

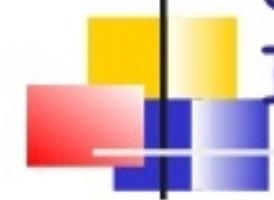
Introduction à la programmation objet

Application au langage Ruby

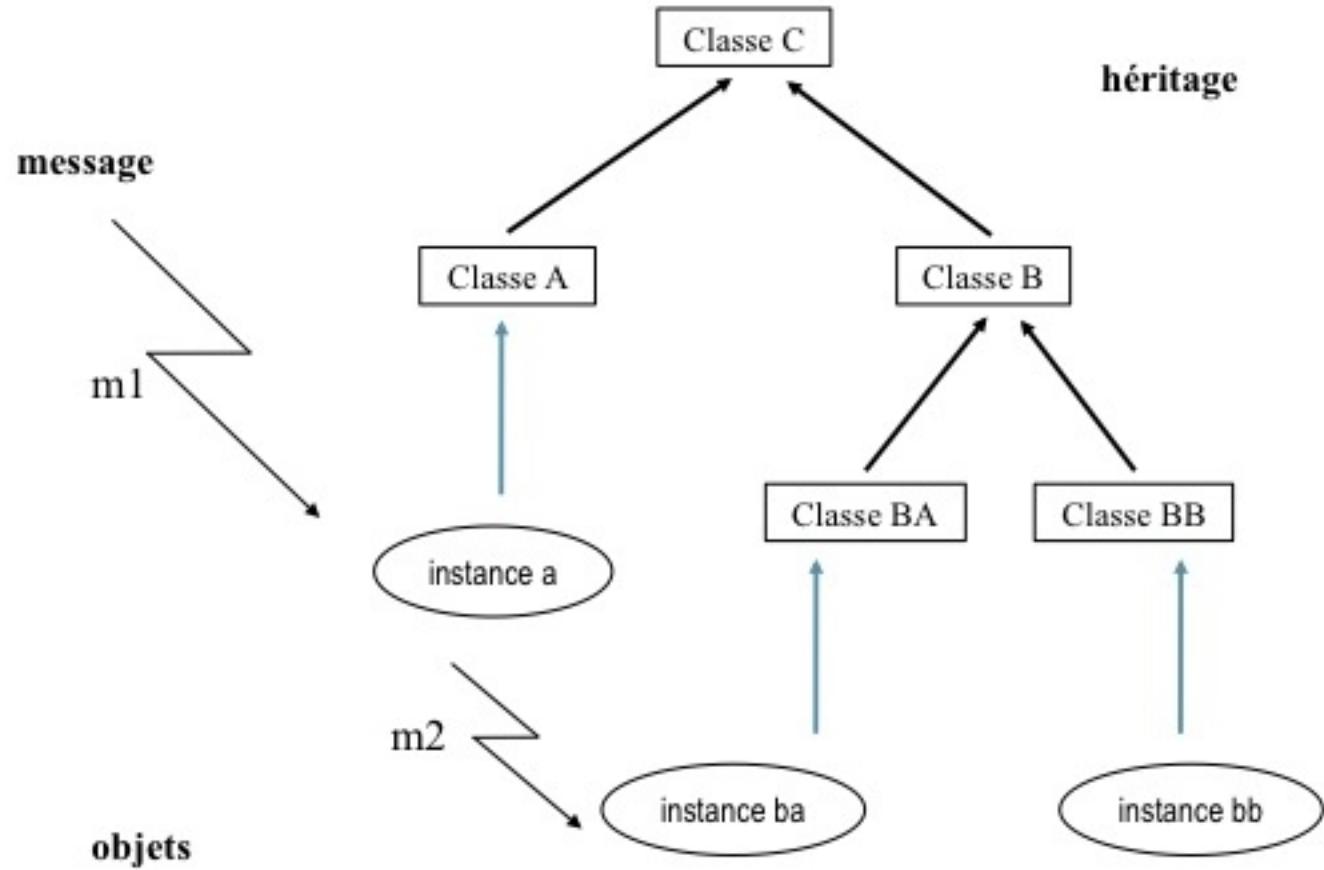
Pierre Jacoboni

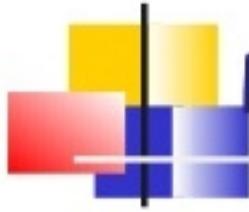
LIUM - Université du Maine

2007



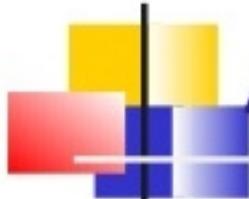
Chapitre 1 : Introduction au monde des Objets





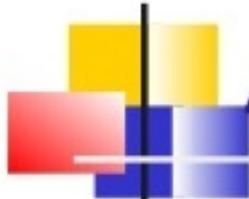
Plan

- A - Présentation
 - 1. Motivations
 - 2. Objectifs de cet enseignement
 - 3. Organisation
 - 4. Bibliographie
- B - Concepts et principes base de la POO
 - 1. Objets et messages
 - 2. Classes et instances
 - 3. Héritage
- C - Résumé des termes employés



A1-Pourquoi Ruby en L3 (1)

- Raison 1 / concepts et pratiques de prog.
 - Lang typés (Caml, Pascal, C++)
lang non typés (Lisp, Smalltalk, Ruby Python)
 - POO :
 - C++ : le plus répandu des LOO
utilisé souvent comme un meilleur C, très peu de POO
 - en Ruby : "le tout objet" empêche de "tricher"
=> obligation de programmer objet
 - comparaison C++/Ruby permet une meilleure compréhension des concepts
 - Ruby est en pleine expansion (en passe de détrôner Python au Japon)
 - utilisation simple de bibliothèques



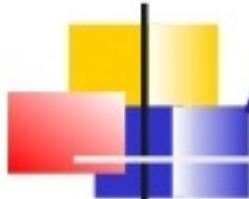
A1-Pourquoi Ruby en L3 (2)

- Raison 2

- Prototypage rapide
- Facilité d'apprentissage
 - Syntaxe proche mais pas trop de C/C++
- Up to date *
- Interface Graphique
- Base de données / réseau

- Raison 3

- initiation à la conception d'interface utilisateur
- TCL/Tk, GTK et FOX Gui



A1-Pourquoi Ruby en L3 (2)

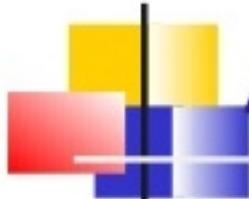
- Raison 2

- Prototypage rapide
- Facilité d'apprentissage

Ruby est aujourd'hui « quasiment » inconnu en France où on commence seulement à s'intéresser à Python alors que celui-ci commence à être supplanté par Ruby au Japon.

A vous de le faire découvrir dans les entreprises.

- initiation à la conception d'interface utilisateur
- TCL/Tk, GTK et FOX Gui



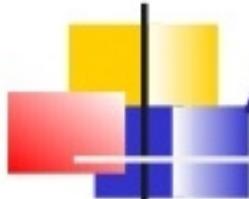
A2-Objectifs de cet enseignement

- Programmation :

- compréhension des concepts de la POO
- initiation à la pratique de la POO
sur des exemples simples savoir concevoir et mettre en œuvre des classes, des hiérarchies de classes
- utilisation de bibliothèques standards
- (faciliter l'apprentissage du C++)

- Ruby : initiation

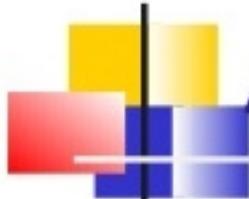
- comprendre la philosophie, maîtriser la syntaxe / prendre en main l'environnement



A2-Objectifs de cet enseignement

- Programmation :

- Ruby est plus petit que Smalltalk,)
i Plus facile à apprendre
s Smalltalk est un "petit langage" voir et mettre
e mais un gros système : de classes
- U 9 mois pour être opérationnel
- C 18 mois pour être expert
- R (entre 3 et 5 ans en C++) syntaxe /
F



A2-Objectifs de cet enseignement

- Programmation :

- Ruby est plus petit que Smalltalk,

- Ruby est petit...
... mais costaud !

- Tout ce que vous pouvez faire avec

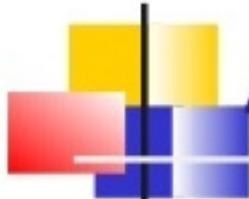
- Un autre langage peut etre fait avec

- Ruby

- Plus vite, plus facilement.

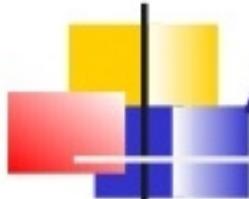
voir et mettre
de classes

syntaxe /



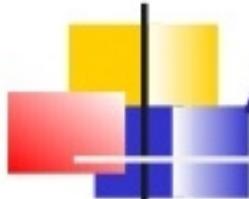
A3-Organisation

- **Chapitres 1 et 2 :**
 - Introduction au cours, au monde des objets et à Ruby
- **Chapitre 3 :**
 - Héritage
- **Chapitre 4 :**
 - les Collections, les itérateurs
 - Les modules, les mixins
- **Chapitre 5 :**
 - Les Exceptions, Les Threads
 - Les Interfaces graphiques avec Ruby



A4-Bibliographie

- Clavel-Veillon : orienté introduction à la POO
 - très bien pour débuter en POO, bonne présentation des principes de base
 - même genre de livre en C++ ce qui permet de comparer
- Ferber : lecture obligatoire pour débuter
 - 100 pages pour avoir des représentations simples et claires sur la COO et la POO
 - des principes à la réalisation informatique... il faut résoudre des problèmes techniques



A4-Bibliographie

Chantal Viallet - cours d'introduction à l'OOD

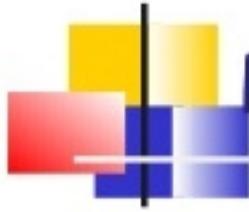
Pour ceux qui acceptent de lire en anglais

(c'est un effort qui paye, tous les informaticiens sont obligés de lire en anglais, autant s'y mettre tout de suite)

- **A quick trip to ObjectLand**

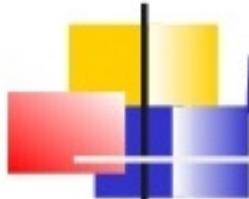
un petit livre sympa pour débuter, facile à lire et rigolo pour comprendre les principes de base et la syntaxe de base (Smalltalk) c'est exactement ce qu'il vous faut pour ce module.

- des principes à la réalisation informatique... il faut résoudre des problèmes techniques



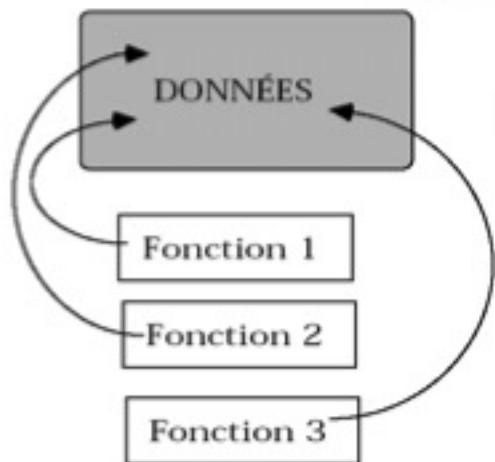
Plan

- A - Présentation
 - 1. Motivations
 - 2. Objectifs de cet enseignement
 - 3. Organisation
 - 4. Bibliographie
- B - Concepts et principes base de la POO
 - 1. Objets et messages
 - 2. Classes et instances
 - 3. Héritage
- C - Résumé des termes employés



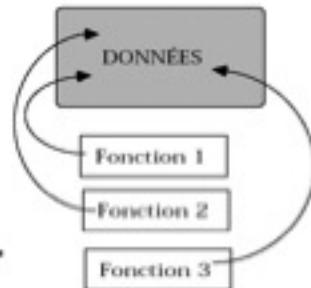
B-Programmation Procédurale

- Un programme = suite d'instructions exécutées par une machine.
 - Exécution = les instructions agissent **sur les données**.
 - Les fonctions et procédures travaillent "**à distance**" **sur les données**.

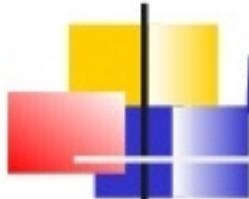


- Accent mis sur les actions. Il s'agit de répondre à la question: **Que veut on faire ?**
- dissociation entre données et fonctions => problème lorsqu'on change les structures de données.

B-Programmation Procédurale

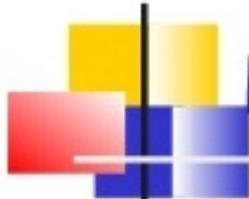


- Les procédures s'appellent entre elles et peuvent modifier les mêmes données
 - => problème lorsqu'on veut modifier une procédure : comment / Ou est -elle appelée ?
- Finalement conception plat de spaghetti dans les appels de procédures.
 - Il faut "responsabiliser" les portions de programmes
- D'où une autre vision de la programmation

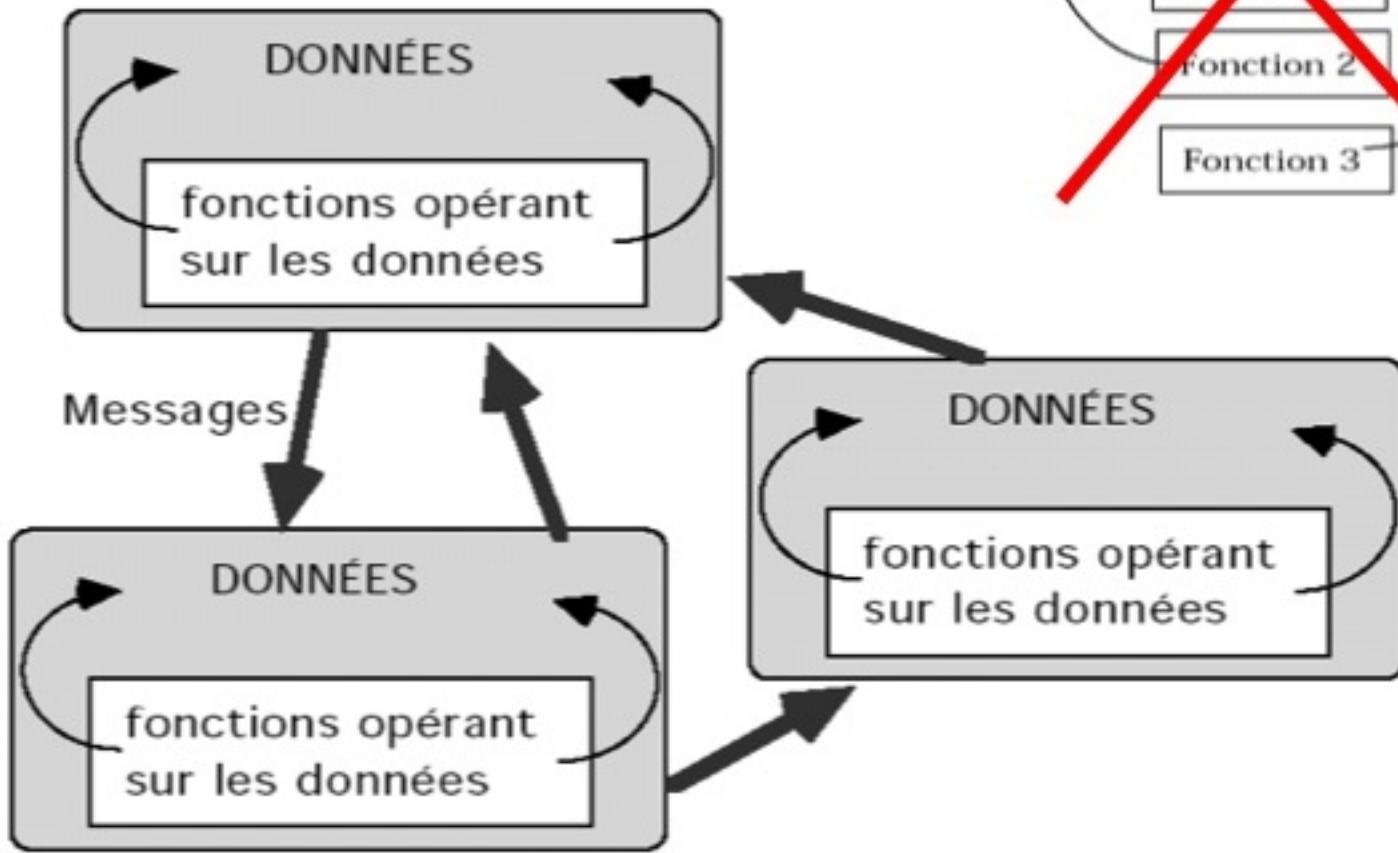


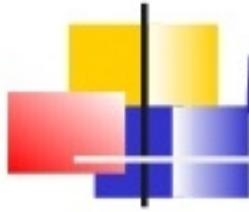
B-Programmation Objet

- Un programme = une société d'entités
- Son exécution : **les entités collaborent pour résoudre le problème final en s'envoyant des messages.**
 - une entité = un objet qui prend en compte sa propre gestion (objet responsable)
 - liaison inévitable entre données et procédures opérant sur ces données.



B-Programmation Objet





B-POO : Les 5 concepts de base



les objets



Les classes et l'instanciation

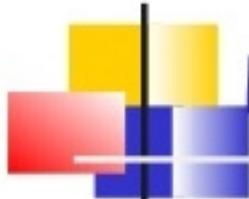
L'envoi de message



L'héritage

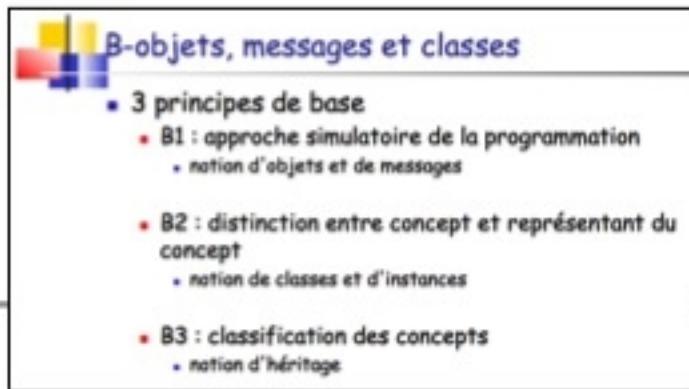
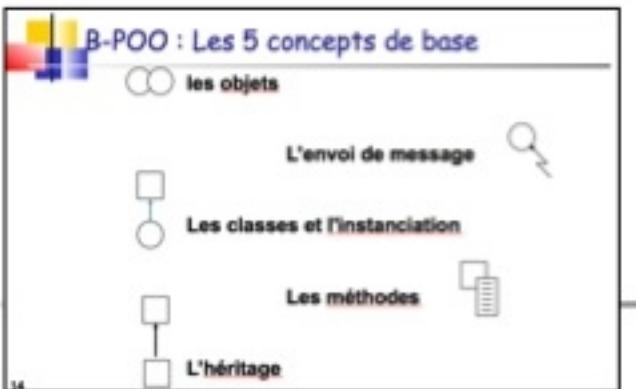
Les méthodes



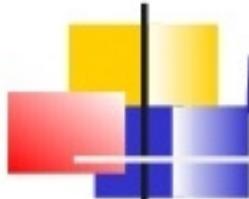


B-objets, messages et classes

- 3 principes de base
 - B1 : approche simulatoire de la programmation
 - notion d'objets et de messages
 - B2 : distinction entre concept et représentant du concept
 - notion de classes et d'instances
 - B3 : classification des concepts
 - notion d'héritage

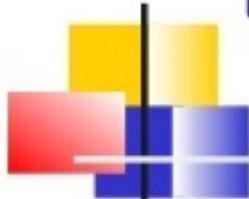


- Uniformité et simplicité des principes de base
 - Tous les LOO sont fondés sur ces Concepts et Principes
- Diversité des langages et systèmes
 - Les LOO peuvent diverger selon leur orientation : Génie Logiciel, IA, BD, Systèmes répartis, Interaction H/M
 - Les points de divergence essentiels concernent
 - la vérification de type (et donc la générativité, les métaclasses) utile en génie logiciel, gênante pour le prototypage, la conception
 - la gestion de la mémoire (présence ou non d'un glaneur de cellules, garbage collector dit encore ramasse-miettes)
 - les attributs actifs (complexes): utiles en représentation des connaissances



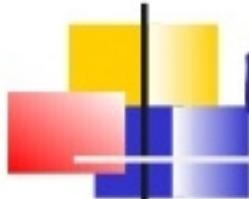
B-objets, messages et classes

- 3 principes de base
 - ➡ ■ B1 : approche simulatoire de la programmation
 - notion d'objets et de messages
 - B2 : distinction entre concept et représentant du concept
 - notion de classes et d'instances
 - B3 : classification des concepts
 - notion d'héritage



B1-P1 : approche simulative, notion d'objet, de message

- Idée de base : raisonner sur les objets (pas sur les traitements)
- Fonctionnement d'un programme en POO :
 - animation d'un modèle réduit d'objets qui réagissent à nos actions et/ou interagissent entre eux
- Activité de POO : activité de simulation
 - on programme en envoyant des messages à des objets qui réagissent à ces messages
- On pense objet et envoi de message
 - pas fonction ou procédure



Exemple 1 : empiler

- Avec un langage procédural :
 - on appelle une procédure à qui l'on passe 2 paramètres la pile et le truc à empiler. La procédure empiler a pour rôle de manipuler la pile
 - empiler(unePile, unTruc)
- Avec un LOO :
 - on envoie à unePile le message "empile" avec en paramètre unTruc.
 - C'est la pile qui est responsable de la façon de réaliser l'action. La pile ne se laisse pas manipuler par une fonction globale. Il faut lui demander (poliment) d'avoir l'obligance d'empiler quelque chose

Exemple 1 : empiler

- Avec un langage procédural:
 - on appelle une procédure avec en paramètres la pile et ce qu'il faut empiler a pour rôle de manipuler la pile
 - empiler(unePile, unTruc)
- Avec un LOO :
 - on envoie à unePile le message "empile" avec en paramètre unTruc.
 - C'est la pile qui est responsable de la façon de réaliser l'action. La pile ne se laisse pas manipuler par une fonction globale. Il faut lui demander (poliment) d'avoir l'obligeance d'empiler quelque chose

- en Smalltalk :

unePile empile: unTruc

- en C++ :

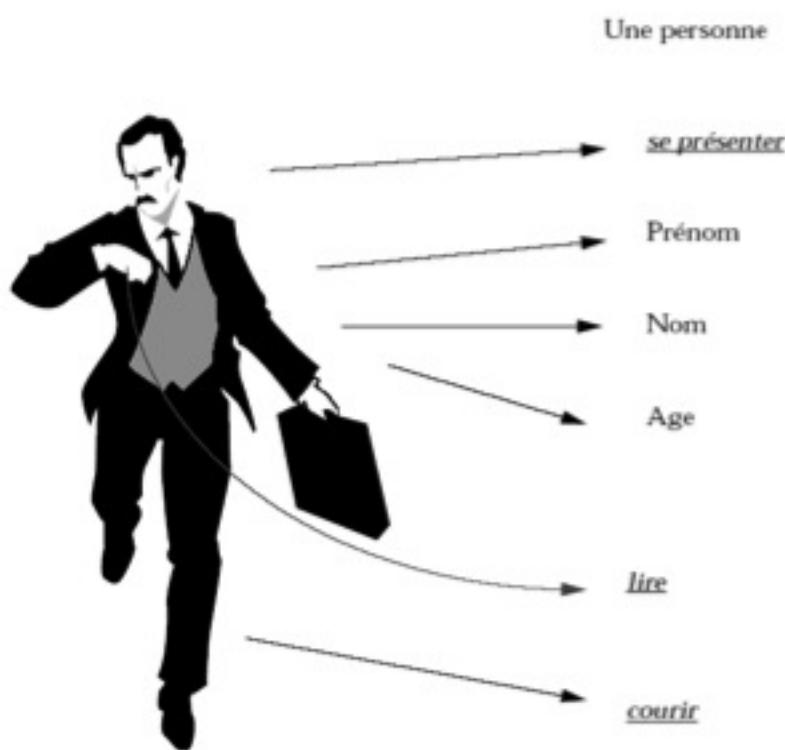
unePile.empile(unTruc)

- en Ruby :

unePile.empile(unTruc)



Présentation du concept Objet



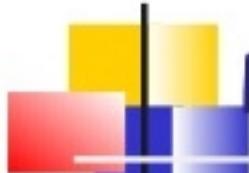
Ce dessin représente une personne.

- Une personne peut se présenter en énonçant son prénom, son nom, son âge, etc.
- Elle peut lire, courir, etc.

Modularisation

C'est le développement logiciel en petits modules, appelés objets.

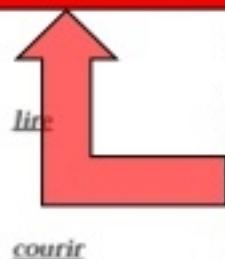
Les objets regroupent des structures de données et les opérations autorisées sur ces données.



Présentation du concept Objet



Tiens !!
ça ressemble à ce
que l'on disait
pour les TDA

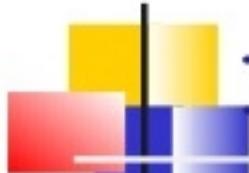


C'est le développement logiciel en petits modules, appelés objets.
Les objets regroupent des structures de données et les opérations autorisées sur ces données.

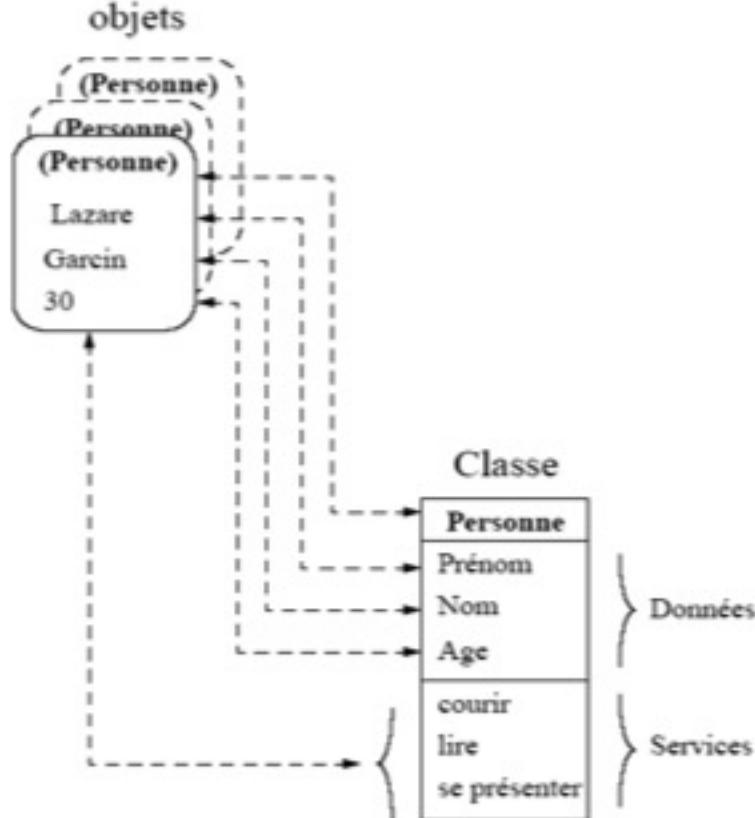
de personne.

se présenter en
i, son nom, son

r, etc.



Identité et Classification

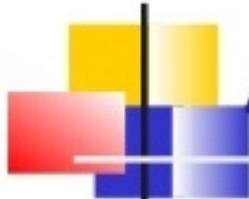


Identité

L'*identité* signifie que les données sont regroupées en entités discrètes et identifiables appelées objets.

Classification

Les objets ayant la même structure de données et les mêmes services sont groupés dans une *Classe*.



Abstraction et Encapsulation

Classe = Type abstrait de données

Personne
Prénom
Nom
Age
courir
lire
se présenter

}

Données

}

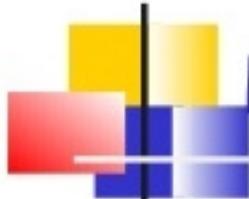
Services

Abstraction

L'Abstraction est une vue générique d'une entité permettant de ne pas être limité par des contraintes d'ordre technique ou matériel.

Encapsulation

L'objet est indépendant parce qu'il peut gérer lui même ses données internes.
Ceci implique que l'objet soit seul à pouvoir accéder à ces données vitales.

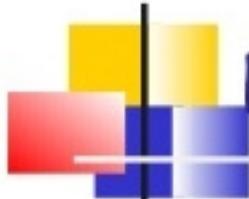


B1-Métaphore du "monde enchanté"

- Dans un système à objets tout traitement se déroule pas-à-pas par envoi de messages d'un objet à un autre
 - sans contrôle global
- Plus un grand manitou de prg principal qui contrôle tout le traitement, mais
 - des objets responsables qui savent
 - ce qu'ils ont à faire et
 - qui doit effectuer la tâche suivante

EXEMPLE de monde enchanté : les soucoupes et théières de la Belle et La Bête

"Un système à objets peut être comparé à une organisation composée d'agents capables d'effectuer chacun un certain type de tâches (en général simples). Pour exécuter un travail on demande à un agent de commencer le traitement. Cet agent après avoir exécuté sa tâche envoie un message à un autre agent qu'il soit capable d'effectuer la tâche suivante. Et ce jusqu'à ce qu'un agent exécute la tâche finale" (Hewitt 1977)



Exemple : Ouvrir une Porte

■ En Procédural

ouvrir(*maPorte*)

 si *maPorte* est de type un seul battant alors

 ouvrirPorteUnSeulBattant (*maPorte*) sinon

 si *maPorte* est de type un deux battants alors

 ouvrirPorteDeuxBattants (*maPorte*) sinon

 si ...

fin

■ En POO

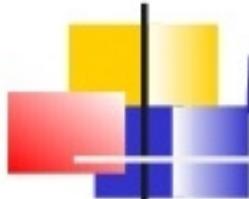
maPorte ouvreToi

En Procédural

L'utilisateur doit connaître le code à exécuter pour tous les types de portes. Source d'erreur, problèmes de maintenance : il faut tripotouiller un peu partout pour ajouter un type de portes

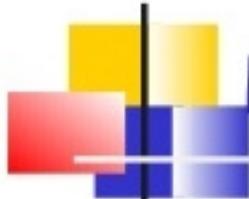
en POO : C'est *maPorte* qui est responsable de sa procédure d'ouverture (en fait nous verrons que c'est un type de porte qui a cette responsabilité et pas une porte individuelle). C'est le concepteur de la porte qui sait comment l'ouvrir. L'utilisateur (comme Ali Baba) doit uniquement se souvenir du message d'ouverture pas du code à déclencher.

Moins d'erreur, maintenance plus facile, conception plus claire...



B1-Métaphore du "monde enchanté"

- Ce style de programmation
 - nécessite une véritable conversion intellectuelle pour les programmeurs "classiques" (procéduraux)
 - facilite la maintenance et diminue la complexité des programmes
- Retenir : **principe d'autonomie des objets :**
 - Les objets sont **responsables** des actions qu'ils savent effectuer.
 - Un même message peut déclencher des traitements différents suivant la nature de l'objet receveur (**polymorphisme**)

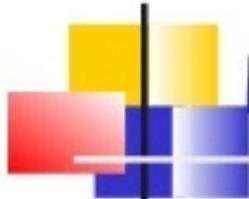


B1-Un Autre exemple simpliste

- Afficher une liste chaînée
 - en procédural de l'extérieur
 - pour i allant de 1 à la fin de la liste répéter
 - afficher le contenu de la cellule
 - en POO , deux temps
 - modélisation : chaque cellule
 - connaît son contenu et connaît sa suivante
 - sait répondre au message `afficheToi` (elle affiche son contenu et envoie le message `afficheToi` à sa suivante)
 - activation :
 - de l'extérieur on envoie le message `afficheToi` à la première cellule, celle-ci affiche sa valeur et si elle a une suivante lui envoie le message `afficheToi` et ainsi de suite jusqu'à la dernière cellule

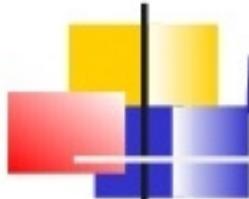
- 1) **itération/délégation** : au lieu de manipuler la liste de loin (itération) on délègue les responsabilités aux objets (aux cellules)
 - le contrôle est aux objets, pas au programme principal
 - mécanisme intellectuel de délégation beaucoup plus "naturel" que l'itération
- 2) **La complexité** : sur cet exemple simpliste on diminue la complexité puisque le client n'a plus à connaître la fin de la liste, ni à la tester. La dernière cellule sait qu'elle est la dernière.
- 3) **L'efficacité** : cette cascade d'envois de messages est-elle moins efficace qu'une itération ?
 - Sur cet exemple simpliste oui.
 - Sur un gros programme non si l'on compte
 - le temps de conception, mise au point, maintenance
 - si l'on applique la célèbre règle des 90-10

- en POO , deux temps
 - modélisation : chaque cellule
 - connaît son contenu et connaît sa suivante
 - sait répondre au message `afficheToi` (elle affiche son contenu et envoie le message `afficheToi` à sa suivante)
 - activation :
 - de l'extérieur on envoie le message `afficheToi` à la première cellule, celle-ci affiche sa valeur et si elle a une suivante lui envoie le message `afficheToi` et ainsi de suite jusqu'à la dernière cellule



B1-Réalisation : objets informatiques

- Qu'est-ce qu'un objet informatique ?
 - un objet réel est une entité permanente qui possède une identité
 - un objet est défini par un terme de
 - structure (l'objet est fait comme ça) : un ensemble de champs dont les valeurs
 - peuvent être des objets
 - définissent l'état de l'objet (état) qui peut varier au cours du temps
 - comportement (l'objet sait faire ça) : un ensemble d'actions
 - que l'objet est capable d'exécuter à la demande
 - qui dépendent de l'état de l'objet au moment de la demande



B1-Réalisation : objets informatiques

- Qu'est-ce qu'un objet informatique ?
 - un objet réel est une entité permanente qui possède une identité
 - un objet est défini un terme de

après ces réflexions philosophiques, passons à la réalisation informatique

1/ entité permanente

une identité = une adresse (un pointeur sur une zone mémoire ou une référence à une zone mémoire)

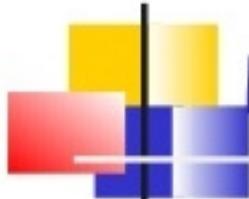
2/ état :

enregistrement (ou une struct) ie un ensemble de champs

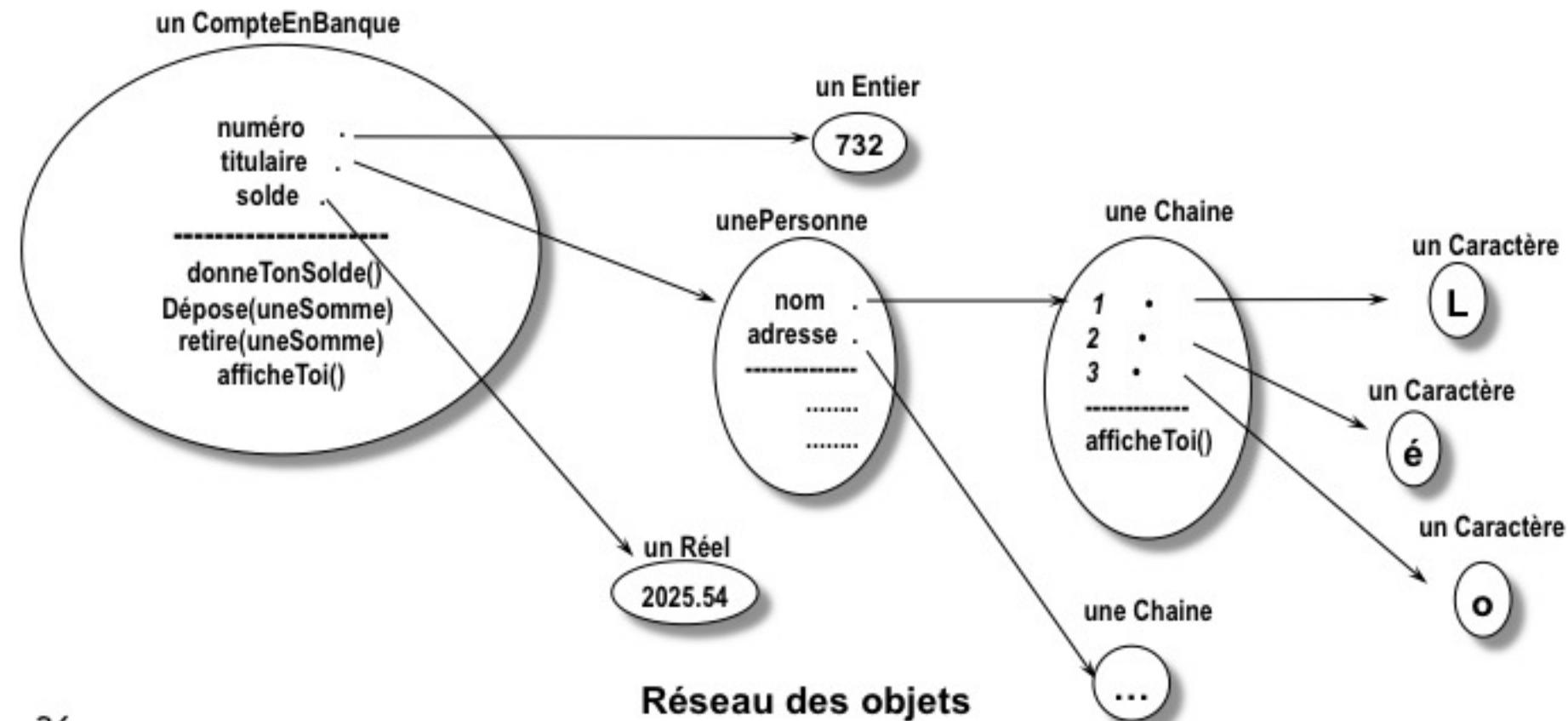
- chaque champ est connu par son nom
- chaque champ contient une valeur (en général un autre objet)
- l'adresse de l'objet (son identité) est l'adresse de la zone mémoire où est mémorisée cet enregistrement
- c'est la mémoire privée de l'objet

3/ comportements :

une batterie de procédures (les méthodes) que l'objet exécute quand il reçoit un message

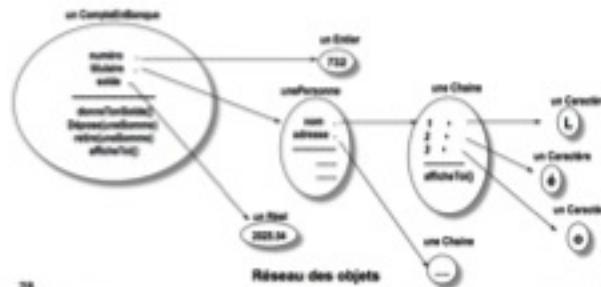


B1- Réalisation : objets informatiques



1) distinguer

- un objet (ie une adresse)
 - ** unCompteEnBanque
- et son état défini par ses attributs (les valeurs de ses champs)
 - ** Ici, l'état de l'objet : 732, unePersonne, 2025.54



732 est un attribut de l'objet unCompteEnBanque ; c'est la valeur de son champ numéro.

2) donner vie à un objet : lui permettre de réagir quand on le titille

- on s'adresse à un objet en lui envoyant un message

unCompteEnBanque.donneTonSolde()

unCompteEnBanque.depose(2000)

- l'objet réagit en exécutant la procédure associée (la méthode) qui dépend de l'état de l'objet

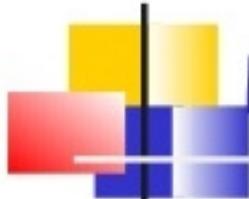
En Smalltalk

```
donneTonSolde      depose: uneSomme  
^solde           solde := solde + uneSomme
```

En Ruby

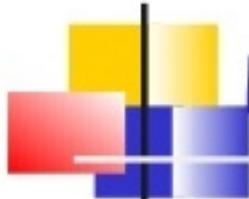
```
def donneTonSolde()      def depose(uneSomme)  
    return @solde          @solde -= uneSomme  
end                      end
```

solde est une variable d'état de l'objet qui reçoit le message (en Ruby/Smalltalk on dit une variable d'instance)



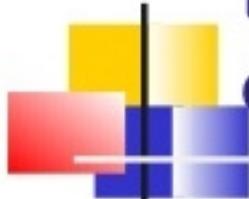
B1-Réalisation : objets informatiques

monde réel	modélisation	réalisation informatique
un objet réel (concret)	un objet qui possède une identité	une adresse unique
<ul style="list-style-type: none">• un état (aspect statique, l'objet est comme ça)	<ul style="list-style-type: none">un ensemble de champs• connus par leur nom• contenant une valeur	enregistrement du nom des champs
<ul style="list-style-type: none">• comportement (aspect dynamique, l'objet sait faire ça)	<ul style="list-style-type: none">un ensemble de messages que l'objet sait traiter en déclenchant l'exécution de procédures	textes définissant les procédures et faisant référence aux champs de l'objet



B-objets, messages et classes

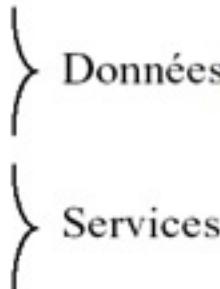
- 3 principes de base
 - B1 : approche similaire de la programmation
 - notion d'objets et de messages
 - ■ B2 : distinction entre concept et représentant du concept
 - notion de classes et d'instances
 - B3 : classification des concepts
 - notion d'héritage



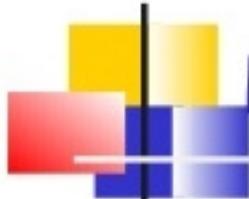
B2-Distinction entre concept (abstrait) et objet (concret)

- L'Abstraction est une vue générique d'une entité permettant de ne pas être limité par des contraintes d'ordre technique ou matériel.
 - Une classe peut être appelée un type abstrait de données

- le texte spécifiant un objet définit une infinité d'objets
 - isomorphes (même structure d'état)
 - comportement identique (partagent l'ensemble des procédures)
- Données : le texte définissant un objet (concret) devient une classe (objet abstrait) définissant une infinité d'instances



Personne
Prénom
Nom
Age
courir
lire
se présenter



B2-Classes/Instances : exemple

objet abstrait
(classe)

Classe Compte

structure

numéro
solde
titulaire

comportement

donneTonSolde
dépose
retire
afficheToi

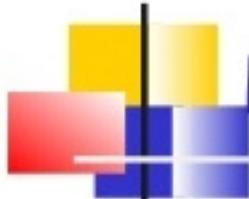
objets concrets
(instances de la classe)

instance de .
numéro **8765**
solde **5000**
Titulaire **Després**

instance de .
numéro **2345**
solde **3 milliards**
titulaire **Jacoboni**

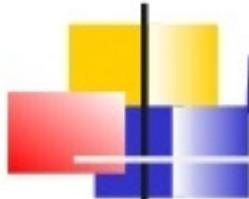
unCompte

unAutreCompte



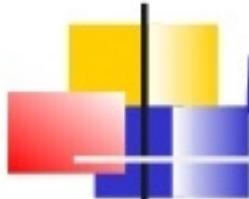
B2-Classes/Instances : réalisation

- Une classe est un texte structuré qui rassemble
 - une liste de noms de champs (**variables d'instances**)
 - un catalogue de procédures (**méthodes**) comportant leur nom et leur code
- Une instance d'une classe est un objet (une adresse)
 - constitué des champs dont la liste est donnée par la classe
 - dont le comportement en réponse aux messages est définie par les procédures fournies par la classe



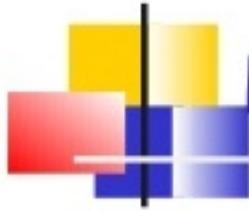
B2-Classes/Instances : programmation

- La réalisation informatique du modèle se fait en deux étapes :
 - 1/ définition (écriture) des classes d'objets
 - 2/ création des instances dont l'ensemble compose le modèle
- L'activation du modèle se fait par envoi d'un message à un objet du modèle



B2-Classes/Instances : programmation

- Programmer objet consiste à
 - concevoir les classes
 - coder les classes (souvent en réutilisant des classes existantes)
 - créer des instances des classes
 - prévoir des séquences de messages entre ces instances



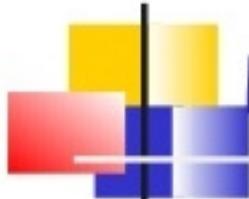
B-Résumé des principes 1 et 2

- **Principe 1**

- **Un Objet** = Une identité + un état + un comportement
 - En réponse à un message l'objet destinataire du message déclenche un comportement
(une méthode / une fonction membre)

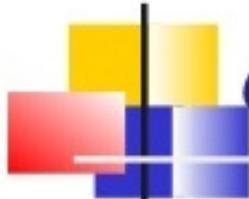
- **Principe 2**

- **Une classe est un moule pour fabriquer des objets.**
 - La classe regroupe ce qui est commun aux objets d'un même type.
 - La classe définit les attributs des objets
 - La structure de l'état des objets
 - Le comportement



B-objets, messages et classes

- 3 principes de base
 - B1 : approche simulatoire de la programmation
 - notion d'objets et de messages
 - B2 : distinction entre concept et représentant du concept
 - notion de classes et d'instances
 - ➡ ■ B3 : classification des concepts
 - notion d'héritage



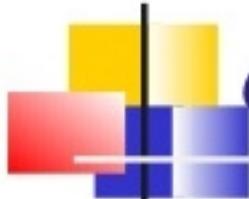
C-Cours 1 en bref...

- **POO**

- **3 principes fondateurs**
 - objets et messages, classes et instances, héritage
- **5 concepts fondateurs**
 - objets, classes et instances, messages et méthodes, héritage, encapsulation

- **Ruby**

- **3 principes de base**
 - principe du tout objet
 - tout objet est instance d'une classe
 - programmer c'est envoyer des messages à des objets



C-Résumé des termes employés

- **POO** : Objet, message, receveur, interface, encapsulation, classe, instance, variable d'instance, méthode, méthode de classe (première approche)