Programmation Impérative et Objets

Licence TechWeb

N.Hameurlain

Motivation

- Difficultés d'imaginer l'état de la mémoire (programme de grande taille!).
- Le découpage en procédures/fonctions permet de comprendre le calcul réalisé par un programme mais pas de comprendre les données.
- Objet = Données + Procédures, une abstraction de la mémoire et des opérations.

L'objet

- Inventé pour unifier données et procédures!
- Il s'agit de mettre ensemble l'agrégat des données représentant une entité particulière et les procédures (appelées méthodes) qui opèrent sur ces données.
- L'état d'un objet est formé par l'ensemble des valeurs courantes de ses variables, qui sont une représentation pour l'entité modélisée par l'objet.
- Les méthodes d'un objet opèrent donc sur cet objet en modifiant son état.

Programmation par objets(1)

- S ’intéresse beaucoup plus aux données qu’au contrôle.
- Utilise la métaphore de communication par messages pour capturer les interactions entre objets.
- Caractérisée par la programmation avec des objets, messages, et hiérarchies des objets (héritage).

Programmation par Objets (2)

- La PO est une abstraction, généralisation de la Programmation impérative (PI):
  - P.I implique un état et un ensemble d ’opérations transformant l ’état;
  - P.O implique une collection d ’objets, chacun avec un état et un ensemble d ’opérations transformant l ’état.

Programmation par Objets(3)

- Programmer en Objets signifie développer des abstractions qui modélisent le problème à résoudre:
  - se focaliser sur les données, les objets, et les opérations nécessaires pour satisfaire une tâche particulière.
- Un objet peut être vu comme un *Serveur*: nécessite des opérations et (parfois) des données.

Programmation VS langages à Objets

- On peut faire de la programmation à objets dans n ’importe quel langage de programmation.
- Un langage à objets offre des mécanismes qui facilitent la programmation par objets:
  - mécanismes *d’encapsulation*,
  - de *classification*,
  - et d *’héritage*

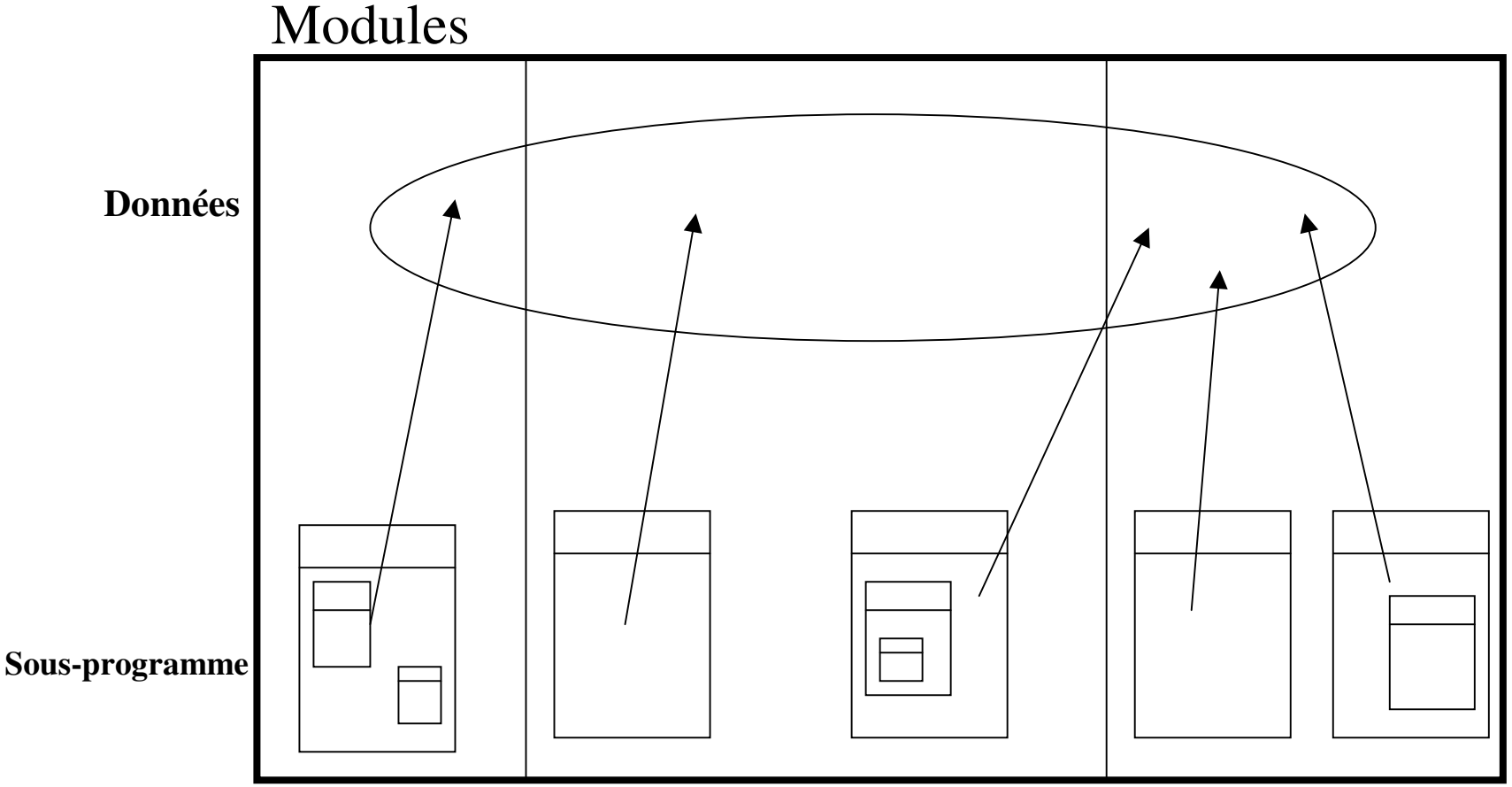
Langage à Objets (1)

- Un Objet est un mécanisme d'**encapsulation**:
  - L *'état* (constitué par les valeurs de ses variables d'instance), accessible seulement aux opérations de l 'objet, est caché depuis le monde extérieur.
- Différents objets receveurs d 'un même message peuvent répondre différemment (**polymorhisme**).
  - Exemple: Différente figures géométriques (objets) comme rectangle, triangle, cercle, .., peuvent avoir la même opération pour le tracer *afficher()*
  - L opération pour tracer un rectangle n'est pas la même que celle d'un tringle,ou celle d'un cercle.
  - Le choix de cette opération dépendra donc de la figure a tracer au moment de l'exécution!

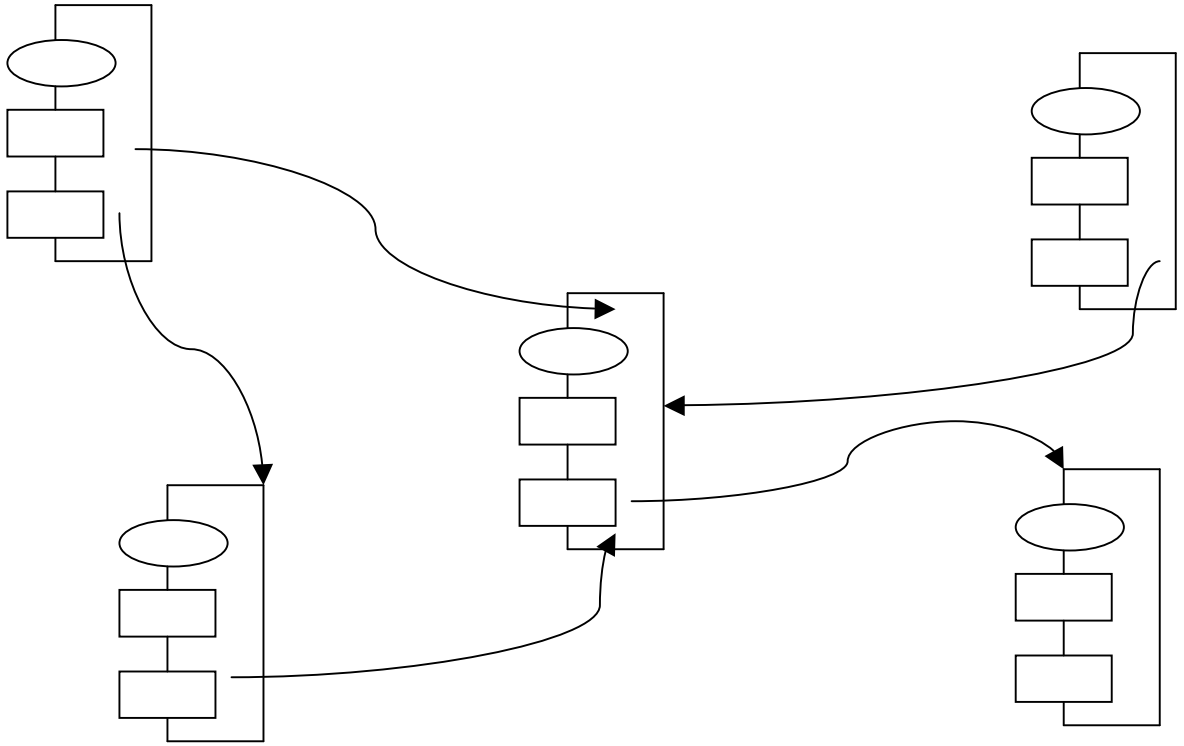
Langage à Objets (2)

- Le code à exécuter en réponse à un message est déterminé dynamiquement (**liaison différée**);
  - Exemple: Attendre l'exécution pour décider si notre objet est un rectangle, triangle, ou un cercle, et appeler l'opération *afficher()* correspondante!
- Les objets sont crées à partir d'un moule (**classe**);
- Une classe peut être construite par **héritage** d'une autre classe déjà existante.
  - La sous-classe peut hériter de la structure et du comportement de sa super-classe.

Historique: approche procédurale



Historique: approche orientée objets



Historique: Objets

- Programmation dirigée par les données et les traitements:
  - la tâche à réaliser est décomposée en un ensemble d'objets.
- Idée : diviser (en sous-programmes autonomes) pour mieux faire évoluer et réutiliser

  Objet = données + traitements

Terminologie (1)

- Objet: collection de données privées et des opérations publics;
- Classe: description d ’un ensemble d ’objets (type encapsulé: partionner en privé et public);
- Instance: une instance d ’une classe est un objet de cette classe;

Terminologie (2)

- *Méthode*: le corps d ’une procédure qui implémente une opération;
- *Message*: appel de procédure. Requête d ’exécuter une méthode;
- *Héritage*: extension d’une classe précédemment définie;
- *Sous-typage* (principe) : il peut apparaître si un objet d ’un supertype est attendu:
  - c’est un mécanisme d ’héritage qui distingue entre les types et les classes

Principe de base de l'objet (1)

Objet = état (variables) + comportement (méthodes)
+ identité (identificateur d'objet)

- Tout objet a un *identificateur unique* et permanent :
  - Sert a lui envoyer des messages.
- Un objet possède des données, *ses variables d'instance*
  - L'état d'un objet observé au cours de l'exécution du programme est constitué de l'ensemble des valeurs de ses variables d'instance
- Un objet possède un comportement ou est doté de savoir-faire: *ses méthodes*
  - Une méthode est constituée :
    - d'un nom, d'un type résultat, d'une liste de par. Formels typés et d'un corps qui explicite ce que fait cette méthode lors de son activation

Principe de base de l'objet (2)

- La seule façon de réaliser une opération sur un objet consiste à appeler l'une de ses méthodes par *envoi de message:*
  - Il faut connaître et utiliser l'identificateur de cet objet
  - Un message est constitué d'un nom de méthode et d'une liste de par. Effectifs;
- L'état d'un objet est *rémanent* entre les envois de messages.
- Les variables d'instance sont généralement privées alors que les méthodes sont généralement publiques
  - L'objet a le contrôle sur ce qu'il veut rendre privé ou public.

Création d'un objet

- Un objet peut être créé puis recevoir de messages auxquels il répond en activant les méthodes correspondantes:

Création d'un objet = allocation d'espace mémoire+ initialisation

- C'est lors de la création de l'objet que lui est attribué son identificateur d'objet unique.
  - Le résultat de cette création est cet idf d'objet, qui devient une valeur manipulée comme n'importe quelle valeur dans le langage;
  - On peut donc affecter cet identificateur d'objet à une variable, le passer en paramètre à une méthode ou encore le retourner comme résultat d'une méthode.

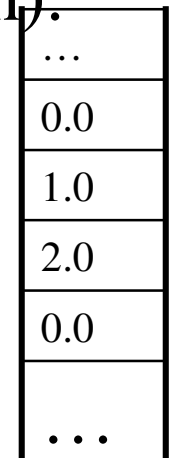
Exemple: objet rectangle

- Le cas d'un objet rectangle, avec quatre variables réelles:
  - (x1, y1)= (0, 1), le coin supérieur gauche
  - (x2, y2) = (2, 0), le coin inférieur droit
- Le système va attribuer un identificateur o1, retourné comme résultat de la création
- La forme précise de l'opération de création dépend du langage (voir le cas du C++, plus loin).

Objet rectangle

o1: rectangle
x1= 0.0
y1= 1.0
x2= 2.0
y2= 0.0

Représentation en mémoire



Utilisation d'un objet: envoi de message

- Une opération est réalisée en demandant à un objet de la faire par envoi de message.

  <receveur>.<nom-méthode> (<arg1>, <arg2>, …)

- L'objet sait répondre à cet envoi de message en exécutant le traitement approprié à l'opération qui lui est demandé.
- La résolution d'un envoi de message demande donc
  - de rechercher la méthode à appliquer en fonction de l'objet receveur,
  - puis de l'appliquer pour répondre au message:

  Envoi de message = rechercher+appliquer

Types d ’Objets

- Objets *Impératifs*: Objets ayant un état que l ’on peut mettre à jour;
- Objets *Actifs*:
  - peuvent être activés quand un message arrive comme un objet impératif;
  - mais contrairement aux objets impératifs qui sont passifs, à moins qu’ils sont activés par un message, ils peuvent avoir une activité interne.
    - Exemple: thread JAVA, tâche ADA, etc

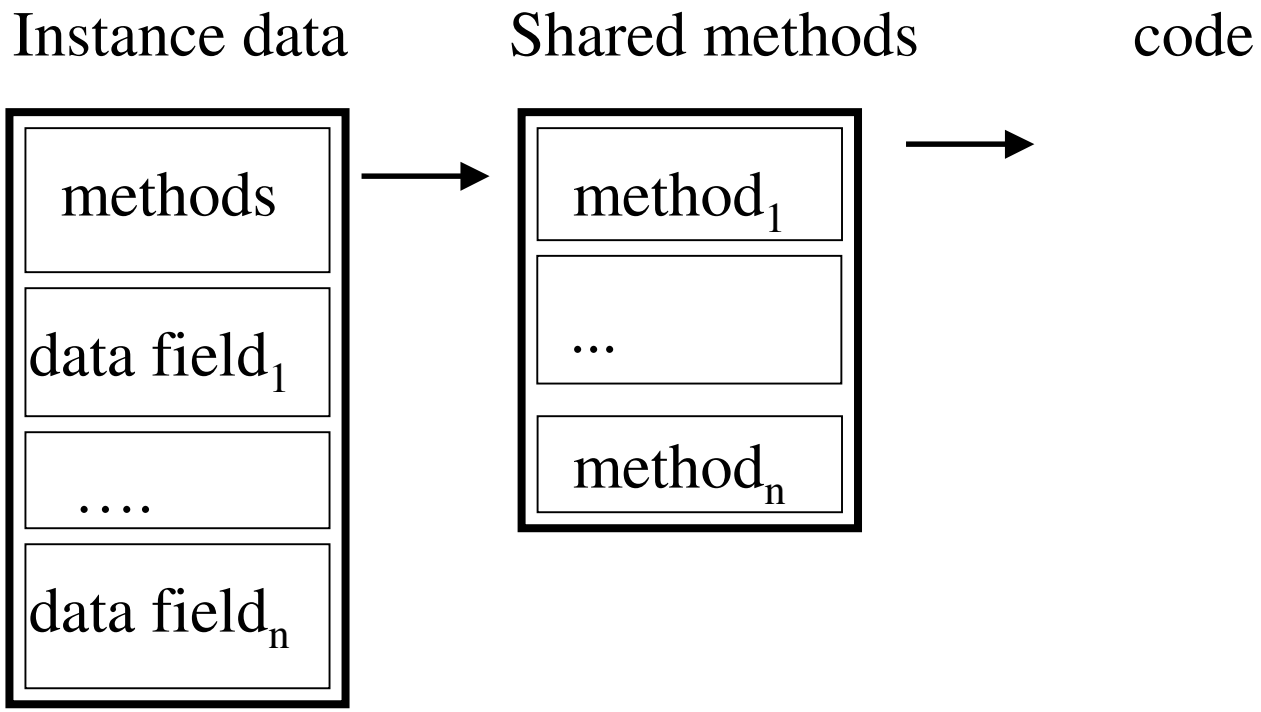
Objets Impératifs

- Sont comme les variables, ayant un état que l ’on peut mettre à jour (Simula, Smalltalk, C++);
- Ont une identité, et une collection de:
  - méthodes qui sont activées par la réception de messages depuis les autres objets;
  - Instances de variables qui sont partagées par les méthodes de l ’objet, mais inaccessibles par les autres Objets.

Objets Actifs

- Les Objets Actifs ont trois modes:
  - quand il y a rien à faire, ils dorment;
  - quand un objet est en train de s ’exécuter il est *actif*;
  - et quand un objet est en attente d ’une ressource ou l ’accomplissement des tâches il est en *attente*.
- Les messages envoyés à l ’objet doivent attendre dans une file jusqu’à ce que l ’objet finisse sa tâche:
  - L ’envoi peut être synchrone ou asynchrone.

Implémentation d'un Objet



Les classes(1)

- Les classes servent de moule à partir duquel les objets peuvent être crées.
  - Regroupe les objets ayant la même structure.
- Les classes ont les mêmes instances de variables et d ’opérations que les objets correspondant mais l ’interprétation diffère:
  - Une instance de variables dans l ’objet représente les variables actuelles,
  - alors que les instances de variables d ’une classe sont potentielles, étant instanciées à la création d ’un objet.

Les classes(2)

- On peut penser d ’une classe comme la spécification du comportement commun à tous les objets de la classe.
  - Ces objets seront dits instances de la classe qui les regroupe
  - Les objets se distinguent que par leur identité (unique), et éventuellement par les valeurs de leurs variables.
- les opérations publics d ’une classe déterminent le comportement,
- Les instances de variables spécifient une structure (de données) pour réaliser ce comportement;

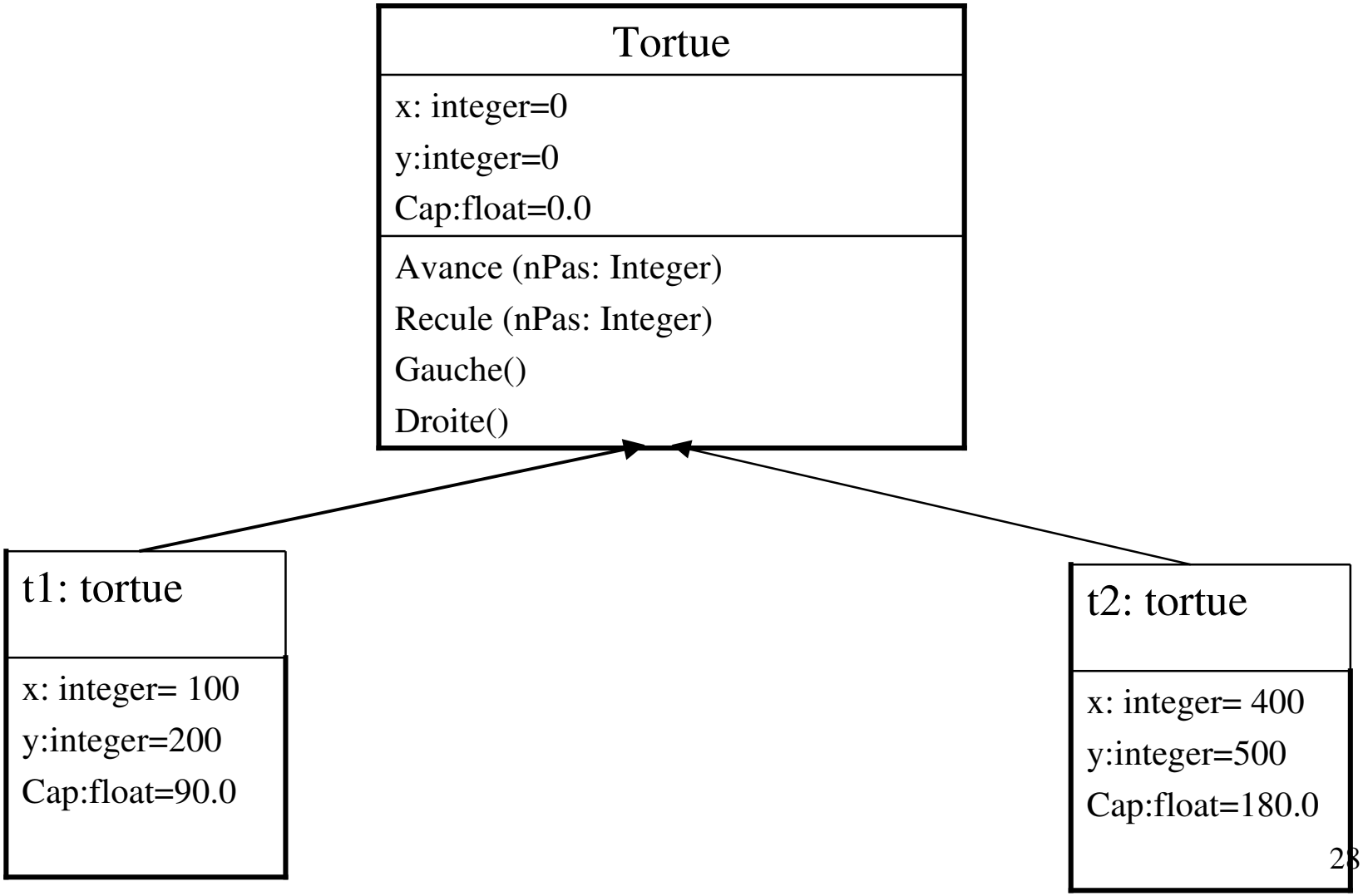
Principe des langages à classes

- Tout objet est une instance d'une classe
- Toutes les instances d'une classe possède le même nombre, les mêmes noms et les mêmes types de variables d'instances.
- Chaque objet possède ses propres valeurs pour ses variables d'instances et donc son propre état.
- Toutes les instances d'une classe ont le même savoir-faire et peuvent donc répondre aux mêmes messages.

Classe: définition

- Un nom unique qui permet de référencer le concept et d'en créer des instances
- La liste des variables, avec leurs types resp., que vont posséder tous les objets instance de cette classe.
- La liste des opérations qui peuvent être appliqués aux objets de la classe, et qui vont donc former le savoir-faire commun à toutes ses instances.
  - Cette liste constitue ce que l'on appelle l'*interface* de l'objet.
- La définition des méthodes qui mettent en œuvre les opérations d'une manière qui dépend de la représentation choisie pour les instances de la classe.

Exemple


Tortue
x: integer=0
y:integer=0
Cap:float=0.0
Avance (nPas: Integer)
Recule (nPas: Integer)
Gauche()
Droite()
t1: tortue
x: integer= 100
y:integer=200
Cap:float=90.0
t2: tortue
x: integer= 400
y:integer=500
Cap:float=180.0

Instantiation

- L'objet est l'entité dynamique:
  - Existe pendant l'exécution du prog. pour faire du calcul
- La création d'un objet à partir d'un moule, lequel est donné par sa classe est appelée *instantiation*

  new <nom-classe> (<liste-paramètres-initilisation>)
- Le résultat de l'expression ***new*** est l'identificateur d'objet de l'objet crée
- Le type de l'identificateur, manipulable comme n'importe quelle autre valeur, est la classe de laquelle l'objet a été crée.

La méthode

- Possède un nom, un type de résultat, une liste de par. formels typés et un corps qui explicite ce que fait cette méthode lors de son activation
- Propose aussi une forme d'abstraction et encapsule l'information propre à chaque opération (les variables locales, et le corps de la méthode)
- Elle peut modifier l 'état de l'objet et/ou retourner un résultat.

Partage d'informations via la classe

- Considérer la classe comme un réservoir d'informations communes à l'ensemble de ses instances
  - noms, et types de variables d'instances, l'interface et les méthodes applicables (cas du C++)
- Cette notion à amener plusieurs langages à objets (cas du JAVA) à ajouter trois nouveaux éléments à la description de classes
  - La définition de constantes et de variables communes (à l'ensemble des instances de la classe) ainsi que la définition de méthode dite de classe.

L ’héritage (1)

- Permet la réutilisation du comportement d ’une classe dans la définition de la nouvelle classe.
- La *sous-classe* hérite des opérations de la classe parente (*super classe*) et peut rajouter de nouvelles opérations et de nouvelles instances de variables.
- L ’héritage capture une forme d ’abstraction, appelée *super-abstraction*, qui est complémentaire à l ’abstraction de données.

L ’héritage (2)

- Permet la classification des classes de la même façon que les classes permettent la classification des valeurs (premier niveau).
  - Offre un pouvoir de modélisation conceptuelle et de classification (du second ordre).
- Offre la possibilité de gérer, de manipuler, et de partager le comportement (du second niveau), complémentaire à celui des classes vis à vis des objets (premier niveau).

Approches de définition de L ’héritage

- *Logique*: exprimé par un prédicat, A(x) est vraie si x $\in$ A. *B hérite de A ssi,*

$$\forall x, \text{ si } A(x) \text{ alors } B(x).$$

- *Ensembliste*: le prédicat est exprimé par une inclusion d ’ensembles. *B hérite de A ssi,*

$$\forall x, \text{ si } x \in A \text{ alors } x \in B.$$

- *Conceptuelle*: exprimée par une spécialisation.

B hérite de A ssi B « est une sorte de » A.

Relation et graphe d ’héritage(1)

- Une relation d ’héritage est préordre (réflexive et transitive), liant une classe à sa super-classe:
  - A hérite (sous-classe) de B est représenté par A $<_h$ B;
- La représentation graphique de l ’héritage forme le *graphe d ’héritage* Gh=(Racine, Nœud, Arc):
  - Racine=classe la plus générale,
  - Nœud=l ’ensemble des classes du système.
  - Arc: il existe un arc orienté A---->B ssi A hérite de B.

Relation et Graphe d ’héritage(2)

- *Simple*: une classe ne possède qu’une seule super-classe directe:
  - L ’ensemble des classes est structuré (graphe d ’héritage) sous forme d ’une arborescence.
- *Multiple*: une classe possède plusieurs super-classes directes:
  - le graphe d ’héritage forme un graphe orienté sans circuit (cas C++).

Les Types et les Classes(1)

- Le concept d ’un type et celui d ’une classe sont liés et peuvent ne pas être distingués:
  - Le typage est utilisé pour la vérification de type;
  - les classes sont utilisés pour gérer et générer des objets ayant des propriétés et des comportements uniformes.
- Toute classe est un type alors qu’un type n ’est pas forcement une classe, puisque les prédicats ne déterminent pas nécessairement des moules d ’un objet.

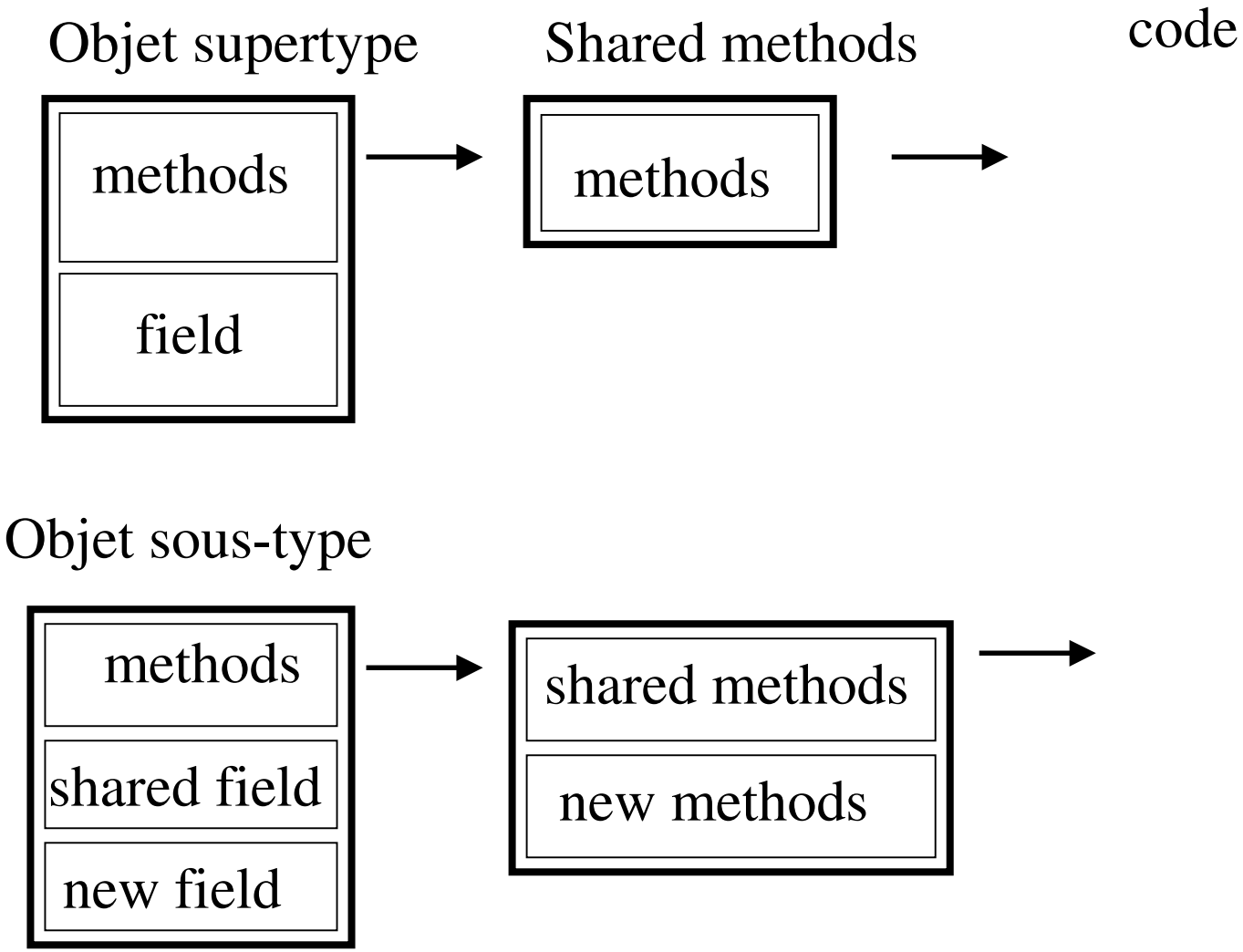
Les Types et les Classes(2)

- La distinction entre les types et les classes peut être détectée quand on compare la relation d ’héritage entre types et sous-types avec la relation d ’héritage entre classes et sous-classes.
- L ’ensemble des valeurs satisfaisant un sous-type est un sous-ensemble de l ’ensemble des valeurs satisfaisant le type.

Sous-types et Sous-classes

- *Exemple1*: les entiers naturels sont un sous-type des réels, mais quand la soustraction est définie pour toutes les paires des réels, elle n ’est pas définies pour toutes les paires des entiers naturels.
- *Exemple 2*: les réels sont une sous-classe des entiers naturels, puisque l ’opération de soustraction sur les entiers naturels peut être étendue à celle des réels.

Implémentation de l ’héritage



Héritage Vs Composition

- L ’héritage multiple permet de définir de nouveaux objets par fusion de la structure et du comportement de plusieurs autres et non par composition.
- La composition d ’objets peut toujours être réalisée par les variables d’instance.
- Exemple:

  Automobile=Moteur+roue+Carosserie, +...