Modèle OO

C. Delobel, V. Benzaken, E. Waller

Les éléments de base

- Un ensemble fini de noms de classe C.
- Un ensemble fini de noms d'attributs {a1,a2,...}.
- Un ensemble fini d'identificateurs d'objet O.
- Un ensemble fini de constantes D. Les constantes peuvent être classées en entier, réel, booléen, chaîne de caractères.
- La notation [a1 :..., a2 :..., ..., ak :...,] dénote un n-uplet. [] représente le n-uplet vide.
- Un ensemble d'éléments est noté {.....}.

Les valeurs

- L'ensemble des valeurs construites sur O sera noté V(O), ou tout simplement V. Il est défini récursivement de la façon suivante.
- V contient D union O.
- si v1,..., vk appartiennent à V alors {v1,v2,...,vk} et [a1:v1,..., ak:vk] appartiennent à V.

La population des classes

- A chaque classe on affecte un ensemble fini d'identificateurs par la fonction de population š : C --> 2^{O}.
- La fonction š sera dite disjointe si š réalise une partition. Dans le cas contraire š sera non-disjointe.
- Cette fonction š modélise la création des objets dans chaque classe lorsque š est disjointe.

Les objets

- Dans les langages à objets, un objet possède une idendité, un état interne, et un comportement spécifié par des opérations. Pour l'instant nous considèrerons que les deux premiers éléments.
- Un *objet* est une paire (i,v) où i est une identité, c'est-à-dire un élément de O, et v une valeur.
- Exemples :
  - (#1, [nom : "dupont", age : 10])
  - (#2, {4, 5, 8 })

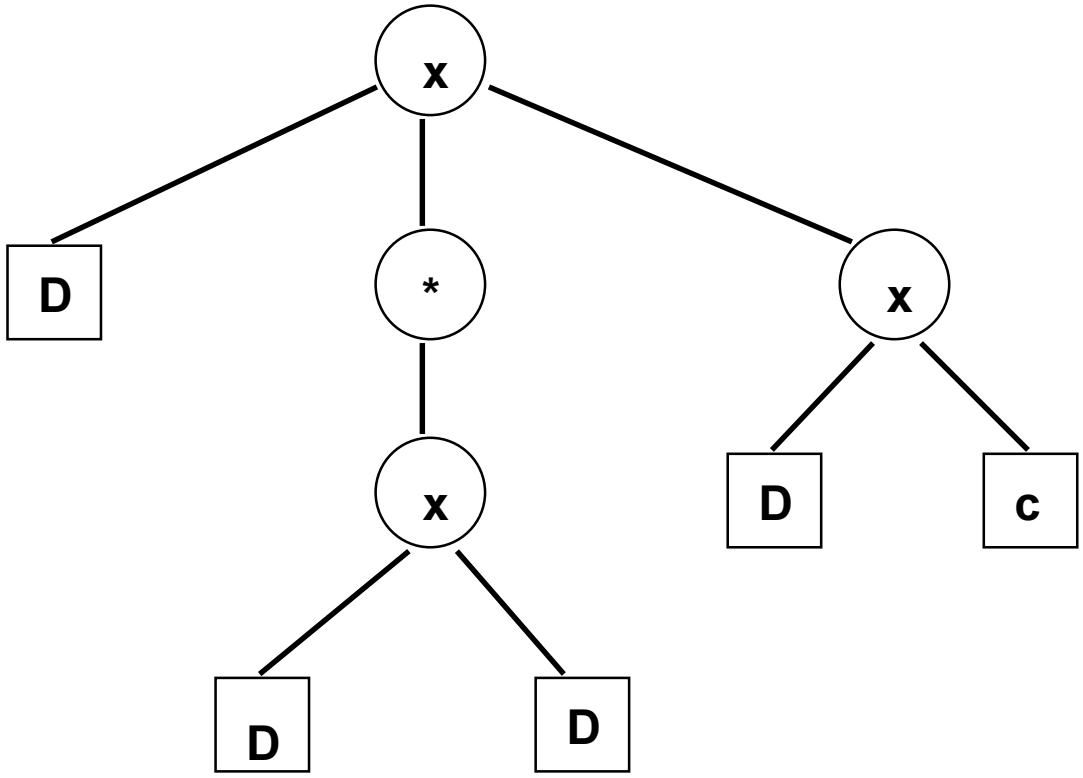
Eléments singuliers

- Parmi l'ensemble des objets O, on considère parfois un objet particulier, l'objet vide dénoté *nil*.
- Parmi l'ensemble des classes C, on considère parfois une classe particulière, la classe *Any*.
- La classe Any ne peut pas apparaître dans la définition d'une classe.

Les types

- Les valeurs associées aux objets sont typées. L'ensemble des expressions de type pris en compte dans un modèle peut être plus ou moins riche.
- Une *expression de type* t est définie parla syntaxe : t ::= Ø | D | C | [a1 : t, ..., ak : t] | { t } | t union t | t inter t . Type (C) dénote l'ensemble des expressions de type.
- Les modèles réels ne possèdent pas ce système de typage, restriction sur l'union les ensembles et les listes
- Représentation par un arbre syntaxique

Arbre syntaxique d'un type



Interprétation d'un type

- L'interprétation d'un type est dépendante de la fonction š et par conséquent de l'ensembles des identificateurs d'objet. Nous noterons $[\![t]\!]_{\check{s}}$ l'interprétation du type t sur la fonction š. Comme š sera toujours sousentendu, on notera tout simplement [[t]].
- L'*interprétation* définit le domaine des valeurs possibles pour un type donné. Il y a deux interprétations qui se sont révélées intéressantes pour la formalisation : l'*interprétation stricte* et l'*interprétation étendue* au sens de Cardelli.

Interprétation stricte

- [[Ø]] = Ø
- [[D]] = D
- [[c]] = š (c) pour chaque classe c de C
- [[t union t']] = [[t]] union [[t']]
- [[t inter t']] = [[t]] inter [[t']]
- [[{ t }]] = { { v1,..., vk } | vi appartient à [[t]], i = 1,k }
- [[[a1 : t1,..., ak : tk]]] = { [a1 : v1,..., ak : vk] | vi appartient à [[ti]], i = 1,k }

Interprétation étendue

- [[Ø]] = Ø
- [[D]] = D
- [[c]] = š (c) pour chaque classe c de C
- [[t union t']] = [[t]] union [[t']]
- [[t inter t']] = [[t]] inter [[t']]
- [[{ t }]] = { { v1,..., vk } | vi appartient à [[t]], i = 1,k }
- [[[a1 : t1,..., ak : tk]]] = { [a1 : v1,..., ak : vk, ak+1 : vk+1,..., al : vl] | il existe ak+1,..., al différent de a1,..., ak, et vi appartient à [[tk]], i = 1,k }.

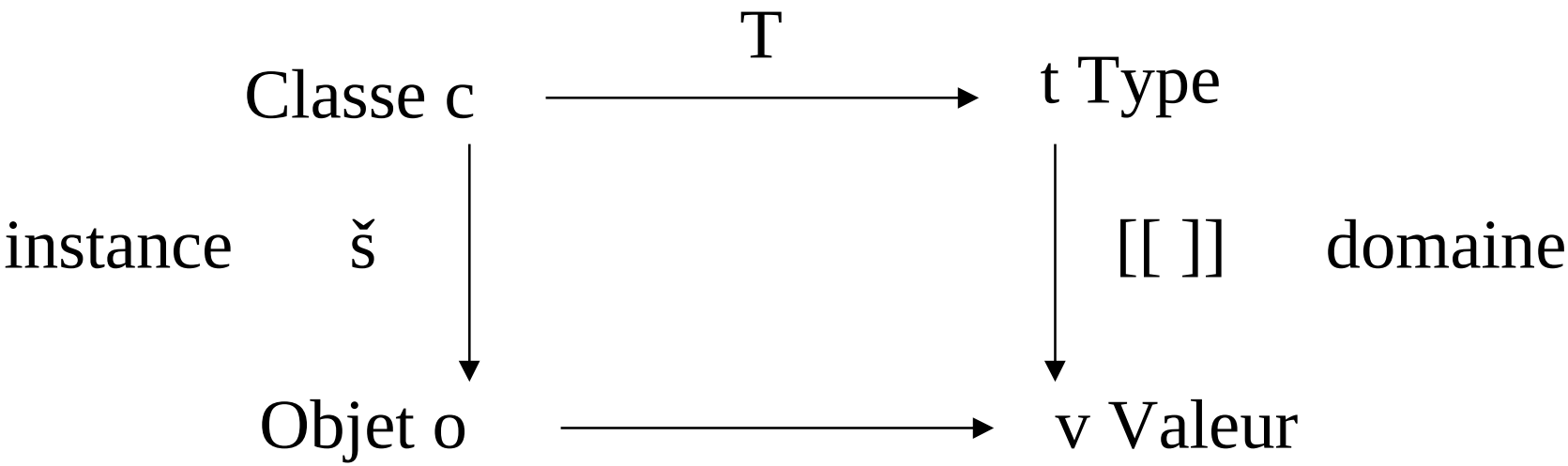
Equivalence de types

- Définition : Deux types sont équivalents s'ils ont la même interprétation.
- Définition : Un type est réduit par rapport à l'intersection si dans l'arbre syntaxique de t aucun nœud intersection est ancêtre de nœuds ensemble, union ou n-uplet. Un type est sans intersection si son arbre syntaxique ne contient pas un nœud d'intersection.
- Proposition : Pour chaque type t il existe une expression équivalente t' qui est sans intersection si la fonction š est disjointe, sinon qui est réduit par rapport à l'intersection.

Schéma et instance (niveau 1)

- Définition : Un *schéma* S est une paire (C, T), où C est un ensemble de noms de classe et T une fonction de C dans Type (C).
- Une instance I d'un schéma S est une paire (š,) où š est une fonction de population disjointe et une fonction de valuation de O dans V telle que : (1) pour chaque classe c de C, si o appartient à à š(c) alors (o) est un élément de [[T(c)]], (2) est total sur š(c) pour tout c tel que T(c) = { t }.

Diagramme 1



Exemple de schéma

Classes : Personne, Realisateur, Acteur, Acteur-Realisateur, Film

Types :

T(Personne) = [nom: string, nat: string, sexe:string]
T(Realisateur) = [nom: string, nat: string, sexe:string, realise:Film]
T(Acteur) = [nom: string, nat: string, sexe:string, joue:{film}]
T(Acteur-Realisateur) = [nom: string, nat: string, sexe:string,
joue:{Film}, realise:Film]
T(Film) = [titre:string, realisateur:Realisateur, acteurs:{Acteur}]

Exemple d’instance

O = {#f1,#a1,#r1}

š(Personne) = Ø
š(Acteur) = {#a1}
š(Realisateur) = {#r1}
š(Acteur-Realisateur) = Ø
š(Film) = {#f1}

(#f1) = [titre: "La strada", realisateur: #r1, acteurs:{#a1}]
(#r1) = [nom:"fellini", nat:"italien", sexe:"m", realise:#f1]
(#a1) = [nom: "loren", nat:"italienne", sexe:"f", joue:{#f1}]

Schéma avec héritage : niveau 2

- Dans la modélisation précédente, il n'est pas possible de construire un objet de type Film dans lequel l'attribut "acteurs" peut prendre comme valeur soit un acteur soit un acteur-réalisateur.
- Pour remédier à ce problème on introduit une relation d'ordre partiel d'héritage entre classes.
- La relation d'héritage est indépendante du type des objets.

Exemple d'héritage

- Un réalisateur est une personne.
- Un acteur est une personne.
- Un acteur-rélisateur est un acteur.
- Un acteur-réalisateur est un réalisateur.

Realisateur < Personne
Acteur < Personne
Acteur-Realisateur < Realisateur
Acteur-Realisateur < Acteur

Schéma avec héritage

- Comment faut-il interpréter les concepts de type et d'héritage pour permettre les manipulations d'objets ?
- Fonction de population héritée : étant donnée une fonction de population š disjointe, on définit $š_{<}$ par $š_{<}(c) = š(c) \cup \{š(c') \mid c' < c\}$.
- Un schéma S est un triplet (C, T, <) et une instance I de S est une paire (š,) telle que : si o est dans š(c) alors (o) est dans $[[T(c)]]_{š<}$

Exemple de schéma

Classes : Personne, Realisateur, Acteur, Acteur-Realisateur, Film

Types :

T(Personne) = [nom: string, nat: string, sexe:string]
T(Realisateur) = [nom: string, nat: string, sexe:string, realise:Film]
T(Acteur) = [nom: string, nat: string, sexe:string, joue:{film}]
T(Acteur-Realisateur) = [nom: string, nat: string, sexe:string,
joue:{Film}, realise:Film]
T(Film) = [titre:string, realisateur:Realisateur, acteurs:{Acteur}]

Héritage :

Realisateur < Personne
Acteur < Personne
Acteur-Realisateur < Realisateur
Acteur-Realisateur < Acteur

Instance de schéma

O = {#f1,#a1,#ar1}
š(Personne) = Ø
š(Acteur) = {#a1}
š(Realisateur) = Ø
š(Acteur-Realisateur) = {#ar1}
š(Film) = {#f1}

(#f1) = [titre: "La strada", realisateur: #ar1, acteurs:{#a1,#ar1}]
(#ar1) = [nom:"fellini", nat:"italien", sexe:"m", realise:#f1]
(#a1) = [nom: "loren", nat:"italienne", sexe:"f", joue:{#f1}]

š< (Personne) = {#a1, #ar1}
š< (Acteur) = {#a1, #ar1}
š< (Realisateur) = {#ar1}
š< (Acteur-Realisateur) = {#ar1}
š< (Film) = {#f1}

Schéma avec héritage : équivalence

- Il est possible d'exprimer la signification de la relation d'héritage < par un schéma sans héritage à condition que le système de type possède l'union de type.

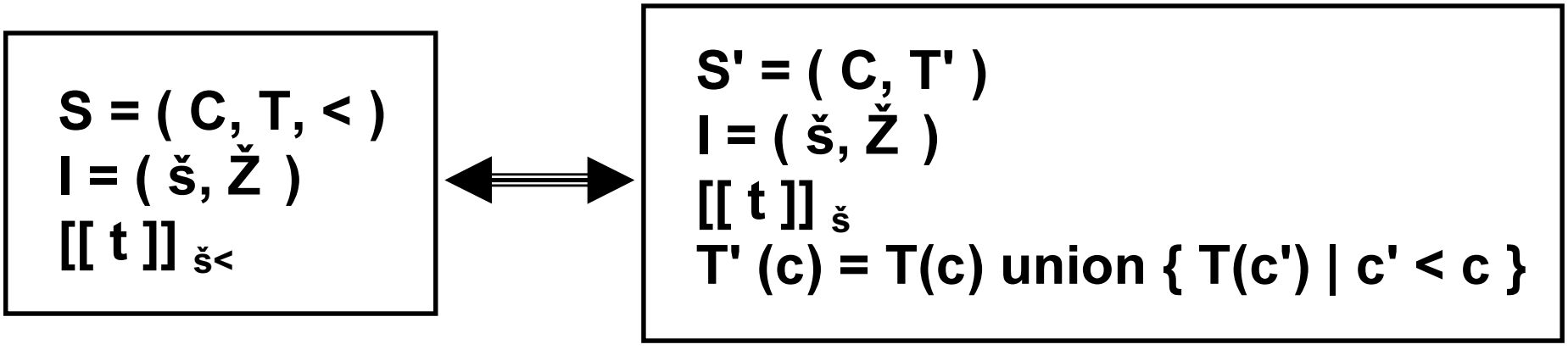
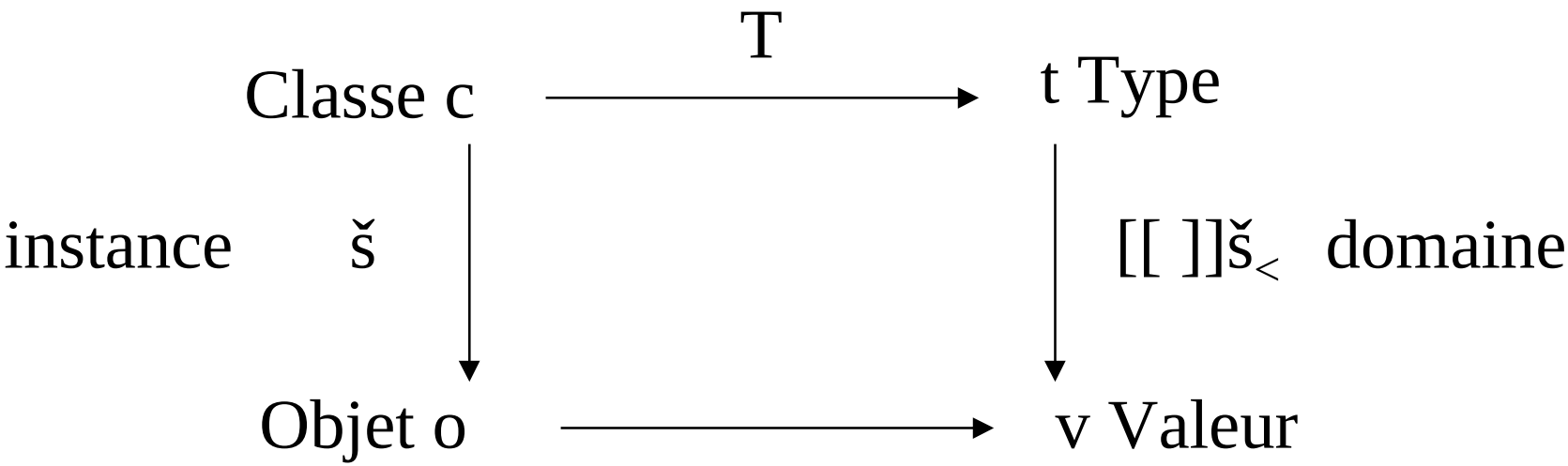


Diagramme 2



Application

Classes : Personne, Realisateur, Acteur, Acteur-Realisateur, Film

Types :

T’(Personne) = [nom: string, nat: string, sexe:string]
U [nom: string, nat: string, sexe:string, realise:Film]
U [nom: string, nat: string, sexe:string, joue:{film}]
U [nom: string, nat: string, sexe:string, joue:{Film}, realise:Film]

T'(Realisateur) = [nom: string, nat: string, sexe:string, realise:Film]
U [nom: string, nat: string, sexe:string, joue:{Film}, realise:Film]

T'(Acteur) = [nom: string, nat: string, sexe:string, joue:{film}]
U [nom: string, nat: string, sexe:string, joue:{Film}, realise:Film]

T'(Acteur-Realisateur) = [nom: string, nat: string, sexe:string,
joue:{Film}, realise:Film]

T'(Film) = [titre:string, realisateur:Realisateur, acteurs:{Acteur}]

Schéma avec héritage : niveau 3

- L'inconvénient de la solution précédente est que des classes en rapport d'héritage peuvent avoir des structures de type incompatible.
- Un schéma S est un triplet (C, T, <).
- Si c < c' alors les types de c et c' sont structurellement compatibles. C'est-à-dire qu'ils vérifient une relation de sous-typage Š.
- Le système de type pris en compte est t ::= Any | D | C | [....] | { t }.

Relation de sous-typage

- c Š c' ssi c < c'
- [a1 : t1, ..., an : tn, ..., ak : tk] Š [a1 : s1, ..., an : tn] ssi ti Š si, i = 1, n.
- { t } Š { s } ssi t Š s.
- t Š Any

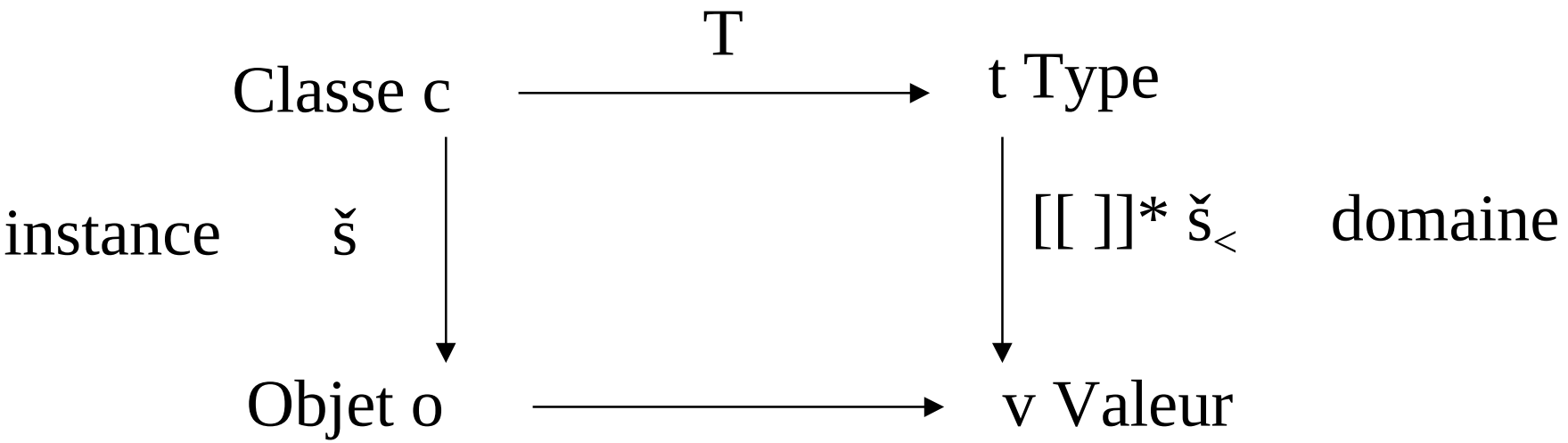
Schéma bien formé

- Un schéma S est bien formé si pour toutes classes c et c' telles que c < c', alors T(c) Š T(c').
- Exemple : T(Acteur) Š T(Personne), T(Acteur-Realisateur) Š T (Acteur)

Sémantique d'un schéma bien formé

- Etant donné un schéma S = (C, T, <) bien formé, une instance I de S est une paire (š,) où š est une fonction de population disjointe et une fonction de valuation des objets.
- L'interprétation du système de types se fait avec les règles suivantes :
  - ➤ on prend pour chaque type t l'interprétation étendue avec la fonction de population héritée,
  - ➤ [[c]]* = $š_{<}$ (c)
  - ➤ [[Any]]* = V(O)

Diagramme 3



Proposition

- Proposition : pour tous types t et t' tels que t Š t', alors [[t]] est incluse dans [[t']].

Schéma, héritage et méthodes : niveau 4

- La relation d'héritage est définie non seulement sur des objets qui ont des structures voisines mais aussi sur des comportements voisins.
- Les méthodes applicables aux objets d'une classe sont classées en méthodes : de création, de construction, d'observation, de modification et de suppression.
- Une méthode comporte une signature et un corps. La signature fait partie de la partie visible d'une classe tandisque le corps correspond à l'implémentation de la méthode.

Exemple de méthodes

```
method imp-nom in class Personne : string
        { if ( self.sexe = "m" )
                return "Mr." + self.nom
        else
                retrun "Mme" + self.nom}

method imp-nom in class Realisateur : string
        { return "Mr. le Réalisateur" + self.nom}

imp-nom : Personne -> string
imp-nom : Relaisateur -> string
```

Signature d'une méthode

- Une signature est une expression de la forme : m : c x t1 x ... x tn --> t, où m est le nom de la méthode, c la classe de définition, et t1, ..., tn, t des types.
- Une méthode m est surchargée si elle est définie dans plusieurs classes reliées entre-elles par la relation d'héritage.

Comportement des méthodes

- La classe Acteur n'a pas de méthode spécifique pour imprimer le nom, elle hérite de la méthode imp-nom définie dans classe Personne.
- Déterminer quelle implémentation doit être utilisée pour une méthode m et une classe c s'appelle la *résolution de la méthode*.
- Supposons que l'objet #p25 soit dans l'interprétation étendue de la classe Personne et que l'on invoque la méthode imp-nom (#p25, quelle implémentation choisir ?

Liaison statique ou dynamique

- Statique : si la variable x contenant l’objet #p25 est déclaré de type c, on utilise l'implémentation relative à cette classe dès la compilation.
- Dynamique : à l'exécution la variable x déclaré de type c, peut être de type c ou de type c' sous-type de c, on utilise alors le vrai type de x, c'est-à-dire le type de l'objet #p25.

Schéma bien formé

- Un schéma S est un quadruplet (C, T, <, M) où M est un ensemble de signatures.
- Un schéma est bien formé s'il vérifie :
  - La relation d'héritage < est compatible avec la relation de sous-typage Š.
  - si une méthode m est définie en deux classes c et c', avec c < c', alors les deux signatures m : c x t1 x ... x tn --> t et m : c' x s1 x ... x sn --> s satisfont ti Š si et t Š s (condition de covariance).
  - s'il existe deux classes c1 et c2 ayant une sous-classe c3 où la méthode m est définie en c1 et c2 mais non en c3, alors il doit exister une classe c4, sous-classe de c1 et c2 et super-classe de c3, où m est aussi définie.

Instance d'un schéma

- Une instance I d'un schéma S est un triplet (š, , µ) où š est une fonction de population disjointe, est une fonction de valuation, et µ une fonction des signatures de M dans l'ensemble des relations d'entrée-sortie.
- doit satisfaire la condition de valuation, à savoir que la valeur associée à un objet doit être prise dans son domaine d'interprétation.
- si une méthode m est surchargée, les relations d'entrée-sortie doivent être compatibles sur une partie du domaine des valeurs.

Schéma et persistance : niveau 5

- Unschéma S = (C, T, <, M, G) est un schéma pour une base de données à objets où G est un ensemble de noms dénotant des racines de persistance de la base ssi :
  - ➤ T est une fonction de C U G dans les types de C
  - ➤ (C, T, <) est une hiérarchie bien formée
  - ➤ M est un ensemble bien formé de signatures

Instance

- Une instance du schéma (C, T, <,M, G) est un quadruplet (š, , µ, ß) où š, , et µ ont les significations précédentes.
- ß est une fonction de G dans V(O).

Typage sûr et héritage

- L'objectif est de vérifier le plus possible les programmes à la compilation pour être sûr qu'à l'exécution il n'y ait pas d'erreurs.
- Héritage et typage sûr sont parfois incompatibles. La condition de co-variance ne garantit pas un typage sûr.

Exemple 1

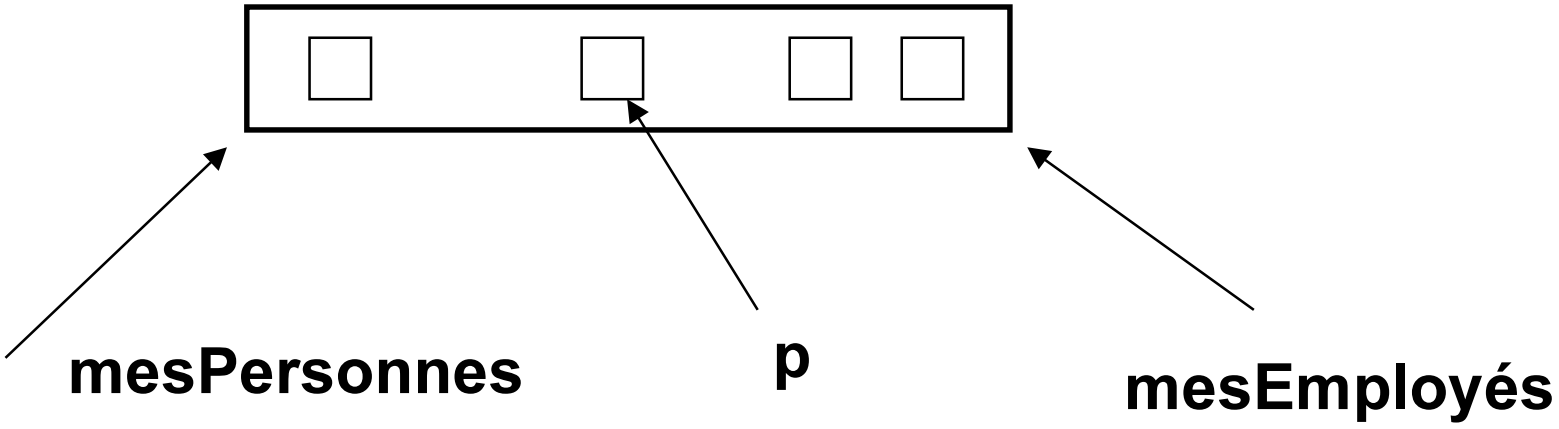
- Personne avec T(Personne) = [nom :D, age : D, chef : Personne] et Employé avec T(Employé) = [nom :D, age : D, chef : Employé, salaire : D].
- Employé < Personne
- Dans la classe Personne une méthode Augmenterage : Personne x D --> Personne, et dans la classe Employé une méthode Augmentersalaire : Employé x D ---> Employé.

Exemple 1 (suite)

```
var e : Employé
e. Augmenterage ( 2 ). Augmentersalaire (100 )
```

Exemple 2

- Etendons le schéma précédent avec des classes de type ensemble.
- LesPersonnes, T(LesPersonnes) = { Personne }
  ajouter : LesPersonnes x Personne --> LesPersonnes
- LesEmployés, T(LesEmployés) = { Employé },
  ajouter : LesEmployés x Employé --> LesEmployés

Exemple 2 (suite)

```
var : mesEmployés : LesEmployés
      mesPersonnes : LesPersonnes
      p : Personne ;
mesPersonnes = mesEmployés ;
mesPersonnes. ajouter (p) ;
```