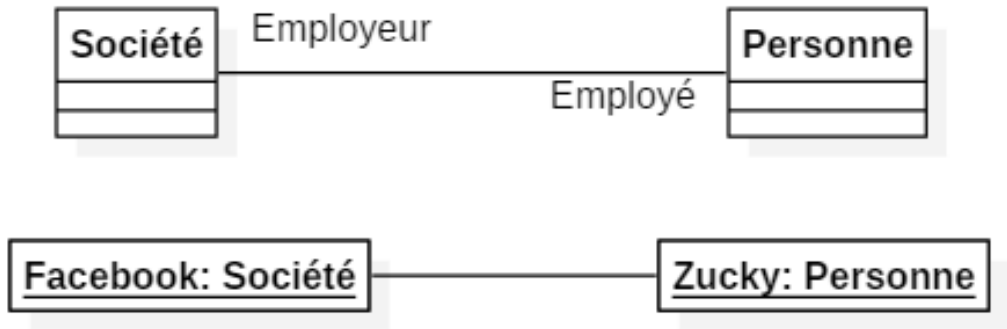
Une association est une relation *structurelle* entre classe d'objets.



Un lien est une *instance* d'une association, càd une relation entre objets.

Associations

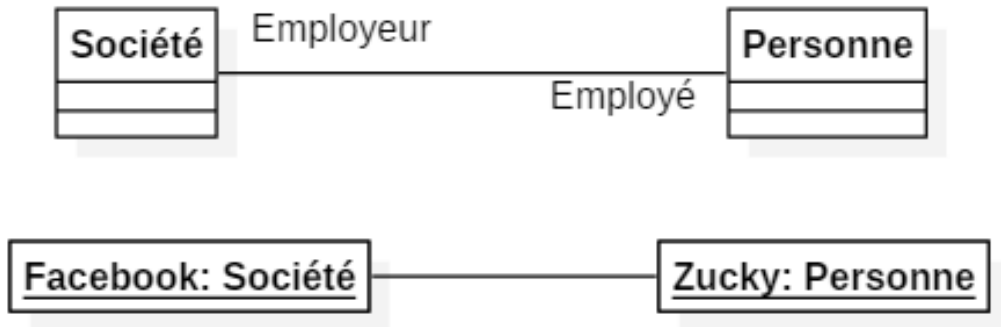
Une association est une relation *structurelle* entre classe d'objets.



Un lien est une *instance* d'une association, càd une relation entre objets.
Similairement aux classes/objets, si une association est vue comme un ensemble, un lien est un élément de cet ensemble.

Associations

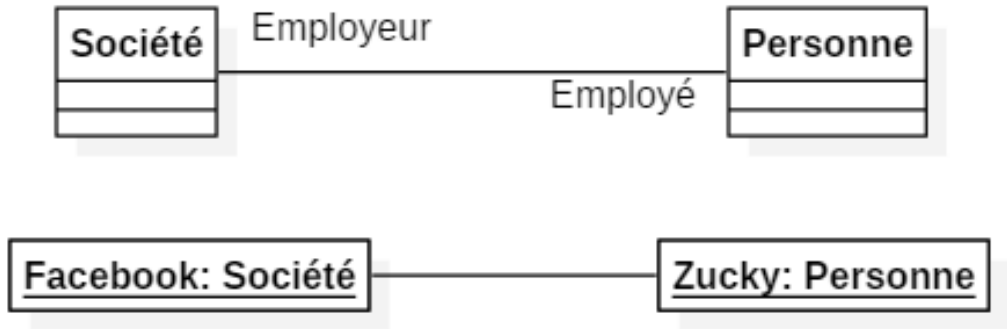
Une association est une relation *structurelle* entre classe d'objets.



Un lien est une *instance* d'une association, càd une relation entre objets.
Similairement aux classes/objets, si une association est vue comme un ensemble, un lien est un élément de cet ensemble.
Ici, le lien est défini par le couple (Facebook,Zucky).
→ Pour une association donnée, il y a généralement **au plus un** lien par couple d'objets.

Associations

Une association est une relation *structurelle* entre classe d'objets.



Un lien est une *instance* d'une association, càd une relation entre objets.

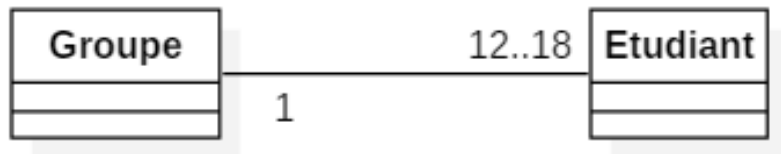
Côté Java

Une association sera le plus souvent implémentée par l'une des deux classes de la relation.
Ici, ce sera par exemple un attribut chez Personne:

```
Société Employeur;
```

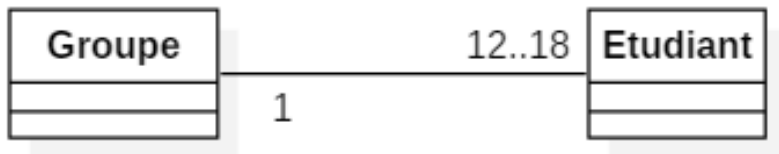
Multiplicité

Une relation peut relier plusieurs objets d'une même classe à un ou plusieurs objets. C'est la *multiplicité* de la relation.
→ Chaque objet Etudiant a un lien avec exactement un objet Groupe.



Multiplicité

Une relation peut relier plusieurs objets d'une même classe à un ou plusieurs objets. C'est la *multiplicité* de la relation.
→ Chaque objet Etudiant a un lien avec exactement un objet Groupe.

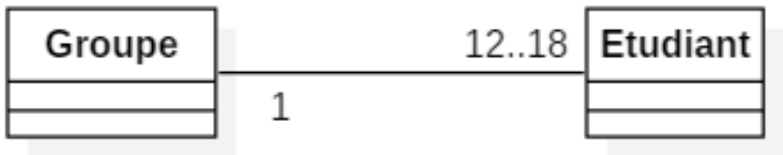


La multiplicité se note de la façon suivante :

n :	Exactement n éléments
$*$:	Un nombre arbitraire (entre 0 et moult)
$n..m$:	entre n et m éléments.

Multiplicité

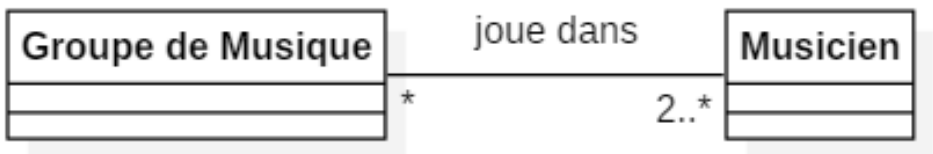
Une relation peut relier plusieurs objets d'une même classe à un ou plusieurs objets. C'est la *multiplicité* de la relation.
→ Chaque objet Etudiant a un lien avec exactement un objet Groupe.



La multiplicité se note de la façon suivante :

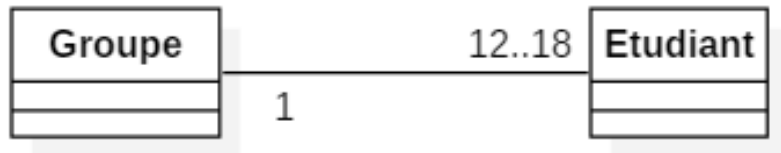
n :	Exactement n éléments
$*$:	Un nombre arbitraire (entre 0 et moult)
$n..m$:	entre n et m éléments.

Example :



Multiplicité

Une relation peut relier plusieurs objets d'une même classe à un ou plusieurs objets. C'est la *multiplicité* de la relation.
→ Chaque objet Etudiant a un lien avec exactement un objet Groupe.



La multiplicité se note de la façon suivante :

n :	Exactement n éléments
$*$:	Un nombre arbitraire (entre 0 et moult)
$n..m$:	entre n et m éléments.

Question :

Quelle est la différence entre $0..*$ et $*$?

Cohérence avec les DCU

Les diagrammes de classes viennent compléter les diagrammes de cas d'usage.
$\rightarrow$ Les différents diagrammes doivent être cohérents entre eux !

Cohérence avec les DCU

Les diagrammes de classes viennent compléter les diagrammes de cas d'usage.
→ Les différents diagrammes doivent être cohérents entre eux !

- Les systèmes et les acteurs sont des classes
- Les cas d'usage sont des opérations du système dont ils font partie.
- Les cas d'usage directement reliés à l'extérieur doivent être public.

Cohérence avec les DCU

Les diagrammes de classes viennent compléter les diagrammes de cas d'usage.
→ Les différents diagrammes doivent être cohérents entre eux !

- Les systèmes et les acteurs sont des classes
- Les cas d'usage sont des opérations du système dont ils font partie.
- Les cas d'usage directement reliés à l'extérieur doivent être public.