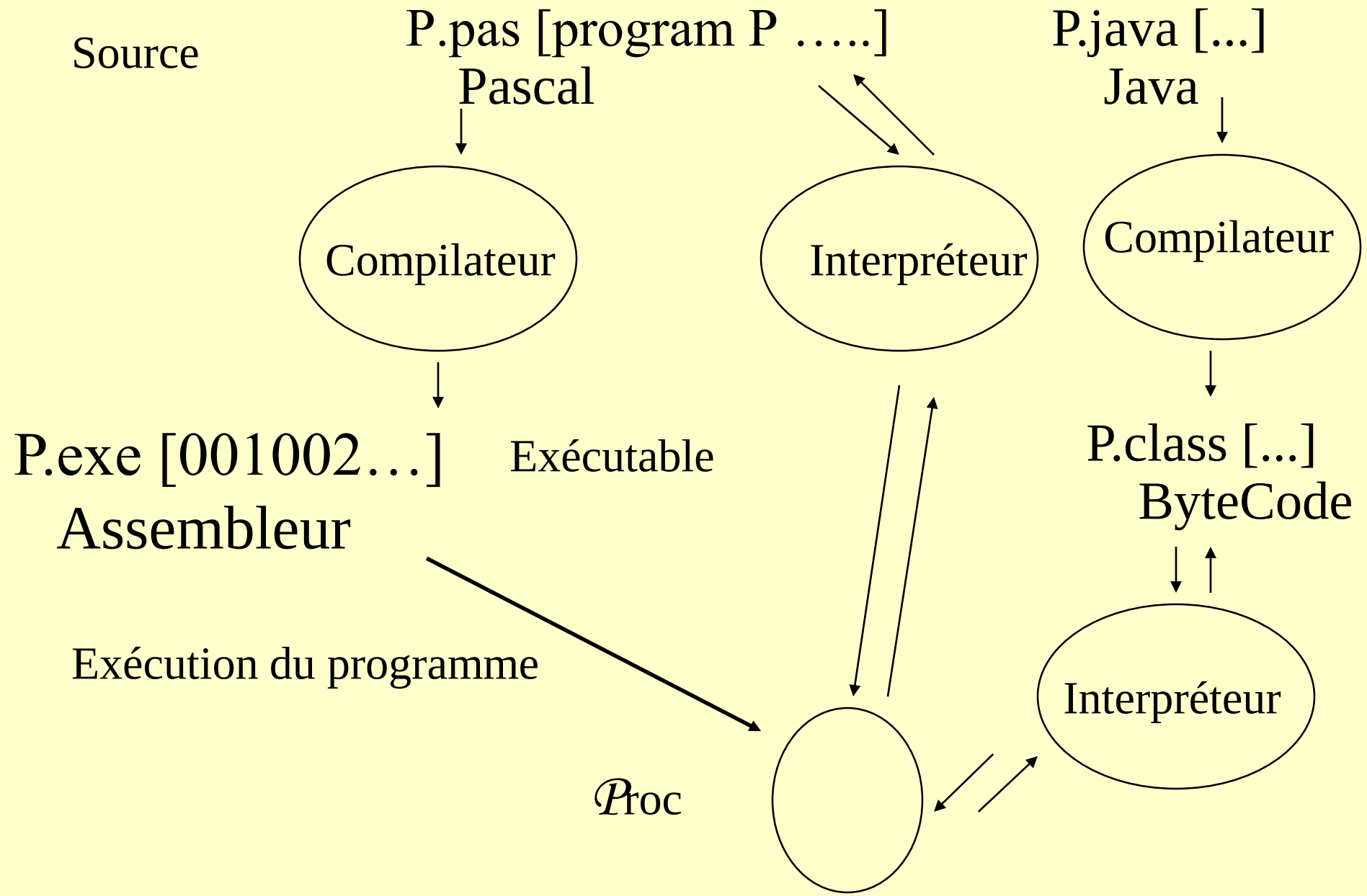
0. Introduction

Stéphane Loiseau, univ. Angers

- Qu'est ce qu'un programme ?
  - une suite d'instructions
  - la métaphore du cours de cuisine
  - Les langages classiques de programmation (dits *procéduraux*)
    - 1 programme
      - un ensemble d'instructions
      - manipulent des données
  - Langages compilés VS interprétés
  - autres langages : fonctionnel et logique
    - fonction mathématique
    - raisonnement

Rappel: Compilation/Interprétation



0. Introduction

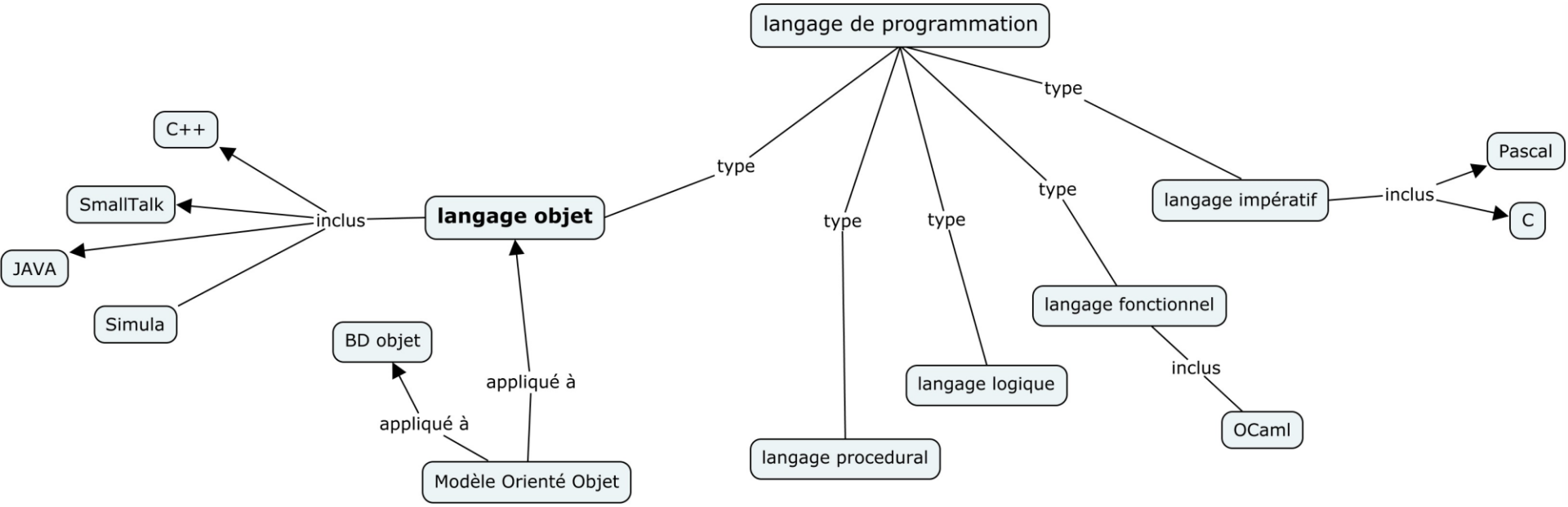
- Autre paradigme : les langages objet
  - Langage Orienté Objet LOO, Programme Orienté Objet POO
  - Programmer (ou concevoir...)
    - Regrouper données et traitement
    - Hiérarchiser
  - Exécuter un programme: des messages entre entités (=objets)

0. Introduction

- SIMULA (65-70) : **classe**
  - principe 1: on raccroche à la structure de données (champs), les codes qui peuvent les manipuler (méthode)
  - Révolution copernicienne
- SMALLTALK (70-75) : **héritage**
  - A. Kay (souris, interfaces graphiques...)
  - Principe 2 : on hiérarchise les classes
- En dehors des labos:
  - 80: SMALLTALK
  - 90: EIFFEL, LISPOBJET, PASCALOBJET, C++, JAVA
- Popularité des langages:
  http://www.tiobe.com/index.php/content/paperinfo/tpci/

0. Introduction

- le cours: programmer en objet, illustrer en JAVA: COURS, TD, TP
- Biblio:
  - conception et programmation par objets J. Ferber Hermes
  - java: la synthèse G. Clavel etc.. Intereditions
  - Les concepts objets à l'épreuve P. Prados Eyrolles
  - JAVA SE6, D. Louis, P. Muller, CampusPress 2007
  - Programmer en JAVA (9) C. Delannoy Eyrolles 2017
  - UML 2: modélisation des objets ENI 2013
  - Web…


langage de programmation
type
type
type
type
type
langage impératif
inclus
Pascal
C
langage fonctionnel
inclus
OCaml
langage logique
langage procedural
langage objet
inclus
C++
SmallTalk
JAVA
Simula
BD objet
appliqué à
appliqué à
Modèle Orienté Objet

1. Principes de la programmation objet

- La programmation classique
  - les procédures agissent sur les données

    1) on a une première idée des procédures

    2) on définit les variables (données)

    3) on écrit les procédures
  - une exécution = appels aux procédures dans un ordre, en donnant les données nécessaires

  $\Rightarrow$données et procédures sont exogènes les unes des autres

$\Rightarrow$ LOO

On pense concepts

Programmer: on raccroche à la structure de données (champs), les codes qui peuvent les manipuler (méthode)

Exécuter: une seule entité homogène: l'objet (aspects données et activations)

- # A. Les concepts de base

  La modélisation amène à concevoir les classes (codes et données) les objets sont activés par envois de messages lors de l'exécution.

  – ## 1. Classes et instances

    - Définitions:
      une *classe* est un moule (= matrice) à partir duquel sont générés les objets.
      Un objet généré à partir d'une classe C est une *instance* de C.

      Une classe a un nom (le nom du concept); elle est constituée d'un ensemble de *champs* (*attributs*, *variables d'instances*) qui décrivent la structure des objets, et d'opérations, appelées *méthodes*, qui leur sont applicables.

- *Création* d'un objet:
  - Un objet est créé par appel à un *constructeur* (`new`) appliqué à sa classe. Le résultat de la création est un objet qui peut être placé dans une variable, un pointeur sur l'objet.
  - Rappel: l'objet créé est dit *instance* de la classe qui l'a créé

- Récapitulatif

  une classe: des champs et des méthodes

  des instances: des objets

– 2. Méthodes et envois de messages

- La partie dynamique des objets est assurée par l'envoi de messages
- Instruction à un objet:
  Envoyer un message à un objet c'est lui dire le comportement qu'il doit adopter, c’est à dire la méthode qu'il doit s’appliquer.
- Définitions:
  - Une *méthode* est une description, au niveau d'une classe, d'un ensemble d'instructions. La méthode contient une liste de paramètres formels; elle peut s'appliquer aux objets instances de la classe
  - Un *message* est constitué du nom de la méthode (m) et d ’une liste de paramètres effectifs
- lorsqu'un objet O reçoit un message M, il s’applique la méthode donnée dans M; celle-ci étant définie dans la classe C de l'objet O

Rappel: tableau/liste 1

1	2	3									12
13	14 10	15 0	16 20	17	18	19	20	21	22	23	24
											36
											…

T: Array[0..4] of integer (integer= 1 octet)

T[0]: @14 ; T[n]: @14+n

Cout = 1 calcul (à peu près 0) + 1 accès

Ex: T[0]=10; T[1]=0; T[2]=20

1	2	3			6	10	@27				12
13	14 20	15 nil	16	17	18	19	20	21	22	23	24
25		0	@14								36
											…

```
Type      maillon= enregistrement
              val: integer,
              suiv: maillon
          Liste= ^maillon
```

```
Var L: Liste
```

L:@6

Coût : n accès pour nème élément

L.suiv.suiv.val=27

Rappel: tableau/liste 2

1 —	2 —	3 —	—	—				—	—	—	12 —
13 —	14	1 15	16	0 17	18	20 19	20	21	22	23	24
	—	0			—						36
											…

T: Array[0..1, 0..2] of integer

T[0,0]: @14 ; T[n,m]: @14+3*n+*m

Ex: T[0,1]=10; T[1,0]=0; T[1,2]=20

Coût = 1 calcul (à peu près 0) + 1 accès

- Tableau
  - En JAVA
    - Tableau à une dimension
      - 1) déclaration
        - `int [] T1, T2`; // tableaux T1 et T2 d'entiers
        - Equivalent à `int T1[], T2[];` // ici x est un simple entier
      - 2) création (réservation mémoire séquentielle)
        - `T1 = new int[5]`

        C'est un objet de type tableau de taille 5 (0, 1,.., 4) créé en mémoire (ne pourra pas être modifié)

        `T1` est un pointeur (pourra pointer sur un autre tableau)
      - 3) utilisation: `T1[i]`
    - Tableau de tableaux (pas tableau à n dimensions en JAVA)
      - Tableau rectangle (matrice) `int T1[][] = new int [2][3]`
      - Tableau irrégulier `int T1 [][];`

        ```
        T1 = new int [2][];
        T1[0]=new int [3];
        T1[1]=new int [2]
        ```

- Collections
  - Classe `Ensemble`

Ensemble
…

`union(_: ensemble): ensemble`
`intersection(_: ensemble): ensemble`
`cardinalité(): nbe`
`ajouteElt(_: Object)`
`enleveElt(_: Object)` …

  - Classe `Dico` (dictionnaire, tables associatives)

Dico
`at (clef: Object): object` //retourne l'élément associé à clef `atput(clef, x)` //associe à la clé clef la valeur de x

  - JAVA
    - Problème du typage, des copies et des algos (java 5 et ultérieur)
    - Ensemble: Classe `HashSet` (`TreeSet`) `addAll(_), retainAll(_), Size(), Add(_), Remove(_)`
    - Dico: Classe `HashMap` (`TreeMap`) `containsValue(_), put(_, _)`

- 3. L'héritage
  - Principe 2: on hiérarchise les classes
  - Méthode:
    - 1/ lors de la conception, on regroupe dans une classe commune les *caractéristiques* (ou *membres)*, c'est à dire les champs et méthodes, communes de classes se ressemblant
    - 2/on hiérarchise les classes avec le lien « est-une-sorte-de »
  - Définitions
    Une *sous classe* SC d'une classe C, est une classe sorte de C.
    SC possède par défaut les caractéristiques de C.
    On dit que SC est une *spécialisation* de C, et que C est une *généralisation* de SC.
  - La hiérarchie permet ainsi de factoriser l'écriture (et MAJ=Mise A Jour) de caractéristiques de plusieurs (sous) classes.
  - On dit qu'une classe (ou objet) *hérite* des caractéristiques de ses sur-classes.

- 4. Synthèse
  - a) classes et instances (objets)
  - b) classes et sous-classes
  - c) méthode et envoi de message
  - d) héritage
  - Programmer "objet" =
    - affiner les classes (classe --> sous classes)
    - hiérarchiser:
      une classe CA qui hérite de CB dispose des attributs et méthodes de CB
  - Hiérarchie des classes (classes métiers, fenêtres...)

- B. L'interprétation des messages
  - L’effet d’un message **dépend du receveur** (l'objet) et **de l'héritage** (hiérarchie).
  - un message envoyé à un objet est composé du nom de la méthode (appelé **sélecteur**) et des paramètres effectifs

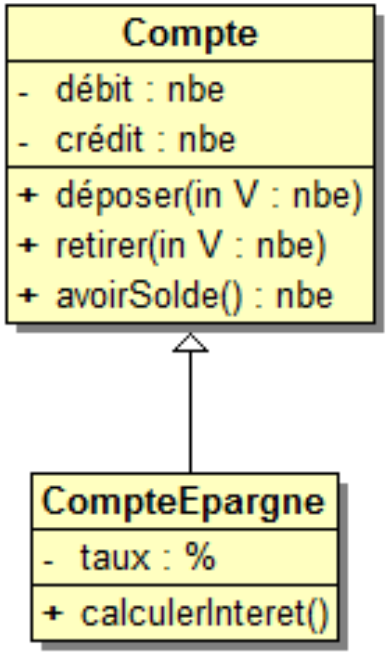
  – 1. L ‘effet dépend du receveur
  - un objet ne peut recevoir un message que si le sélecteur appartient à la classe de l'objet ou à une de ses sur-classes
  - il est possible d'envoyer deux messages avec le même sélecteur à deux objets différents qui ne sont pas de la même classe. Chaque classe implémentant sa propre méthode

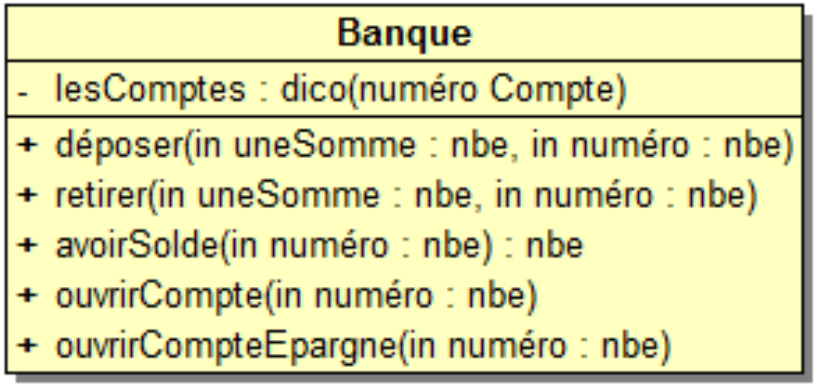
– 2. L ‘effet dépend de la hiérarchie d'héritage

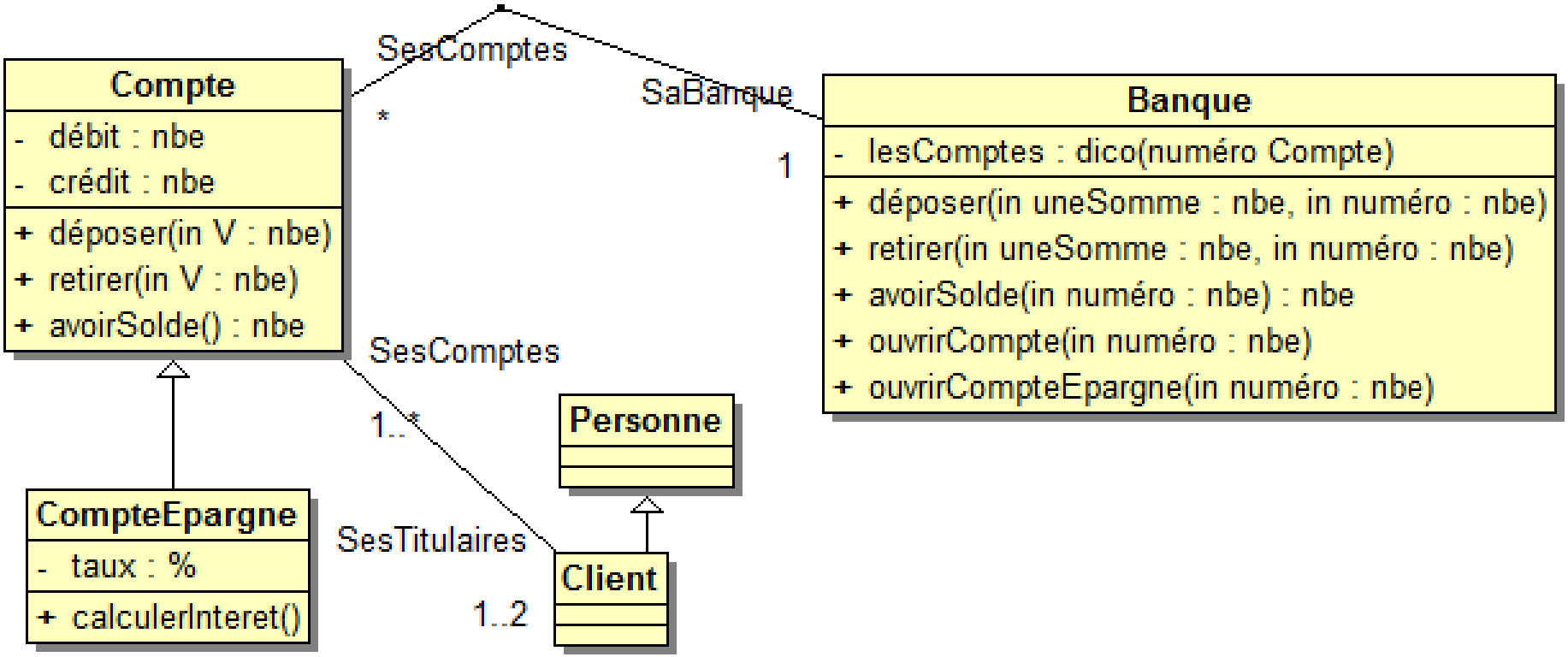
- Règle:
  Lorsqu'un objet reçoit un message de sélecteur M, il recherche d'abord la méthode de nom M dans sa classe, puis s'il ne la trouve pas, il recherche en remontant dans la hiérarchie des classes héritées.

– 3. La composition de messages
  - Définition
    Lorsqu'un message envoyé à un objet retourne un objet qui lui même reçoit un message, on dit qu'il y a *composition* de messages.

  - exemple bancaire:
    – une banque a des comptes indexés sur leur numéros


Compte
- débit : nbe
- crédit : nbe
+ déposer(in V : nbe)
+ retirer(in V : nbe)
+ avoirSolde() : nbe
CompteEpargne
- taux : %
+ calculerInteret()


Banque
- lesComptes : dico(numéro Compte)
+ déposer(in uneSomme : nbe, in numéro : nbe)
+ retirer(in uneSomme : nbe, in numéro : nbe)
+ avoirSolde(in numéro : nbe) : nbe
+ ouvrirCompte(in numéro : nbe)
+ ouvrirCompteEpargne(in numéro : nbe)


Compte
- débit : nbe
- crédit : nbe
+ déposer(in V : nbe)
+ retirer(in V : nbe)
+ avoirSolde() : nbe
SesComptes
*
SaBanque
1
Banque
- lesComptes : dico(numéro Compte)
+ déposer(in uneSomme : nbe, in numéro : nbe)
+ retirer(in uneSomme : nbe, in numéro : nbe)
+ avoirSolde(in numéro : nbe) : nbe
+ ouvrirCompte(in numéro : nbe)
+ ouvrirCompteEpargne(in numéro : nbe)
SesComptes
1..*
Personne
CompteEpargne
- taux : %
+ calculerInteret()
SesTitulaires
1..2
Client

• • •

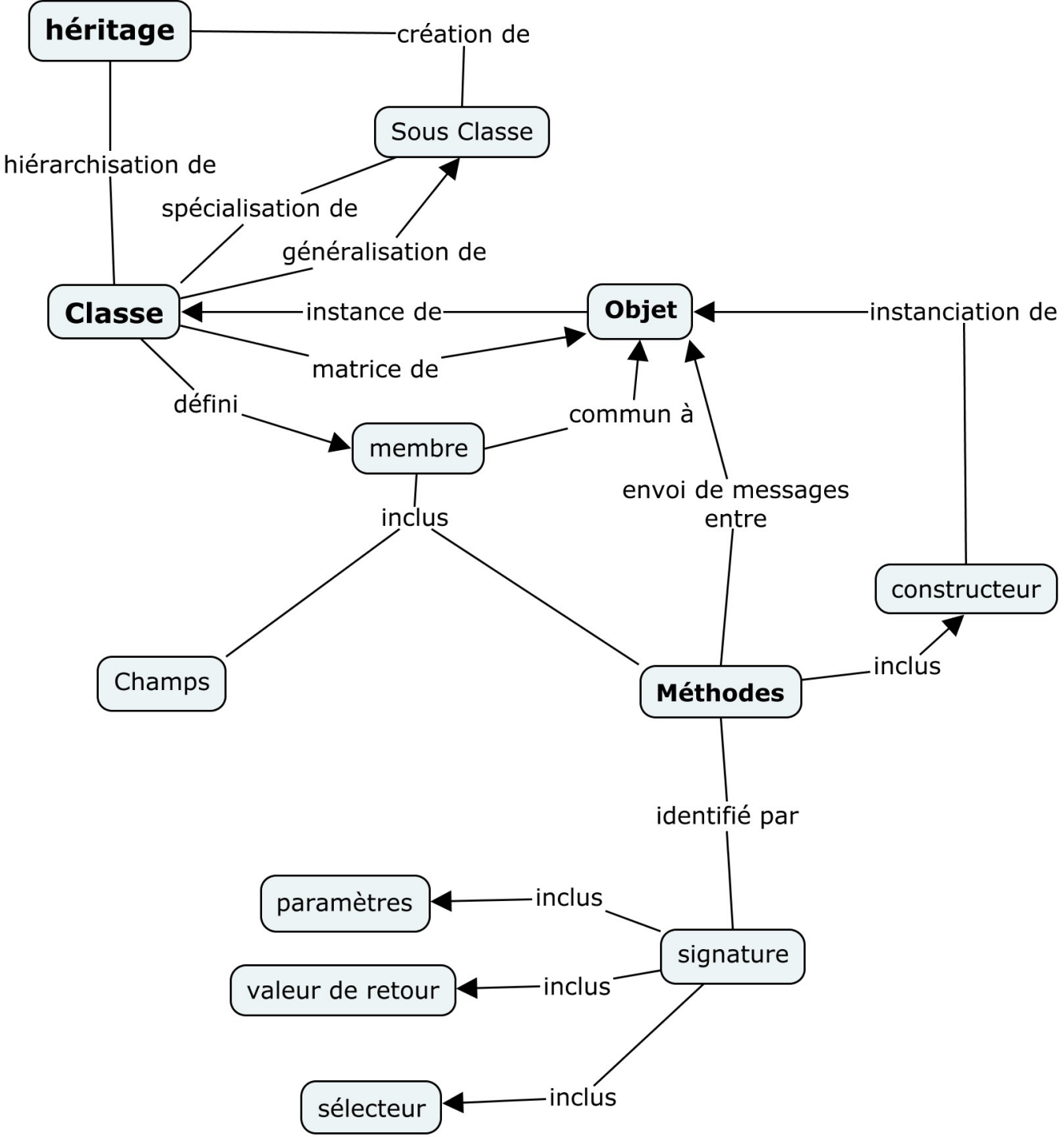
– 4. Les messages à soi-même

- les objets ont besoin de s'envoyer des messages à eux-même
  - pour appeler une autre méthode sur le même objet
- il existe une « variable » pour cela appelée
  - `self` (smalltalk)
  - `this` (C++, JAVA)

- ex bancaire: pour faire un virement

– 5. Les méthodes quasi-génériques
  - grâce a l'héritage et `self` (`this`)
  - on peut définir des méthodes attachées à une classe indépendantes de leurs réalisations dans des sous-classes

  - ex: des objets graphiques,
    – tous les objets savent s'effacer et se visualiser
    – mais ces méthodes dépendent de la nature des objets
    – Le déplacement = effacer, translaterCoordonnée, visualiser (classe générique)


héritage
création de
Sous Classe
hiérarchisation de
spécialisation de
généralisation de
Classe
instance de
Objet
instanciation de
matrice de
défini
commun à
membre
envoi de messages
entre
inclus
constructeur
Champs
Méthodes
inclus
identifié par
paramètres
inclus
signature
valeur de retour
inclus
sélecteur
inclus

Fin chapitre 1