# Bases de programmation

# Paradigme Objet

*DUT Réseaux & Télécommunications 1ière année*
*IUT d'Aix Marseille*

*J. Seinturier*

Laboratoire des Sciences de l'Information et des Systèmes
umr CNRS 7296
Equipe Images et Modèles

seinturier@gmail.com
http://www.seinturier.fr

# Paradigme Objet

## Etat des connaissances

### Algorithmique

- Permet de formaliser des traitements simples
- Types de données basiques (nombres, booléens, caractères)
- Syntaxe simple
- Echangeable, code strict
- Facilement implantable (C, Java, …)

### Limitations

- Difficulté de représentation des données complexes
- Difficulté de représentation de traitements complexes

# Etat des connaissances

## Types de données complexes

- Notion de données **structurées**
- Répond au problème de représentation des données
- Utilisable avec des algorithmes classiques

## Limitations

- Pas de liaison données / traitement explicite
  - Données définies séparément des traitements
  - Maintient de la cohérence des traitements difficile
- Difficulté de représentation de traitements complexes

# Problématique

## Types de données complexes

**Exemple:** *Créer un logiciel capable de simuler la gestion d'une bibliothèque.*

- *Une bibliothèque est composée d'un ensemble de livres et d'un ensemble d'adhérents.*

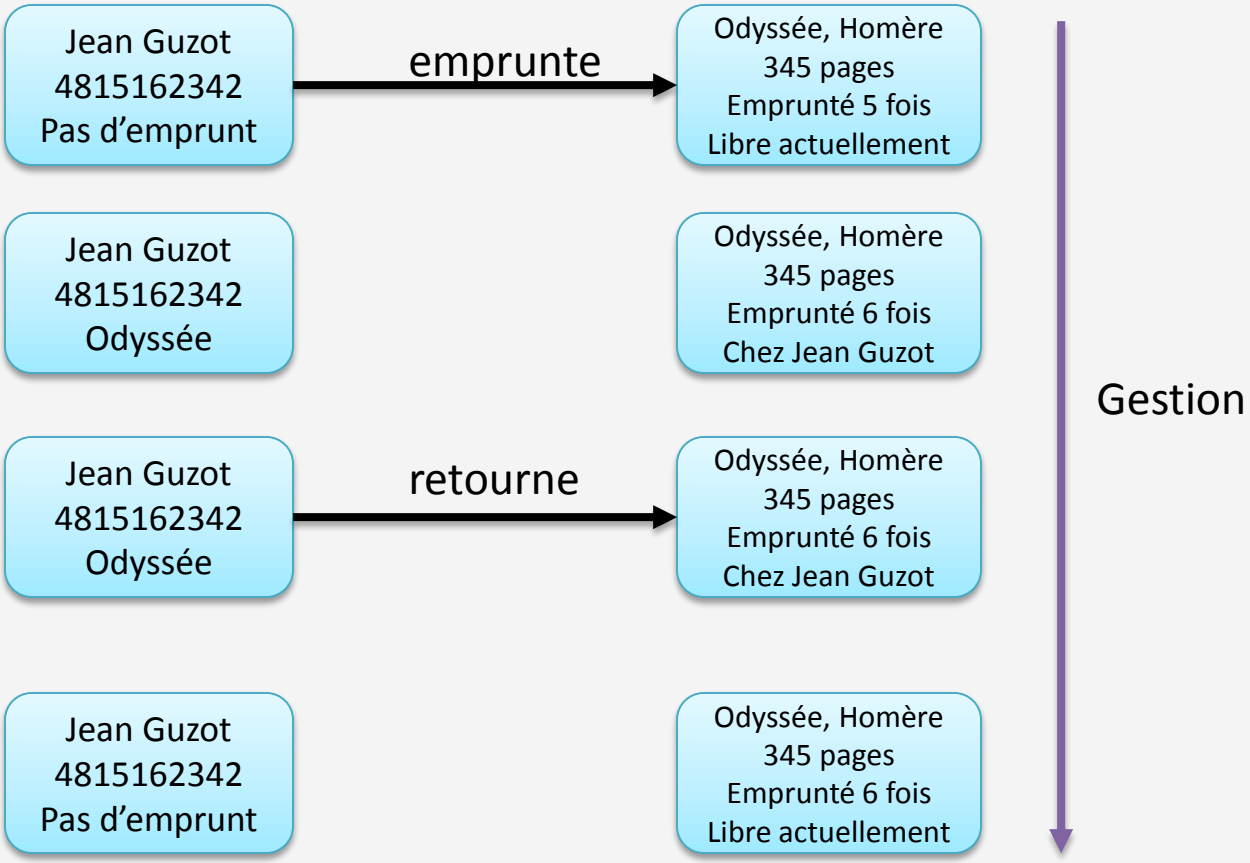
- *Un adhérent est décrit par un nom, un numéro de membre et peut emprunter et rendre un et un seul livre à la bibliothèque, sachant qu'il ne peut emprunter un livre si celui-ci est déjà prit par quelqu'un d'autre.*

- *Un livre est décrit par son titre, son auteur, son nombre de pages, l'adhérent l'ayant emprunté actuellement et son nombre total d'emprunts.*

*Le logiciel devra être capable de simuler l'inscription de nouveaux adhérents, l'achat de nouveaux livres et l'emprunt d'un livre par un adhérent ainsi que son retour.*

# Problématique

## Types de données complexes



# Problématique

## Approche algorithmique

- 2 types complexes liés

```
type Livre
données
  titre: chaine
  emprunteur: Adherent
  nbEmprunt: entier
fintype
```

```
type Adherent
données
  nom: chaine
  auteur: chaine
  num: entier
  livre: Livre
fintype
```

- Gestion des ensembles d'adhérents et de livres par des tableaux.

```
adherents: Adherent[]

livres: Livre[]
```

# Problématique

## Approche algorithmique

- Emprunt et rendu de livre = algorithmes

```
algorithme emprunte
entrées
  ad: Adherent
  l: Livre
sorties
  ok: booléen
début
  si l.emprunteur = null alors
    ad.livre ← l
    l.emprunteur ← ad
    l.nbEmprunt ← l.nbEmprunt + 1
    ok ← vrai
  sinon
    ok ← faux
  finsi
fin
```

```
algorithme rend
entrées
  ad: Adherent
  l: Livre
sorties
  ok: booléen
début
  ad.livre ← null
  l.emprunteur ← null
  l.nbEmprunt ← l.nbEmprunt - 1
  ok ← vrai
fin
```

# Problématique

## Approche algorithmique

- Réponse au problème posé
- Représentation optimale ? ***NON***
  - emprunter / rendre relatif à **Adhérent**
  - Nombre d'emprunts relatifs seulement à **Livre**
- Cohérence de la solution ? **SOUMISE A CAUTION**
  - **ex:** *un adhérent rend un livre qu'il n'a pas emprunté*)
  - **ex:** *un livre peut avoir un nombre négatif d'emprunts*)

# Problématique

## Approche algorithmique

```
algorithme rend
entrées
  ad: Adherent
  l: Livre
sorties
  ok: booléen
début
  ad.livre ← null
  l.emprunteur ← null
  l.nbEmprunt ← l.nbEmprunt - 1
  ok ← vrai
fin
```

```
algorithme test
données
  adA: Adherent
  adB: Adherent
  l1: Livre
  l2: Livre
début
  emprunte(ad1, l1)
  rend(ad2, l1)
  rend(ad1, l1)
  emprunte(ad1, l2)
fin
```

- ad2 rend le livre l1 alors qu'il ne l'a pas emprunté
- ad1 rend le livre l1 alors qu'il ne l'a plus, le nombre d'emprunts de l1 étant 0, le nouveau nombre devient -1

# Problématique

## Approche algorithmique

- Possibilité de corriger les algorithmes afin de les rendre cohérents

```
algorithme rend
entrées
  ad: Adherent
  l: Livre
sorties
  ok: booléen
Début
  si ad.livre = l et l.emprunteur = ad alors
    ad.livre ← null
    l.emprunteur ← null
    l.nbEmprunt ← l.nbEmprunt - 1
    ok ← vrai
  sinon
    ok ← faux
fin
```

# Problématique

## Approche algorithmique

- Solution obtenue (types + algorithmes) lourde
- Problème de modularité:
  - Les algorithmes **doivent accéder** à tous les champs
  - Les algorithmes **peuvent modifier** tous les champs
- Problème principal: **séparation des données et des traitements**
- Nécessité de lier certains traitements aux données

# Problématique

## Approche objet

- Voir les acteurs de notre problème comme des entité ayant des **attributs** et des **comportements**
  - Un adhérent n'est **pas seulement** un nom, un numéro et un livre emprunté
  - Un adhérent est capable de réaliser des **actions** comme *emprunter* un livre ou le *rendre*
  - Un livre est capable de dire quelle est la personne qui l'a emprunté actuellement et combien de fois au total il l'a été
- Formaliser cette approche: le **Paradigme Objet**

# Paradigme Objet

## Définition

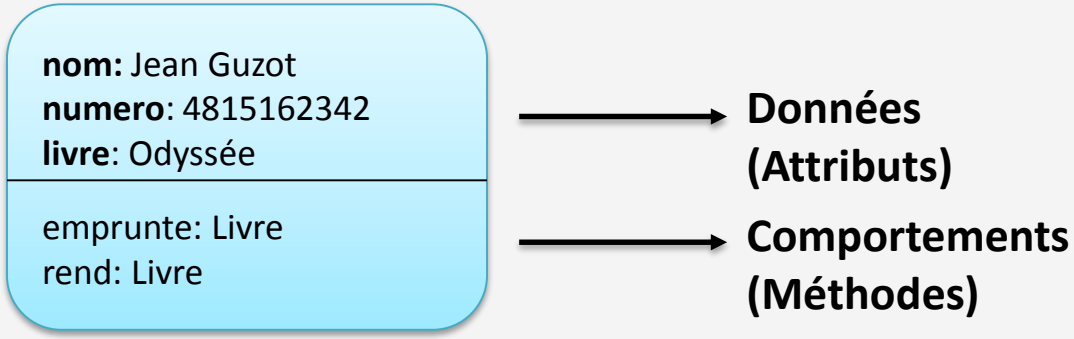
**Représentation** d'entités réelles ou abstraites par un ensemble d' **attributs** et de **comportements**. Chaque entité est appelée **objet**.

- **Paradigme**: manière dont les solutions aux problèmes doivent être formulées dans un langage de programmation.
- On peut également parler de **Modèle Objet**
- Vocabulaire:
  - Les **attributs** (ou **champs**) représentent les données liées à l'objet
  - Un comportement est également appelé **méthode**

# Paradigme Objet

## Définition

**Exemple:** *Un adhérent est décrit par un nom, un numéro de membre et peut emprunter et rendre des livres à la bibliothèque, sachant qu'il ne peut emprunter un livre si celui-ci est déjà prit par quelqu'un d'autre.*




- Les notations emprunte: Livre et rend: Livre signifient que l'objet représenté peut effectuer l'action emprunte / rend sur un objet Livre

# Paradigme Objet

## Cas réel

- Représentation de grands **ensembles** d'objets
  - Bibliothèque universitaire (milliers d'adhérents / livres)
  - Parc automobile (dizaines de véhicules)
- Nécessité d'identifier chaque objet de manière unique (**référence**)
  - Numéro d'adhérent, ISBN
  - Numéro d'immatriculation

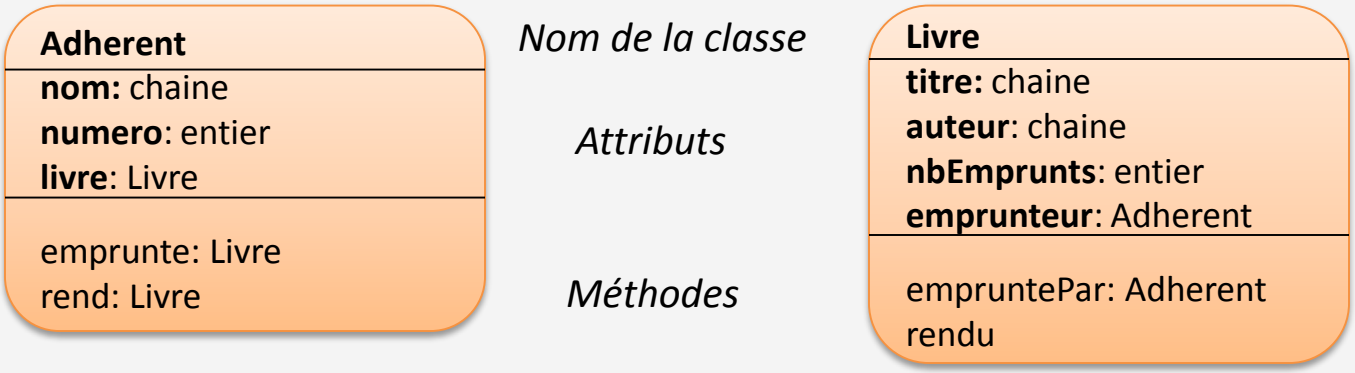
# Paradigme Objet

## Classe

**Une classe** est un ensemble d'attributs et de comportements **communs** à un ensemble d'objet.

- Permet de regrouper des objets partageant des **traits communs**.
- **Identifiée** par un **nom unique**.

**Exemple:** *La classe Adhérent et la classe Livre*

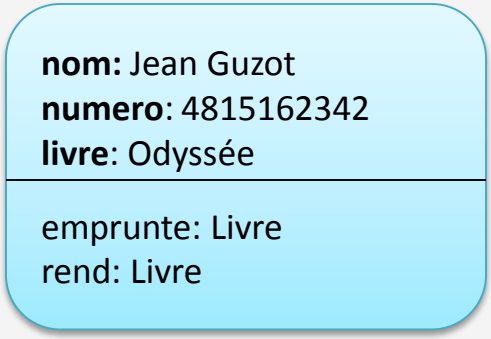


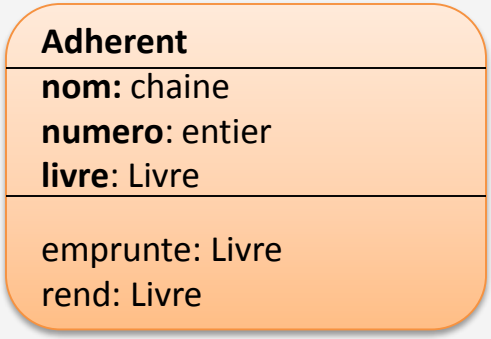
# Paradigme Objet

## Instance

**Une instance de classe** est un objet particulier appartenant à une classe.

**Exemple:** *Jean Guzot est une **instance** de la classe Adherent*




- En langage naturel, on marque l'instanciation par le verbe « **est** »

**Exemple:** *Jean Guzot **est** un Adherent*

# Paradigme Objet

## Instanciation

**Action** de **créer** un nouvel objet depuis une classe.

- La classe est vue comme un moule permettant de créer des objets partageant les mêmes attributs et méthodes (**existence**).
- En informatique, instancier un objet peut souvent être assimilé à la création ou à la réservation de ressources systèmes (mémoire) destinée a rendre le nouvel objet utilisable.
- Les langages orientés objet permettent la création et la gestion d'objets, c'est le cas de Java.

# Paradigme Objet

## Référencement

- Une instance doit pouvoir être identifiée et retrouvée par le système. Pour cela, chaque objet instancié reçoit une **référence**.

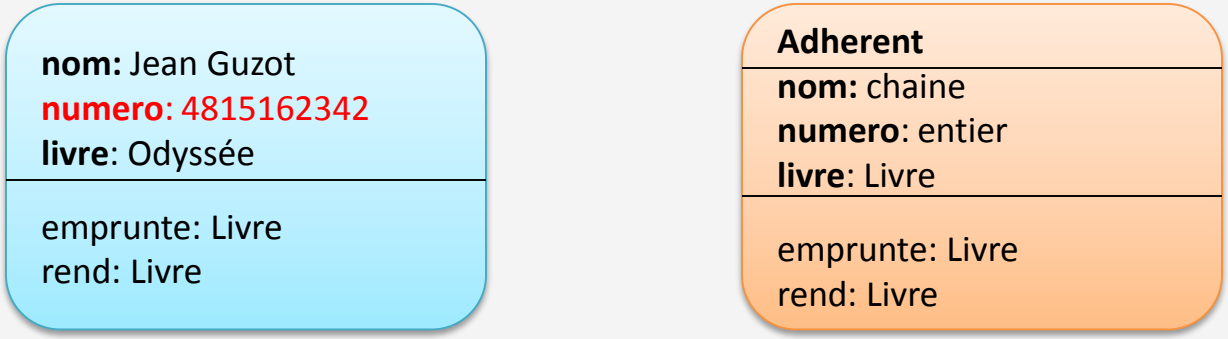
Une **référence** est un **identifiant** permettant au système gérant les objets de les identifier de façon **unique**.

- Dans le cadre de l'utilisation du paradigme objet sur un **système informatique**. La référence peut être par exemple l'**adresse mémoire** à laquelle se trouve l'objet.

# Paradigme Objet

## Référencement

**Exemple:** *Jean Guzot est une **instance** de la classe Adherent*




- Dans notre exemple, la référence d'un objet de la classe Adhérent peut être son numéro d'adhérent.
- **ATTENTION:** Dans la plupart des système de programmation orienté objets, le référencement est géré en interne et il n'est pas possible de spécifier explicitement les références. Chaque objet obtient une **référence** lors de son **instanciation**.

# Paradigme Objet

## Représentation

- La représentation d'un problème dans le paradigme objet est obtenus par:
  - Le recensement des objets intervenant dans le problème
  - L'extraction des données et des comportements à représenter selon le point de vue du problème (**abstraction**)
  - L'expression du problème en **classes**, **attributs** et **méthodes**
  - La mise en place d'un mécanisme d'**instanciation**
  - La mise en place d'un mécanisme de **référencement**

# Paradigme Objet

## Représentation

**Exemple:** *Créer un logiciel capable de simuler la gestion d'une bibliothèque.*

- *Une bibliothèque est composée d'un ensemble de livres et d'un ensemble d'adhérents.*

- *Un adhérent est décrit par un nom, un numéro de membre et peut emprunter et rendre un et un seul livre à la bibliothèque, sachant qu'il ne peut emprunter un livre si celui-ci est déjà prit par quelqu'un d'autre.*

- *Un livre est décrit par son titre, son auteur, son nombre de pages, l'adhérent l'ayant emprunté actuellement et son nombre total d'emprunts.*

*Le logiciel devra être capable de simuler l'emprunt d'un livre par un adhérent ainsi que son retour.*

# Paradigme Objet

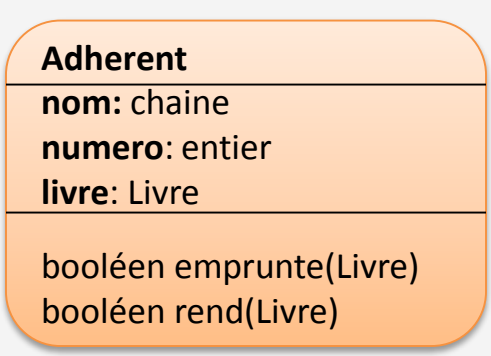
## Représentation

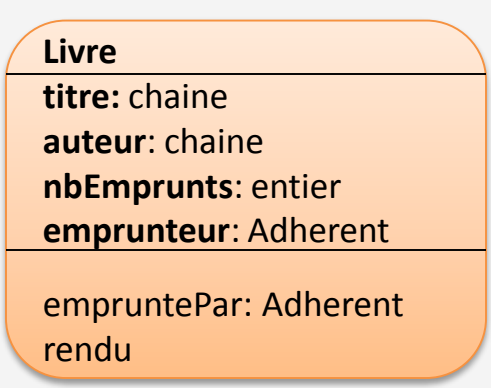
**Objets intervenants:** des adhérents, des livres

***Abstraction:***

- *un livre représenté par un titre, un nombre d'emprunts total et un emprunteur actuel. Un livre peut être emprunté ou rendu par un adhérent*
- *Un adhérent est représenté par un nom, un numéro est un livre actuellement emprunté. Un adhérent peut emprunter un livre et le rendre.*

***Classes, attributs, méthodes:***



# Paradigme Objet

## Représentation

Les mécanismes d'**instanciation** et de **référencement** seront assurés par le langage informatique utilisé pour implanter le système (Java).

- A partir de la représentation du problème, il est possible de passer à l'implantation des fonctionnalités demandées.