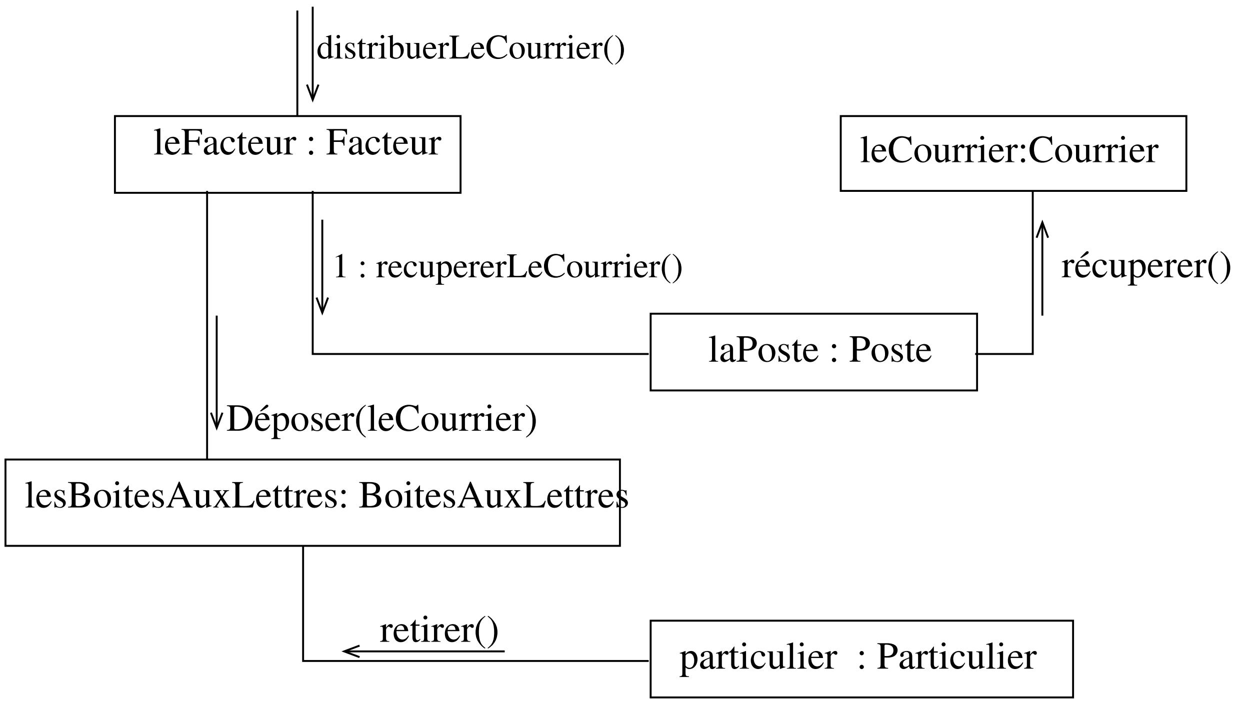
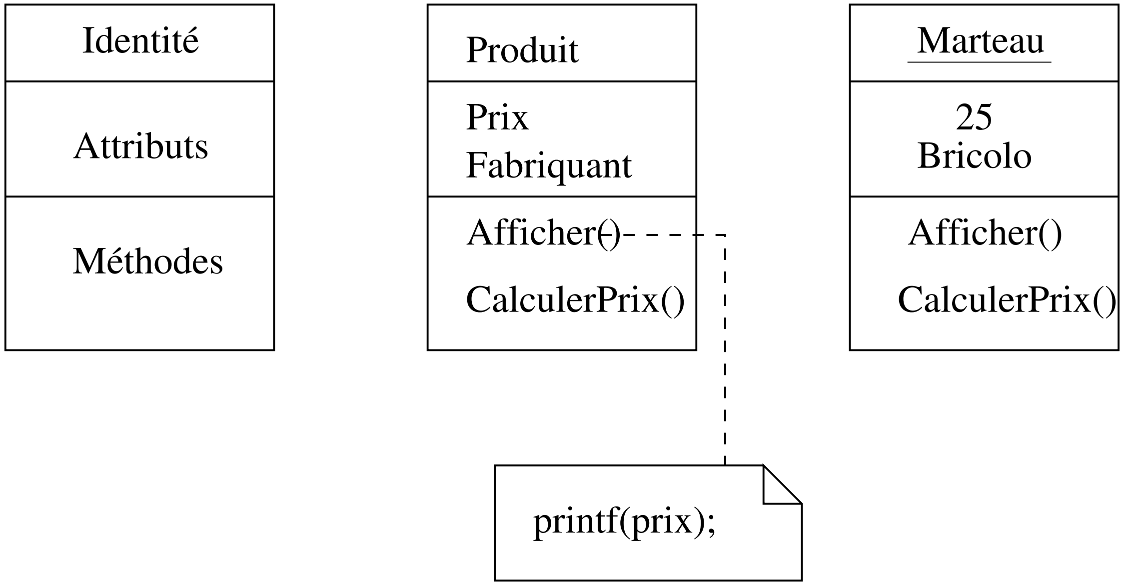

distribuerLeCourrier()
leFacteur : Facteur
leCourrier:Courrier
1 : recupererLeCourrier()
récuperer()
laPoste : Poste
Déposer(leCourrier)
lesBoitesAuxLettres: BoitesAuxLettres
retirer()
particulier : Particulier

Objets et Classe d'objets

$$\boxed{\text{Objet} = \text{Etat} + \text{Comportement} + \text{Identité}}$$

➔ Conventions graphiques de UML

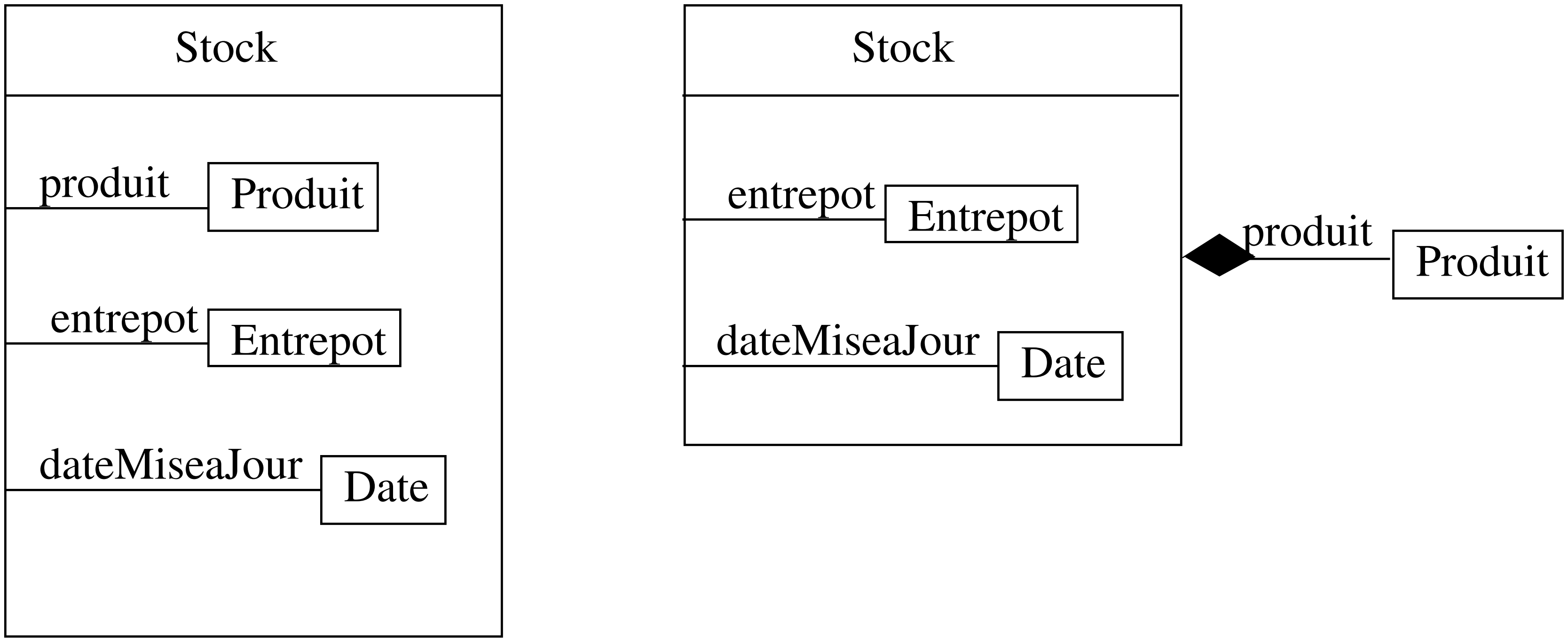


Les classes abstraites sont représentées uniquement par leur nom dans un rectangle. Il est possible de faire apparaitre le mot `abstract` sous ce nom.

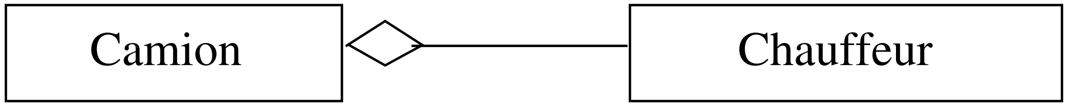
La composition d'objets

Les attributs d'un objet peuvent être eux-même des objets.

➜ **La composition par valeur** : la construction d'un objet physique implique la construction de ses attributs par valeur.



➔ **La composition par référence** : c'est un lien de référence qui peut être partagé par plusiseurs objets.



La construction du conteneur n'implique pas la construction de l'objet référencé.

NB : le losange se place du coté de l'objet référençant.

La visibilité des attributs et des méthodes

- ➔ Publique : un attribut ou une méthode publique est spécifée avec le signe +.

- ➔ Privée : un attribut ou une méthode privée est spécifée avec le signe −.

- ➔ Protected : un attribut ou une méthode protégée est spécifée avec le signe #.

Signature

La signature d'une méthode se compose de :

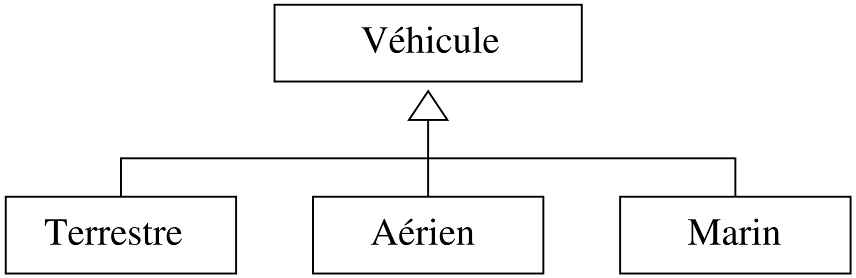
- son nom,
- le nombre et le type de ses paramètres en entrée,

Exemple : `public void afficher(String , Integer);`

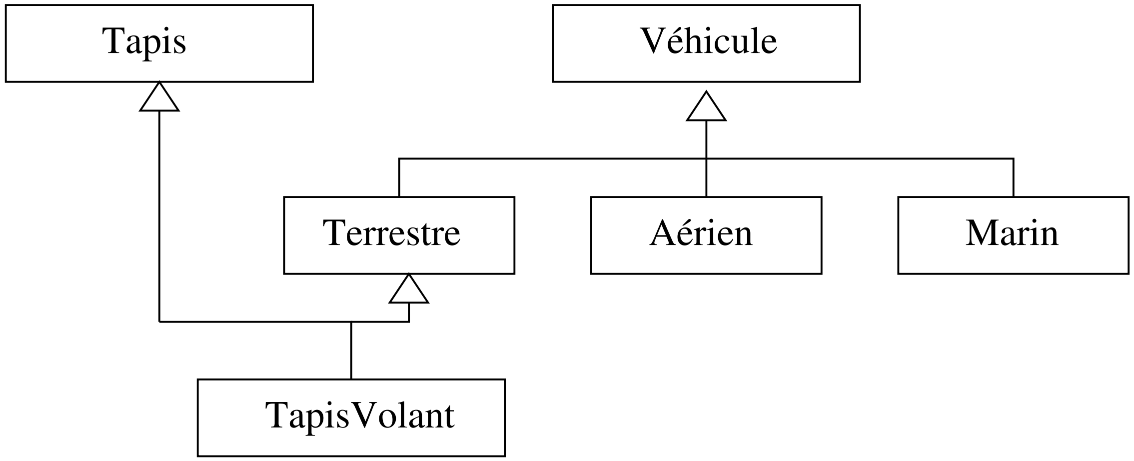
Dans un espace défini, deux méthodes peuvent avoir le même nom si elles n'ont pas la même signature.

Représentation de l'héritage

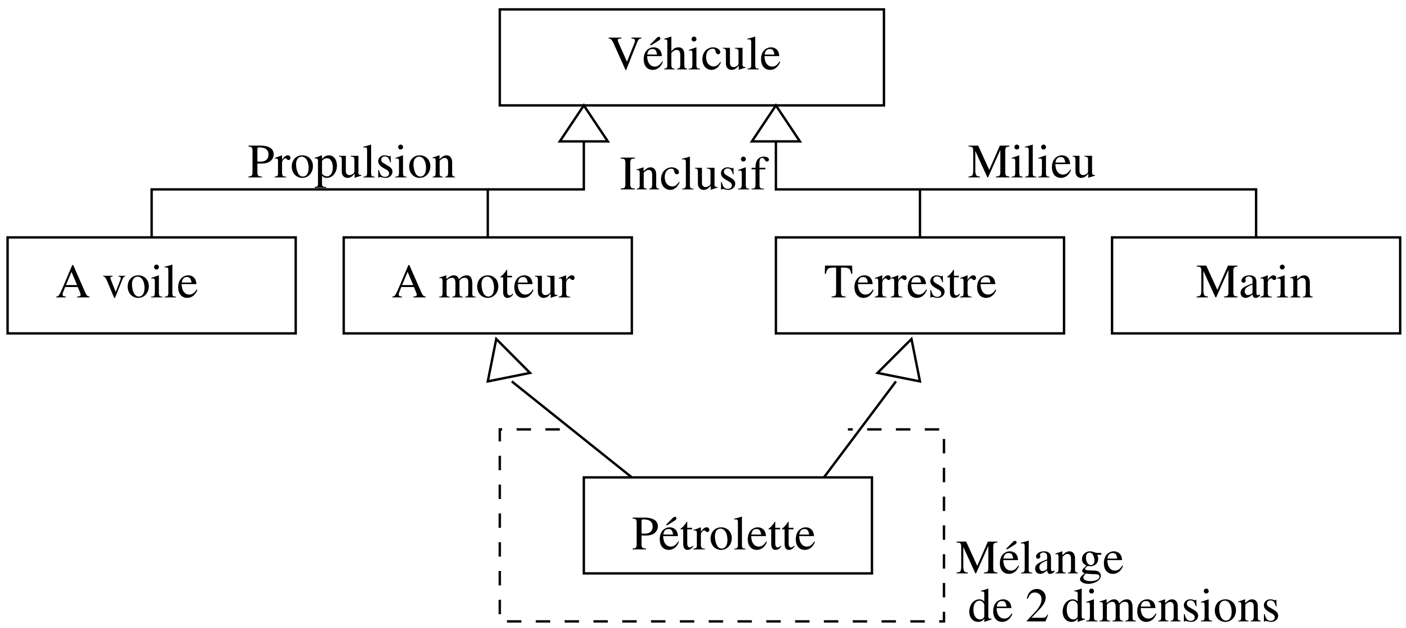
Héritage simple



Héritage multiple



Héritage multiple inclusif



Un exemple

Contexte :

Un utilisateur d'interface graphique compose des dessins complexes à partir d'éléments simples tel que les carrés, les rectangles . . . Une représentation simple définit des classes primitives graphiques teles que `Texte`, `Ligne` . . . et quelques autres classes classes jouant le rôle de containers.

Problème :

Cette approche ne permet pas de traiter de façon identique les objets containers et les objets primitives alors que l'utilisateur les traite de la même manière.

L'exemple du pattern `Composite`