

Chassez le naturel... il revient à l'OO

DISC - 26 février 2004



**« L'informatique ne s'écrit pas...
...elle se ré-écrit »**

**« La civilisation progresse quand on
augmente le nombre d'opérations
importantes que nous pouvons
réaliser sans y penser »**

Alfred N. Whitehead

Programme



I. Aux origines et raisons d'être de l'objet

- Un petit écosystème
- Flashback, au temps de l'assembleur
- Victoire de l'homme ou de la nature sur la machine

II. L'orienté objet en quelques mots

- Une boîte noire
- Un objet sans classe n'a pas de classe
- Les objets parlent aux objets
- Héritage
- Le modèle devient la référence → UML
- Avantages de l'OO
- Langages et plateformes?

III. Quelques aspects pratiques

- Distribution d'objets
- Persistance d'objets

IV. Evolutions récentes / thèmes à la mode

- Conception par contrat
- Model-Driven Architecture

Chassez le naturel... il revient à l'OO

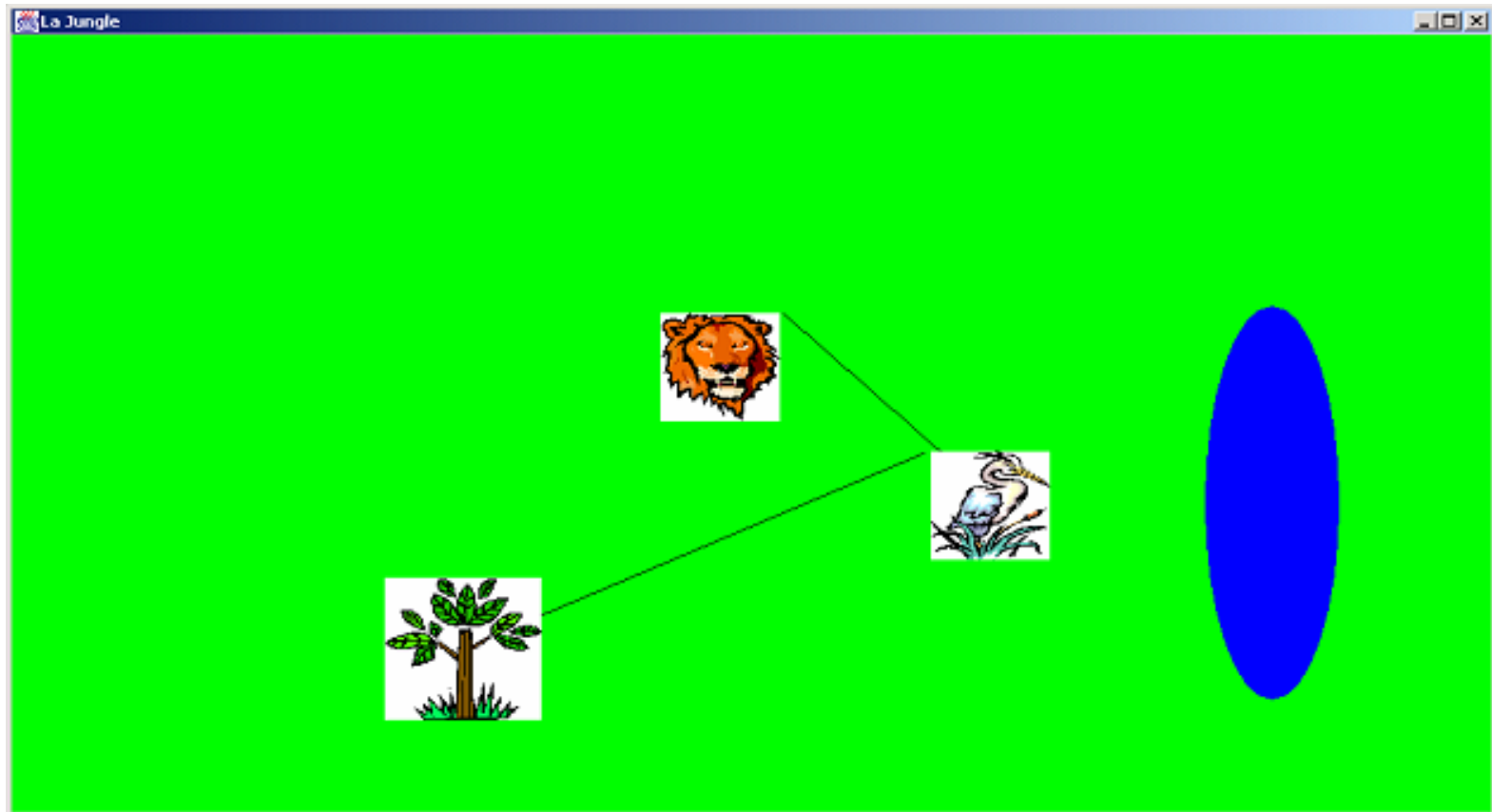
I. Aux origines de l'objet



Un petit écosystème

- **But du jeu?**

→ Simuler la vie dans une petite jungle composée de prédateurs, de proies, de plantes et de points d'eau



Un petit écosystème

• Règles du jeu?

- Les proies se déplacent soit vers l'eau, soit vers une plante, soit pour fuir un prédateur en fonction du premier de ces objets qu'elles repèrent
- Les prédateurs se déplacent soit vers l'eau, soit vers une proie en fonction du premier de ces objets qu'ils rencontrent
- Les plantes poussent lentement au cours du temps, tandis que l'eau s'évapore peu à peu
- Les animaux épuisent leurs ressources énergétiques peu à peu, au fur et à mesure de leurs déplacements
- Lorsque les animaux rencontrent de l'eau, ils s'abreuvent et récupèrent de l'énergie tandis que le niveau de l'eau diminue en conséquence
- Lorsqu'un prédateur rencontre une proie, il la dévore et récupère de l'énergie
- Lorsqu'une proie rencontre une plante, il la dévore et récupère de l'énergie

Un petit écosystème

- **Comment procéder?**

→ En découpant le problèmes en quelques grand traitements:

1. La proie et le prédateur doivent se déplacer

→ On pense/code le déplacement de la proie et du prédateur ensemble

→ Chacun des animaux doit pour cela repérer une cible

→ On conçoit une fonctionnalité de repérage commune à tous les acteurs

→ Les animaux cherchent l'eau

→ Le prédateur cherche la proie

→ La proie cherche la plante et à éviter le prédateur

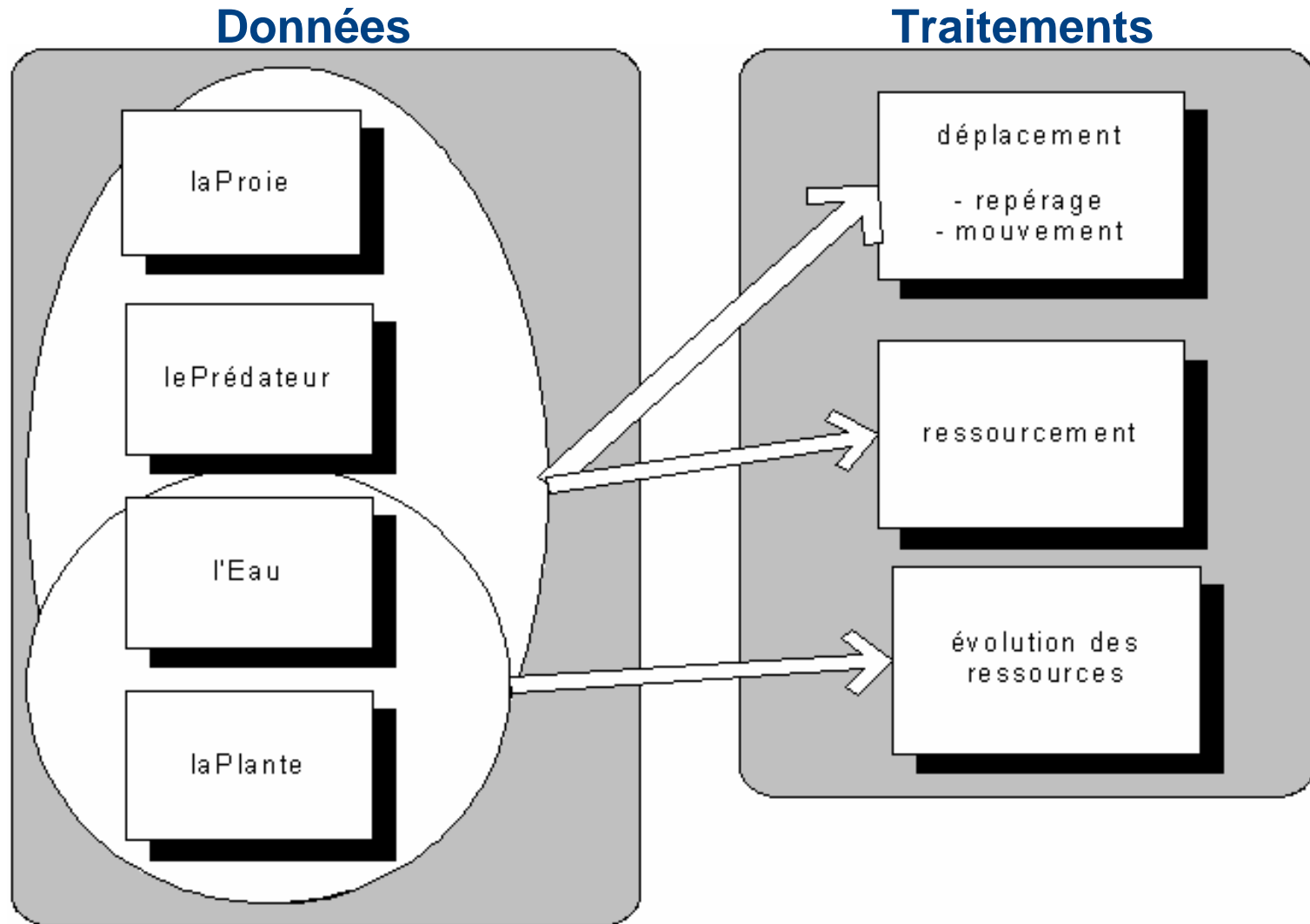
2. Les animaux se ressourcent

→ On conçoit une fonctionnalité qui concerne tous les acteurs

3. Les ressources de la plante et de l'eau évoluent

→ On pense / code une fonction d'évolution des ressources

Un petit écosystème



Un petit écosystème

- Enseignements?

- Les données constituent des ensembles globaux traités globalement
- Les traitements sont regroupés dans quelques grandes fonctions qui peuvent être imbriquées et faire appel les unes aux autres
- Les données sont donc partagées collectivement et les fonctions sont interpénétrées
- On a donc pensé aux traitements et à leur enchaînement AVANT de penser aux données

→ Approche « TOP-DOWN »

- Modus operandi de plus en plus complexe à mesure que la complexité et interdépendances des traitements s'intensifient
- Difficulté de réutiliser les fonctions dans un autre contexte
- Difficulté de maintenance: toutes les activités impliquant tous les objets
→ Si on modifie une entité, il faut vérifier l'entièreté du code



Flashback au temps de l'assembleur

- **Aux origines de l'informatique**

- Programmation des ordinateurs dictée par le fonctionnement des processeurs
- Programme = Succession d'instructions
- Organisation du programme et nature des instructions doit être le plus proche possible de la façon dont le processeur les exécute
 - Modification des données mémorisées
 - Déplacement des données d'un emplacement à un autre
 - Opérations arithmétiques et de logique élémentaire
- Programmation en langage « machine »
- Exemple: « $c = a + b$ » se programme
 - LOAD a, REG1
 - LOAD b, REG2
 - ADD REG3, REG1, REG2
 - MOVE c, REG3

Flashback au temps de l'assembleur

- **Langages de programmation procéduraux**

- Mise au point d'algorithmes plus complexes
- Nécessité de simplifier la programmation
 - Distance par rapport au fonctionnement des processeurs
- Apparition de langages procéduraux
 - Cobol, Basic, Pascal, C, etc.
- S'intercalent entre langage ou raisonnement naturel et instructions élémentaires
- Nécessité donc d'une traduction des programmes en instructions élémentaires → Compilation
- Raisonnement reste néanmoins conditionné par la conception des traitements et leur succession
 - Eloigné de la manière humaine de poser et résoudre les problèmes
 - Mal adapté à des problèmes de plus en plus complexes

Victoire de l'homme ou de la nature sur la machine

- **L'avènement de l'objet, il y a 40 ans...**

- Afin de
 - Libérer la programmation des contraintes liées au fonctionnement des processeurs
 - Rapprocher la programmation du mode cognitif de résolution des problèmes
- Mise au point d'un nouveau style de langages de programmation
 - ➔ Simula, Smalltalk, C++, Eiffel, Java, C#, Delphi, PowerBuilder
- Idée?
 - ➔ Placer les entités, objets ou acteurs du problème à la base de la conception
 - ➔ Etudier les traitements comme des interactions entre les différentes entités
 - ➔ **Penser aux données AVANT de penser aux traitements**
 - ➔ Penser un programme en modélisant le monde tel qu'il nous apparaît

Chassez le naturel... il revient à l'OO

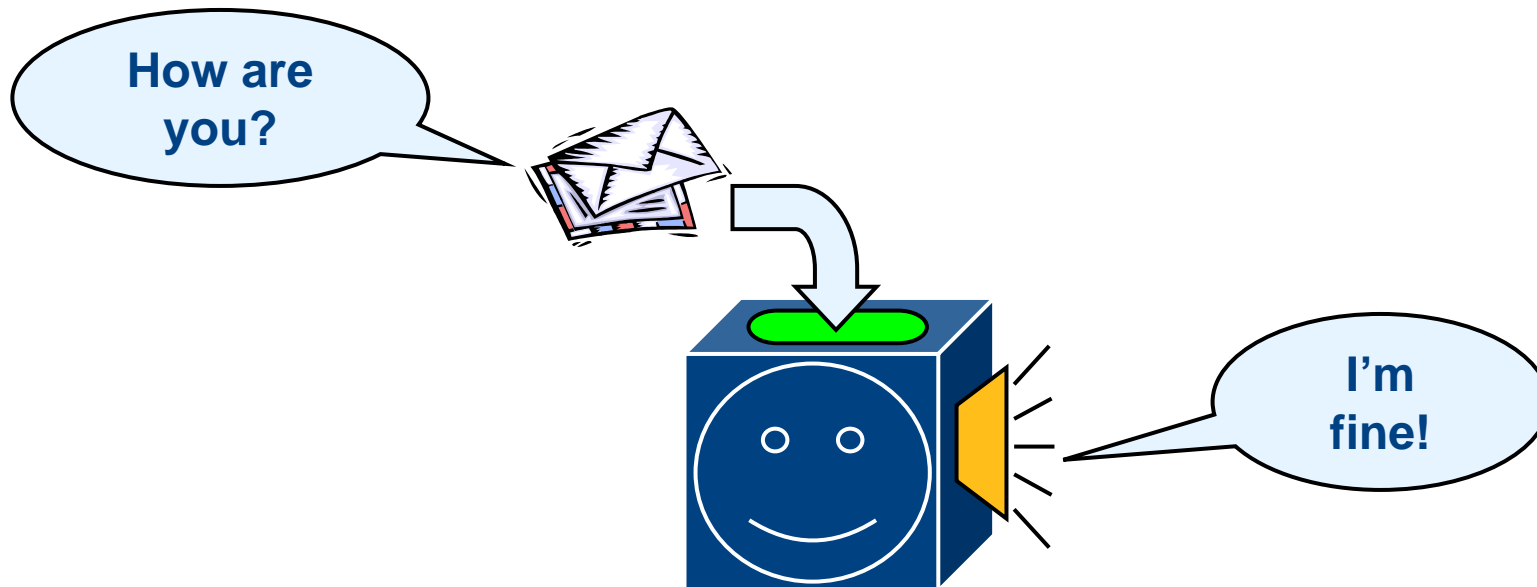
II. L'orienté objet en quelques mots



Une boîte noire

- **Qu'est-ce qu'un objet?**

→ Une boîte noire qui reçoit et envoie des messages



Une boîte noire

- **Que contient cette boîte?**

- Du code → Traitements composés d'instructions → **Comportements**
- Des données → L'information traitée par les instructions et qui caractérise l'objet → **Etats** ou **Attributs**
- Données et traitements sont donc indissociables
- Les traitements sont conçus pour une boîte en particulier et ne peuvent pas s'appliquer à d'autres boîtes

- **En procédural: code et données sont toujours séparés**

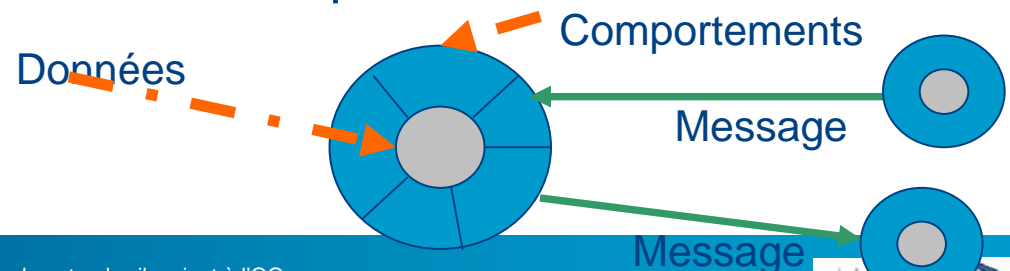
- Code → Fonctions
- Données → Structures ou unités de données
- Code et données ne sont donc pas liés
- Une même fonction peut traiter plusieurs types de données et plusieurs fonctions peuvent traiter les mêmes données

Une boîte noire

- **Pourquoi une boîte noire?**

- L'utilisateur d'un objet ne devrait jamais avoir à plonger à l'intérieur de la boîte
- Toute l'utilisation et l'interaction avec l'objet s'opère par messages
- Les messages définissent l'interface de l'objet donc la façon d'interagir avec eux
- Il faut et il suffit de connaître l'interface des messages pour pouvoir exploiter l'objet à 100%, sans jamais avoir à connaître le contenu exact de la boîte ni les traitements qui l'animent
- L'intégrité de la boîte et la sécurité de ses données peut être assurée
- Les utilisateurs d'un objet ne sont pas menacés si le concepteur de l'objet en change les détails ou la mécanique interne

- **ENCAPSULATION**



Une boîte noire

- Quelques exemples?

| Objet | États | Comportements |
|-----------|---|--|
| Chien | Nom, race, âge, couleur | Aboier, chercher le baton, mordre, faire le beau |
| Compte | N°, type, solde, ligne de crédit | Retrait, virement, dépôt, consultation du solde |
| Téléphone | N°, marque, sonnerie, répertoire, opérateur | Appeler, Prendre un appel, Envoyer SMS, Charger |
| Voiture | Plaque, marque, couleur, vitesse | Tourner, accélérer, s'arrêter, faire le plein, klaxonner |

Un petit écosystème

- **Comment procéder?**

1. Identifier les acteurs du problème: Proie/Prédateur/Plante/Eau

2. Identifier les actions ou comportements de chaque acteur

- Le prédateur:

- se déplace en fonction des cibles, peut boire l'eau et manger la proie

- La proie:

- se déplace en fonction des cibles, peut boire l'eau et manger la plante

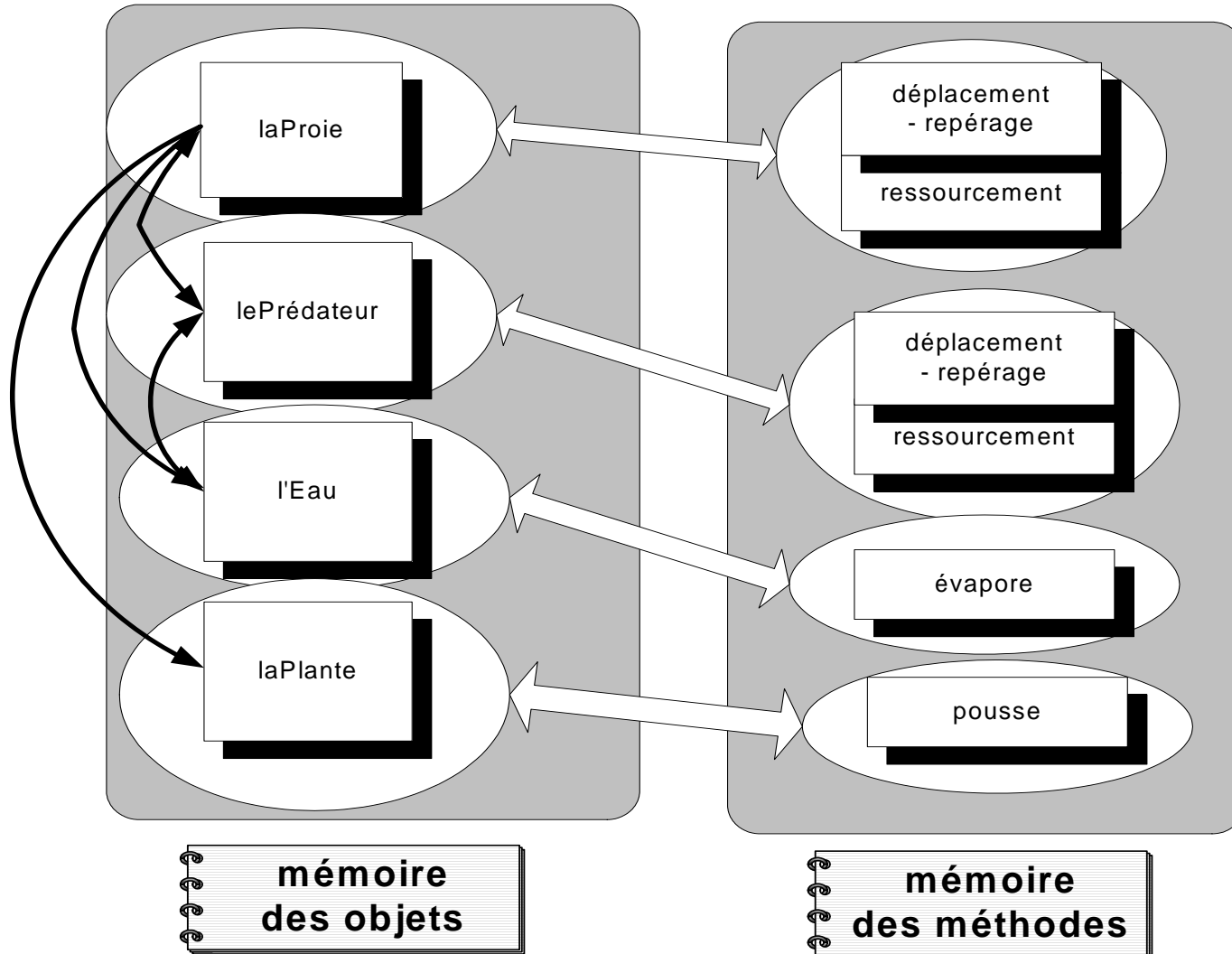
- La plante:

- pousse et peut diminuer sa quantité

- L'eau:

- s'évapore et peut diminuer sa quantité

Un petit écosystème



Un petit écosystème

• Enseignements?

- Le découpage s'effectue à partir des acteurs (et non plus des activités) de la situation
 - Découpage plus naturel
- Les traitements sont propres à chaque acteur et ne dépendent que de la structure de l'acteur concerné
- On a donc pensé aux entités/acteurs/données AVANT de penser aux traitements

→ Approche « BOTTOM - UP »



- Modus operandi remarquablement modulable et adaptable
- Les objets peuvent être réutilisés dans d'autres contextes

Un objet sans classe n'a pas de classe

- **Comment les objets sont-ils définis?**

- Par leur classe, qui détermine toutes leurs caractéristiques
 - Nature des attributs et comportements possibles
- La classe détermine tout ce que peut contenir un objet et tout ce qu'on peut faire de cet objet
- Classe = Moule, Définition, ou Structure d'un d'objet
- Objet = Instance d'une classe
- Une classe peut-être composite ⇔ peut contenir elle-même des objets d'autres classes
 - Ex: Une voiture contient un moteur qui contient des cylindres...
 - Ex: Un chien possède des pattes...

Un objet sans classe n'a pas de classe

• Quelques exemples?

- La classe « chien » définit:
 - Les attributs d'un chien (nom, race, couleur, âge...)
 - Les comportements d'un chien (Aboier, chercher le baton, mordre...)
- Il peut exister dans le monde plusieurs objets (ou instances) de chien

| Classe | Objets |
|---------|--|
| Chien | Mon chien: Bill, Teckel, Brun, 1 an Le chien de mon voisin: Hector, Labrador, Noir, 3 ans |
| Compte | Mon compte à vue: N° 210-1234567-89, Courant, 1.734 €, 1250 € Mon compte épargne: N° 083-9876543-21, Epargne, 27.000 €, 0 € |
| Voiture | Ma voiture: ABC-123, VW Polo, grise, 0 km/h La voiture que je viens de croiser: ZYX-987, Porsche, noire, 170 km/h |

Un objet sans classe n'a pas de classe

- **Similitudes avec les bases de données?**
 - Classe → Table
 - Attribut → Champ / Colonne
 - Objet → Enregistrement (ligne) de la table
 - Valeur → Valeur du champ ou de la colonne

| Marque | Modele | Serie | Numero |
|---------|--------|--------|-------------|
| Renault | 18 | RL | 4698 SJ 45 |
| Renault | Kangoo | RL | 4568 HD 16 |
| Renault | Kangoo | RL | 6576 VE 38 |
| Peugeot | 106 | KID | 7845 ZS 83 |
| Peugeot | 309 | chorus | 7647 ABY 82 |
| Ford | Escort | Match | 8562 EV 23 |

Les objets parlent aux objets

- **Quid des comportements?**

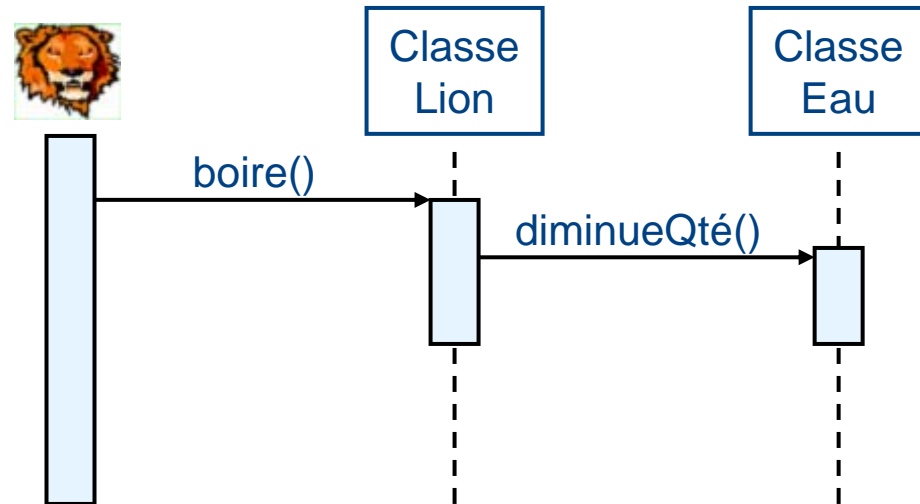
- La classe définit les comportements des objets
- Les comportements sont les messages que les objets de la classe peuvent comprendre → « Aboie », « Va chercher », « Fais le beau »
Les traitements ou instructions à réaliser lorsqu'un message en particulier est envoyé à un objet sont définis dans la classe mais restent cachés (boîte noire)
- Certains messages requièrent des renseignements complémentaires → « Va chercher... le bâton », « Aboie... 3 fois très fort »
- Pour déclencher un traitement, il faut donc envoyer un message à l'objet concerné → On n'invoque jamais une fonction dans le vide
- Pour envoyer un message, il faut toujours préciser à quel objet en particulier le destine

Ex: monChien.vaChercher(leBaton)

Les objets parlent aux objets

• Exemple?

- Quand le lion s'abreuve, il provoque une diminution de la quantité d'eau
- Ce n'est pas le lion qui réduit cette quantité
- Le lion peut seulement envoyer un message à l'eau, lui indiquant la quantité d'eau qu'il a consommée
- L'eau gère seule sa quantité, le lion gère seul son énergie et ses déplacements



Les objets parlent aux objets

- **Comment le lion connaît-il le message à envoyer à l'eau?**
 - Pour pouvoir envoyer un message à l'eau, il faut que le lion connaisse précisément l'interface de l'eau
 - La classe lion doit pour cela « connaître » la classe eau
 - Cette connaissance s'obtient par l'établissement d'une communication entre les deux classes
 - De tels liens peuvent être
 - Persistants → **Association**
 - Circonstanciels et passagers → **Dépendance**

Les objets parlent aux objets

• Communications entre classes



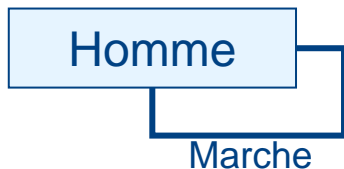
Agrégation directionnelle faible
→ Un objet Thermostat contient un objet Thermomètre



Agrégation directionnelle forte (composition)
→ Si l'objet de type Bank est détruit, l'objet de type BankAccount l'est en même temps



Association bi-directionnelle (réciproque)



Auto association
→ Appels imbriqués de méthodes



Dépendance
→ Une méthode de Dessin exige un objet ContexteGraphique en argument

Orienté objet?

Un programme orienté objet est uniquement constitué de classes interagissant par envoi de messages

L'intégralité du code d'un programme orienté objet se trouve donc à l'intérieur de classes

Orienté objet?

Déclaration
de la classe

Variables d'instance
ou « champs »

Définition du
constructeur

Méthodes d'accès

Définition
des
méthodes

```
package bank;
import java.lang.*;
public class BankAccount {
    private String name ;
    private int solde ;
    private int interest ;
    public BankAccount(String n,int s,int i) {
        name=n ;
        solde=s;
        interest=i;
    }
    public String getName() { return name; }
    public void setName (String n) {name= n;}
    ...
    public void deposit (int amount) {
        solde += amount ;
    }
    public void withdrawal (int amount ){
        solde-=amount ;
    }
}
```

Class Body

Héritage

- **En quoi consiste l'héritage?**

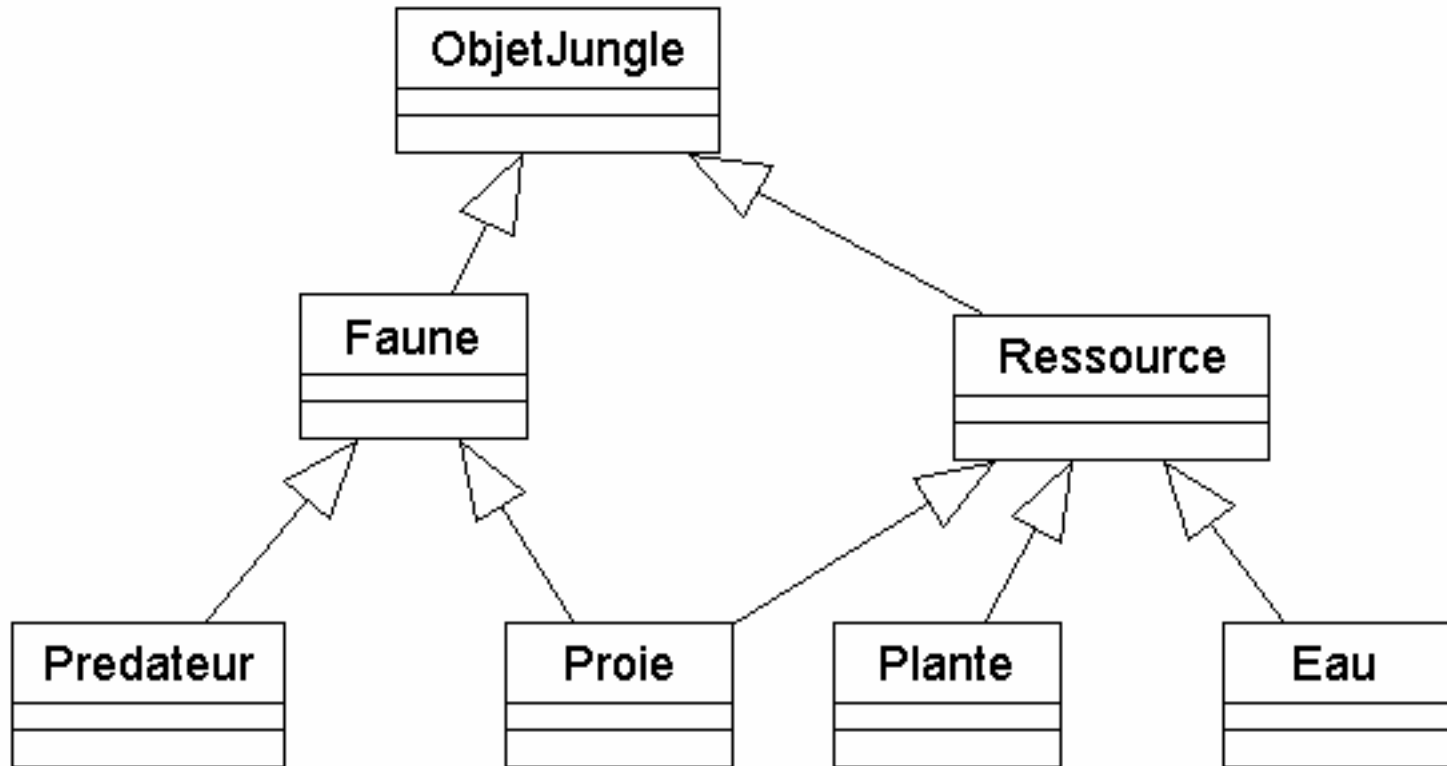
- Supposons qu'il existe déjà une classe qui définit un certain nombre de messages et qu'on ait besoin d'une classe identique mais pourvue de quelques messages supplémentaires
- Comment éviter de réécrire la classe de départ?
- Regrouper les classes en super-classes en factorisant et spécialisant
- La sous-classe hérite des attributs et méthodes et peut en rajouter de nouveaux
- Quid du multi-héritage?



Héritage

• Ex: Dans l'écosystème

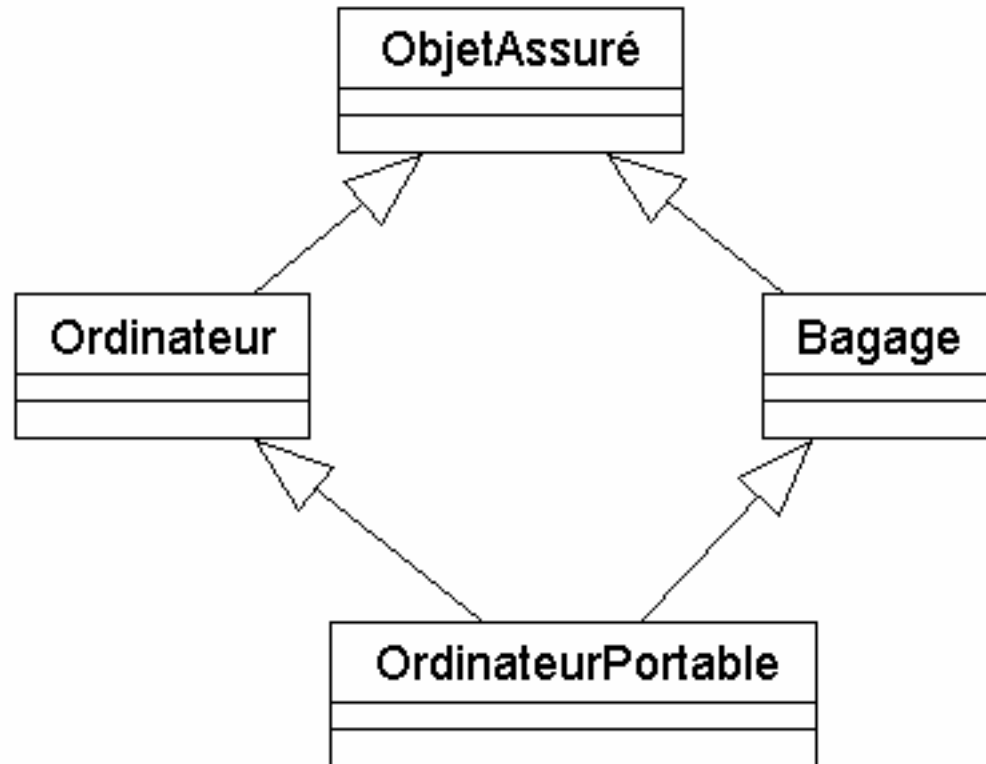
- La classe Faune regroupe les animaux
- La classe Ressource regroupe l'eau et la plante



Héritage

- **Quid du multi-héritage?**

- Possible ou non d'un langage à un autre



Héritage

- **Qu'est-ce que le polymorphisme?**

- Concept basé sur la notion de redéfinition de méthodes
- Permet à une tierce classe de traiter un ensemble de classes sans connaître leur nature ultime
- Permet de factoriser des dénominations d'activité mais pas les activités elles-mêmes
- Consiste à permettre à une classe de s'adresser à une autre en sollicitant un service générique qui s'appliquera différemment au niveau de chaque sous-classe du destinataire du message
- En d'autres termes, permet de changer le comportement d'une classe de base sans la modifier → Deux objets peuvent réagir différemment au même appel de méthode
- Uniquement possible entre classes reliées par un lien d'héritage

- **Exemple dans l'écosystème?**

- Demander à tous les animaux de se déplacer (selon leurs propres règles) en leur adressant un message en tant qu'objets de type "Faune"

La modélisation devient la référence

- **Pourquoi la modélisation?**

- La conception OO est entièrement batié sur une modélisation des objets intervenant dans le problème
- Avant de programmer quoi que ce soit, il faut donc modéliser les classes et leurs relations au minimum

- **Comment?**

- Sur base d'UML (Unified Modeling Language)
 - Notation standardisée pour toute le développement OO de la conception au déploiement
 - Définition de 9 diagrammes

1. Identifier les classes

- Attributs, comportements, polymorphisme

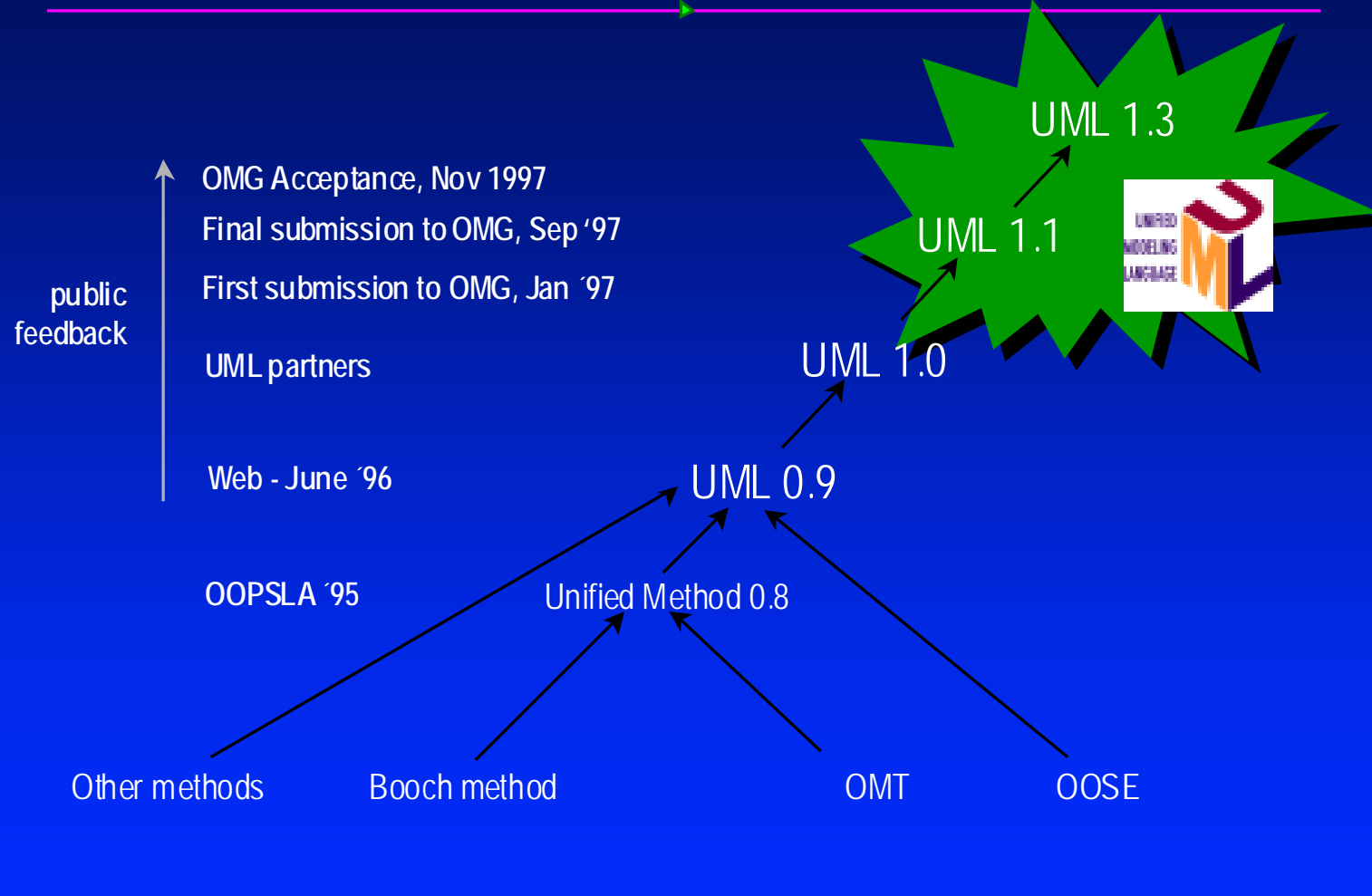
2. Déterminer les relations entre les classes

- Associations / Dépendance / Hériage

3. Construire les modèles

La modélisation devient la référence

Creating the UML

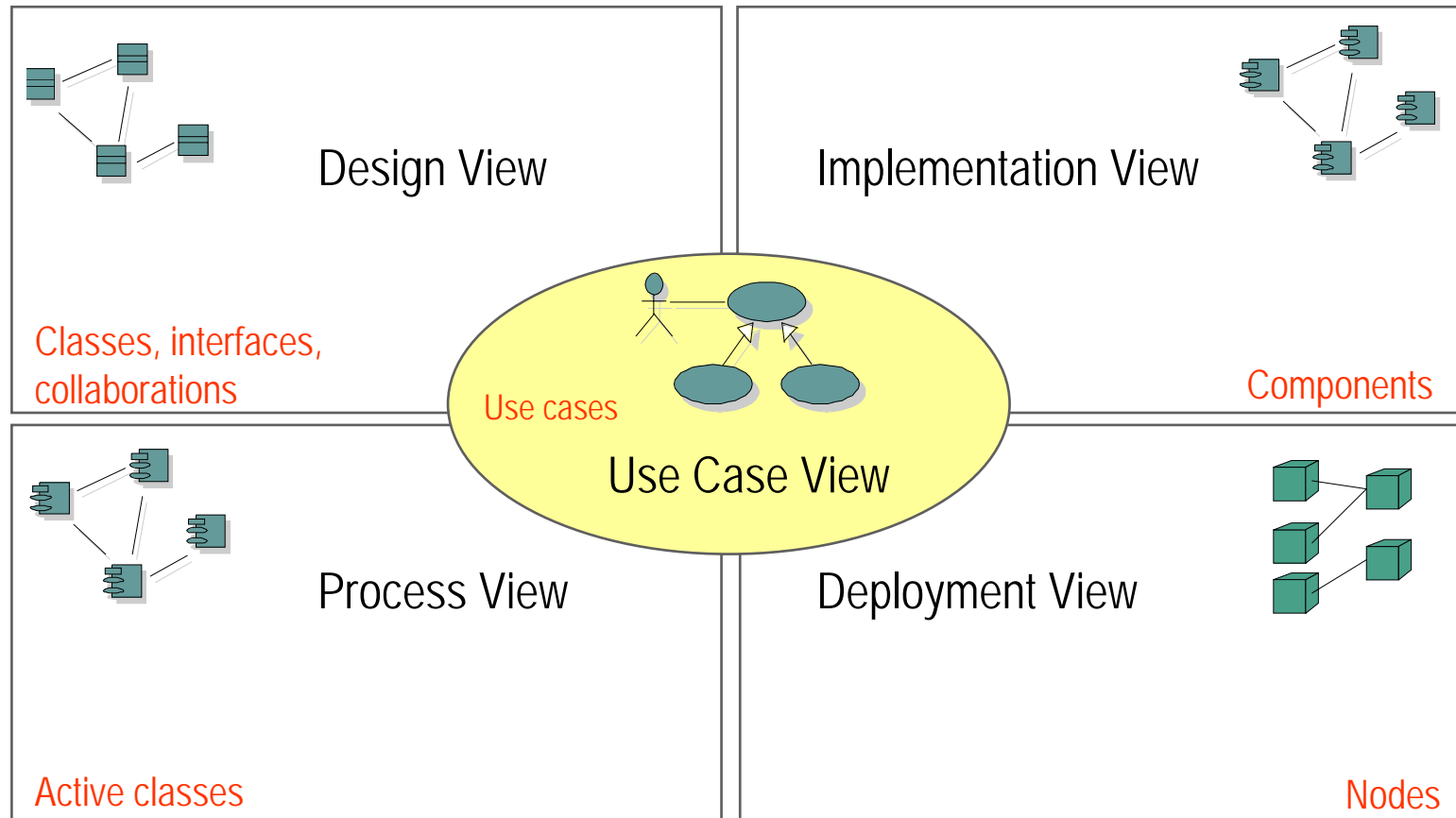


La modélisation devient la référence

• Qu'est-ce qu'UML?

- UML est un langage objet graphique - un formalisme orienté objet, basé sur des diagrammes (9)
- UML permet de s'abstraire du code par une représentation des interactions statiques et du déroulement dynamique de l'application.
- UML prend en compte, le cahier de charge, l'architecture statique, la dynamique et les aspects implémentation
- Facilite l'interaction, les gros projets
- Générateur de squelette de code
- UML est un langage PAS une méthodologie, aucune démarche n'est proposée juste une notation

La modélisation devient la référence



La modélisation devient la référence

• Diagrammes UML

- Les diagrammes des cas d'utilisation: les fonctions du système, du point de vue de l'utilisateur ou d'un système extérieur - l'usage que l'on en fait
- Les diagrammes de classes: une description statique des relations entre les classes
- Les diagrammes d'objet: une description statique des objets et de leurs relations. Une version « instanciée » du précédent
- Les diagrammes de séquence: un déroulement temporel des objets et de leurs interactions
- Les diagrammes de collaboration: les objets et leurs interactions en termes d'envois de message + prise en compte de la séquentialité

La modélisation devient la référence

• Diagrammes UML

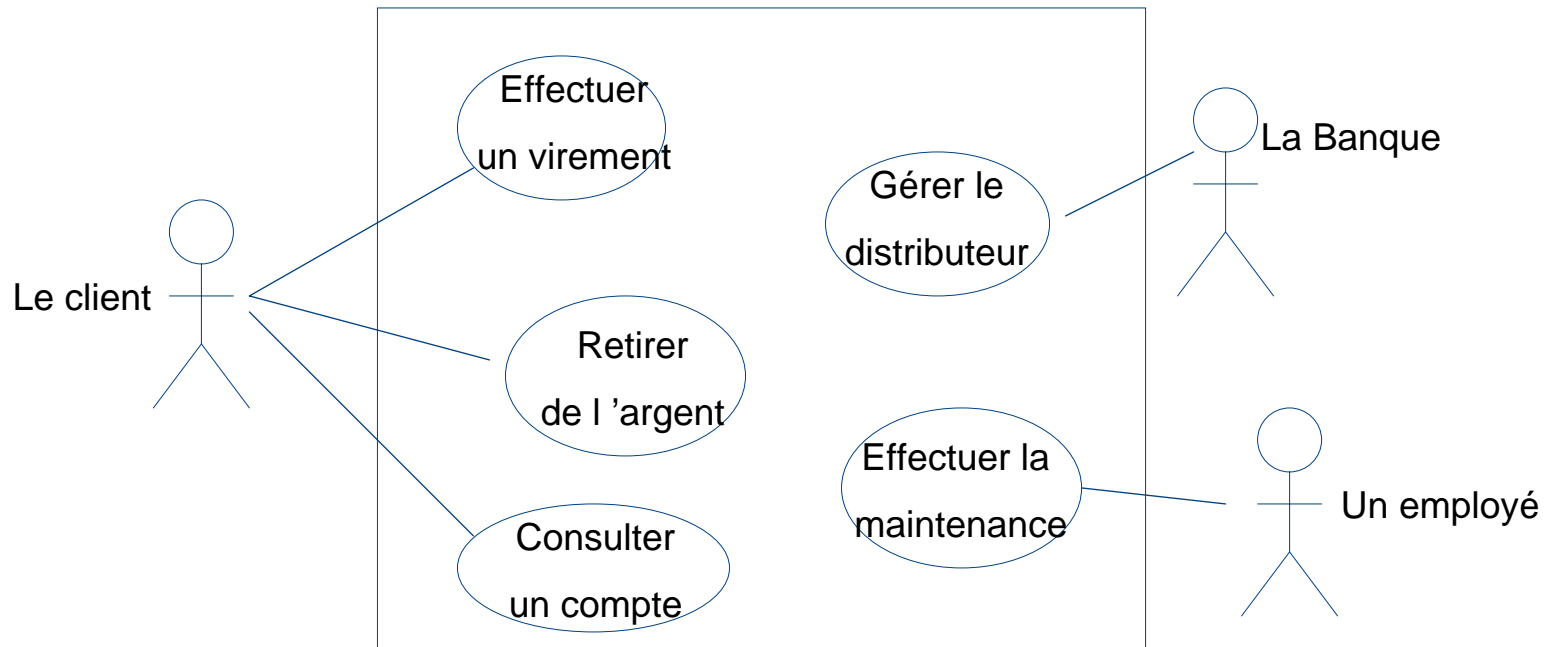
- Les diagrammes d'états-transitions: scrute les cycles de vie d'une classe d'objet, la succession d'états et les transitions
- Les diagrammes d'activité: le comportement des différentes opérations en termes d'actions
- Les diagrammes de composants: représente les composants physiques d'une application
- Les diagrammes de déploiements: le déploiement des composants sur les dispositifs et les supports matériels

La modélisation devient la référence

- **Diagramme de cas d'utilisation**

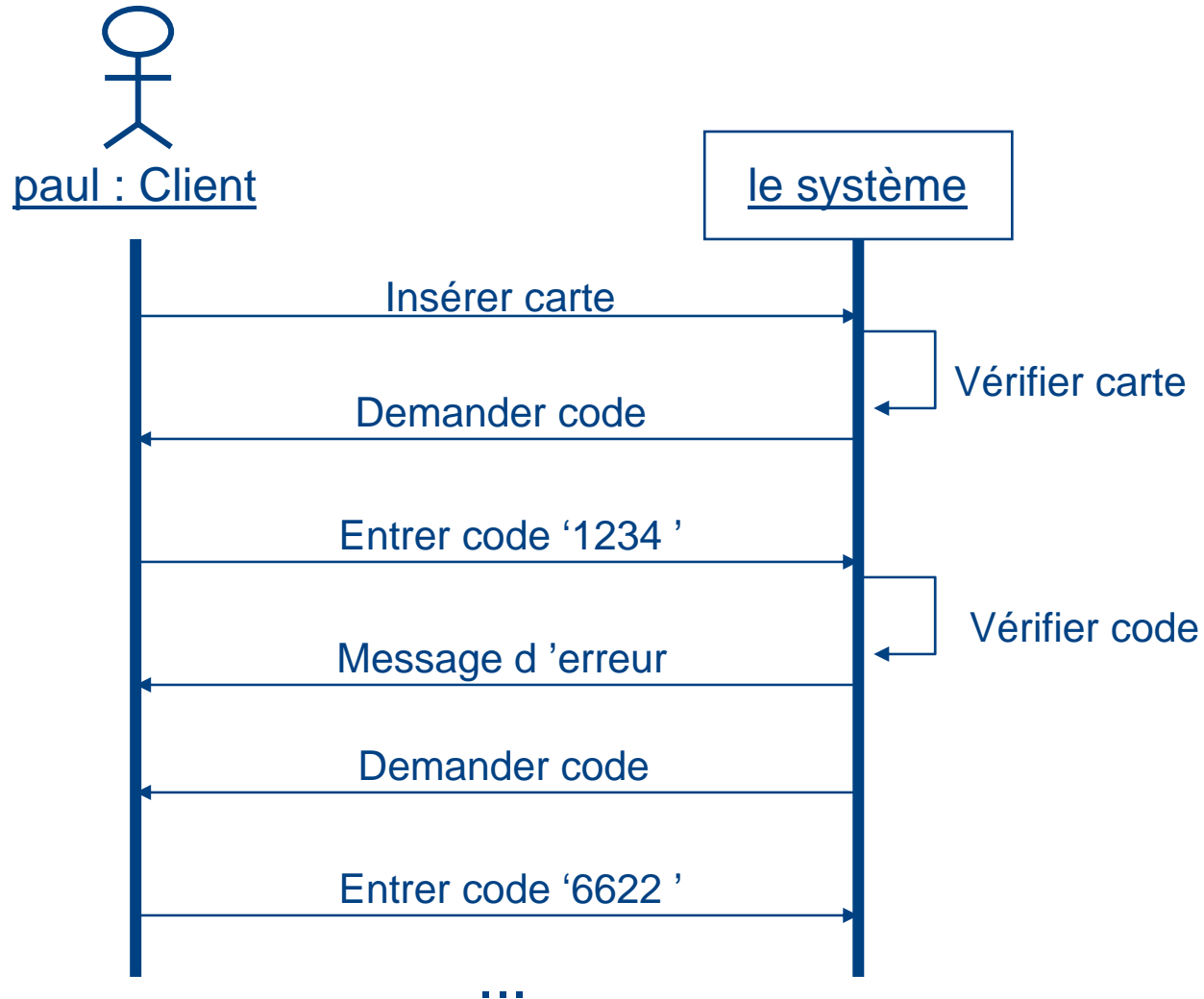
- Cela répond aux spécifications du système:
 - Ses fonctionnalités, son utilisation, les attentes de l'utilisateur

- **Ex: Distributeur MisterCash**



La modélisation devient la référence

- Exemple de diagramme de cas d'utilisation détaillé



La modélisation devient la référence

• Diagramme de classes

- Le but du diagramme de classes est de représenter les classes au sein d'un modèle
- Dans une application OO, les classes possèdent:
 - Des attributs (variables membres)
 - Des méthodes (fonctions membres)
 - Des relations avec d'autres classes
- C'est tout cela que le diagramme de classes représente
- L'encapsulation est représentée par:
 - (private), + (public), # (protected)
- Les attributs s'écrivent:
+/-/# nomVariable : Type
- Les méthodes s'écrivent:
+/-/# nomMethode(Type des arguments) : Type du retour ou « void »

| Nom Classe |
|------------|
| Attributs |
| Méthodes() |

La modélisation devient la référence

- Les relations d'héritage sont représentées par:

- A  B signifie que la classe A hérite de la classe B

- L'agrégation faible est représentée par:

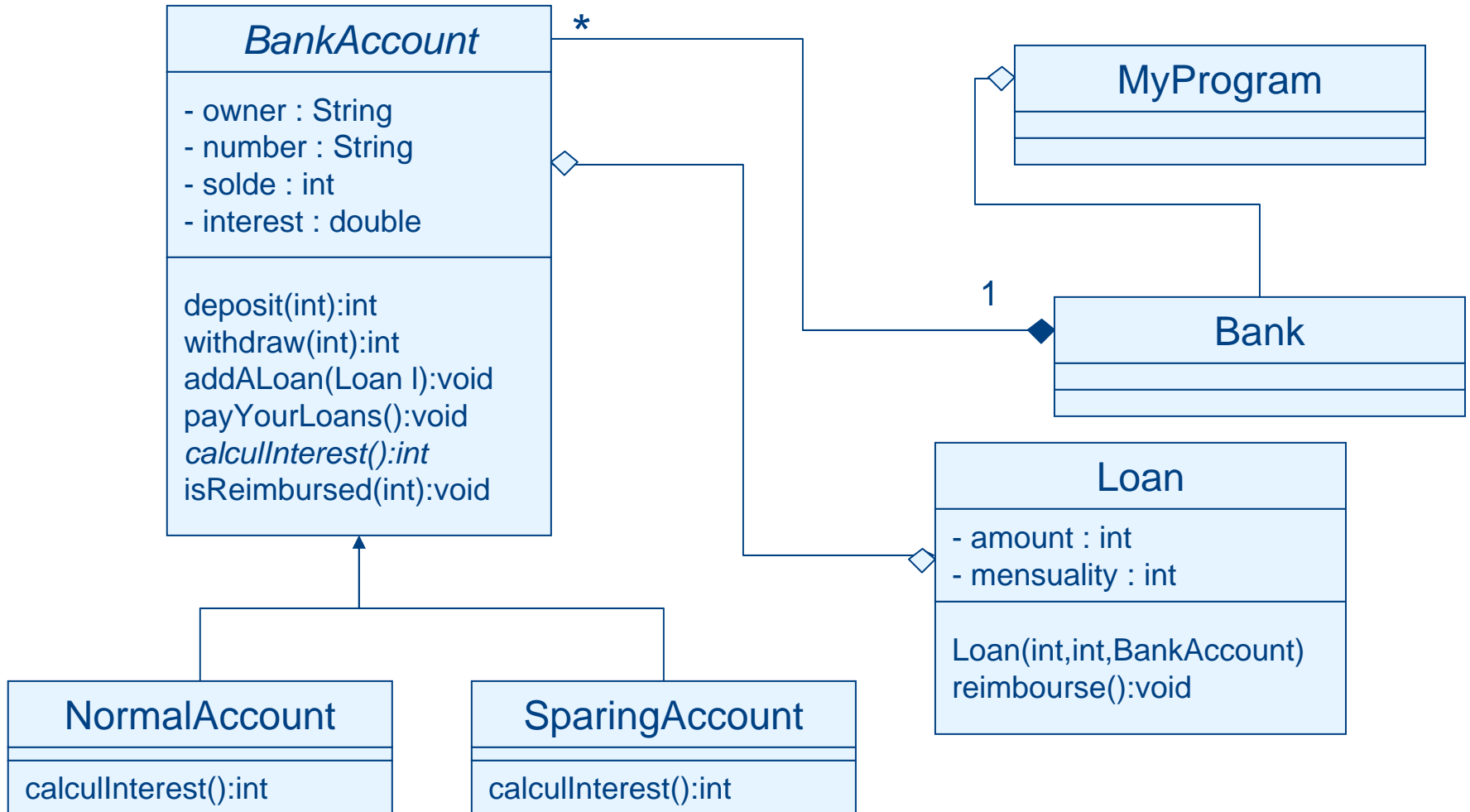
- A  B signifie que la classe A possède un ou plusieurs attributs B

- L'agrégation forte (ou composition) est représentée par:

- A  B signifie que les objets de la classe B ne peuvent exister qu'au sein d'objets de type A

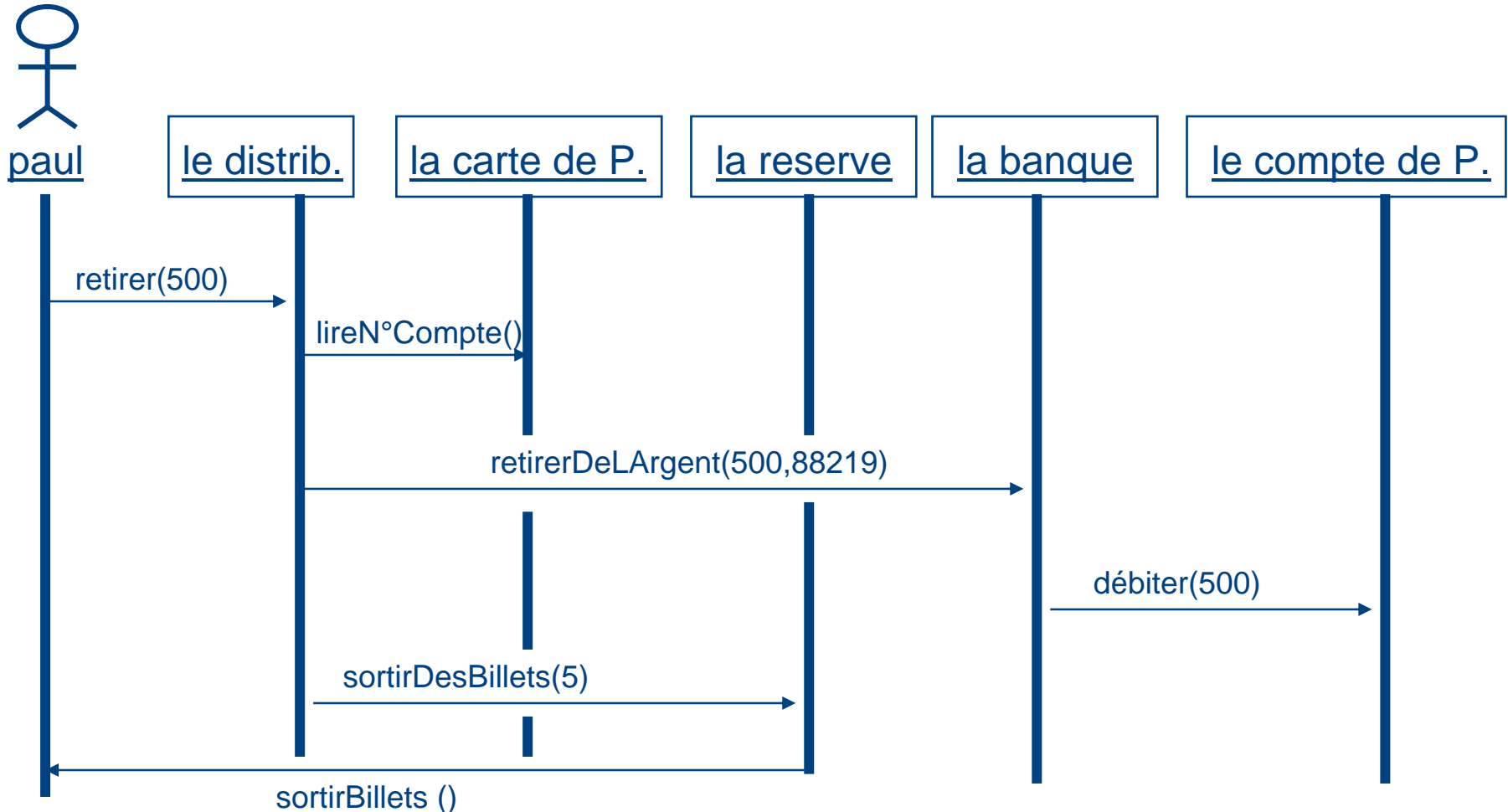
La modélisation devient la référence

- Exemple de diagramme de classes



La modélisation devient la référence

- Exemple de diagramme de séquence



Les avantages de l'OO

- **Les programmes sont plus stables, plus robustes et plus faciles à maintenir car le couplage est faible entre les classes («encapsulation»)**
- **elle facilite grandement le ré-emploi des programmes: par petite adaptation, par agrégation ou par héritage**
- **émergence des «design patterns»**
- **il est plus facile de travailler de manière itérée et évolutive car les programmes sont facilement extensibles. On peut donc graduellement réduire le risque plutôt que de laisser la seule évaluation pour la fin.**
- **l'OO permet de faire une bonne analyse du problème suffisamment détachée de l'étape d'écriture du code - on peut travailler de manière très abstraite → UML**
- **l'OO colle beaucoup mieux à notre façon de percevoir et de découper le monde**

Les avantages de l'OO

- **Tous ces avantages de l'OO se font de plus en plus évidents avec le grossissement des projets informatiques et la multiplication des acteurs. Des aspects tels l'encapsulation, l'héritage ou le polymorphisme prennent vraiment tout leur sens. On appréhende mieux les bénéfices de langage plus stable, plus facilement extensible et plus facilement ré-employable**
- **JAVA est un langage strictement OO qui a été propulsé sur la scène par sa complémentarité avec Internet mais cette même complémentarité (avec ce réseau complètement ouvert) rend encore plus précieux les aspects de stabilité et de sécurité**

Langages et plateformes

- **Quels sont les principaux langages orienté objet aujourd'hui?**
 - **C++**
 - Hybride, permet la coexistence d'OO et procédural
 - Puissant et plus complexe
 - Pas de « ramasse-miettes », multihéritage, etc.
 - **Java**
 - Très épuré et strictement OO
 - Neutre architecturalement (Multi-plateformes)
 - Ramasse-miettes, pas de multihéritage, nombreuses librairies disponibles
 - **C#**
 - Très proche de Java
 - Conçu par Microsoft
 - **Eiffel**
 - Le plus pur(itain) des langages OO
 - Conçu par Bertrand Meyer
 - **Autres**
 - PowerBuilder, Delphi, Smalltalk (I.A.), etc.

Chassez le naturel... il revient à l'OO

III. Quelques aspects pratiques



Objets distribués

- **Faire d'internet un ordinateur géant?**

- Utiliser des objets qui s'exécutent indépendamment sur des processeurs séparés
- En s'envoyant des messages à travers le réseau (ou internet)

- **Objectif?**

- Accroître la capacité de calcul à disposition d'une application en répartissant son traitement sur différentes machines
 - « *The computer IS the network* »
 - Cf. [Seti@home](#)
- Permettre à des applications d'interagir lorsque leurs données sont stockées sur des serveurs distincts
 - Cf. Architecture 3-tier ou multi-tier des applications Web
 - Les allers-retours entre les 3 machines peuvent se faire au moyen d'objets distribués, typiquement Java RMI

Objets distribués

- **Solutions**

- **OMG → CORBA**
 - CORBA se veut indépendant des plateformes, des OS et des langages
- **Microsoft → OLE/DCOM → ActiveX**
- **Java → RMI**
 - uniquement java mais indépendant également des plateformes et des OS
 - La grande différence avec CORBA est donc la flexibilité additionnelle du langage

Objets distribués

- **CORBA (Common Object Request Broker Architecture)**

- Norme d'applications distribuées prenant en compte: la communication, l'hétérogénéité, l'intégration et l'interopérabilité
- Intégration: il est possible de récupérer des applications "legacy" par l'encapsulation de celles-ci dans des objets ré-utilisables
- Plus universel mais plus complexe que RMI
- Défini et standardisé par l'OMG
- Middleware: vaste gamme de services logiciels (tous objets) pour construire des applications distribuées
- Objectif: Rendre l'appel distant à des méthodes semblable à l'appel local

Objets distribués

- **CORBA (Common Object Request Broker Architecture)**
 - Architecture globale:
 - le bus d'objets répartis: ORB
 - les services objet communs: Common Object Services: nommage, persistance, cycle de vie, transactions, sécurité
 - utilitaires communs: CORBA facilities: logiciels de haut niveau pour l'interface utilisateur, la gestion de l'information, administration du système
 - les interfaces de domaine: pour des segments d'activité tels que la santé, la finance, les télécoms ou les échanges commerciaux = objets métiers

Objets distribués

- **Architecture globale:**

- le bus d'objets répartis: ORB
- les services objet communs: Common Object Services: nommage, persistance, cycle de vie, transactions, sécurité
- utilitaires communs: CORBA facilities: logiciels de haut niveau pour l'interface utilisateur, la gestion de l'information, administration du système
- les interfaces de domaine: pour des segments d'activité tels que la santé, la finance, les télécoms ou les échanges commerciaux = objets métiers

Objets distribués

- **Le bus d'objets répartis: Object Request Broker:**
 - les invocations sont transparentes: quelles soient locales ou distantes. Indépendantes des langages ou des OS
 - Il y a deux types d'invocation:
 - statique (très proche de RMI)
 - dynamique: plus complexe car il n'y a pas de stub capturant la fonctionnalité de l'objet serveur. On peut découvrir sa fonctionnalité pendant le cours même de l'exécution - grâce à un référentiel des interfaces. A partir des informations extraites, un programme peut construire dynamiquement les invocations. C'est parfait quand les objets changent souvent ---> plus de flexibilité
 - Différentes stratégies d'activation automatique des objets: un processus par objet, un processus partagé, ou un processus par exécution de méthode.
 - Communication standard entre différents bus

Persistance des objets

• Kesako?

- Les objets créés au cours de l'exécution d'un programme sont normalement détruits lorsque l'application se termine
- Les objets peuvent être sauvés (sérialisés) → Photo instantannée
- La sauvegarde peut se faire
 - Sur disque, sur une autre machine, sur une base de données, vers un fichier XML...
- Seuls les attributs de chaque objet sont sauvés
- Pour cela les objets doivent faire partie d'une classe "serializable"
- Tout le graphe d'objet est sauvé sauf si l'un des objets est "non-serializable"
 - C'est le cas des classes qui changent constamment (Thread, FileInputStream,...)
 - On peut permettre la sauvegarde en rendant certains attributs "transient"
- La lecture renvoie des "Objets" et donc ils doivent être "castés"

Persistance des objets

- **Les solutions immédiates**

- **séparant le monde du relationnel et de l'OO**

- La persistance dans les langages OO
- Des requêtes SQL dans les programmes OO

- **Comment interagir avec un base de données relationnelle dans un programme OO?**

- Avec JDBC par exemple

- C'est un niveau d'abstraction qui permet une interaction indifférenciée avec la grande majorité des bases de données relationnelles

- JDBC est compatible ANSI-SQL2 et très proche de ODBC

- JDBC nécessite un pilote de communication avec la BD qui réalise la communication, l'envoi de requêtes SQL et la réception de "ResultSet"

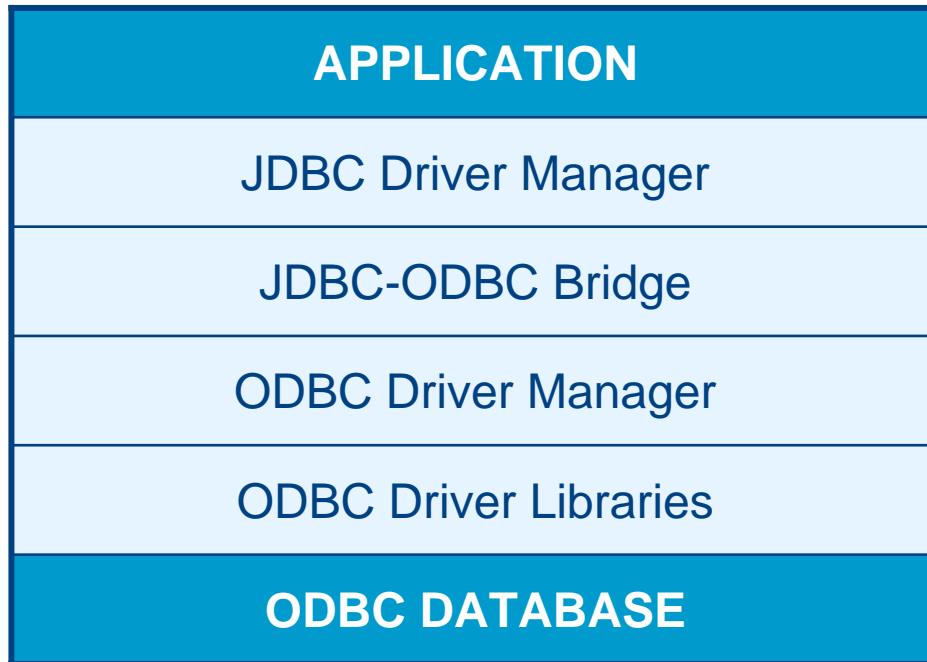
- Il faut d'abord installer le pilote au sein du programme

- Puis lancer la connection

- Possibilité d'établir un pont JDBC vers ODBC

Persistence des objets

- **JDBC**



Persistance des objets

• Quelques questions méthodologiques

- Doit-on séparer le comportement propre à l'application des aspects bases de données?
- Les classes doivent-elles contenir tous les détails qui permettent d'interagir avec la base de données ou alors ne rien savoir et laisser cela à un module externe?
 - Dans le premier cas, la classe possède les méthodes nécessaires à la sauvegarde des données. Mais les différences entre l'OO et le relationnel rendent cette solution difficile.
 - Dans l'autre cas, les deux aspects sont bien tenus séparés

Persistance des objets

- **Bases de données relationnelles et OO?**

- Tous deux ont pour objectif d'établir des relations entre des données
- Une équivalence attirante mais inexacte est:
→ Classe = Table et Objet = Enregistrement
- Cela devient d'autant plus faux que l'on s'intéresse aux aspects relationnels.
- Et cela même si on se maintient aux données statiques, sans parler de méthodes, héritage, polymorphisme
- Les relations sont plus explicites et plus riches en OO. Les pointeurs jouent un rôle capital. Un join est différent d'un attribut d'une classe
- L'objet préserve une identité malgré des changements d'attributs
- Dénaturation du programme OO au moment de la sérialisation vers une base de données relationnelle

Persistance des objets

- **Bases de données OO**

- Offrent

- un mode de lecture et écriture d'objets aussi simple que sérialisation
- Les mécanismes habituels des bases de données:
 - archivage
 - accès sécurisés, fiabilisés et concurrentiels
 - existence d'un langage d'interrogation détaché de la programmation → OQL au lieu de SQL

- Emergent depuis une dizaine d'années

- ObjectStore, Versant, O2, Poet, Gemstone, etc.

- Tardent à percer

Chassez le naturel... il revient à l'OO

IV. Evolutions récentes/thèmes à la mode



Conception par contrat

- **Comment concevoir des applications OO fiables?**

- Un élément crucial de la programmation: la fiabilité (absence de bugs)
 - Capacité d'un système à s'exécuter conformément aux spécifications (exactitude) et à gérer les situations anormales (robustesse)
- Mécanismes de gestion mémoire? Réutilisation de composants validés?
 - Insuffisant pour garantir la fiabilité
- Nécessité d'une approche systématique pour spécifier et implémenter des éléments OO et leurs relations dans un système
 - Conception par contrat (Bertrand Meyer)

Conception par contrat

- **En quoi consiste la conception par contrat?**

- Les classes fournissent en fait des prestations pour des « clients » (via leurs messages)
- Elles devraient donc le faire sous forme de « contrat » clairement établi et spécifié qui définit:
 - Les pré-conditions nécessaires à la bonne exécution
 - Les post-conditions que la classe prestataire s'engage à fournir à son client
 - De plus, chaque classe a la responsabilité personnelle de préserver son intégrité en respectant un certain nombre d'invariants
- Cette méthode peut en fait, en triturant l'écriture, être implémentée dans n'importe quel langage orienté objet
- Elle gagne néanmoins à être intégrée au sein de la syntaxe du langage → Le cas d'Eiffel uniquement

Conception par contrat

- **Exemple de pré- et post-conditions**

- Soit une méthode qui calcule la racine carrée d'un nombre réel
 - Pré-condition: le nombre réel indiqué ne peut être négatif
 - Post-condition: le résultat est une approximation (avec un degré de précision spécifié), de la racine carrée réelle

- **Exemple d'invariant**

- Soit une classe compte en banque
 - Champs « Solde actuel » et « liste des dépôts et retraits depuis l'ouverture »
 - ➔ Un invariant de classe pourrait définir que la valeur du champ « Solde actuel » doit toujours être égal au total de tous les dépôts moins tous les retraits

Conception par contrat

- **Avantages de la conception par contrat**

- Meilleure compréhension de la méthode OO et, plus généralement, de la construction de logiciels
- Approche systématique pour réaliser des systèmes OO sans bugs
- Cadre efficace pour le débogage, le test et la garantie de qualité
- Méthode pour documenter les composants logiciels
- Meilleure compréhension et meilleur contrôle du mécanisme d'héritage
- Technique pour gérer les cas anormaux, menant à une construction sûre et efficace de la gestion d'exceptions

Model Driven Architecture

- **L'environnement informatique aujourd'hui, c'est**
 - Systèmes distribués sur toute la planète via internet
 - Des plateformes, langages et applications hétérogènes
 - Une demande croissante d'inter-connexion au sein des entreprises et entre entreprises différentes
 - L'émergence des appareils de poche et sans-fil
 - L'avènement de nouvelles technologies: XML, Services Web, etc.
- **Applications e-business**
 - Doivent être écrites, modifiées et déployées rapidement
 - Nécessite de connecter diverses applications nouvelles et anciennes au sein de nouvelles applications qui doivent répondre rapidement
 - A fait relever les besoins d'une approche systématique de l'intégration d'applications

Model Driven Architecture

- **Quelques bonnes nouvelles de la dernière décennie**

- Standardisation largement accrue
 - Protocoles Internet, SQL, UML
- Ouverture accrue
 - Apache, Linux, Java
- Moins de développement spécifique
 - Réutilisation de composants, Applications ERP, CRM, etc. packagées

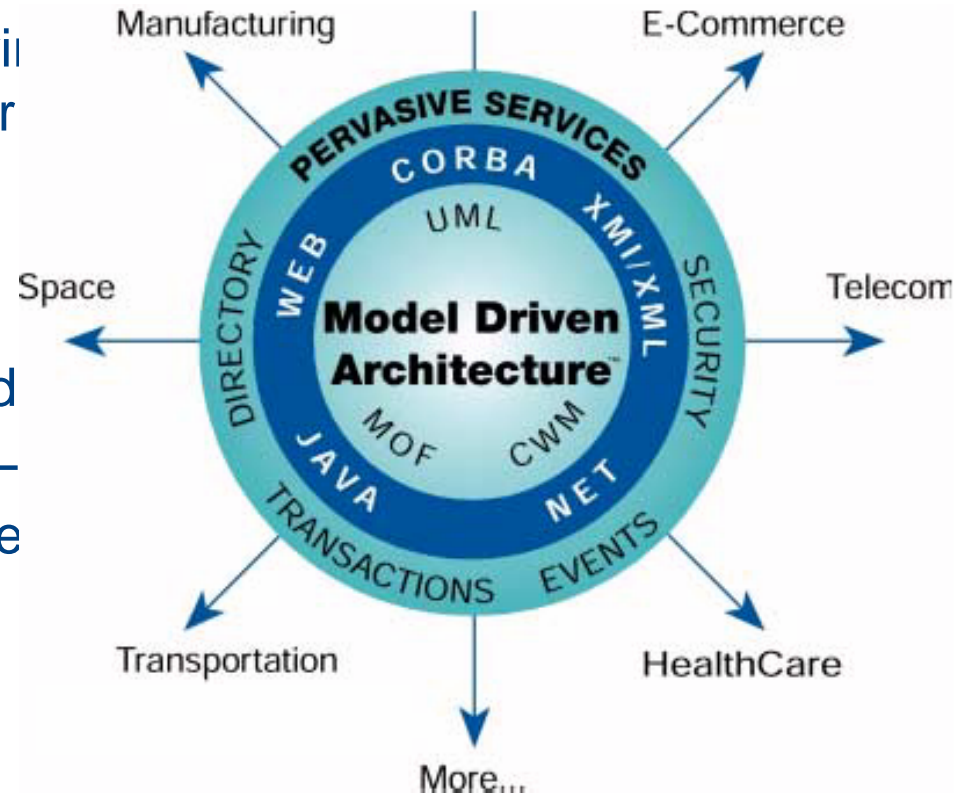
- **Quelques défis présents**

- Maintenance et connexion des applications existantes et des bases de données
- Difficulté de modifier les applications ERP et autres
- Systèmes middleware concurrents et multiples
- Mise en place d'une architecture et d'une infrastructure d'entreprise opérationnelle, la plus adaptable possible

Model Driven Architecture

• Qu'est-ce que le MDA?

- Une nouvelle methode pour écrii des spécifications et développer des applications sur base d'un modèle indépendant des plate-formes (PIM)
- Au départ, MDA est une méthod plus sophistiquée d'utilisée UML
- Basé sur des standards largeme établis: ULM, XML, CORBA
- Consiste à séparer
 - la logique fondamentale d'un composant (UML)
 - Les spécificités du middleware qui l'implémente (PSM) → Platform-specific models



Model Driven Architecture

• Ambitions du MDA

- Augmentation de la portabilité et de la réutilisation des applications
- Interopérabilité entre plateformes
 - Garantit que les standards basés sur différentes technologies implémentent des fonctions business identiques
- Indépendance par rapport aux plateformes
- Gain en efficacité et productivité du développement du fait que chaque développeur, concepteur ou administrateur système peut travailler dans les langages et les concepts qu'il maîtrise le mieux tout en permettant la communication et l'intégration transparente entre différentes équipes