Classe Abstraite

- Soit **ObjetGeometrique**, qui possède deux méthodes **surface()** et **perimetre()**, une super-classe des classes **Triagle** et **Ellipse**.

- L'implémentation des méthodes surface() ou perimetre() dépend de la forme géométrique qui hérite de la classe ObjetGeometrique.

Classe Abstraite

- Les classes Triangle et Ellipse doivent remplacer ces méthodes par une implémentation significative.

- Alors que nous ne devons pas écrire une implémentation de ces méthodes dans la classe ObjetGeometrique.

- Ces méthodes serons déclarées comme abstraites (abstract) dans la classe ObjetGeometrique.

Classe Abstraite

- Pensez à des méthodes abstraites comme «placeholders» pour les méthodes qui seront éventuellement définis dans certaines sous-classes de la classe actuelle.

- En faisant cela, nous reconnaissons que ObjetGeometrique n'est pas complète dans un certain sens

- Il serait inapproprié d'instancier un objet de cette classe.

Classe Abstraite

- Plus généralement, les classes avec des méthodes abstraites doivent se déclarer abstraite.

- Autrement dit, une méthode abstraite ne peut être contenue dans une classe non abstraite.

- De même, si une sous-classe d'une super-classe abstraite n’implémente pas toutes les méthodes abstraites de la super-classe, elle doit être déclarée comme abstraite.

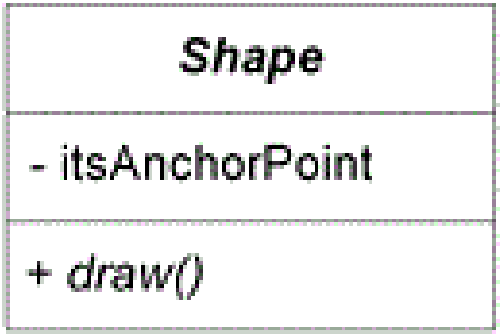
Classe Abstraite

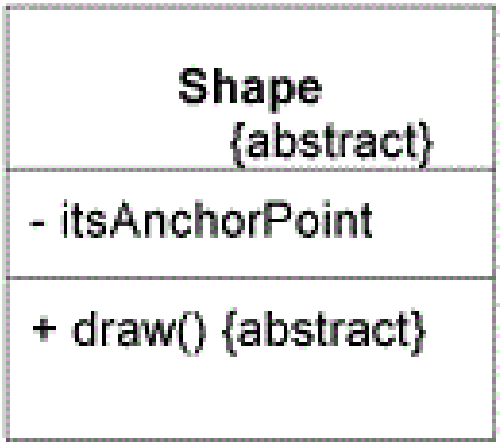
- Donc, si on désire étendre une classe abstraite, toutes les méthodes de la superclasse doivent être implémentées, même si elles ne sont pas utilisées dans la sous-classe.

- Une sous-classe peut être abstraite, même si sa superclasse est complète. Par exemple, la classe Objet est complète, mais peut avoir une sous-classe abstraite comme ObjetGeometrique.

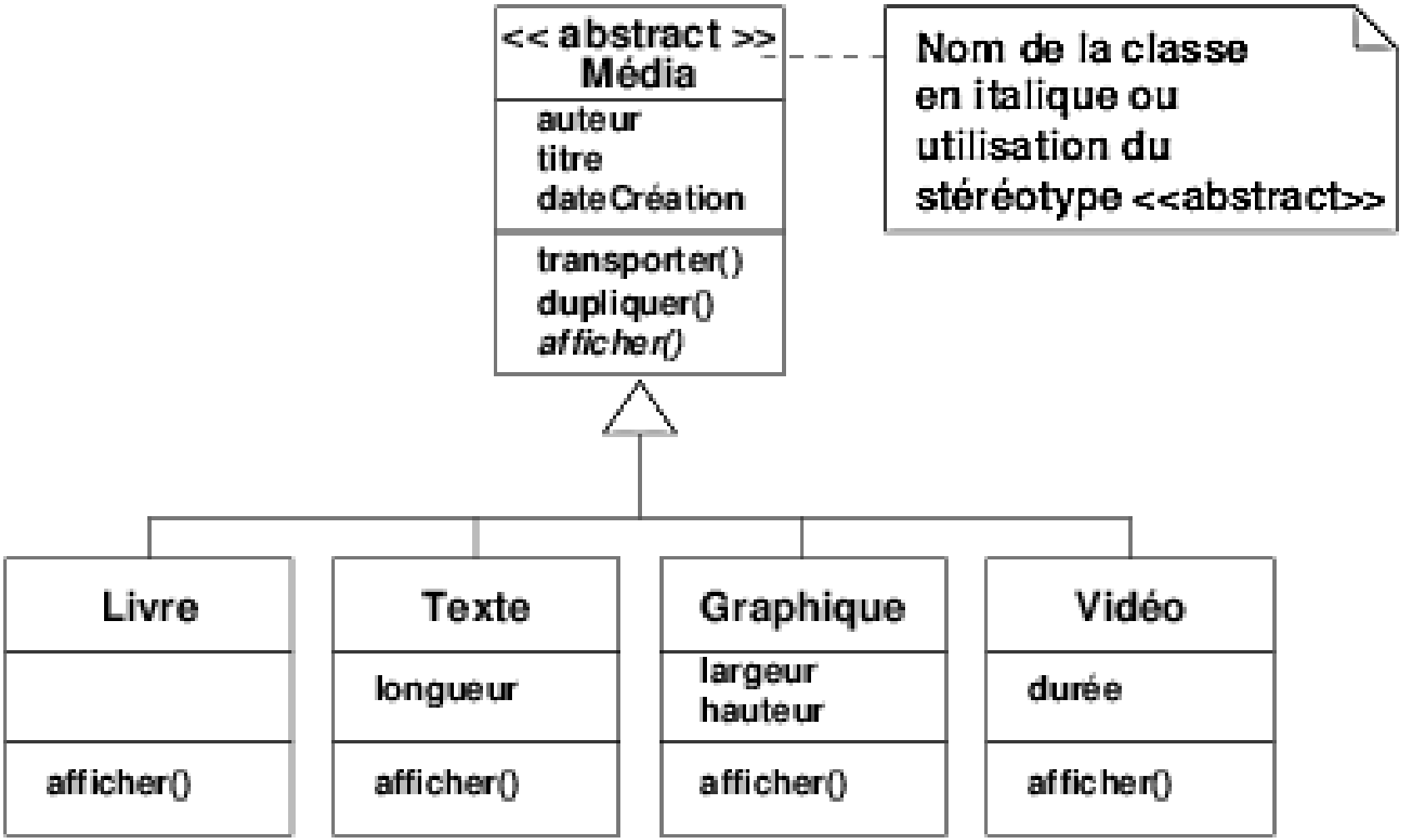
Classe Abstraite : Notation graphique

- Il y a deux façons d'indiquer qu'une classe ou une méthode est abstraite:
  - Vous pouvez écrire le nom en italique,
  - ou vous pouvez utiliser la propriété {abstract} ou <<abstract >>

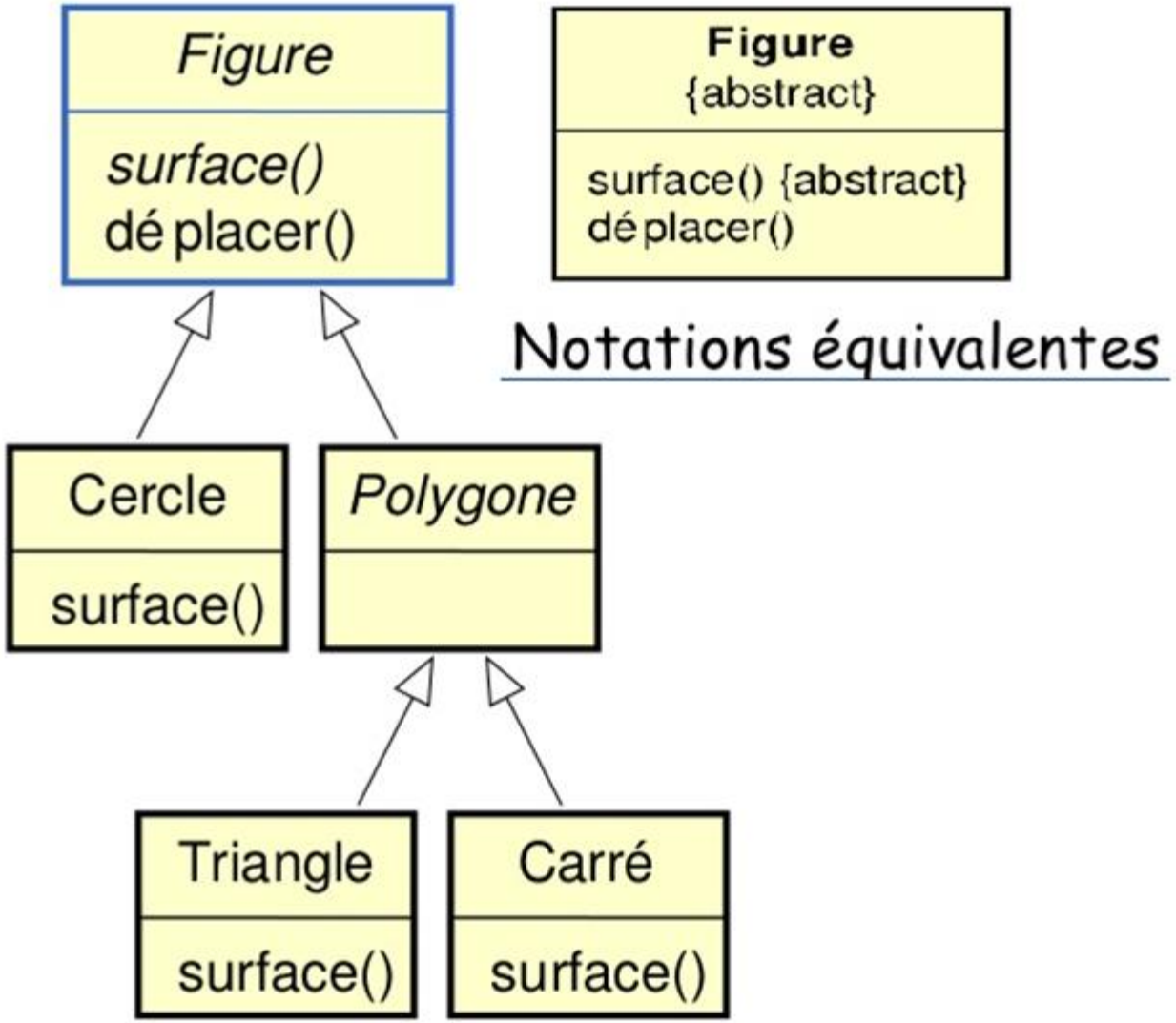




Classe Abstraite : exemple



Classe Abstraite : exemple



Classe Abstraite : exemple

- Dans certains cas, nous pouvons désirer implémenter deux objets proches possédant une partie de méthodes et de propriétés en commun comme *Voiture* et *Moto*.
- Certaines de ces méthodes comme *rouler()* sont totalement identiques d'un objet à l'autre alors que certaines autres diffèrent comme la méthode *seGarer()* qui est différente pour une *Voiture* et pour une *Moto*.

- Pour ce faire, l'OO introduit la notion de classe abstraite.

Classe Abstraite : règles et définitions

- Une méthode est dite abstraite lorsqu'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 ou lorsqu'une classe parent contient une méthode abstraite non encore réalisée.

- On ne peut instancier une classe abstraite : elle est vouée à se spécialiser.

Classe Abstraite : règles et définitions

- Une classe abstraite peut très bien contenir des méthodes concrètes.
- Spécialisation nécessaire jusqu'à obtenir des classes instanciables
- Impose un comportement commun à toutes les classes dérivées
- Une classe abstraite pure ne comporte que des méthodes abstraites.
- Une telle classe est appelée une interface.

Exercice 1 - Classes abstraites

- La classe *Vector* est utilisée de deux façons pour créer une FIFO (First In / First Out) et une LIFO (Last In / First Out).
- Créer la classe abstraite *Stack* qui contient les méthodes abstraites:
  - *Object get();*
  - *void put(Object);*
- Implémenter cette classe de deux façons:
  - par une classe *FIFOStack*
  - par une classe *LIFOStack* selon le comportement de pile désiré.
- Utiliser les méthodes *addElement()* et *removeElement()* de la classe *Vector*.

INTERFACE

Interface

- Une interface définit le comportement **visible** d'une classe.
- Ce comportement est défini par une liste d'opérations ayant une visibilité « public ».
- Aucun attribut ou association n'est défini pour une interface.
- Une interface est en fait une classe particulière (avec le stéréotype « « interface » »).

Interface

UML représente les interfaces :

- soit au moyen de petits cercles reliés par un trait à l'élément qui fournit les services décrits par l'interface
- soit au moyen de classes avec le mot clé ««interface»».
- Cette notation permet de faire figurer dans le compartiment des opérations la liste des services de l'interface.

Interface

- Une interface doit être réalisée par au moins une classe et peut l'être par plusieurs.

- Graphiquement, cela est représenté par un trait discontinu terminé par une flèche triangulaire et le stéréotype « ».

Interface

Les relations possibles sur une interface sont :

- **l’utilisation**

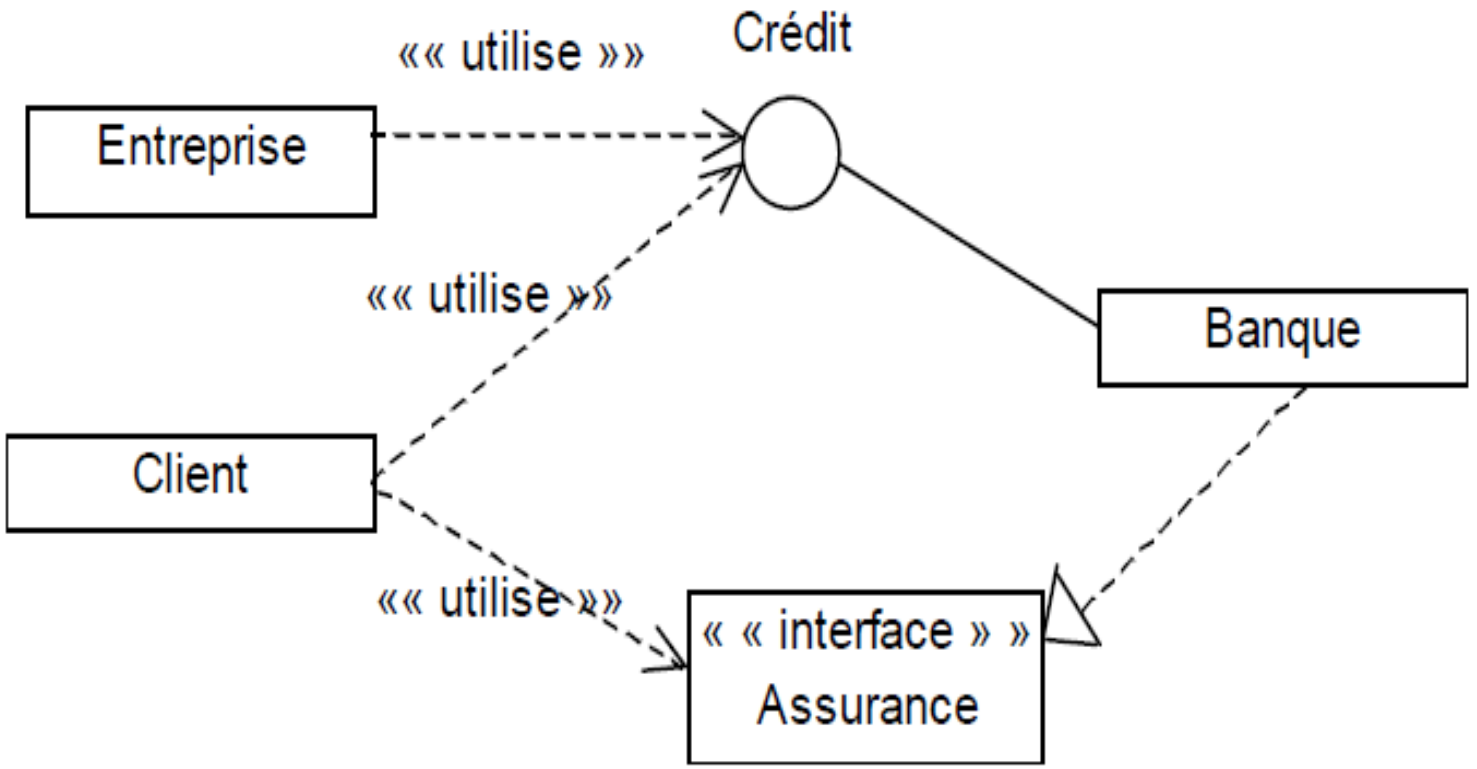
Cette relation concerne toute classe client qui souhaite accéder à la classe interface de manière à accéder à ses opérations. C’est une relation d’utilisation standard.

- **la réalisation**

Cette relation n’est utilisée que pour les interfaces. Une réalisation est une relation entre une classe et une interface. Elle montre que la classe réalise les opérations offertes par l'interface.

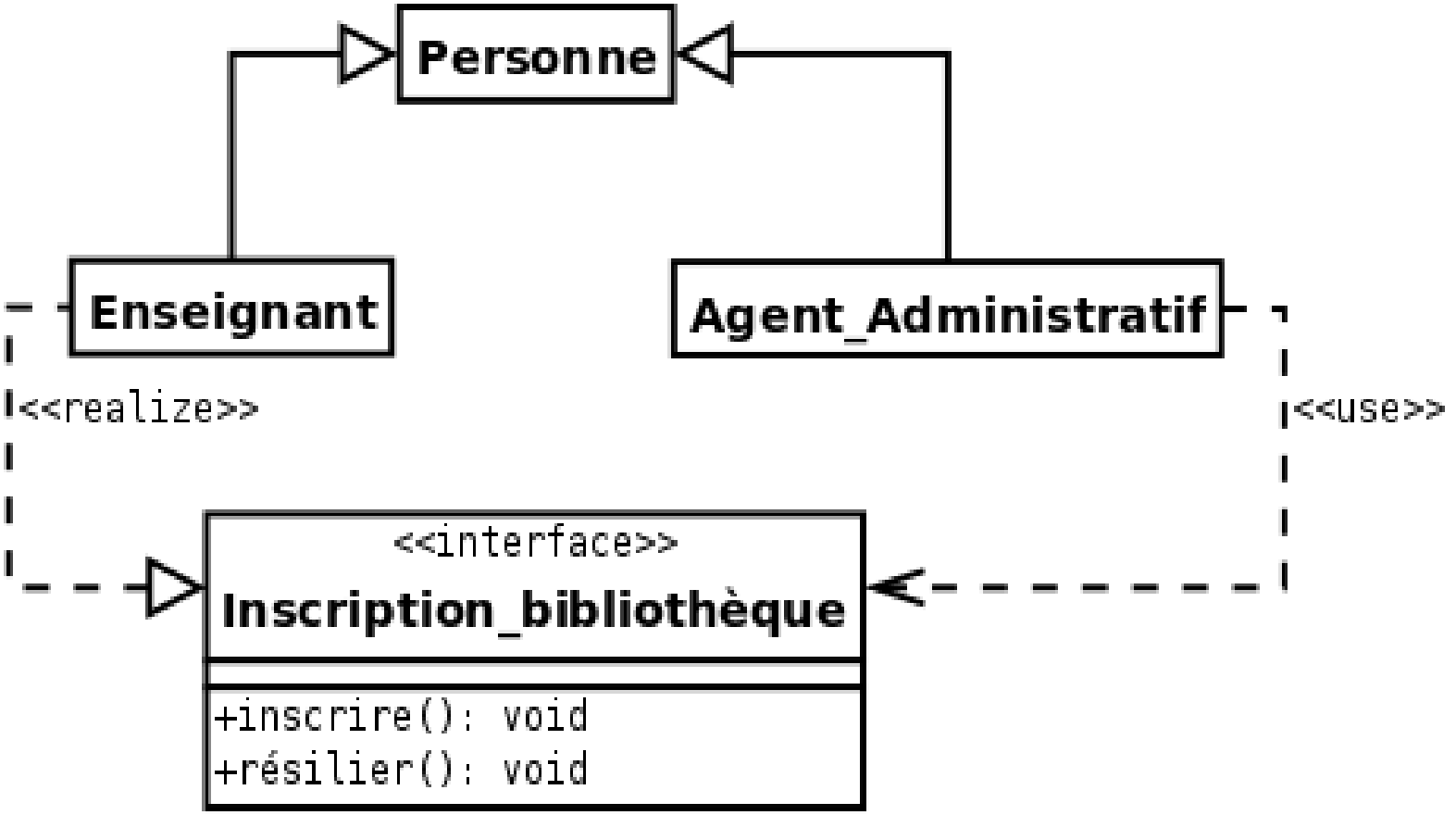
Interface

- Représentations des Interfaces



Interface

- Représentations des Interfaces



Interface : Règles et Définitions

- Une interface doit être réalisée par au moins une classe et peut l'être par plusieurs.
- Graphiquement, cela est représenté par un trait discontinu terminé par une flèche triangulaire et le stéréotype « *realize* ».
- Une classe (classe cliente de l'interface) peut dépendre d'une interface (interface requise).
- On représente cela par une relation de dépendance et le stéréotype « *use* ».

Interface

Une classe C qui implémente une interface I a 2 seuls cas possibles :

- soit la classe C implémente toutes les méthodes de I
- soit la classe C doit être déclarée abstract ; Les méthodes manquantes seront implémentées par les classes filles de C