PROGRAMMATION EN SCALA

Cheikh SARR

Ingénieur
Electronique
et Informatique

sarr.cheikh08@yahoo.com

SCALA

Objectifs

- *Comprendre les apports du langage Scala et de la programmation fonctionnelle*
- *Pouvoir maitriser la programmation Scala*
- *S'avoir s'interfacer avec des programmes Java*

AVANTAGES DE SCALA

- *Langage multiparadigme (prend en charge la programmation orientée objet et la programmation fonctionnelle)*
- *Langage qui peut croître et évoluer avec la demande*
- *Hautement compatible avec Java*
- *Moins de code à écrire que Java*
- *Moins de dettes techniques*
- *Réduction des coûts du projet*
- *Cycle de développement court*
- *Possibilité de combiner facilement avec les autres langages*

Plan

1. ***Les constructions de base du langage***
   - *Les types de base*
   - *Contrôle: if et match – case*
   - *Les boucles: while et for*
   - *Structures: Arrays, Lists, Sets, Maps*
   - *Fonctions*

2. ***Modèle Objet***
   - *Définitions de classes et de constructeurs*
   - *Méthodes: redéfinition et surcharge*
   - *Objets Singletons*

3. ***Interfaçage avec le java***
   - *Fonctionnement de scala byte code*
   - *Différences entre Java et scala*
   - *Appel de classes scala depuis du code Java*
   - *Utilisation de bibliothèques Java dans un programme scala*

Les constructions de base du langage

INTRODUCTION

- *Scala pour « Scalable Language »*
  - *La **“scalabilité”** c’est la capacité à grandir beaucoup, changer d’échelle, rapidement et avec des moyens réduits.*
- *Conçu par **Martin Odersky** de **L’EPFL** **(Ecole polytechnique Fédérale de Lausanne)***
  - *Un des concepteurs du compilateur Java*
- *Compatible avec java*
- *Un compilateur et un interprète(Intégrés dans l’IDE Eclipse)*
- *Il est récent mais se propage très rapidement*

LE CONCEPT DE CLASSE

Une classe est une structure abstraite qui décrit des objets du monde réel sous deux angles

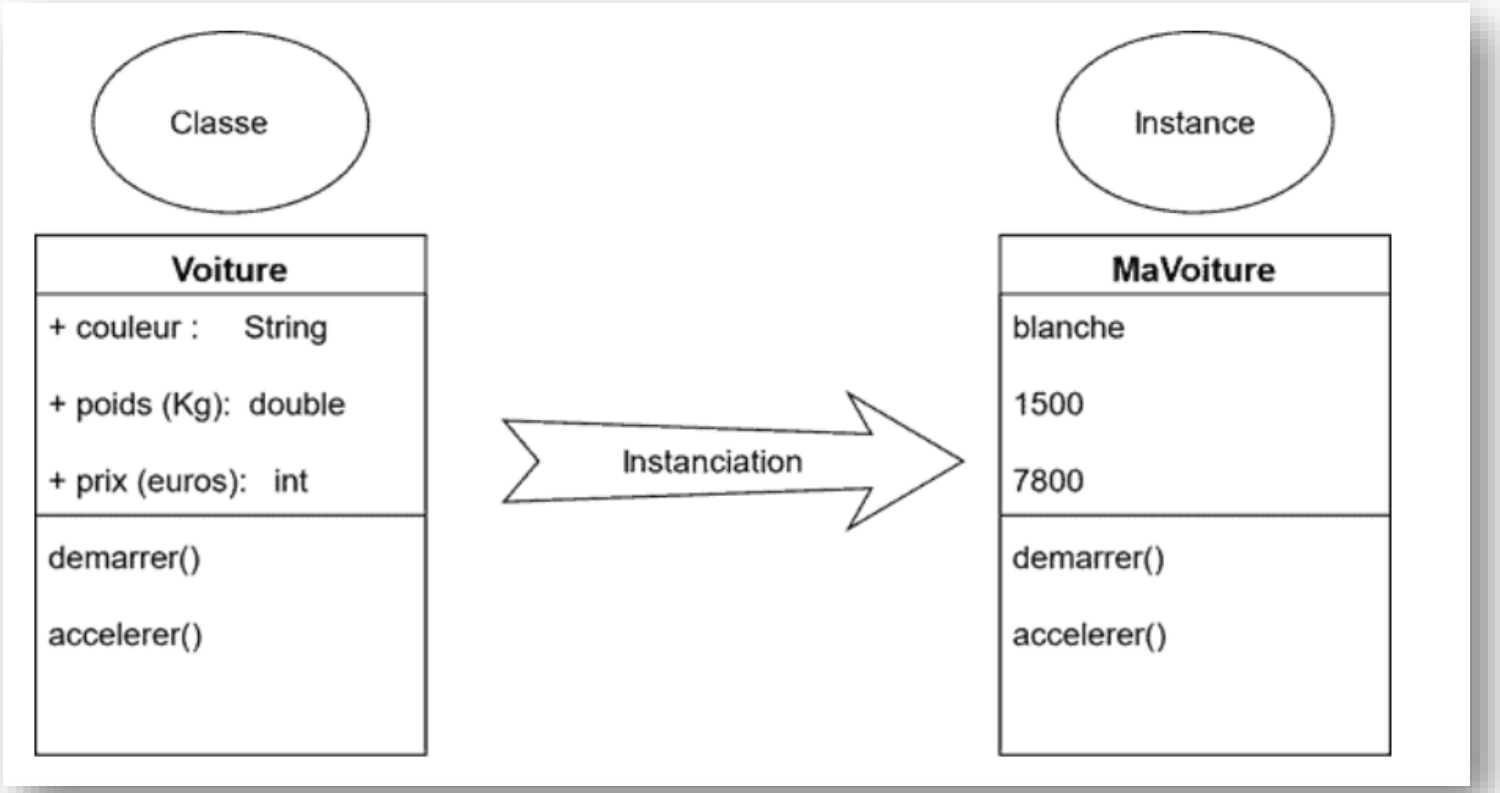
- *Ses propriétés* (ses caractéristiques)
- *Ses méthodes* (les actions qu'elle peut effectuer ou son comportement).

Exemples:

- *La classe **Véhicule** représente un véhicule*
  Couleur: une propriété
  Accélérer/freiner: deux méthodes
- *La classe **Employés** représente tous les employés*
  Propriétés: nom, prénom, adresse, date de naissance
  Opérations: changement de salaire, prise de congé, prise de retraite, etc

LE CONCEPT DE CLASSE

- *Une classe est une sorte de **modèle***
- *Toutes les instances s'appellent des **objets***
- *Les objets sont construits à partir de la classe **« appelé instanciation »***



SCALA, TOUT est OBJET

- *Un langage purement orienté objet dans le sens où tout est un objet*
- *Il diffère du Java:*
  - *Java distingue les types primitifs (comme Boolean et Int) des types référentiels.*
  - *Les nombres sont des objets*
  - *Exemple: 1 + 2 * 3 / x équivalent à 1.+(2.*(3)./(x)*

SCALA, TOUT est OBJET

- *Un langage purement orienté objet dans le sens où tout est un objet*
- *Il diffère du Java:*
  - *Java distingue les types primitifs (comme Boolean et Int) des types référentiels.*
  - *Les nombres sont des objets*
  - *Exemple: 1 + 2 * 3 / x équivalent à 1.+(2.*(3)./(x)*

LES TYPES DE BASE

- ***Variables et constantes***
  - *La déclaration d'une variable se fait de la manière suivante*
    - ***Var nomVariable: Type***
    - ***Val nomVariable: Type***

  - *Les variables (dont les valeurs changent) sont introduites par le mot clé **var**.*
    - ***var j = 10;*** ***// ici, j est considéré comme entier***

  - *Les constantes à valeurs fixes sont introduites par le mot clé **val**.*
    - ***Val i : Int = 10;*** ***//constante i de valeur 10***

NB: Il est préférable de préciser le type des variables (entier, réel, booléen ....) mais ce n'est pas obligatoire si elles sont initialisées sans ambiguïté...

LES TYPES DE BASE

- ***Variables et constantes***
  - ***Byte*** *: entiers signés de 8-bits*
  - ***Short*** *: entiers signés de16-bits*
  - ***Int*** *: entiers signés de 32-bits*
  - ***Long*** *: entiers signés de 64-bits*
  - ***Float*** *: réels sur 32-bits*
  - ***Double*** *: réels 64-bits*
  - ***Boolean*** *: true or false*
  - ***Char*** *: charactère Unicode de16-bit*
  - ***String*** *: une suite de caractères*

LES TYPES DE BASE

❖ *Conversions entre les nombres*

- *Pour transformer un type à un autre type, il faut utiliser les opérations comme:*
  - ***toDouble, toFloat, toInt, toLong***

- ***Exemple 1:***
  - ***var valeur:Int = 10 ;*** *// ici, valeur est considéré comme entier*
  - ***var valeur = valeur.toDouble ;*** *// ici, valeur est castée comme un double*

- ***Exemple 2:***
  - ***var valeur:Double = 10;*** *// ici, valeur est considéré comme entier*
  - ***var valeur = valeur.toInt ;*** *// ici, valeur est castée comme un Int*

LES OPERATEURS

❖ **Les opérateurs arithmétiques**

opérateurs	description
+	plus
-	moins
*	signe de multiplication
/	Signe de division
%	reste

LES OPERATEURS

❖ **Les opérateurs relationnels**

opérateurs	description
==	égal
! =	N'est pas égal
>	plus de
<	moins que
> =	Supérieur ou égal
<=	Inférieur ou égal

LES OPERATEURS

❖ **Les opérateurs logiques**

opérateurs	description
&&	logique et
||	logique ou
!	NON logique

LES CONDITIONS

❖ **Condition « IF »**

```
if(condition)
{
    // Code à Exécuter
}
```

```
var age:Int = 20;
if(age > 18)
{
    println ("Age is greate than 18")
}
```

LES CONDITIONS

❖ Condition « IF- Else»

```
if(condition){

    // Code à exécuter
}
else
{
    // Code à exécuter
}
```

```
var number:Int = 21
if(number%2==0)
{
    println("Even number")
}
else
{
    println("Odd number")
}
```

LES CONDITIONS

❖ Condition « IF-Else-IF»

```
if (condition1){
    //Code à exécuter
}
else if (condition2)
{
    //Code à exécuter
} else if (condition3)
{
    //Code à exécuter
}
```

```
var number:Int = 85

if(number>=0 && number<50)
{
    println ("fail")
}
else if(number>=50 && number<60)
{
    println("D Grade")
}
else if(number>=60 && number<70)
{
    println("C Grade")
}
```

LES BOUCLES

❖ Boucles for dans une plage

```
for( var x <- Range ){
   //code à exécuter;
}
```

- **Exemple**

```
object Demo {

   def main(args: Array[String]) {
      var a = 0;

      for( a <- 1 to 10){
         println( "Valeur de a: " + a );
      }
   }
}
```

LES BOUCLES

❖ Boucles for dans une collection

```
for( var x <- List ){
   //code à éxécuter;
}
```

- **Exemple**

```
object Demo {

   def main(args: Array[String]) {
      var a = 0;
      val numList = List(1,2,3,4,5,6);

      // for loop execution with a collection
      for( a <- numList ){
         println( "Valeur a: " + a );
      }
   }
}
```

LES BOUCLES

❖ Boucles while

```
while(condition){
    //code à exécuter ;
}
```

- **Exemple**

```
object Demo {
   def main(args: Array[String]) {
      var a = 10;

      while( a < 20 ){
         println( "Valeur de a: " + a );
         a = a + 1;
      }
   }
}
```

LES BOUCLES

❖ Boucles do ……..while

```
do {

  //Code à exécuter

} while(condition);
```

- **Exemple**

```
object dowhileLoopDemo
{

    def main(args: Array[String])
    {
        var a = 10;

        do
        {
            print(a + " ");
            a = a - 1;
        }while(a > 0);
    }
}
```

EXERCICES D'APPLICATIONS

APPLICATION 1

Ecrire un programme qui affiche « Hello, World » et affiche la version du langage scala installer

APPLICATION 2

Écrivez un programme Scala pour calculer la somme des deux valeurs entières données. Si les deux valeurs sont identiques, retournez alors le triple de leur somme.

APPLICATION 3

Ecrire un programme Scala pour vérifier deux entiers donnés, et retourner vrai si l'un d'eux vaut 30 ou si leur somme vaut 30.

APPLICATION 4

Écrivez un programme Scala pour vérifier un entier donné et renvoyer vrai s'il est compris entre 20 et 100 ou 300.

APPLICATION 5

Écrire un programme Scala pour échanger le premier et le dernier caractère d'une chaîne donnée et renvoyer la nouvelle chaîne.

APPLICATION 6

Écrivez un programme Scala pour vérifier si l'une des températures données est inférieure à 0 et l'autre supérieure à 100.

APPLICATION 7

Écrivez un programme Scala pour vérifier deux entiers donnés si l'un d'eux est dans la plage 100..200 inclus

APPLICATION 8

Écrivez un programme Scala pour vérifier quel nombre est le plus proche de la valeur 100 parmi deux entiers donnés. Renvoie 0 si les deux nombres sont égaux.

APPLICATION 9

Écrire un programme Scala pour vérifier si deux entiers positifs donnés ont le même dernier chiffre.

APPLICATION 10

Écrivez un programme Scala pour convertir les 4 derniers caractères d'une chaîne donnée en majuscules. Si la longueur de la chaîne est inférieure à 4, alors mettez en majuscule tous les caractères.

APPLICATION 11

Écrivez un programme Scala pour comparer une chaîne donnée à une autre chaîne, en ignorant les majuscules.

APPLICATION 12

Écrivez un programme Scala pour convertir tous les caractères minuscules en majuscules

Le modèle Objet

La modélisation UML

Qu’est ce que l’UML

*UML(**Unified Modeling Language) un langage de modélisation unifié***

***Langage** = syntaxe + sémantique :*

syntaxe : notations graphiques consistant essentiellement en des représentations conceptuelles d'un système

sémantique : sens précis pour chaque notation

Qu’est ce que l’UML

UML est caractérisé par :

- *un travail d'expert*
- *utilise l’approche orientée objet*
- *normalisé, riche*
- *Formel : sa notation limite les ambiguïté et les incompréhensions*
- *langage ouvert*
- *INDÉPENDANT du langage de programmation*

Qu’est ce que l’UML

UML est caractérisé par :

- *un travail d'expert*
- *utilise l’approche orientée objet*
- *normalisé, riche*
- *Formel : sa notation limite les ambiguïté et les incompréhensions*
- *langage ouvert*
- *INDÉPENDANT du langage de programmation*

Diagramme D’UML

UML comprend 9 de diagrammes :

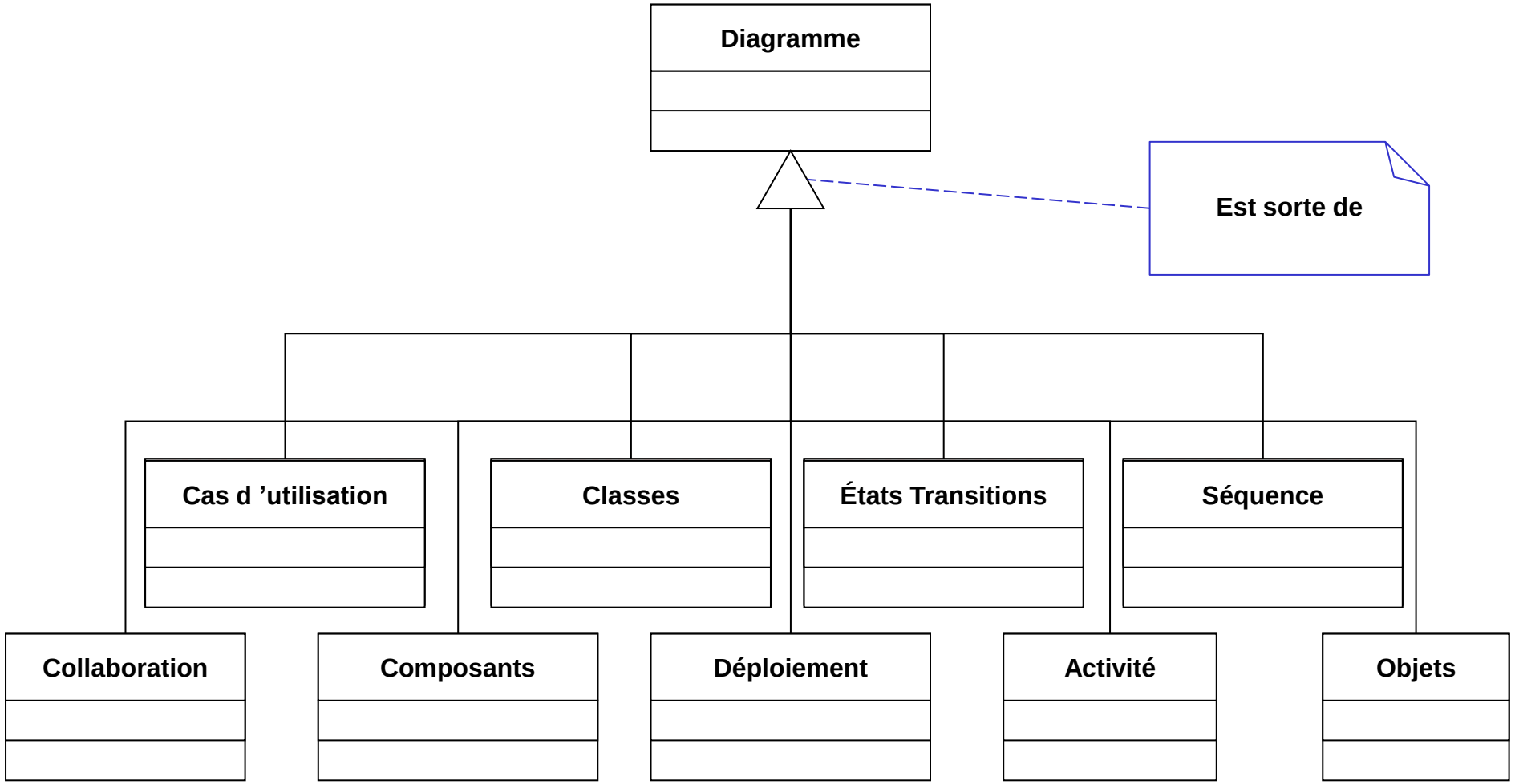


Diagramme D'UML

Chacun de ces diagrammes correspond

- *Soit à la description d'une partie du système*
- *Soit à la description du système selon un point de vue particulier*

- *Diagramme de cas d'utilisation : destiné à représenter les besoins des utilisateurs par rapport au système. Point de vue de l'utilisateur*
- *Diagramme de classes : représente la partie statique du système en intégrant dans chaque classe la partie dédiée aux données et celles consacrée aux traitements. C'est le diagramme pivot de la modélisation du système.*
- *Diagramme objets : représente les instances des classes du diagramme de classes.*
- *Diagramme d'état/transitions : montre les différents états par lesquels passe un objet en réactions aux événements.*

Diagramme D’UML

- *Diagramme d’activités : donné une vision des enchaînements des activités propres à une opération ou un cas d’utilisation.*

- *Diagramme de séquences : permet de décrire les scénarios de chaque cas d’utilisation en mettant l’accent sur la chronologie des interactions entre objets.*

- *Diagramme de collaboration : une autre représentation des scénarios des cas d’utilisation qui met l’accent sur les objets et les messages échangés.*

- *Diagramme de composants : représente les différents constituants logiciel d’un système en terme de modules : fichiers source, librairies, exécutables, etc.*
- *Diagramme de déploiement : montre la disposition physique des matériels qui composent le système et la répartition des composants sur ces matériels.*

Diagramme D’UML

UML définit deux types de diagrammes, structurels (statiques) et comportementaux (dynamiques)

- ***Modélisation de la structure***
  - *diagramme de classes*
  - *diagramme d’objets*
  - *diagramme de composants*
  - *diagramme de déploiement*

- ***Modélisation du comportement***
  - *diagramme de cas d'utilisation*
  - *diagramme d’états*
  - *diagramme d’activités*
  - *diagramme de collaboration*
  - *diagramme de séquence*

Diagramme D’UML

Les diagramme d’UML peuvent être utilisés pour représenter différents points de vues :

- ***Vue externe*** *: vue du système par ses utilisateurs finaux*
- ***Vue logique statique*** *: structure des objets et leurs relations*
- ***Vue logique dynamique*** *: comportement du système*
- ***Vue d’implémentation*** *: composants logiciels*
- ***Vue de déploiement*** *: répartition des composants*

Diagramme D’UML

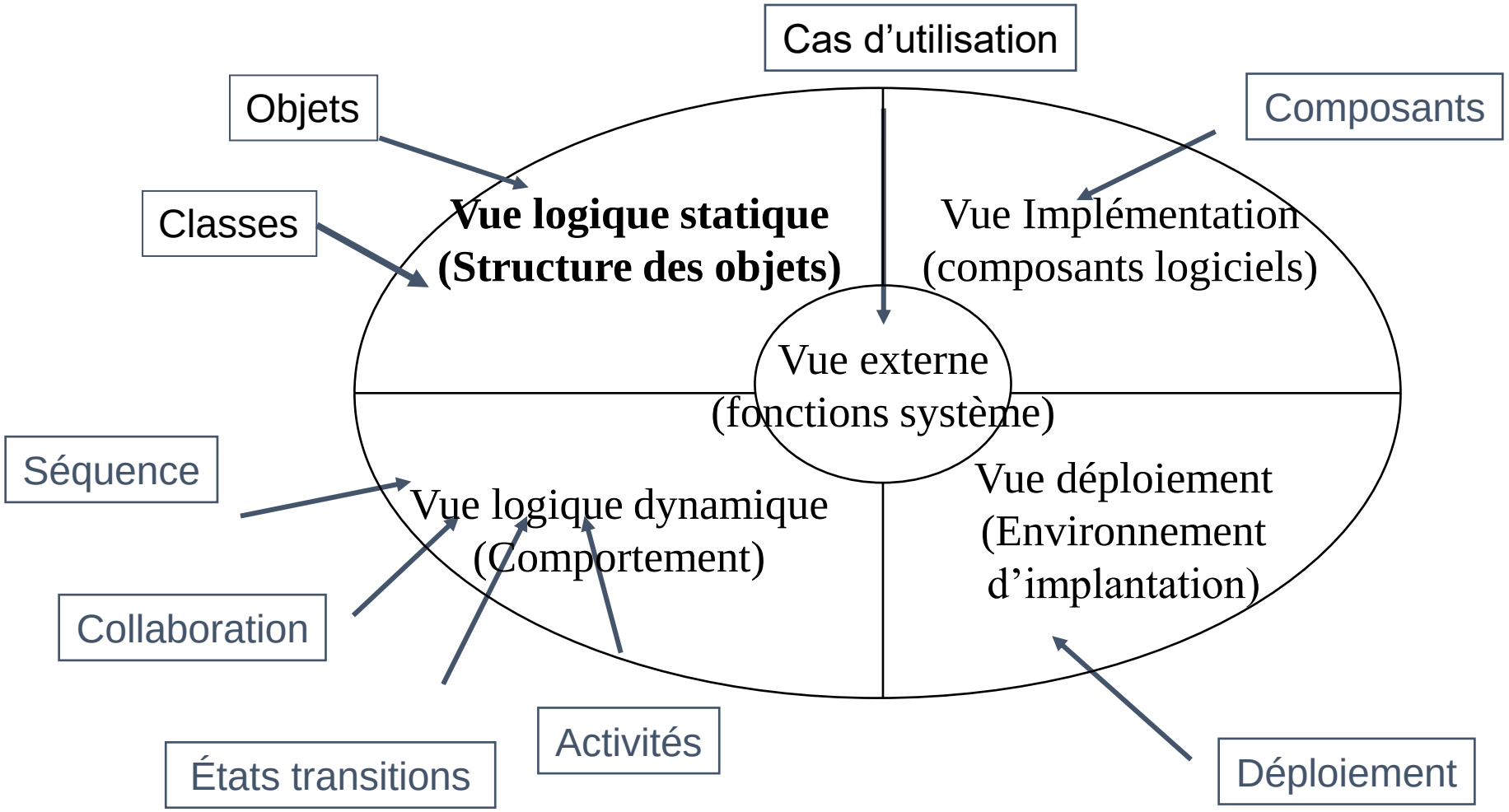


DIAGRAMME DE CLASSE ET D'OBJETS

- *Permet de donner une vue statique du système en terme de :*
  - *Classes d'objets*
  - *Relations entre classes*
    - *Associations*
    - *agrégation/composition*
    - *Héritage*

- *La description du diagramme de classes est centrée sur trois concepts :*
  - *Le concept d'objets*
  - *Le concept de classes d'objets comprenant des attributs et des opérations*
  - *Les différents types de relations entre classes.*

DIAGRAMME DE CLASSE ET D'OBJETS

❖ *Concept d'objets*

- ***Objet** = un concept, abstraction ou une chose autonome qui a un sens dans le contexte du système à modéliser*
- *Un objet doit :*
  - *Être autonome*
  - *Avoir une signification dans le système*
  - *En relation avec d'autres objets*
- *Ne pas confondre "autonomie" avec "indépendance"!!*
- *Exemples*
  - *Gestion de stock : Clients, Commandes, Articles, …*
  - *Gestion scolaire : Étudiants, Modules, Filières, …*

DIAGRAMME DE CLASSE ET D'OBJETS

❖ *Concept d'attributs*

- *Un attribut est une propriété, caractéristique d'un objet. Par exemple :*
  - *Un client a un **nom**, un **prénom**, une **adresse**, un **code client**, …*
  - *Un compte bancaire a un **numéro**, un **solde**, …*
- *Un attribut doit (généralement) avoir une **valeur atomique***

DIAGRAMME DE CLASSE ET D'OBJETS

- *Concept d'attributs*
  - *La description d'un attribut comporte :*

    Visibilité attribut:type[= valeur initiale]

    Où :

    - *Visibilité :*
      - *+ (publique, public) : visible par tous*
      - *- (privée, private) : visible seulement dans la classe*
      - *# (protégée, protected) : visible seulement dans la classe et dans les sous-classes de la classe.*
  - *Nom d'attribut*
  - *Type de l'attribut*
  - *Valeur initiale (facultative)*

DIAGRAMME DE CLASSE ET D'OBJETS

- *Concept d'attributs*

Le type d'un attribut peut être :

- *Un type de base : entier, réel, …*
- *Une expression complexe : tableaux, enregistrements, …*
- *Une classe*

Exemples d'attributs :

- *- couleur : enum{Rouge, Vert, Bleu}*
- *# b : boolean = vrai*
- *- Client : Personne*

DIAGRAMME DE CLASSE ET D'OBJETS

❖*Concept d'attributs*

Lorsqu'un attribut peut être dérivé ou calculé à partir d'autres attributs, il est précédé d'un /. Par exemple, une classe « Rectangle » peut contenir les attributs suivants :

- *longueur : réel,*
- *largeur : réel,*
- */surface : réel.*

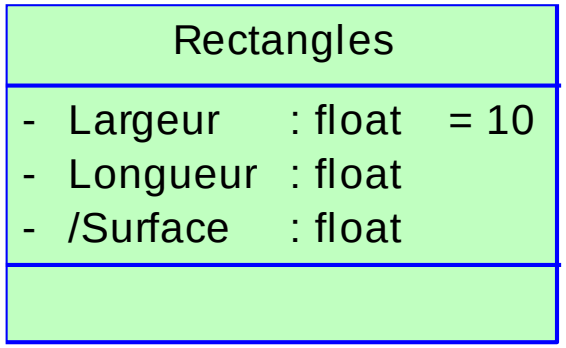


DIAGRAMME DE CLASSE ET D'OBJETS

❖*Concept d'attributs*

On distingue deux types d'attributs :

- *Attribut d'instance :*
  - *Chaque instance de la classe possède une valeur particulière pour cet attribut*
  - *Notation :* ***Visibilité attribut:type[= valeur initiale]***
- *Attribut de classe*
  - *Toutes les instances de la classe possède la même valeur pour cet attribut*
  - *Notation :* ***Visibilité attribut:type[= valeur initiale]***
  - *Équivalent en C++, Java : static*

DIAGRAMME DE CLASSE ET D'OBJETS

- ***Concept d'opération et méthode***

Une opération est :

- *Un service offert par la classe*
- *Une fonction ou une transformation qui peut être appliquée aux objets d'une classe.*
- *permet de décrire le comportement d'un objet. Par exemple, « Embaucher », « Licencier » et « Payer » sont des opérations de la classe « Société ».*

DIAGRAMME DE CLASSE ET D'OBJETS

❖ ***Concept d'opération et méthode***

Une méthode est

- *l'implémentation d'un service offert par la classe (opération).*
- *de différents types :*
  - *accesseurs (get...): renvoie une information sur l'état d'un objet (fonction)*
  - *modifieurs (set...): modifie l'état de l'objet (procédure)*
  - *constructeurs: initialise une nouvelle instance*

DIAGRAMME DE CLASSE ET D'OBJETS

❖ *Concept d'opération et méthode*

La description d'une opération comporte :

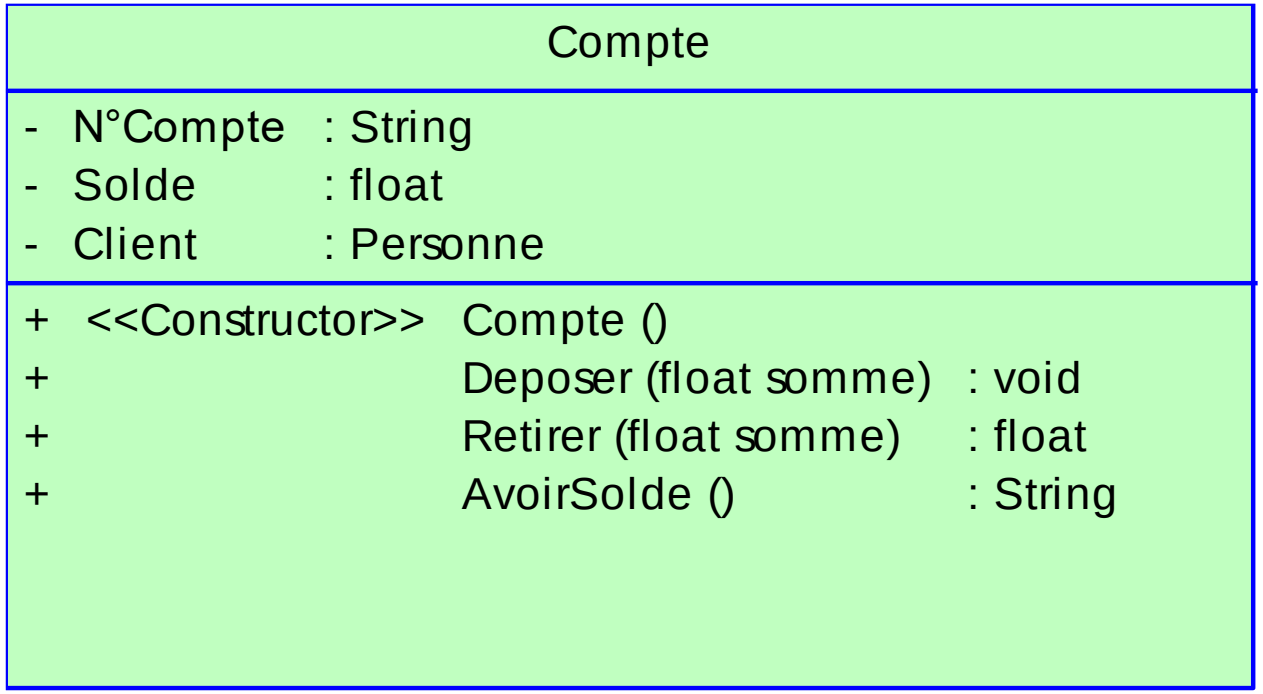
Visibilité opération([arguments:type[=valeur initiale]]):type de résultat

- *Visibilité de l'opération (-, +, #)*
- *Nom de l'opération*
- *Liste des arguments avec leurs types et éventuellement leurs valeurs par défaut*
- *Le type du résultat retourné*

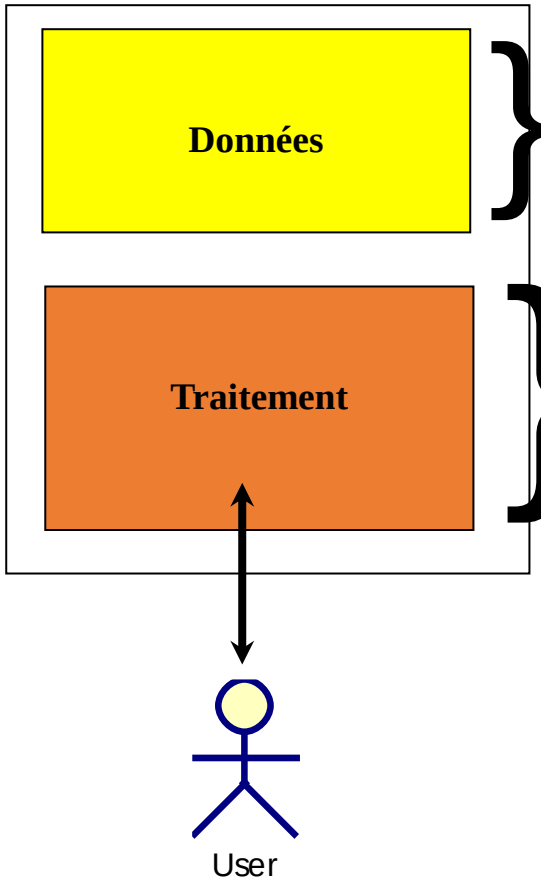
DIAGRAMME DE CLASSE ET D'OBJETS

❖ *Concept d'opération et méthode*

Exemple d'opération



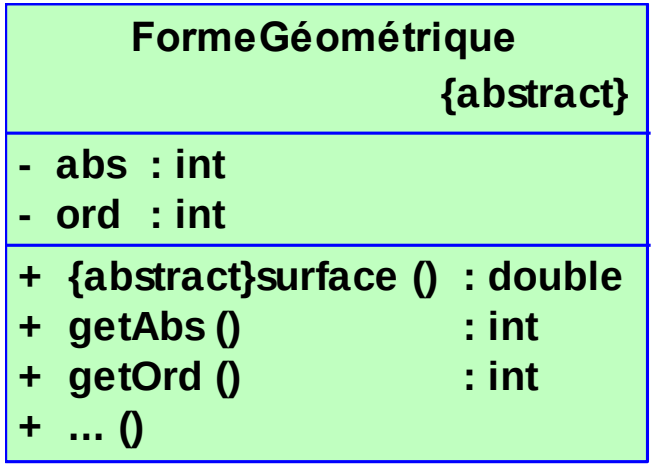
- *Encapsulation est le mécanisme de regrouper les attributs et les opérations au sein d'une même entité (classe)*
- *Ce regroupant permet d'empêcher d'accéder directement aux données par un autre moyen que les services proposés (opérations)*
- *Ces services offerts aux utilisateurs définissent ce que l'on appelle l'interface de la classe.*



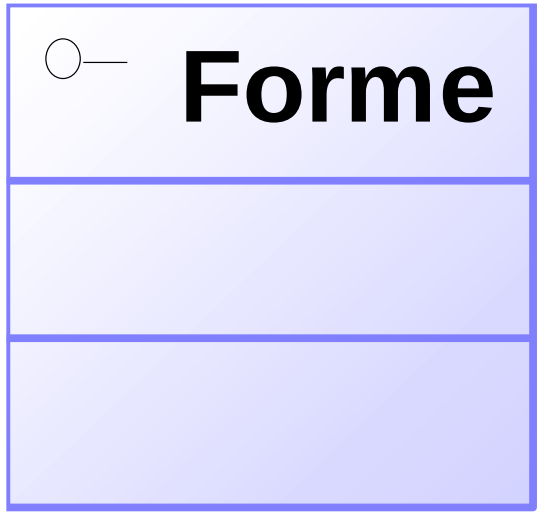
- *Partie statique, passive*
- *Partie cachée, privée*

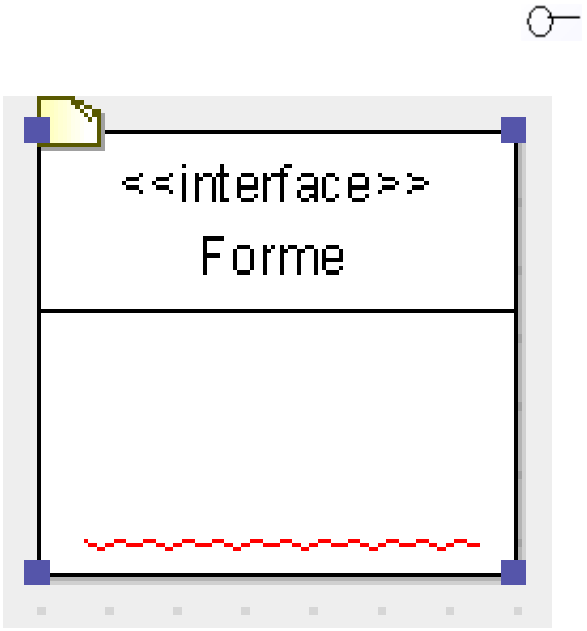
- *Partie dynamique, comportementale*
- *Partie visible, publique*
- *Interface avec l'extérieur*

- *Une méthode est dite abstraite si on connaît son entête, mais pas la manière dont elle peut être réalisée*
- *Une classe est dite abstraite lorsqu'elle définit au moins une méthode abstraite*



- *Une interface est une classe spéciale dont toutes les méthodes sont abstraites*
- *Une interface se note en UML avec le stéréotype <<interface>> ou symbole*



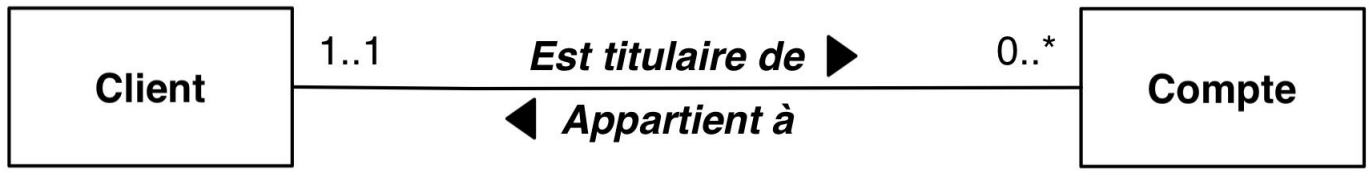


- *Relation existant entre une, deux ou plusieurs classes.*
- *Une association porte un nom (signification)*
- *Représentée par une ligne rectiligne*

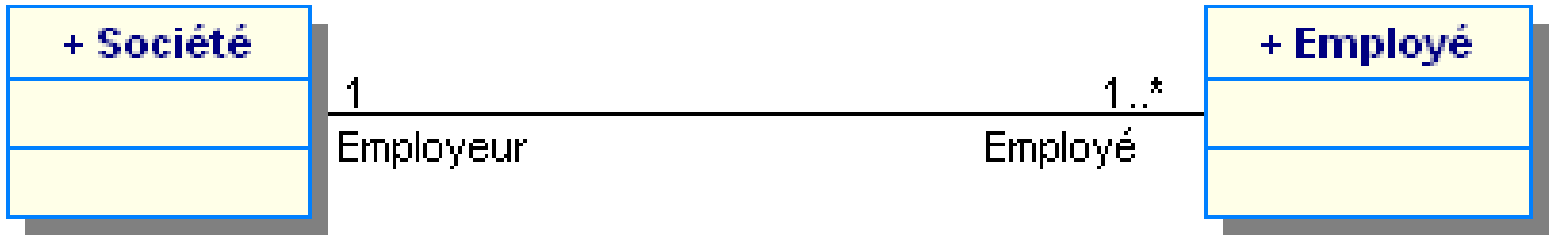
Remarques

- *une association fonctionne (généralement) dans les 2 sens (bidirectionnelle)*
- *termes associés : Nom, Sens de lecture, degré (arité), Multiplicité, Rôle, navigabilité et le qualificateur*

- *Décrit la nature (signification) de l'association*
- *Montre la direction de lecture de l'association*

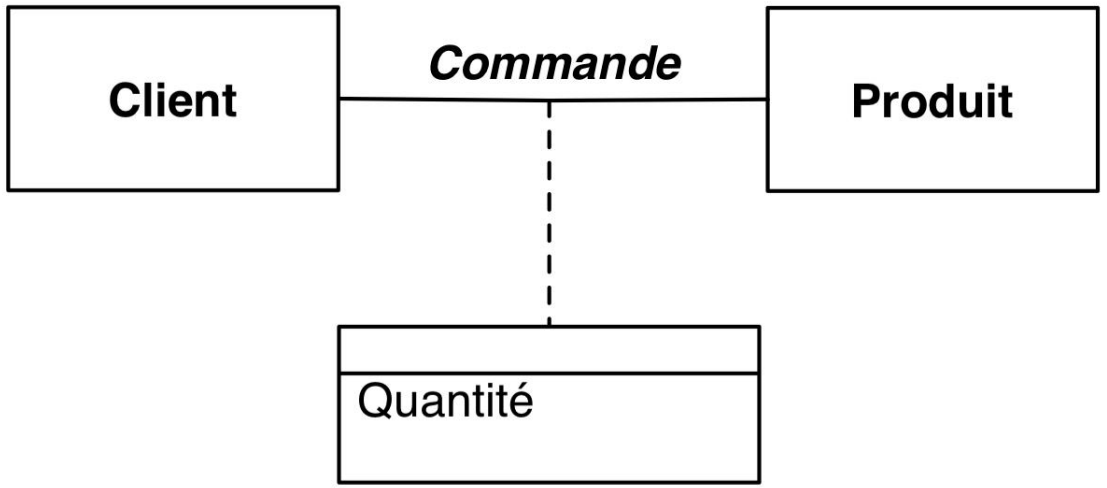


Décrit le rôle d'une classe dans une association

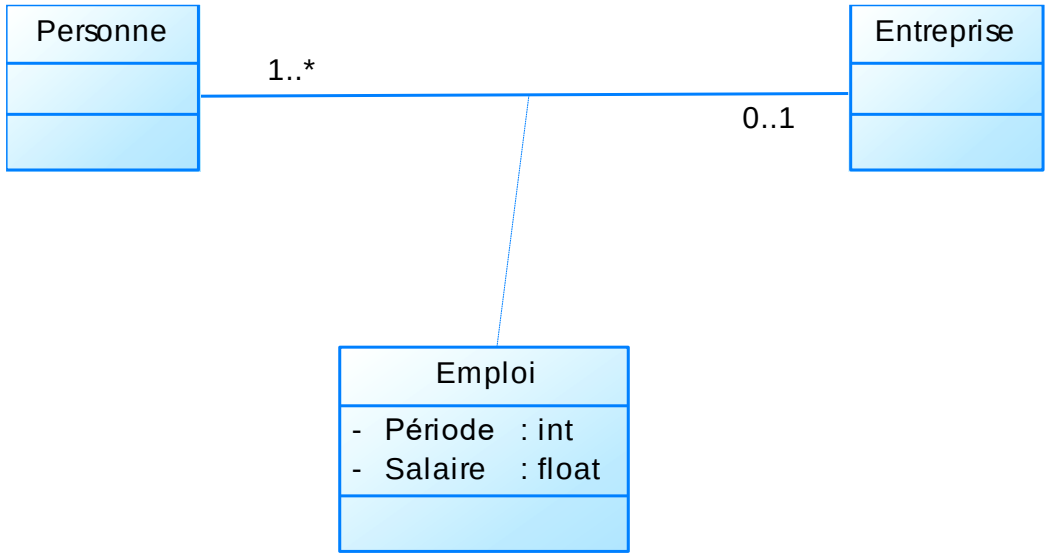


Associations
Classe association

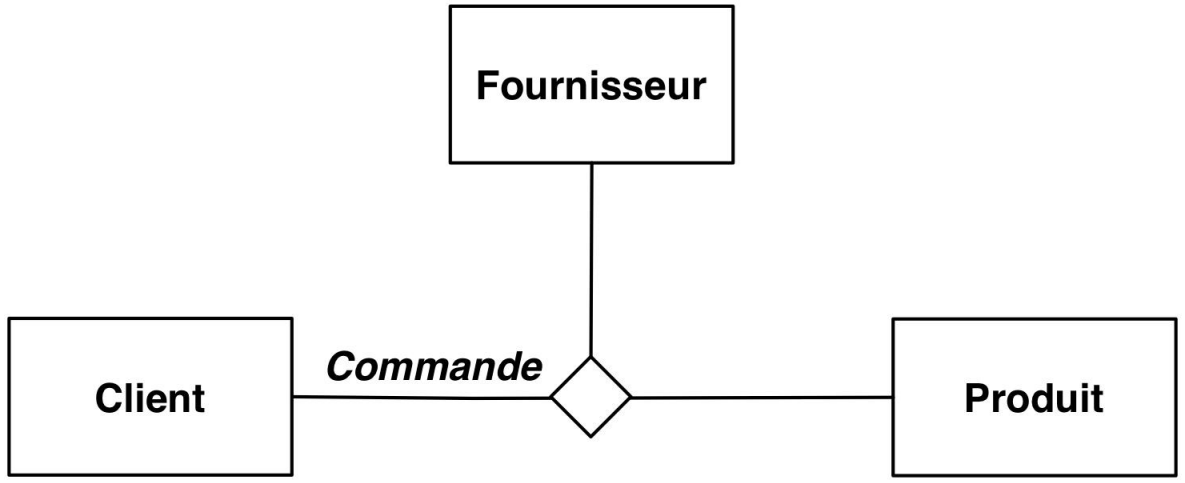
Une association peut avoir des attributs = ***classe-association***



- *Les classes association sont utiles quand il y a des attributs qui sont pertinents à l'association, mais à aucune des classes impliquées.*



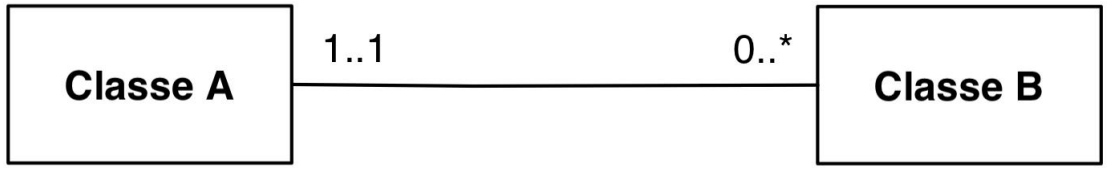
- ***degré d'une association*** *= nombre de classes participantes*
  - Association **unaire** : relie 2 instances d'une classe
  - Association **binaire** : relie 2 classes
  - Association **ternaire** : relie 3 classes
  - Association **n-aire** : relie n classes



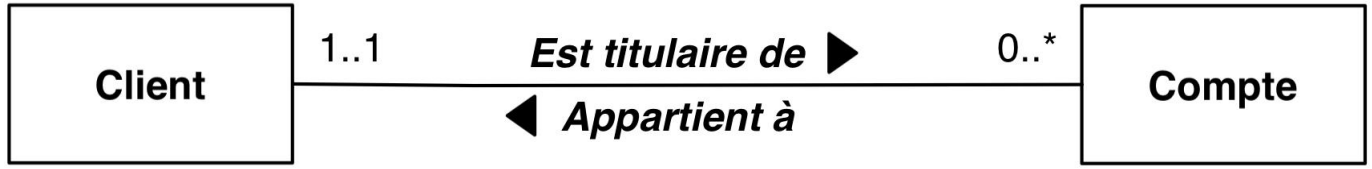
M**ultiplicité** = nombre de participations d'une classe dans une association

- indiquée **à chaque extrémité** d'une association
- sous la forme **min..max**
- min, max = **0, 1, ***

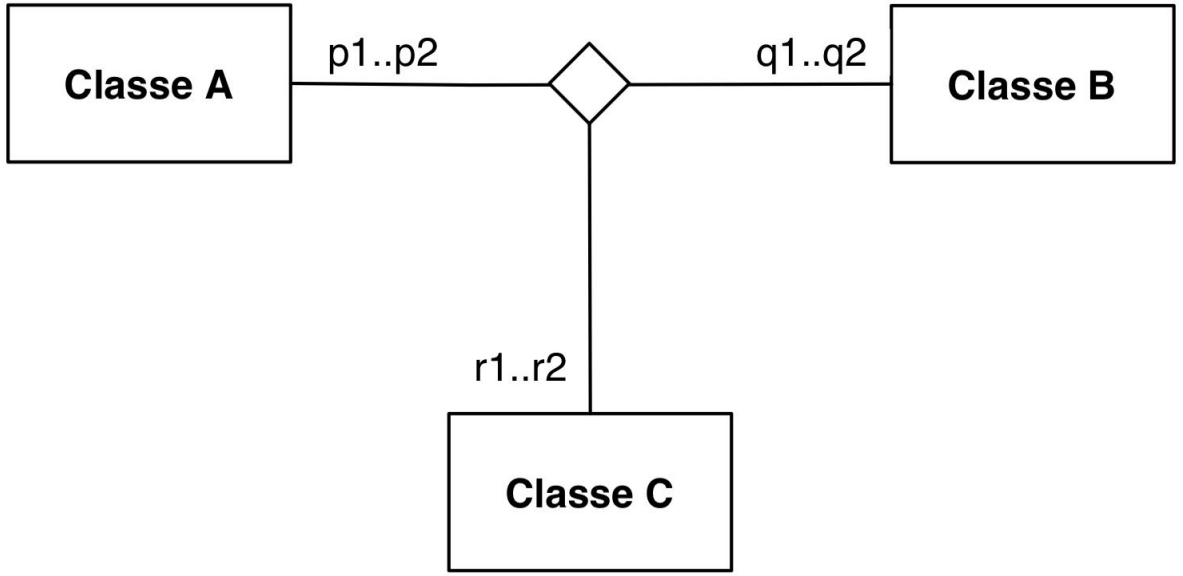
Exemple général



Exemple concret



Exemple ternaire



- Pour un couple d'instances de la classe A et de la classe B, il y a au min. **r1** instances de la classe C et au max. **r2** instances,
- …
- …

Notation abrégée des multiplicités **:**

1 ⇔ 1..1 (exactement 1)

***** ⇔ 0..* (0 ou plusieurs)

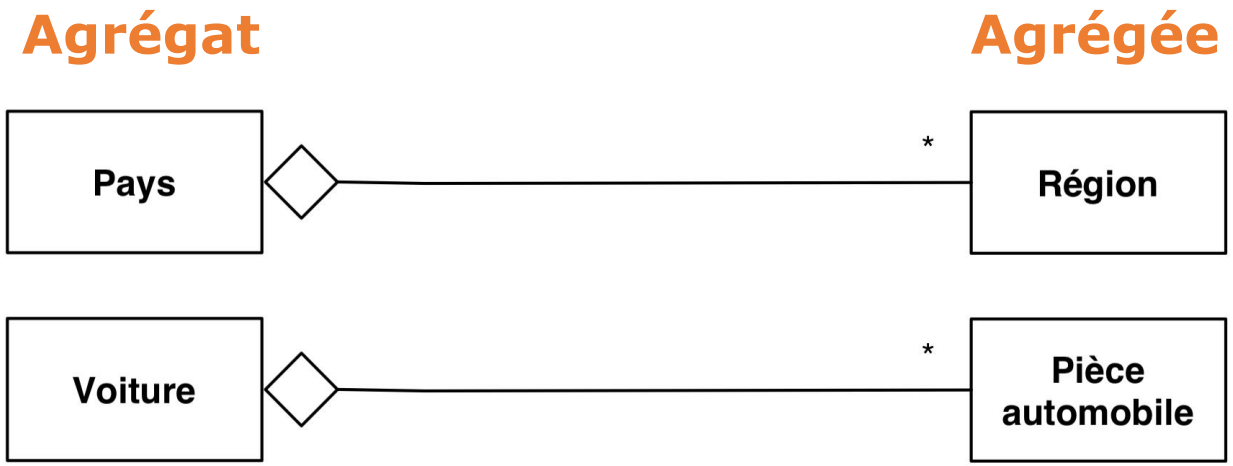
n ⇔ n .. n (exactement n)

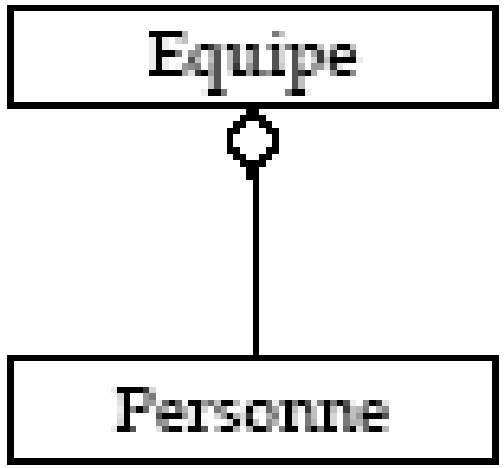
1..* ⇔ 1 ou plusieurs (1 ou plus)

0..1 ⇔ 0 ou 1 (au plus un)

1..100 ⇔ entre 1 et 100

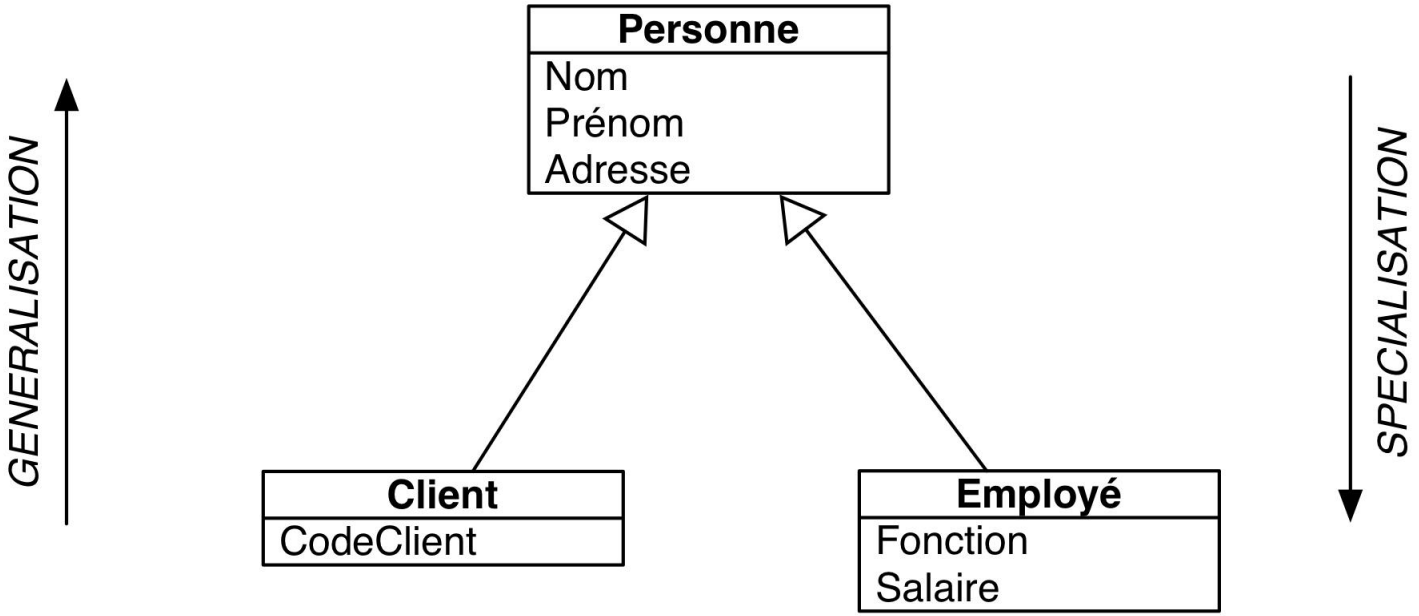
2,4,5 ⇔ 2, 4 ou 5


Agrégat
Agrégée
Pays
*
Région
Voiture
*
Pièce automobile


Equipe
Personne

- *La généralisation est la relation entre une classe et une ou plusieurs de ses versions raffinées.*
- *On appelle la classe dont on tire les précisions la super-classe et les autres classes les sous-classes.*
- *C'est une relation de type « est un (is a) » ou « est une sorte de ».*
- *La notation utilisée pour la généralisation est le triangle*

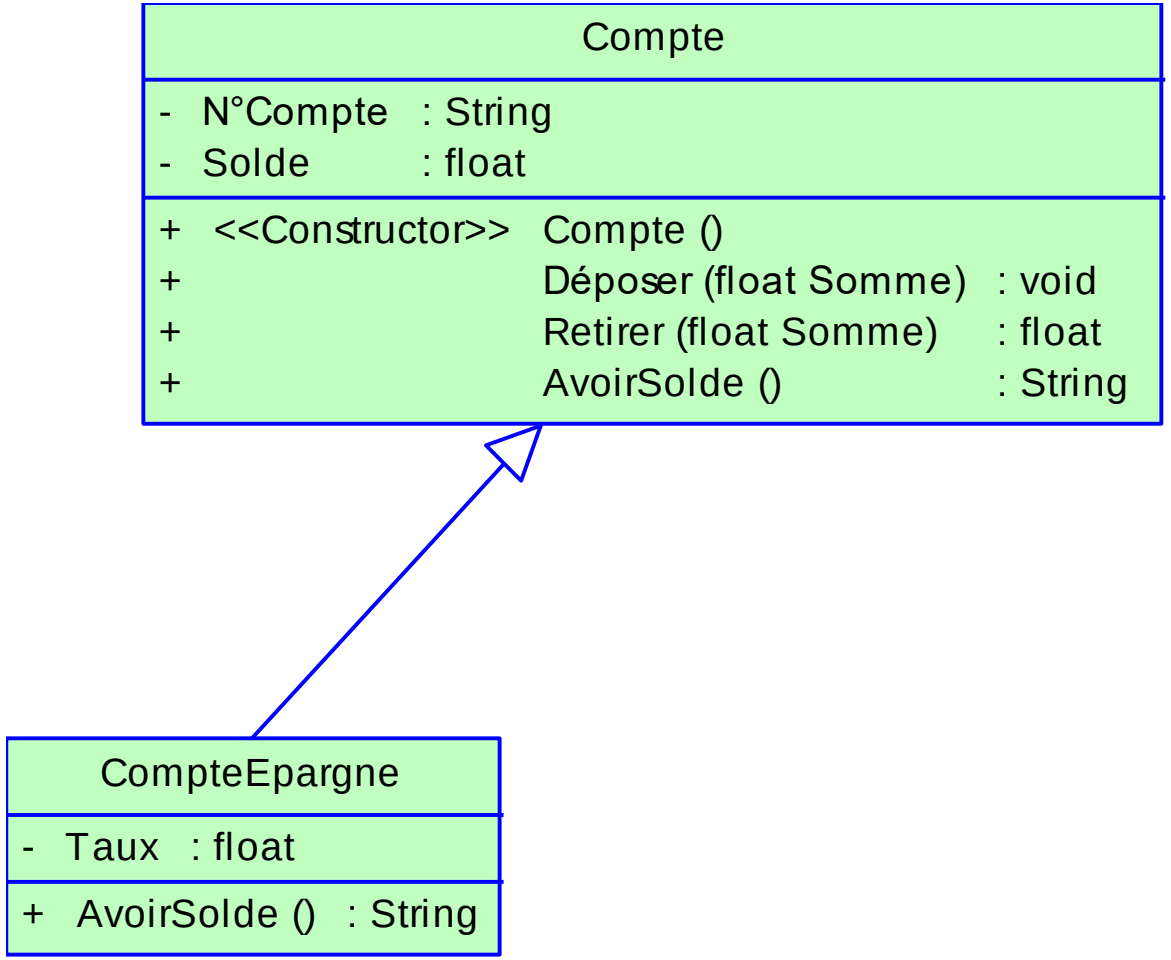
- **généraliser** = mettre en facteur des classes → « super-classe »
- **spécialiser** = décrire de nouveaux détails → « sous-classes »



- comparable à une association de type « est un, is a, kind of »
- une sous-classe **hérite** des attributs et opérations de sa super-classe (classe mère)

La classe spécialisée (sous-classe)

- *hérite les méthodes et les attributs de la classe générale (super-classe)*
- *peut ajouter ses propres attributs et méthodes.*
- *peut redéfinir le comportement d'une méthode.*


Compte
- N°Compte : String
- Solde : float
+ <<Constructor>> Compte ()
+ Déposer (float Somme) : void
+ Retirer (float Somme) : float
+ AvoirSolde () : String
CompteEpargne
- Taux : float
+ AvoirSolde () : String

Remarques

- *La généralisation et la spécialisation sont deux façons pour voir la même relation, top-down (spécialisation) ou bottom-up (généralisation).*
- *L'héritage est l'implémentation de la relation de la généralisation/spécialisation.*
- *Une classe peut hériter de plusieurs classes, on parle alors d'un héritage multiple.*

Spécialisation

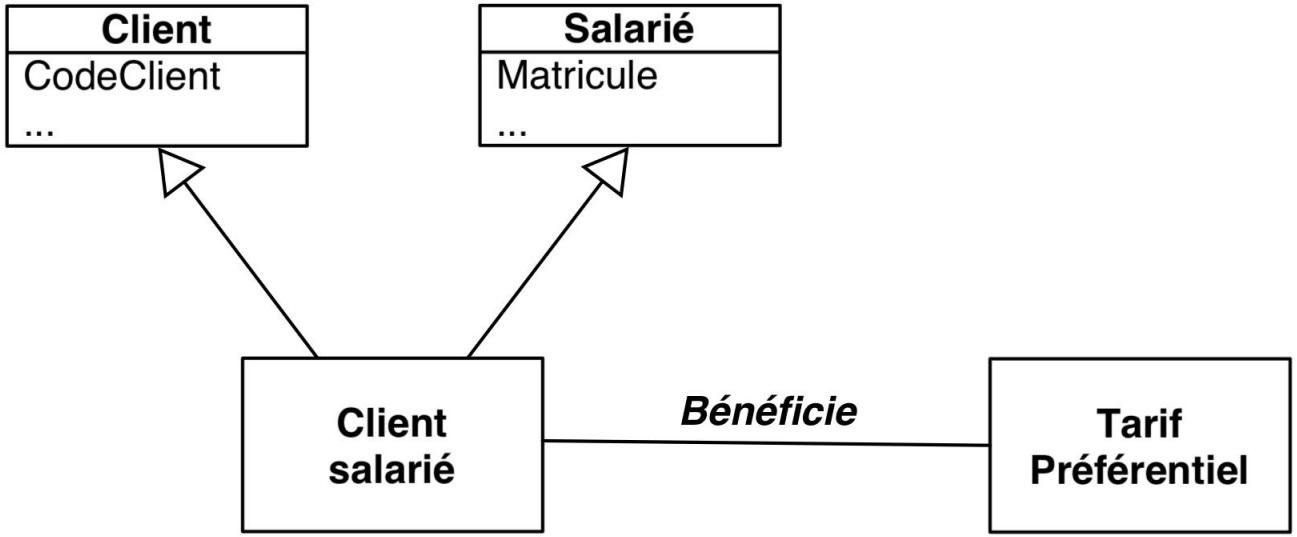
Sous classes
Classes filles
Classes dérivées

Super classe, classe mère

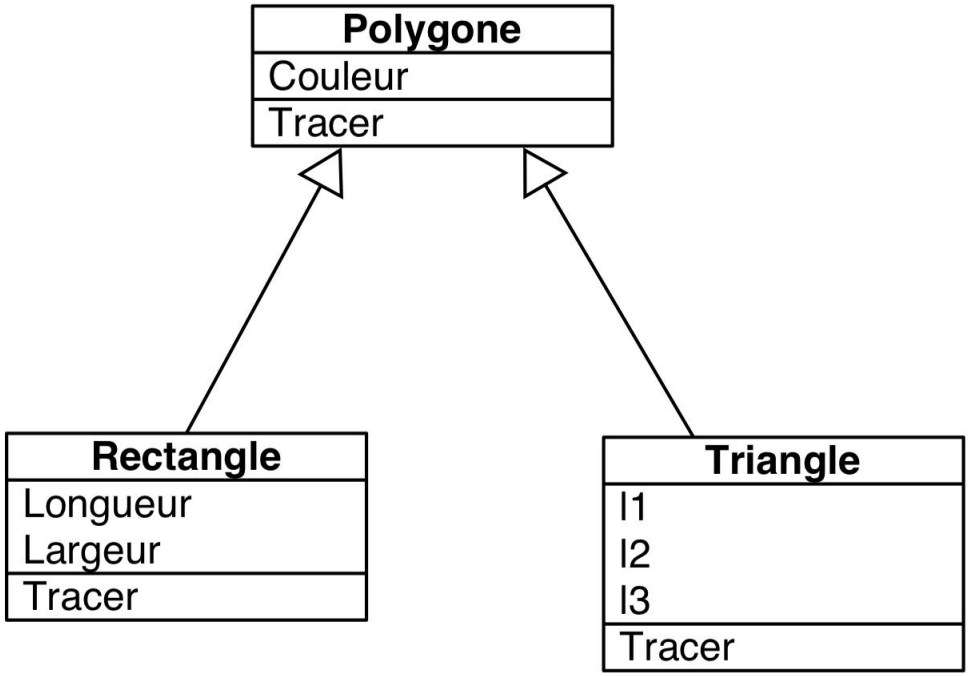
Généralisation

- une classe peut hériter de plusieurs super-classes

= **héritage multiple**



- **polymorphisme** = opérations de même nom, comportement spécifique



EXERCICE D'APPLICATION 1

Une personne est caractérisée par son nom, son prénom, son sexe et son âge. Les objets de classe Personne doivent pouvoir calculer leurs revenus et leurs charges. Les attributs de la classe sont privés ; le nom, le prénom ainsi que l'âge de la personne doivent être accessibles par des opérations publiques.

***Question : Donnez une représentation UML de la classe Personne, en remplissant tous les compartiments adéquats**.*

Deux types de revenus sont envisagés : d'une part le salaire et d'autre part toutes les autres sources de revenus. Les deux revenus sont représentés par des nombres réels (float). Pour calculer les charges globales, on applique un coefficient fixe de 20% sur les salaires et un coefficient de 15% sur les autres revenus.

***Question : Enrichissez la représentation précédente pour prendre en compte ces nouveaux éléments**.*

Un objet de la classe Personne peut être créé à partir du nom et de la date de naissance. Il est possible de changer le prénom d'une personne. Par ailleurs, le calcul des charges ne se fait pas de la même manière lorsque la personne décède.

Question : Enrichissez encore la représentation précédente pour prendre en compte ces nouveaux éléments

EXERCICE D'APPLICATION 2

Question : Pour chacun des énoncés suivants, donnez un diagramme des classes :

Tout écrivain a écrit au moins une œuvre

Les personnes peuvent être associées à des universités en tant qu'étudiants aussi bien qu'en tant que professeurs.

Un rectangle a deux sommets qui sont des points. On construit un rectangle à partir des coordonnées de deux points. Il est possible de calculer sa surface et son périmètre, ou encore de le translater.

Les cinémas sont composés de plusieurs salles. Les films sont projetés dans des salles. Les projections correspondantes ont lieu à chacune à une heure déterminée.1

Tous les jours, le facteur distribue des recommandés dans une zone géographique qui lui est affectée. Les habitants sont aussi associés à une zone géographique. Les recommandés sont de deux sortes : lettres ou colis. Comme plusieurs facteurs peuvent intervenir sur la même zone, on souhaite, pour chaque recommandé, le facteur qui l'a distribué, en plus du destinataire.

EXERCICE D’APPLICATION 3

Un hôtel est composé d'au moins deux chambres. Chaque chambre dispose d'une salle d'eau : douche ou
bien baignoire. Un hôtel héberge des personnes. Il peut employer du personnel et il est impérativement dirigé par un directeur. On ne connaît que le nom et le prénom des employés, des directeurs et des occupants. Certaines personnes sont des enfants et d'autres des adultes (faire travailler des enfants est interdit). Un hôtel a les caractéristiques suivantes : une adresse, un nombre de pièces et une catégorie. Une chambre est caractérisée par le nombre et de lits qu'elle contient, son prix et son numéro. On veut pouvoir savoir qui occupe quelle chambre à quelle date. Pour chaque jour de l'année, on veut pouvoir calculer le loyer de chaque chambre en fonction de son prix et de son occupation (le loyer est nul si la chambre est inoccupée). La somme de ces loyers permet de calculer le chiffre d’affaires de l'hôtel entre deux dates.

Question : Donnez une diagramme de classes pour modéliser le problème de l'hôtel.

EXERCICE D’APPLICATION 4

Considérons une station-service de distribution d'essence. Les clients se servent de l'essence et le pompiste remplit les cuves.
Question : Le client se sert de l'essence de la façon suivante : il prend un pistolet accroché à une pompe et appuie sur la gâchette pour prendre de l'essence. Qui est l'acteur du système ? Est-ce le client, le pistolet ou la gâchette ?
C'est le client. Un acteur est toujours extérieur au système. Définir les acteurs d'un système, c’est aussi en définir les bornes.

Question : Jojo, dont le métier est pompiste, peut se servir de l'essence pour sa voiture. Pour modéliser cette activité de Jojo, doit-on définir un nouvel acteur ? Comment modélise-t-on ça ?

EXERCICE D'APPLICATION 5

Dans un établissement scolaire, on désire gérer la réservation des salles de cours ainsi que
du matériel pédagogique (ordinateur portable ou/et Vidéo projecteur).

Seuls les enseignants sont habilités à effectuer des réservations (sous réserve de disponibilité de la salle ou du matériel).

Le planning des salles peut quant à lui être consulté par tout le monde (enseignants et étudiants).

Par contre, le récapitulatif horaire par enseignant (calculé à partir du planning des salles) ne
peut être consulté que par les enseignants.

Enfin, il existe pour chaque formation un enseignant responsable qui seul peut éditer le récapitulatif horaire pour l'ensemble de la formation.

EXERCICE D'APPLICATION 6

Pour faciliter sa gestion, un entrepôt de stockage envisage de s'informatiser. Le logiciel à produire doit allouer automatique un emplacement pour le chargement des camions qui convoient le stock à entreposer.

Le fonctionnent du système informatique doit être le suivant :

- *déchargement d'un camion : lors de l'arrivée d'un camion, un employé doit saisir dans le système les caractéristiques de chaque article ; le système produit alors une liste où figure un emplacement pour chaque article ;*
- *chargement d'un camion : les caractéristiques des articles à charger dans un camion sont saisies par un employé afin d'indiquer au système de libérer des emplacements.*

Le chargement et le déchargement sont réalisés manuellement.

Les employés de l'entrepôt sont sous la responsabilité d'un chef dont le rôle est de superviser la bonne application des consignes.

1. *Dégager les Acteurs*
2. *Donner le Diagramme des cas d'utilisation*

EXERCICE D’APPLICATION 7

Question : Les étudiants et les enseignants sont deux sortes de personnes. Proposez un modèle de classes correspondant.

Question : Un doctorant est un étudiant qui assure des enseignements. Complétez le modèle de classes précédent.

Question : Les doctorants et les étudiants doivent s'inscrire au début de l'année et éventuellement modifier leur inscription. On connaît le nom et le prénom de toutes les personnes. On doit pouvoir calculer le salaire des doctorants aussi bien que celui des enseignants. Ajoutez ces éléments au modèle précédent.

EXERCICE D'APPLICATION 8

Une académie souhaite gérer les cours dispensés dans plusieurs collèges. Pour cela, on dispose des renseignements suivants :

- *Chaque collège possède d'un site Internet*
- *Chaque collège est structuré en départements, qui regroupent chacun des enseignants spécifiques. Parmi ces enseignants, l'un d'eux est responsable du département.*
- *Un enseignant se définit par son nom, prénom, tél, mail, date de prise de fonction et son indice.*
- *Chaque enseignant ne dispense qu'une seule matière.*
- *Les étudiants suivent quant à eux plusieurs matières et reçoivent une note pour chacune d'elle.*
- *Pour chaque étudiant, on veut gérer son nom, prénom, tél, mail, ainsi que son année d'entrée au collège.*
- *Une matière peut être enseignée par plusieurs enseignants mais a toujours lieu dans la même salle de cours (chacune ayant un nombre de places déterminé).*
- *On désire pouvoir calculer la moyenne par matière ainsi que par département*
- *On veut également calculer la moyenne générale d'un élève et pouvoir afficher les matières dans lesquelles il n'a pas été noté*
- *Enfin, on doit pouvoir imprimer la fiche signalétique (, prénom, tél, mail) d'un enseignant ou d'un élève.*

Elaborez le diagramme de classes correspondant. Pour simplifier l'exercice, on limitera le diagramme à une seule année d'étude

EXERCICE D'APPLICATION 9

On souhaite gérer les réservations de vols effectués dans une agence.
D'après les interviews réalisées avec les membres de l'agence, on sait que :

- *Les compagnies aériennes proposent différents vols*
- *Un vol est ouvert à la réservation et refermé sur ordre de la compagnie*
- *Un client peut réserver un ou plusieurs vols, pour des passagers différents*
- *Une réservation concerne un seul vol et un seul passager*
- *Une réservation peut être confirmée ou annulée*
- *Un vol a un aéroport de départ et un aéroport d'arrivée*
- *Un vol a un jour et une heure de départ, et un jour et une heure d'arrivée*
- *Un vol peut comporter des escales dans un ou plusieurs aéroport(s)*
- *Une escale a une heure de départ et une heure d'arrivée*
- *Chaque aéroport dessert une ou plusieurs villes*

A partir des éléments qui vous sont fournis ci-dessus, élaborez le diagramme de classes

EXERCICE D'APPLICATION 10

Le déroulement normal d'utilisation d'un distributeur automatique de billets est le suivant :

- *le client introduit sa carte bancaire*
- *la machine vérifie alors la validité de la carte et demande le code au client*
- *si le code est correct, elle envoie une demande d'autorisation de prélèvement au groupement de banques. Ce dernier renvoie le solde autorisé à prélever.*
- *le distributeur propose alors plusieurs montants à prélever*
- *le client saisit le montant à retirer*
- *après contrôle du montant par rapport au solde autorisé, le distributeur demande au client s'il désire un ticket*
- *Après la réponse du client, la carte est éjectée et récupérée par le client*
- *les billets sont alors délivrés (ainsi que le ticket)*
- *le client récupère enfin les billets et son ticket*

Modéliser cette situation à l'aide d'un diagramme de séquence en ne prenant en compte que le cas où tout se passe bien. NB : on identifiera les scénarios qui peuvent poser problème en incluant des commentaires dans le diagramme

Scala avec les classes