- **Semestre 2** R2.01 Développement orienté objets : UML
- **Semestre 3** R3.03 Analyse : UML

CM N°1	Système et Modèle	UML	Cas d'utilisation	
CM N°2	**Déploiement**	**Classes**		
CM N°3	Dépendances	Associations		
CM N°4	Code et dépendances	Code et associations		
CM N°5	Héritage	Stéréotypes	Abstraites	Interfaces
CM N°6	Rappels: Cas d'utilisation, classes, associations			
CM N°7	Rappels : Héritage, abstraites, interfaces			
CM N°8	Objets	Communication	Séquences	
CM N°9	Fragments	Scénarios	Séquences et code	
CM N°10	États-Transitions	Activités	Paquets	

R2.01 Développement orienté objets

Cours N°2 : UML : Les diagrammes de déploiement et les classes

Sommaire

E.Porcq : R2.01 UML

Département : IUT Caen Informatique

Année universitaire : 2024-2025

Sources bibliographiques :

-Cours "M2104 Approche qualité" de P.Brutus

- Cours de Laurent AUDIBERT

1) Les différents diagrammes

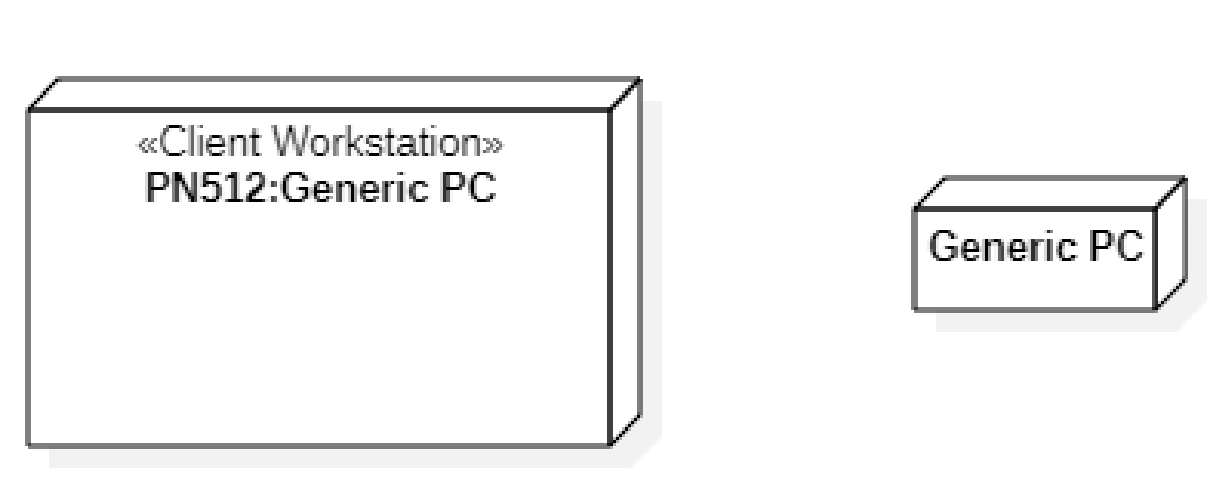
- Diagrammes de structure
  - Diagramme de déploiement (à voir lors de ce cours)
  - Diagramme de composant (à voir lors de ce cours)
  - Diagramme de classes (à voir lors de ce cours)
  - Diagramme d'objets (à voir)
  - Diagramme de paquetages (à voir)
  - Diagramme de structure composite
- Diagrammes de comportement
  - Diagramme de cas d'utilisation (vu)
  - Diagramme d'activités (à voir en partie en TD PL/SQL)
  - Diagramme états-transitions (à voir)
  - Diagramme d'interaction
    - Diagramme de séquences (à voir)
    - Diagramme global d'interaction
    - Diagramme de communication (à voir)
    - Diagramme de temps

2-1) Présentation du diagramme de déploiement

- Rappel : UML propose différents points de vue sur le S.I.
  - des diagrammes de structure : aspects liés à l'architecture
    - Diagramme de **déploiement et de composants**
  - des diagrammes de comportement : pour quoi faire ?
    - Diagramme de cas d'utilisation (vu)
- Le diagramme de déploiement
  - Montre l'affectation de composants logiciels à des composants physiques
  - Présente le déploiement des éléments sur l'architecture physique ainsi que la communication entre des composants
  - Possède des ressources matérialisées par des nœuds,
  - Précise comment les composants sont répartis sur les nœuds et quelles sont les connexions entre les composants

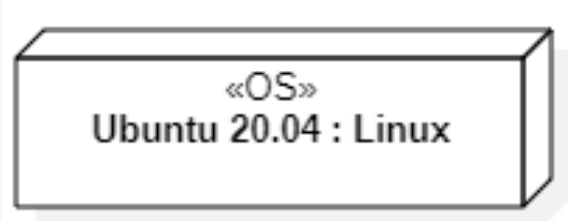
2) Les nœuds

- Chaque ressource matérielle est représentée par un nœud La représentation d'un nœud est un cubes en 3D avec le nom à l'intérieur. Il peut être plus intéressant de remplacer ces cubes par des représentations plus parlantes des matériels.
- On peut ajouter des propriétés au nœud (pour indiquer le type de processeur ou le système d'exploitation). Un stéréotype informe de la propriété.
- La encore, on peut montrer des classes ou des instances de nœud (même façon de le montrer ...)

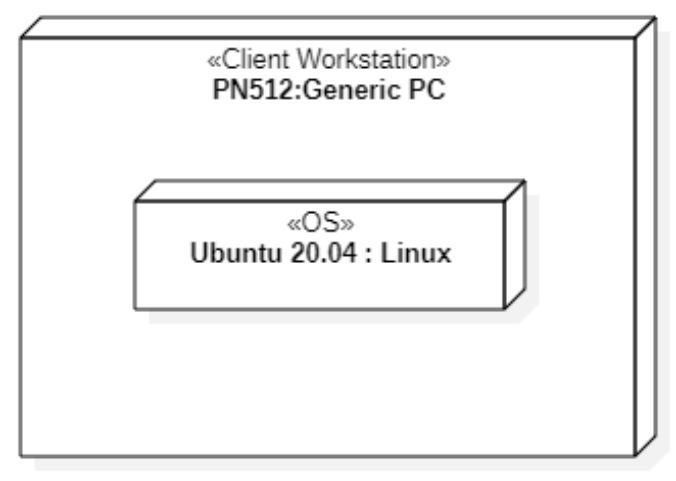


2) Les nœuds

- Un nœud peut aussi être un environnement d'exécution, une ressource logicielle ((système d'exploitation, machine virtuelle, SGBD, navigateur web, serveur http, serveur de messagerie…)

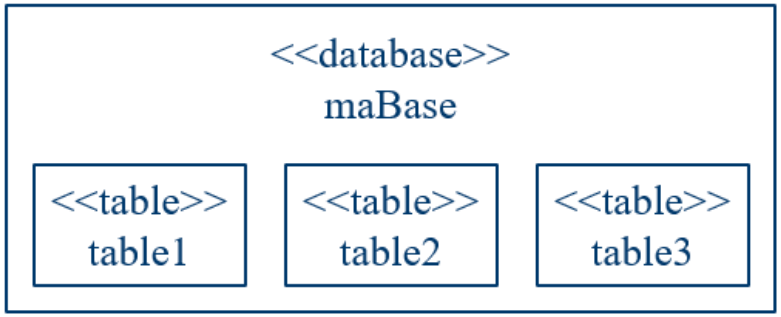


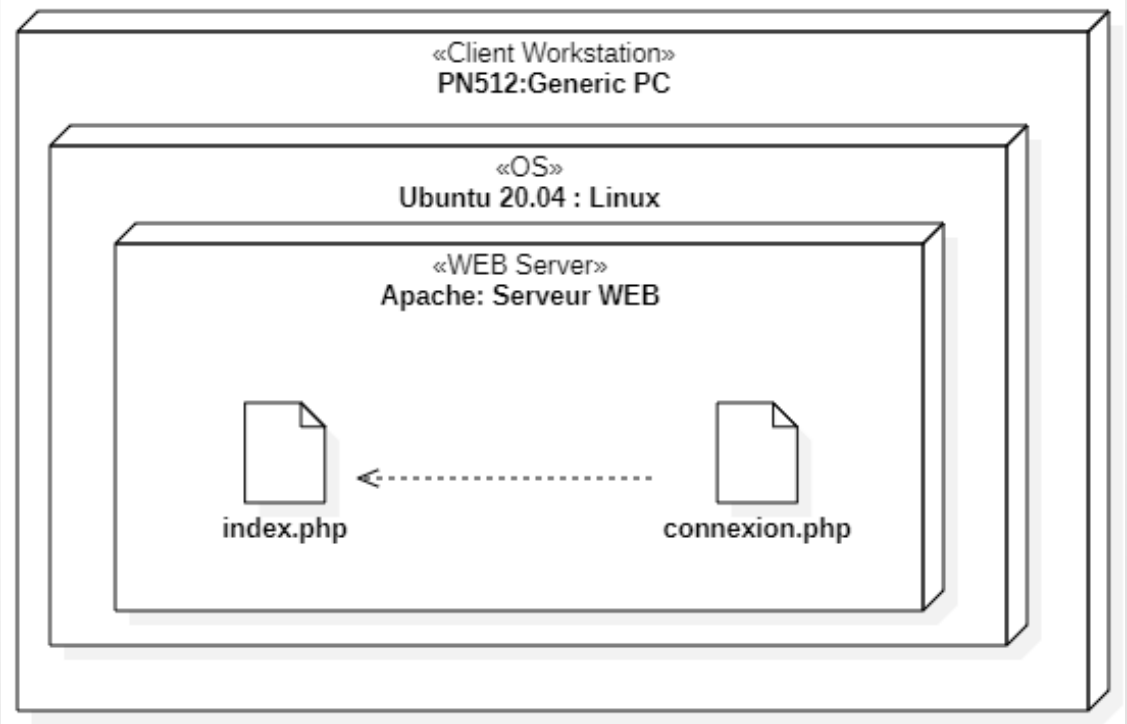
- On peut donc représenter un nœud à l'intérieur d'un autre



3) Les artefacts

- Un artefact correspond à un élément concret existant dans le monde réel (document, exécutable, fichier, tables de bases de données, script…). Il se représente comme un classeur par un rectangle contenant le mot-clef « artifact » suivi du nom de l'artefact. On peut préférer un stéréotype plus précis

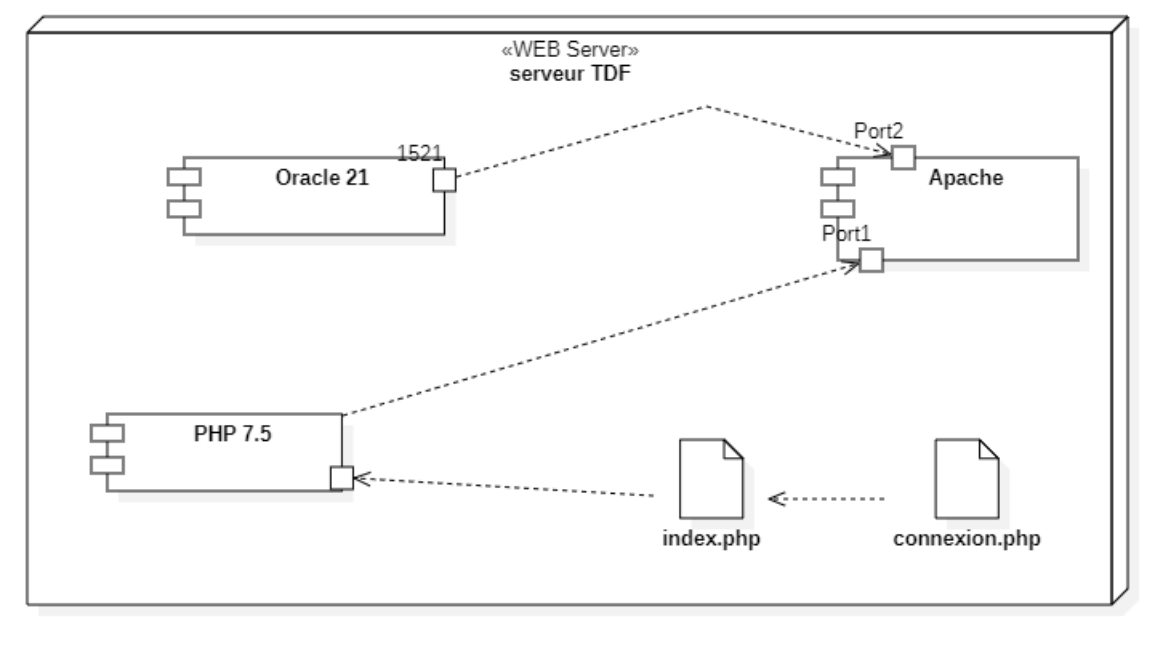




R2.01 Développement orienté objets
Cours N°2 : UML : Les diagrammes de déploiement
Les artefacts

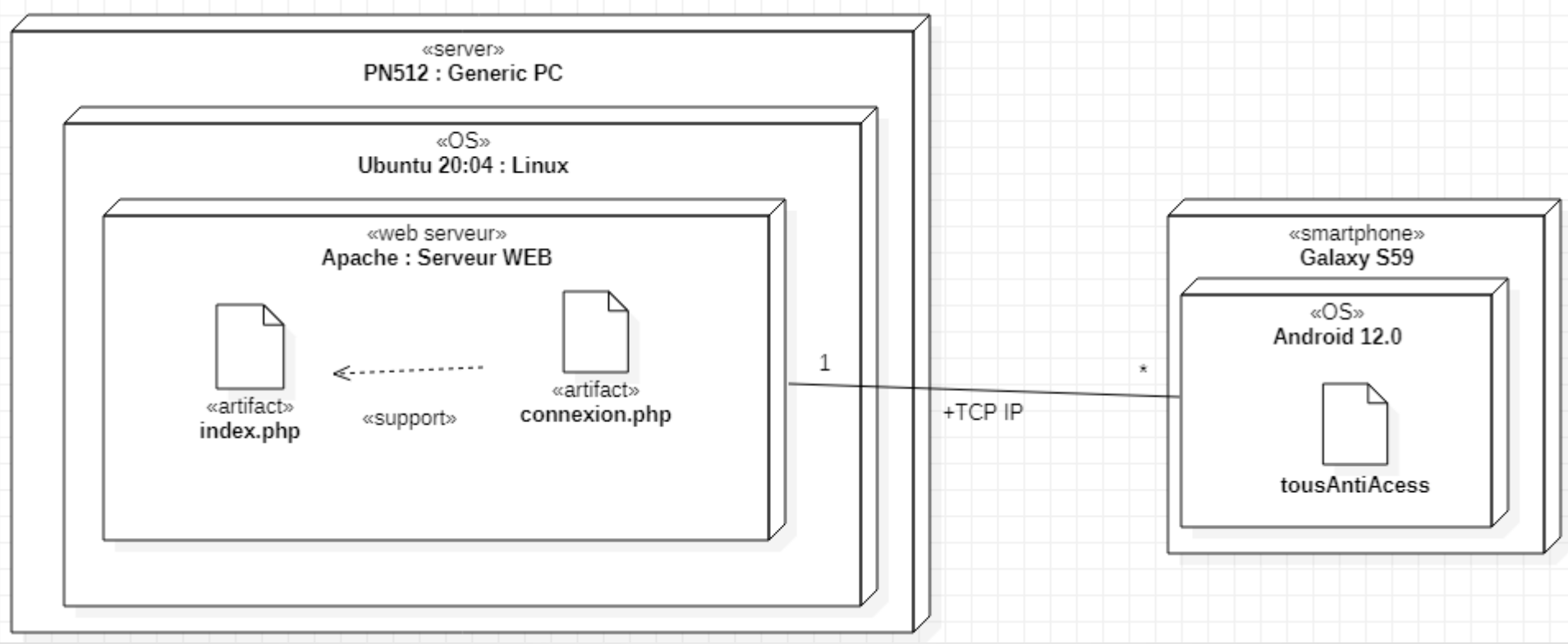
4) Les composants

- Ils représentent un élément logiciel indépendant du noeud auquel il est associé.
- Un composant est une unité autonome
- Un composant doit fournir un service bien précis
- Un port est un point de connexion entre un composant et son environnement.



4) Connexion entre les nœuds

- Les relations entre les nœuds désignent le type de support communication. Ce dernier peut désigner une couche du modèle OSI.
- Les informations de multiplicités peuvent être présentes dans les diagrammes de classes de nœud



4) Connexion entre les noeuds

- Entre nœuds physiques, le chemin de connexion est étiqueté par les caractéristiques de la ligne de communication :
  - paire torsadée, fibre optique
  - 3G, 4G, WiFi
  - ...
- Entre nœuds logiques (environnement d'exécution ou artefact), le chemin de communication est étiqueté par le protocole, le langage utilisé pour les échanges :
  - HTTP
  - SQL
  - ...
- Il n'est pas interdit d'être exhaustif
  - WIFI/TCP IP/IMAP

5) Stéréotypes des diagrammes de déploiement

<<device>>	<<équipement>>
<<client workstation>>	<<poste client>>
<<digital pad>>	<<tablette>>
<<smartphone>>	<<ordiphone>>
<<server>>	<<serveur>>
<<industrial station>>	<<ordinateur industriel>>
<<execution environnement>>	<<env. D'exécution>>
<<OS>>	<<SE>>
<<web server>>	<<serveur web>>
<<web browser>>	<<navigateur>>
<<virtual machine>>	<<machine virtuelle>>
<<DBMS>>	<<SGBD>>
<<artifact>>	<<artefact>>
<<database>>	<<BdD>>
<<table>>	<<table>>
<<folder>>	<<dossier>>
<<file>>	<<fichier>>

6) Les diagrammes de classe

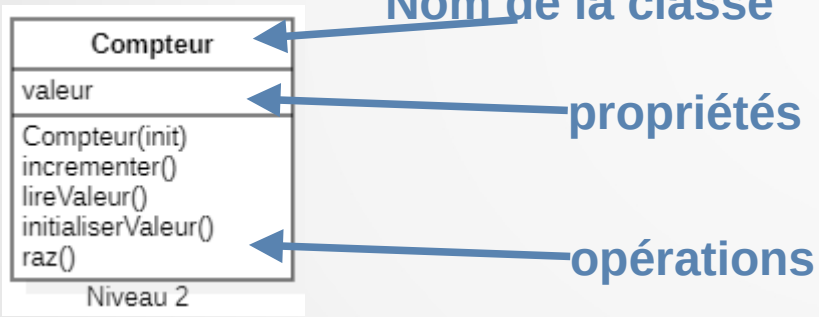
- Rappel : UML propose différents points de vue sur le S.I.
  - des diagrammes de structure : aspects liés à l'architecture
    - Diagramme de déploiement (vu)
    - **Diagramme de classe**
  - des diagrammes de comportement : pour quoi faire ?
    - Diagramme de cas d'utilisation (vu)
- Le diagramme de classe
  - est considéré comme le plus important de la modélisation orientée objet, il est le seul obligatoire lors d'une telle modélisation.
  - montre la structure interne. Il permet de fournir une représentation abstraite des objets du système qui vont interagir ensemble pour réaliser les cas d'utilisation.
  - n'est pas adapté (sauf cas particulier) pour détailler, décomposer, ou illustrer la réalisation d'un cas d'utilisation particulier.
  - est une vue statique car on ne tient pas compte du facteur temporel dans le comportement du système.

7) Les classes

- Une classe est une représentation **abstraite** d'un objet. Elle donne le prototype (type) d'une entité complexe. Elle n'a donc pas d'existence réelle.
- Pour utiliser un élément typé (comme Individu) dans un programme, on crée une **instance** d'une classe qui est une variable complexe, image d'une classe, mais ayant une réalité au niveau du programme informatique. **L'objet** ainsi créé va prendre de la place en mémoire et, par son comportement, va modifier ses données membres (**attributs**) ou communiquer avec d'autres objets.
- En en langage Java, on **instancie** une classe (on crée un **objet**) en exécutant

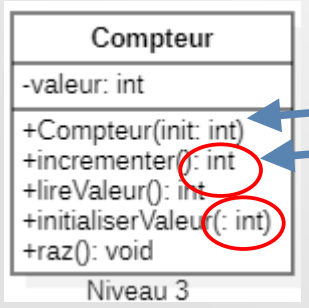
```
NomDeLaClasse nomDeLobjet;
nomDeLobjet = new NomDeLaClasse();
```

8) Représentation d'une classe



Menu starUML pour niveaux

Compteur monCompteur ;
monCompteur = new Compteur(25) ;



Paramètre d'entrée

Paramètre de retour

(:int) signifie que le nom du paramètre n'est pas indiqué

9) Les propriétés

- A chaque instant, un objet est dans un **état** donné.
- Cet état est conditionné par la valeur de tous les **attributs** ou propriétés de l'objet.
- Chaque attribut peut prendre une valeur dans un **domaine** de définition donné.
- Rq : il arrive parfois que l'on considère un objet (informatique) comme suffisamment primitif pour ne pas avoir d'état.
- Une propriété possède un **nom**, un **type** (qui peut être une classe) et un niveau de **visibilité**
- Il existe quatre visibilités prédéfinies.
  - **Public** ou + : tout élément qui peut voir le conteneur peut également voir l'élément indiqué.
  - **Protected** ou # : seul un élément situé dans le conteneur ou un de ses descendants peut voir l'élément indiqué.
  - **Private** ou - : seul un élément situé dans le conteneur peut voir l'élément.
  - **Package** ou ~ : seul un élément déclaré dans le même paquetage peut voir l'élément.

9) Les propriétés

- Le nom de l'attribut doit être unique dans la classe.
- Le type de l'attribut peut être un nom de classe, un nom d'interface ou un type de donné prédéfini. La multiplicité d'un attribut précise le nombre de valeurs que l'attribut peut contenir. Lorsqu'une multiplicité supérieure à 1 est précisée, il est possible d'ajouter une contrainte ({<contrainte>}) pour préciser si les valeurs sont ordonnées ({ordered}) ou pas ({list}).
- Par défaut, chaque instance d'une classe possède sa propre copie des attributs de la classe. Les valeurs des attributs peuvent donc différer d'un objet à un autre. Cependant, il est parfois nécessaire de définir un attribut de classe (static en Java ou en C++) qui garde une valeur unique et partagée par toutes les instances de la classe. L'accès à cet attribut ne nécessite pas l'existence d'une instance.
- Graphiquement, un attribut de classe est souligné.

10) Les opérations

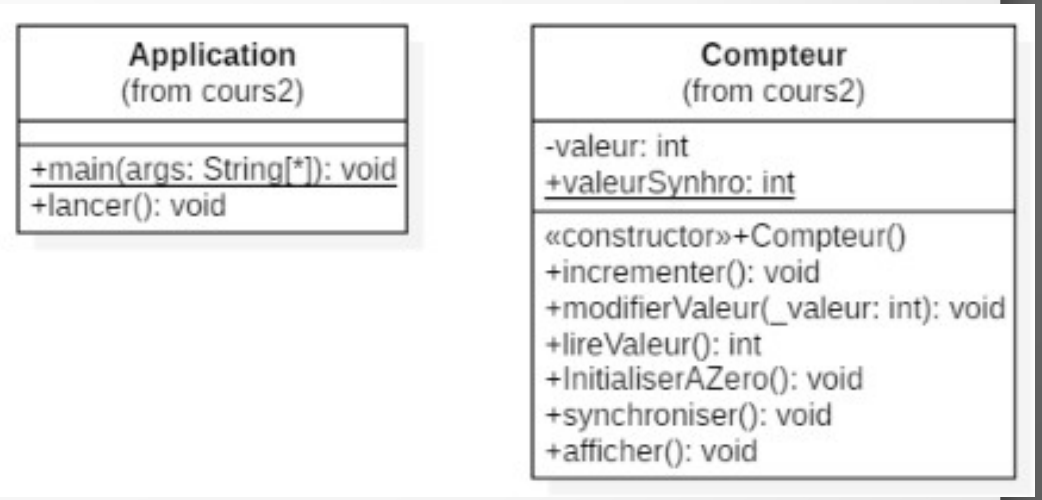
- Un objet possède un certain nombres de **comportements**
- Un comportement regroupe toutes les **compétences** d'un objet et décrit les actions et les réactions de cet objet. Chaque atome de comportement est appelé **opération** ou **méthode**.
- Les opérations sont déclenchées suite à une stimulation, représentée sous la forme d'un message envoyé par un autre objet ou par lui-même.
- Rq : il arrive parfois que l'on considère un objet (informatique) comme suffisamment basique pour ne pas avoir de comportement.
- Dans une classe, une opération (même nom et même types de paramètres) doit être **unique**. Quand le nom d'une opération apparaît plusieurs fois avec des **paramètres d'entrée** différents, on dit que l'opération est **surchargée**.
- En revanche, il est **impossible** que deux opérations ne se distinguent que par leur valeur retournée.

10) Les opérations

- Quand on passe un paramètre à une méthode, on peut préciser la direction
  - in : Paramètre d'entrée passé par valeur. Les modifications du paramètre ne sont pas disponibles. pour l'appelant. C'est le comportement par défaut.
  - out : Paramètre de sortie uniquement. Il n'y a pas de valeur d'entrée et la valeur finale est disponible pour l'appelant.
  - inout : Paramètre d'entrée/sortie. La valeur finale est disponible pour l'appelant
- Le type du paramètre peut être un nom de classe, un nom d'interface ou un type de donné prédéfini.
- Comme pour les attributs de classe, il est possible de déclarer des méthodes de classe. Une méthode de classe ne peut manipuler que des attributs de classe et ses propres paramètres. Cette méthode n'a pas accès aux attributs de la classe. L'accès à une méthode de classe ne nécessite pas l'existence d'une instance de cette classe.
- Graphiquement, une méthode de classe est soulignée.

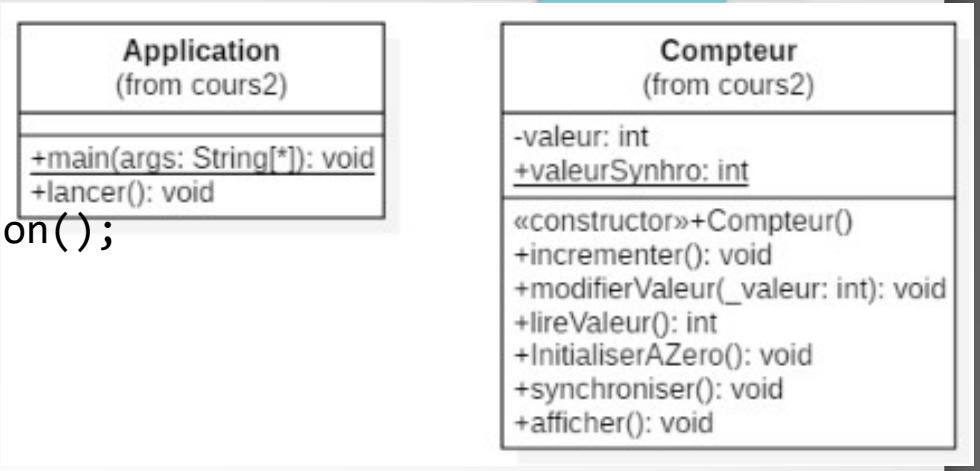
11) Exemple

```
public class Compteur {
     private int valeur;
     public static int valeurSynhro;
     public Compteur() {
          valeur = 0;
     }
     public void incrementer()   {
          valeur++;
     }
     public void modifierValeur(int _valeur){
          valeur = _valeur ;
     }
     public int lireValeur()     {
          return valeur;
     }
     public void InitialiserAZero()  {
          valeur = 0;
     }
     public void synchroniser()  {
          valeur = valeurSynhro;
     }
     public void afficher()
     {
          System.out.println("valeur       : "+valeur);
          System.out.println("valeurSynhro : "+valeurSynhro);
     }
}
```



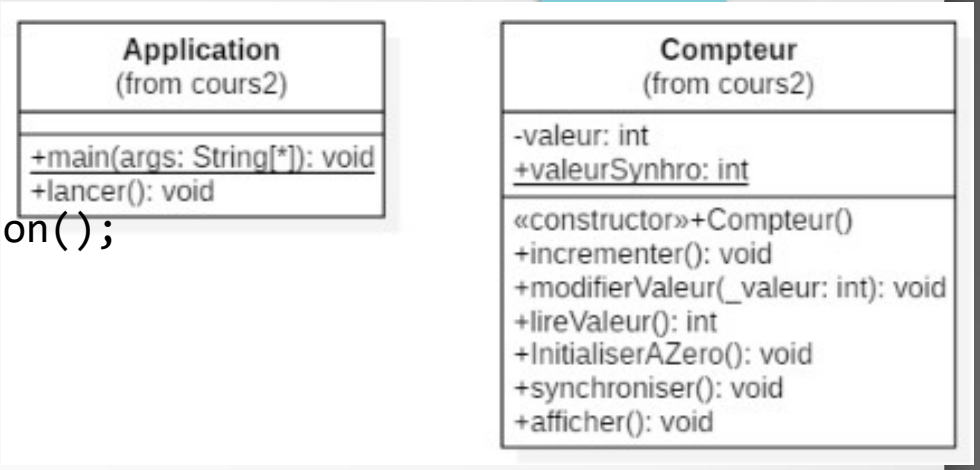
11) Exemple

```
public class Application {
    public static void main(String[] args) {
        Application monAppli = new Application();
        monAppli.lancer();
    }
    public void lancer()
    {
        Compteur cp1 = new Compteur();
        Compteur cp2 = new Compteur();

        cp1.incrementer();
        cp1.incrementer();
        cp2.incrementer();
        cp1.valeurSynhro = 15;
        cp1.afficher();
        cp2.afficher();
        Compteur.valeurSynhro = 10;
        cp1.afficher();
        cp2.afficher();
        cp1.synchroniser();
        cp1.afficher();
        cp2.afficher();
    }
}
```



11) Exemple

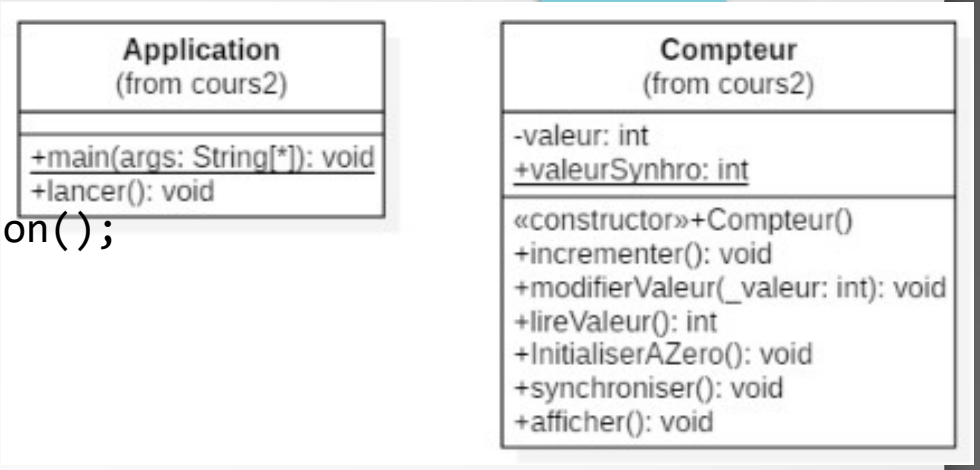
```
public class Application {
    public static void main(String[] args) {
        Application monAppli = new Application();
        monAppli.lancer();
    }
    public void lancer()
    {
        Compteur cp1 = new Compteur();
        Compteur cp2 = new Compteur();

        cp1.incrementer();
        cp1.incrementer();
        cp2.incrementer();
        cp1.valeurSynhro = 15;
        cp1.afficher();
        cp2.afficher();
        Compteur.valeurSynhro = 10;
        cp1.afficher();
        cp2.afficher();
        cp1.synchroniser();
        cp1.afficher();
        cp2.afficher();
    }
}
```




```
valeur          : 2
valeurSynhro    : 15
valeur          : 1
valeurSynhro    : 15
```

11) Exemple

```
public class Application {
    public static void main(String[] args) {
        Application monAppli = new Application();
        monAppli.lancer();
    }
    public void lancer()
    {
        Compteur cp1 = new Compteur();
        Compteur cp2 = new Compteur();

        cp1.incrementer();
        cp1.incrementer();
        cp2.incrementer();
        cp1.valeurSynhro = 15;
        cp1.afficher();
        cp2.afficher();
        Compteur.valeurSynhro = 10;
        cp1.afficher();
        cp2.afficher();
        cp1.synchroniser();
        cp1.afficher();
        cp2.afficher();
    }
}
```




```
valeur          : 2
valeurSynhro    : 10
valeur          : 1
valeurSynhro    : 10
```

valeur : 10
valeurSynhro : 10
valeur : 1
valeurSynhro : 10

11) Exemple

```
public class Application {
    public static void main(String[] args) {
        Application monAppli = new Application();
        monAppli.lancer();
    }
    public void lancer()
    {
        Compteur cp1 = new Compteur();
        Compteur cp2 = new Compteur();

        cp1.incrementer();
        cp1.incrementer();
        cp2.incrementer();
        cp1.valeurSynhro = 15;
        cp1.afficher();
        cp2.afficher();
        Compteur.valeurSynhro = 10;
        cp1.afficher();
        cp2.afficher();
        cp1.synchroniser();
        cp1.afficher();
        cp2.afficher();
    }
}
```