Modélisation UML

Diagramme de classe en analyse
Diagramme de classe en conception
Dérivation conception vers code (java ou PHP)

Plan du cours

- Diagramme de classe en analyse
- Diagramme de classe en conception
- Dérivation diagramme de conception vers code

Diagramme de classe en analyse

- Utilisé essentiellement pour la modélisation conceptuelle des données

- Un des 3 schémas de modélisation pour l'EDC, avec Merise et modèle relationnel

Plan du cours

- Diagramme de classe en analyse
- Diagramme de classe en conception
- Dérivation diagramme de conception vers code

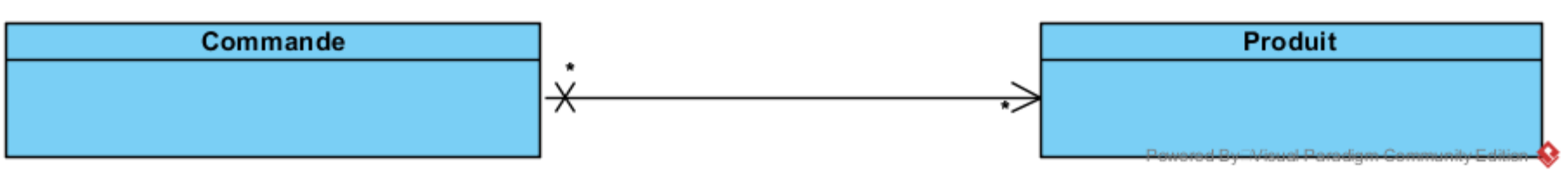
Contenu diagramme de conception

- Attributs avec ***type*** et ***portée***
- Operations (méthodes)
- **Navigabilité** d'une association
- Prépare et documente la partie ***applicative*** (code)

Navigabilité d'une association

- Jusqu'à présent, nous avons considéré des associations navigable
- La navigabilité indique s'il est possible de traverser une association
- 4 cas sont possibles. En considérant 2 objets, A et B
  - ✓ Navigation possible dans les 2 sens
  - ✓ Navigation possible uniquement de A vers B: A connaît B, mais B ne connaît pas A
  - ✓ Navigation possible uniquement de B vers A
  - ✓ Lien non navigable

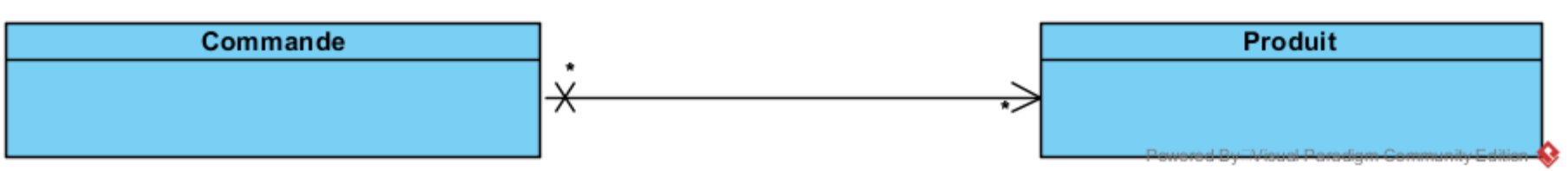
Exemple: commande et produit



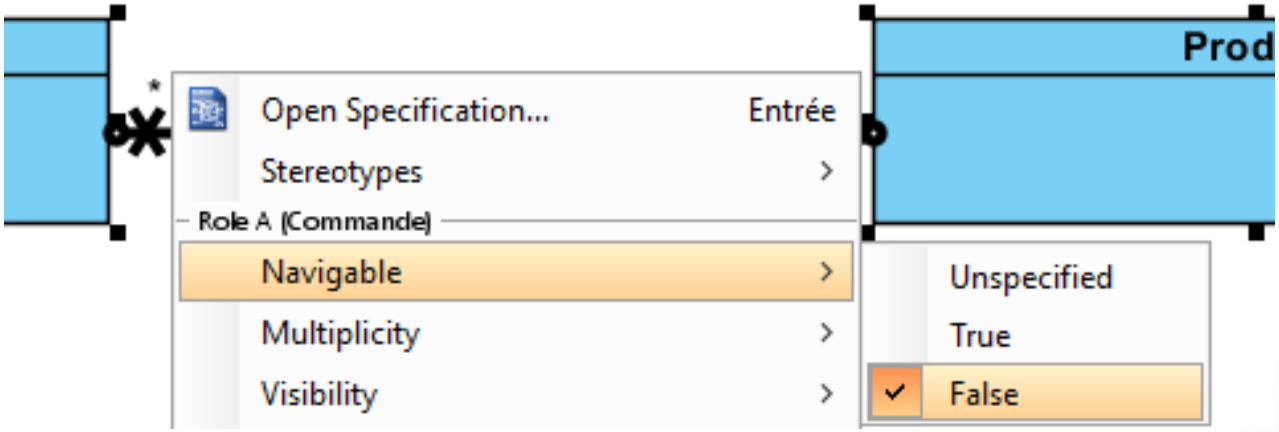
Graphiquement, la navigabilité est indiquée par une flèche du côté navigable.

Du côté non navigable, il y a une croix

Utilisation dans Visual Paradigm



Clic droit du côté où on souhaite indiquer la non navigabilité puis Navigable → False



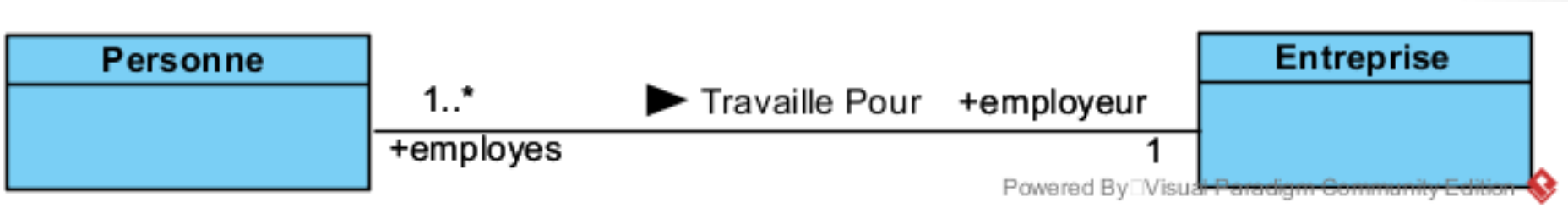
Comment caractériser complètement une association ?

- Une association a un nom à chaque extrémité, appelé ***rôle***, qui permet parfois d'apporter un complément d'information
- ***Visibilité***: la terminaison d'une association a une visibilité comme un attribut
  - ✓ Private, protected, package ou public

Comment caractériser complètement une association ?

- ***Multiplicité*** (aux 2 extrémités)
- ***Navigabilité*** (aux 2 extrémités)
- ***Propriétaire***: indique qui est propriétaire de la terminaison d'association (l'association ou à la classe à l'autre extrémité)

Exemple complet Visual Paradigm



Le + à côté de employes et employeur indique une visibilité publique des 2 rôles définis

L'association est navigable dans les 2 sens

Plan du cours

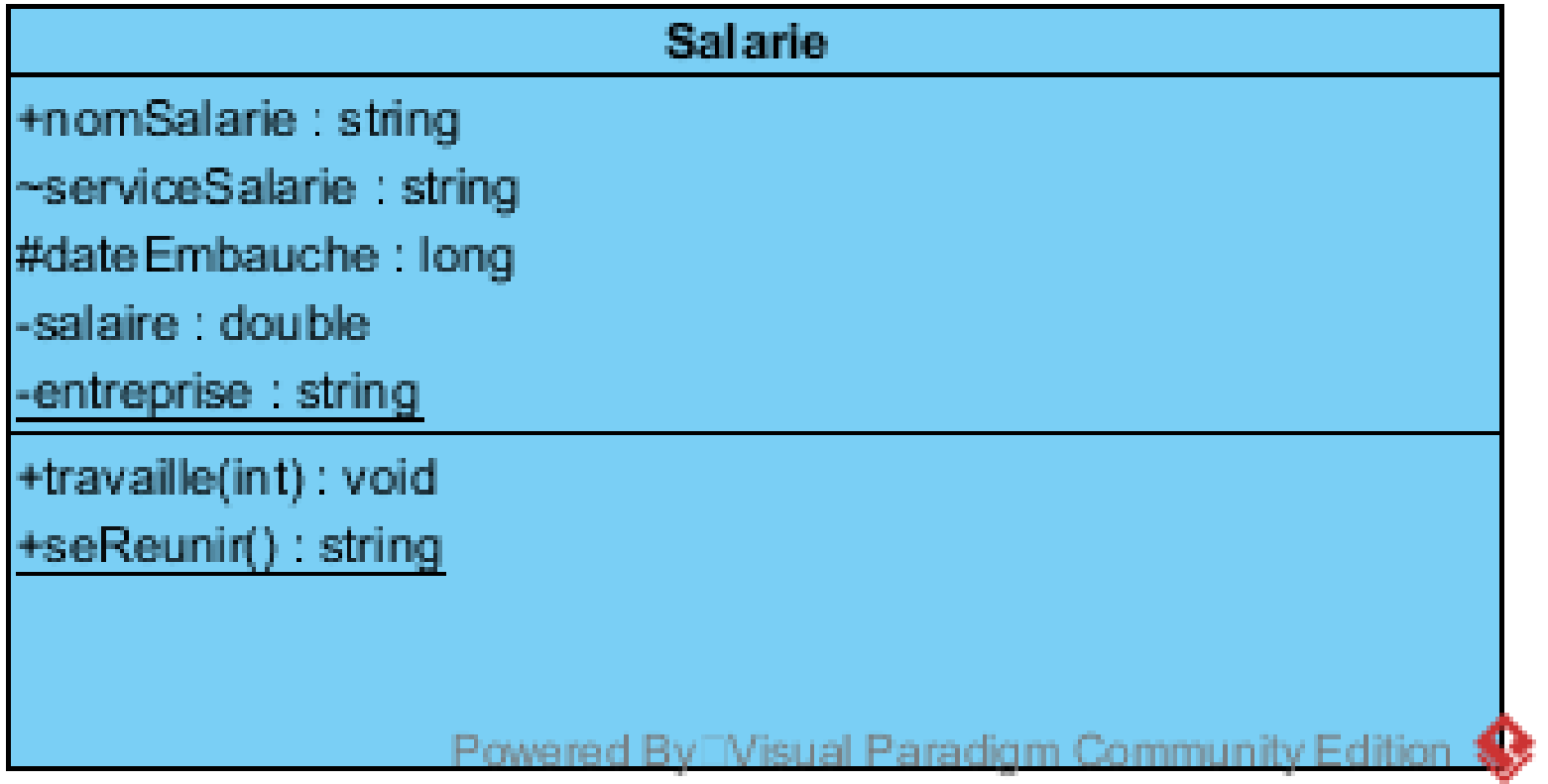
- Diagramme de classe en analyse
- Diagramme de classe en conception
- Dérivation diagramme de conception vers code

Implémentation en PHP ou Java

- UML étant plus abstrait et plus large qu'un langage objet comme Java ou PHP, il s'agit de traduire un diagramme de classe en un code Java ou PHP
- Les classes UML donnent des classes Java / PHP
- Une interface UML donne une interface Java / PHP
- Un héritage simple entre 2 classes Parent et Enfant s'implémente avec
  - 1 classe parente Parent
  - 1 classe Enfant qui « étend » (extends) Enfant

Exercice

- Reproduire sous Visual Paradigm la classe suivante
- Traduire sous Intelle, en Java, ou sous PHPStorm, en PHP la classe suivante
- Préciser la visibilité de chaque attribut et opération



Différence fondamentale analyse / conception

- En Analyse **la multiplicité de l'association** indique la ***localisation des clefs étrangères***

- En Conception la **navigabilité** indique la ***localisation des attributs*** *liés à l'association, sous forme de rôle* (une classe A possède comme attribut une occurrence ou une collection de la classe B)

Association unidirectionnelle

En Java, cela donne:

```
public class ClasseA {

private ClasseB  maClasseB;
…
}
```

Pour la classe B

```
public class ClasseB {

// la classe B ne connaît pas
// la classe A
}
```

Association bidirectionnelle 1 - 1

En Java, cela donne:

public class ClasseA {

private ClasseB roleB;
…
}

Pour la classe B

public class ClasseB {

private ClasseA roleA ;
…
}

Association unidirectionnelle 1 - *

En Java, cela donne:

```
public class ClasseA {

private Set<ClasseB>  roleB
  = new HashSet<ClasseB>();
…
}
```

Pour la classe B

```
public class ClasseB {

// la classe B ne connaît pas
// la classe A
…
}
```

Association bidirectionnelle 1 - *

En Java, cela donne:

```
public class ClasseA {

private Set<ClasseB>  roleB
  = new HashSet<ClasseB>();
…
}
```

Pour la classe B

```
public class ClasseB {

private ClasseA roleA ;
…
}
```