Plan

- Introduction au Java
  - Généralités
  - Syntaxe de base
- Concepts et modélisation orientée objets
  - **Objet, classe et modélisation UML**
  - Les principes fondamentaux: encapsulation, abstraction, héritage et polymorphisme
- Programmation Orientée Objets en Java
  - Classes, objets et bonnes pratiques
  - Héritage, interfaces, agrégation, composition et association
  - Packages
  - Généricité
  - Exceptions
  - Flux d'Entrée/Sortie et fichiers
  - Empaqueter et déployer son programme

Constat

- Etude réalisée en 1995 aux Etats-Unis
  - 31% des logiciels abandonnés ou jamais utilisés
  - 53% des logiciels ont un coût dépassant largement les prévisions initiales
  - 16% réalisés suivant les prévisions initiales

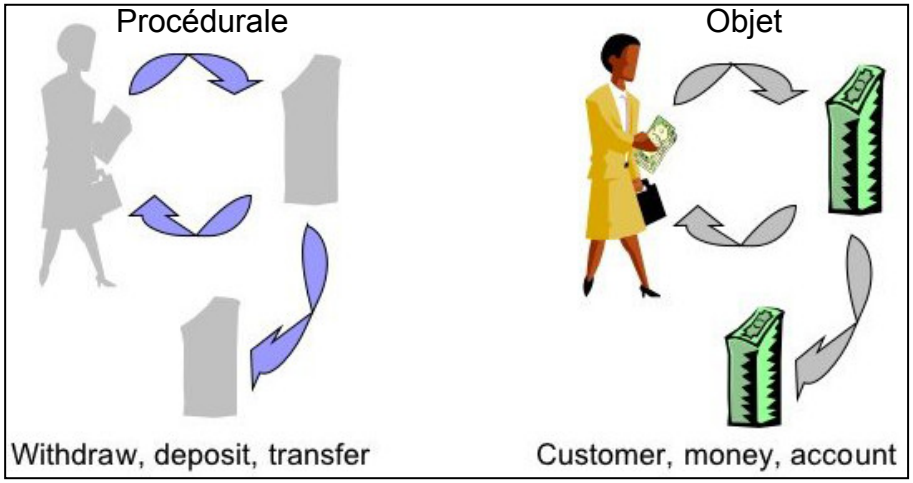
➢ Une partie de ces problèmes était liée à la façon de concevoir un logiciel
  - Pas de réelle méthodologie de conception et des langages non adaptés à des projets importants

- Les limites de la programmation impérative structurée (p.ex. C)
  - Difficile à faire évoluer et à maintenir
  - Favorise peu la réutilisabilité du code

➢ Nécessité d'un nouveau modèle de programmation ➔ introduction de la modélisation orientée objets
  - Réfléchir et modéliser avant de coder
  - Une modélisation indépendante du langage de programmation

Procédurale versus objet

- Approche procédurale : que doit faire mon programme ?
- Approche objet : de quoi doit être composé mon programme ?
  - Nécessité d'identifier les entités mises en jeu
  - Nécessité d'identifier les relations entre ces identités



*Procedural VS. Object Oriented Programming, Alphanso Tech, 2016

- Principales différences en orientée objets :
  - Division du code, focus sur les données plutôt que sur les fonctions, approche de conception ascendante (*bottom-up*), mécanismes d'encapsulation et de masquage du code, factorisation des entités, et possibilité de surcharger (spécifier) un comportement

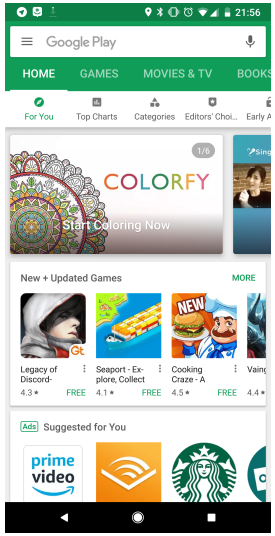
La notion d'objet

- Qu'est-ce qu'un objet ?
  - **Modélise** toute entité identifiable, concrète ou abstraite, manipulée par l'application logicielle
    - objets concrets : une ville, un étudiant, une horloge, un bouton sur l'écran
    - objets abstraits : une réunion, un compte bancaire, une chaîne de caractères
  - Réagit à certains messages qu'on lui envoie de l'extérieur; la façon dont il réagit détermine le **comportement** de l'objet
  - Ne réagit pas toujours de la même façon à un même message; sa réaction dépend de l'**état** dans lequel il se trouve

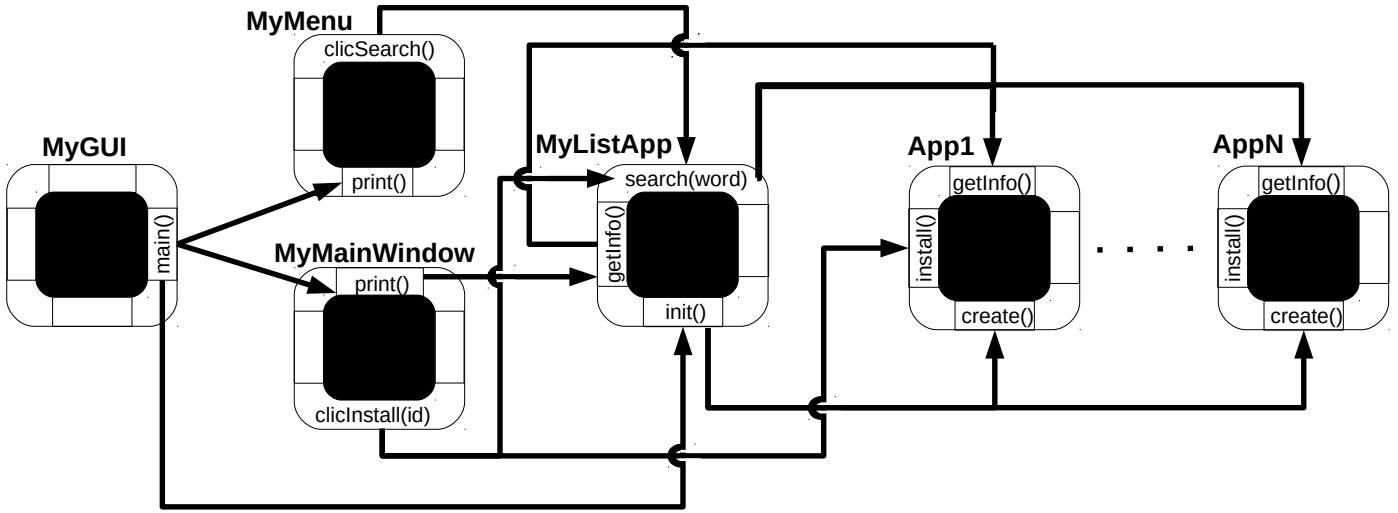
Objet : une maison
Etat : adresse, couleur, superficie
Comportement : ouvrir la porte, fermer la porte

- Autrement dit, un ensemble de données et des fonctions qui lui sont associées
  - Etat interne = un ensemble **d'attributs** (des valeurs de variables)
  - Comportement = des **méthodes** (des fonctions et des procédures)
  - Message = invocation d'une méthode

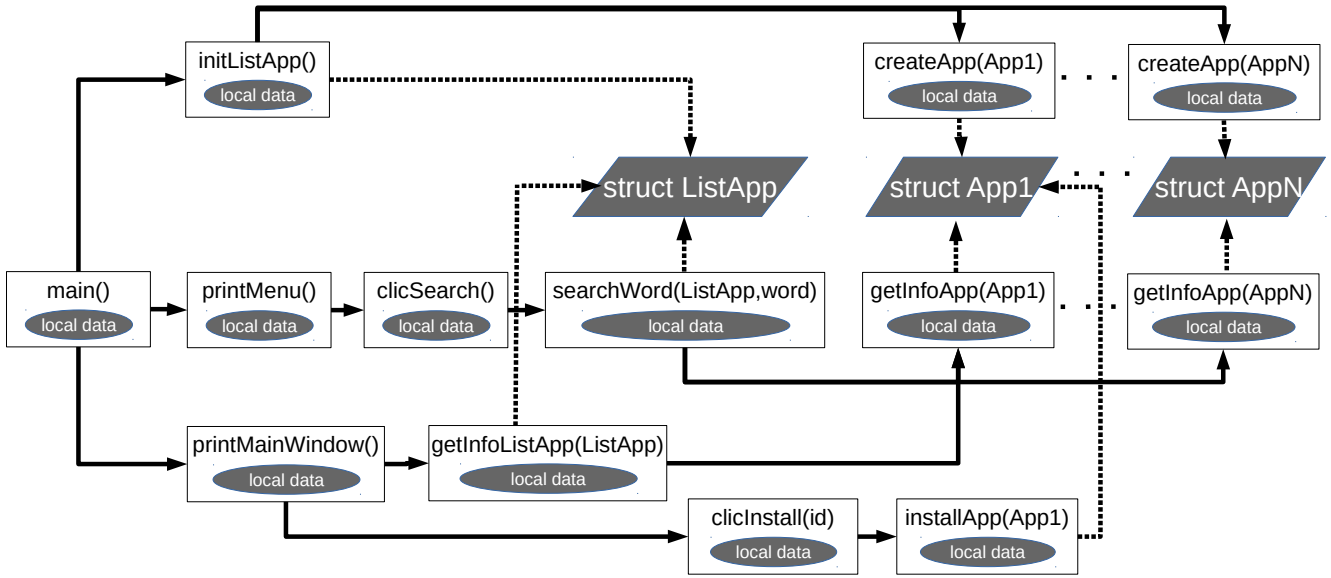
Exemple (très très) simplifié d'un logiciel du type Google Play (magasin d'applications)



Programmation Orientée Objets (POO)



Programmation Orientée Procédures (POP)

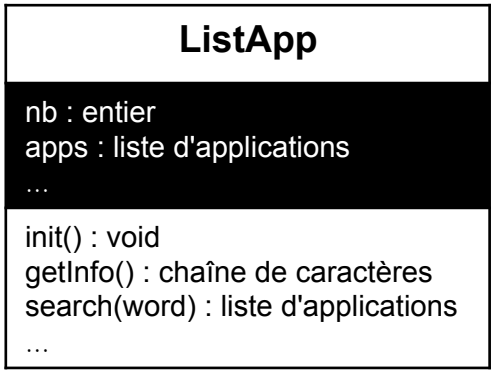


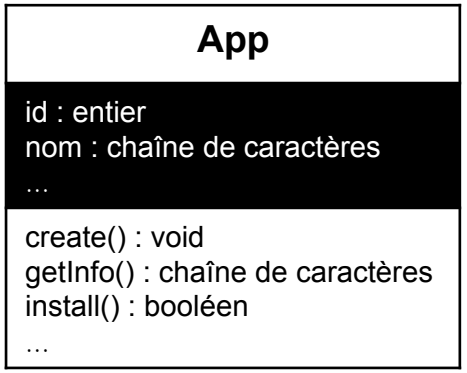
La notion de classe

- Dans un programme, on utilise souvent plusieurs objets du même type
- Introduction de la notion de classe

- Qu'est-ce qu'une classe ?
  - Un type / une description d'objets (un "moule")

  - Une classe se compose
    - d'une description de l'état interne, i.e. des déclarations d'attributs
    - d'une description du comportement de ce type d'entité, i.e. des déclarations de méthodes





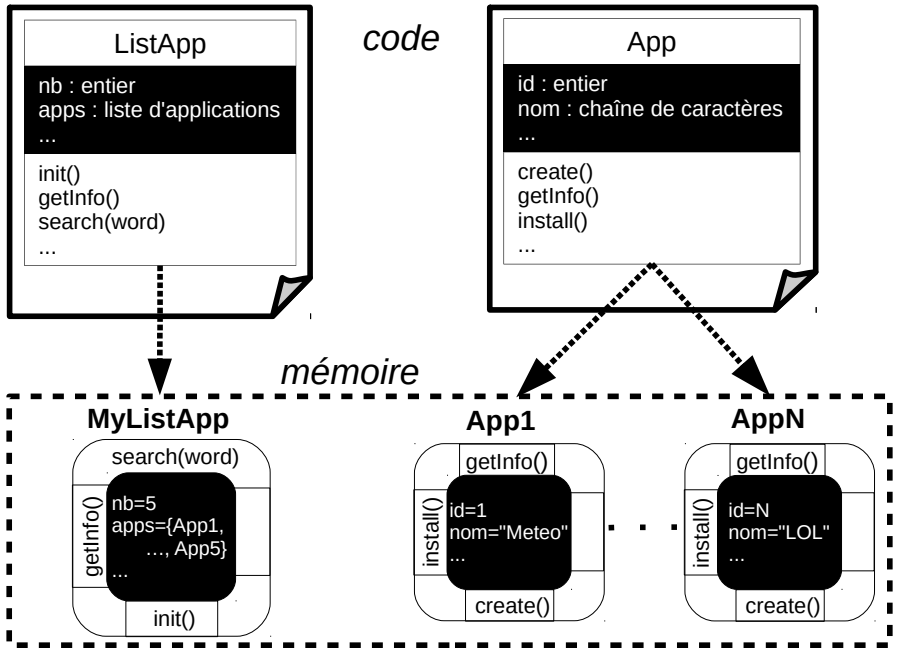
La notion de classe

- Relation classe/objet similaire à celle entre les types primitifs et les variables

```
int numéro ;
ListApp myListApp ;
```

- On dit qu'un objet est une **instance** d'une classe

- Des objets instanciés à partir d'une même classe possèdent
  - des valeurs d'attributs distinctes, mais des attributs avec les mêmes types et les mêmes noms
  - des méthodes identiques



Modéliser les logiciels avec UML (Unified Modeling Language)

- Langage de modélisation permettant de représenter sous forme de diagrammes
  - l'architecture d'un logiciel
    - **diagramme de classes**, diagramme de composants, diagramme de structure composite (fonctionnement interne d'une classe), diagramme de déploiement, diagramme d'objets, diagramme de paquetages (espaces de nommages), …
  - son fonctionnement
    - diagramme de cas d'utilisation (fonctionnalités globales), diagramme d'activités (les processus), diagramme de séquences (interactions entre les acteurs et le logiciel), diagramme de communication, diagramme d'états, …

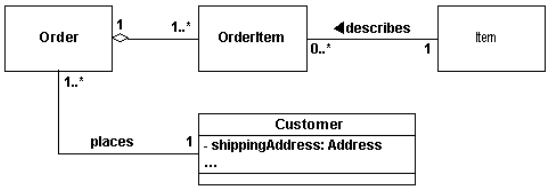


Diagramme de classes *

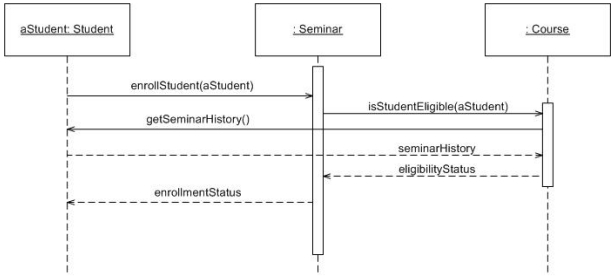


Diagramme de séquences *

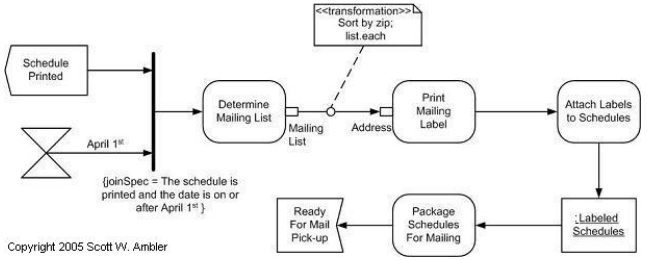
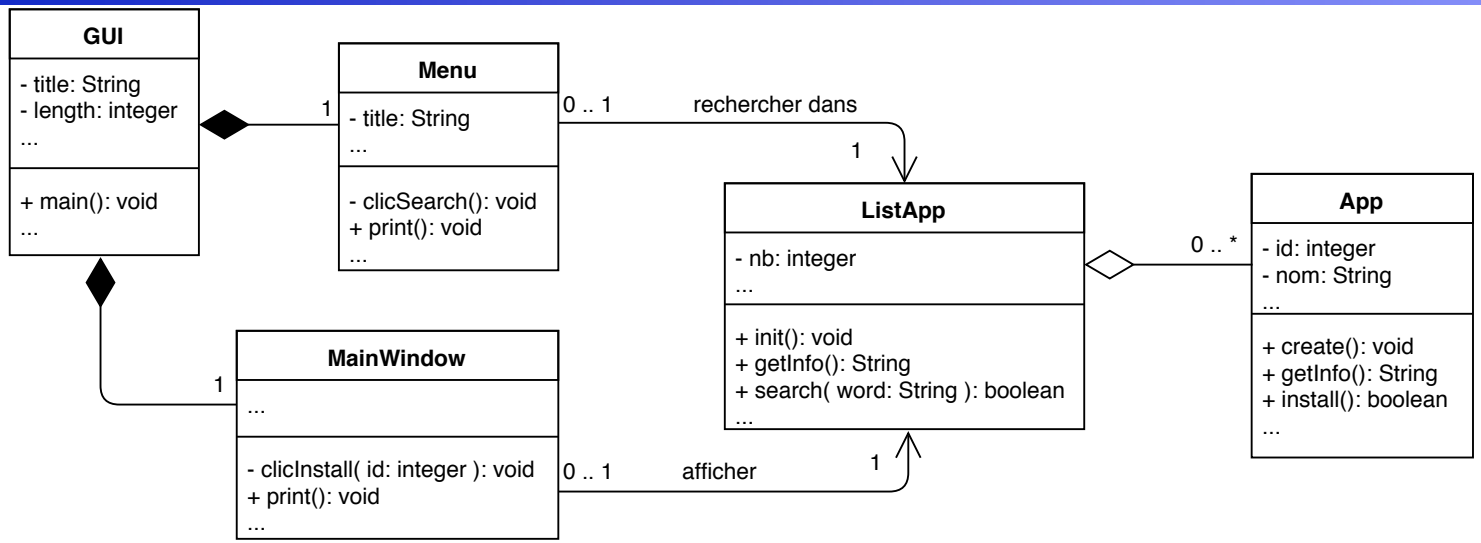


Diagramme d'activités *

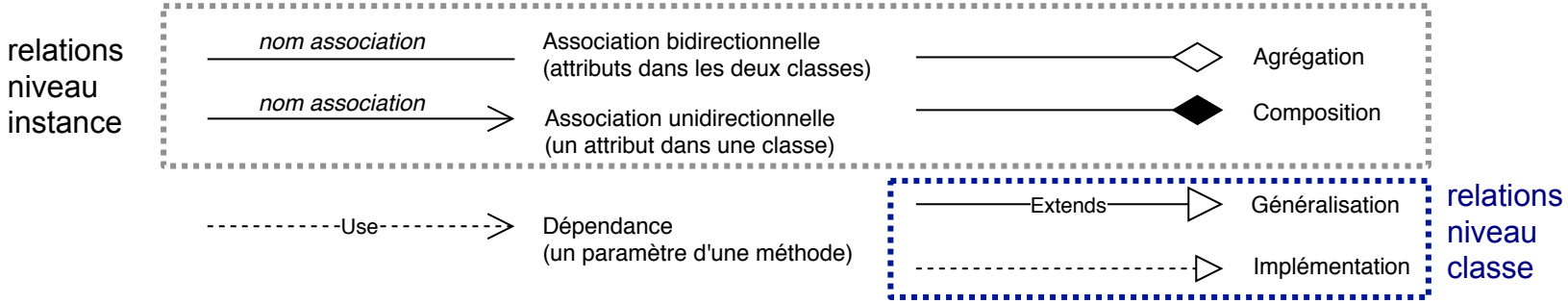
* http://agilemodeling.com/essays/umlDiagrams.htm

- Un standard de l'OMG (Object Management Group) et de l'ISO (International Organization for Standardization)
- ➢ Facilite le dialogue entre les différents acteurs du développement

Diagramme de classes UML



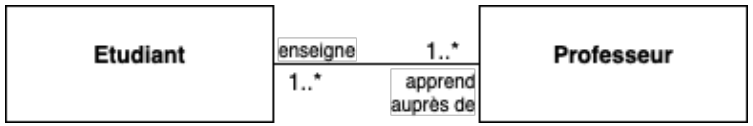
- Visibilité des attributs et méthodes : + (public), - (privé) et # (protégé)
- Principales relations entre classes



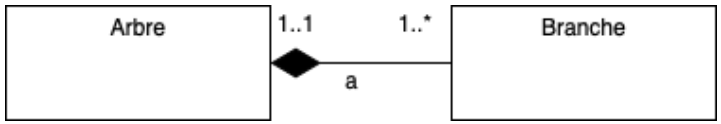
- Cardinalités : 0..1, 1, 1..1, 0..*, *, 1..*

Exercice: quelques cas simples

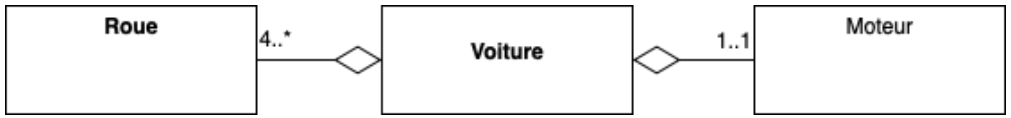
- Modéliser les cas suivants:
  - un étudiant étudie auprès de professeurs



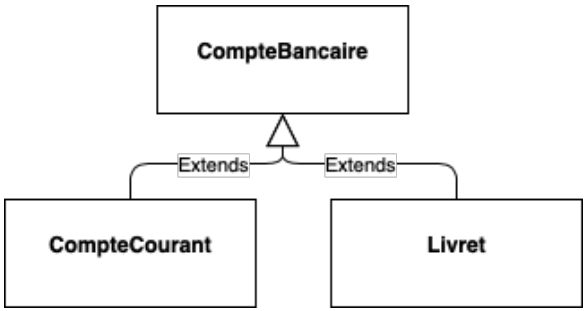
  - un arbre a des branches



  - une voiture a un moteur et des roues



  - un compte bancaire peut être un compte courant ou un livret



  - un répertoire contient des fichiers
  - une pièce contient des murs
  - les modems et claviers sont des périphériques d'entrée/sortie
  - une transaction boursière est un achat ou une vente
  - un compte bancaire peut appartenir à une personne physique ou morale

Plan

- Introduction au Java
  - Généralités
  - Syntaxe de base
- Concepts et modélisation orientée objets
  - Objet, classe et modélisation UML
  - **Les principes fondamentaux: encapsulation, abstraction, héritage et polymorphisme**
- Programmation Orientée Objets en Java
  - Classes, objets et bonnes pratiques
  - Héritage, interfaces, agrégation, composition et association
  - Packages
  - Généricité
  - Exceptions
  - Flux d'Entrée/Sortie et fichiers
  - Empaqueter et déployer son programme

Les principes fondamentaux en POO

- **Encapsulation**
  - Regrouper attributs (états) et méthodes (comportements) ensemble

  ➢ Intérêt : une meilleure organisation du code
    ➢ facilite la maintenance et l'évolution de l'application

App
- id: integer - nom: String ...
+ create(): void + getInfo(): String + install(): boolean ...

- **Abstraction**
  - Montrer les attributs/méthodes pertinents (+)
  - Masquer les détails d'un objet (- et #)
    – mise en en place de méthodes "*getter*" et "*setter*" pour modifier les attributs cachés

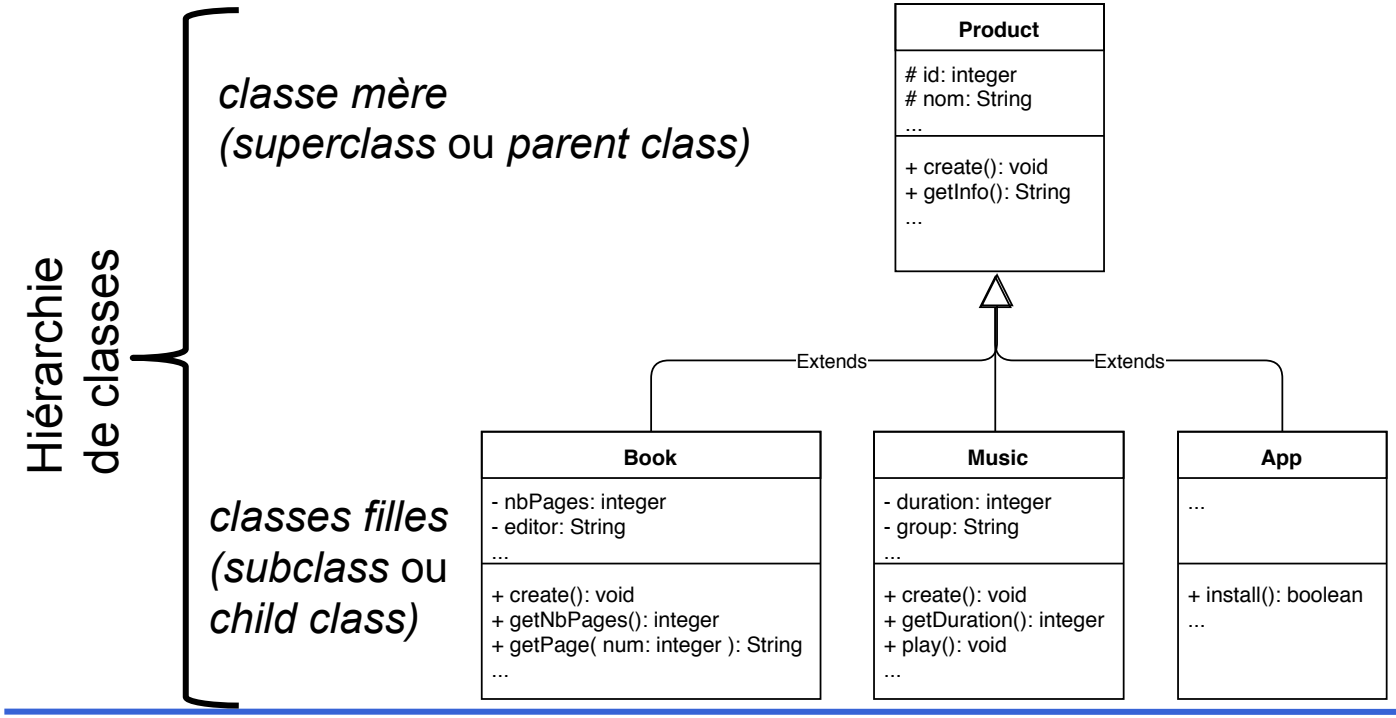
+ getId(): integer + getNom(): String

  ➢ Intérêt : une modification du code "caché" n'impact pas le reste de l'application

Les principes fondamentaux en POO

- **Héritage** (*inheritance en anglais*)
  - Acquisition par une **classe** des attributs et méthodes, publiques (+) ou protégées (#), d'une autre **classe**

➢ Intérêt : favorise la réutilisation du code
  – facilite la maintenance et l'évolution de l'application



Possibilité d'ajouter de nouveaux attributs et/ou méthodes

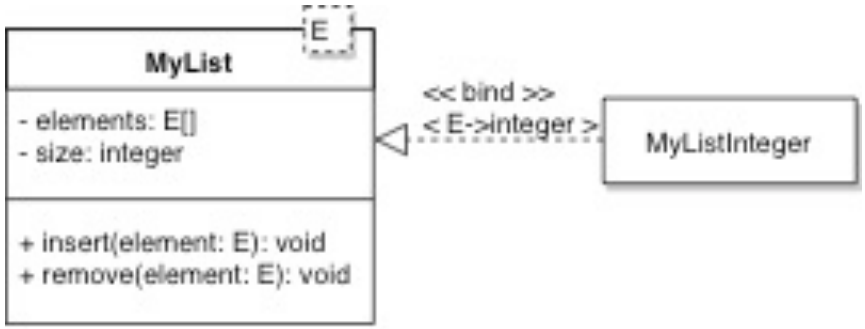
Possibilité de modifier ou redéfinir les méthodes héritées

Les principes fondamentaux en POO

- **Polymorphisme** (*polymorphe=qui peut se présenter sous plusieurs formes*)
  - Le polymorphisme statique (i.e. résolu à la compilation)
    - La **surcharge** de méthode (*method overloading*): méthodes ayant le même nom mais des signatures différentes
      signature = nom de la méthode + types des paramètres

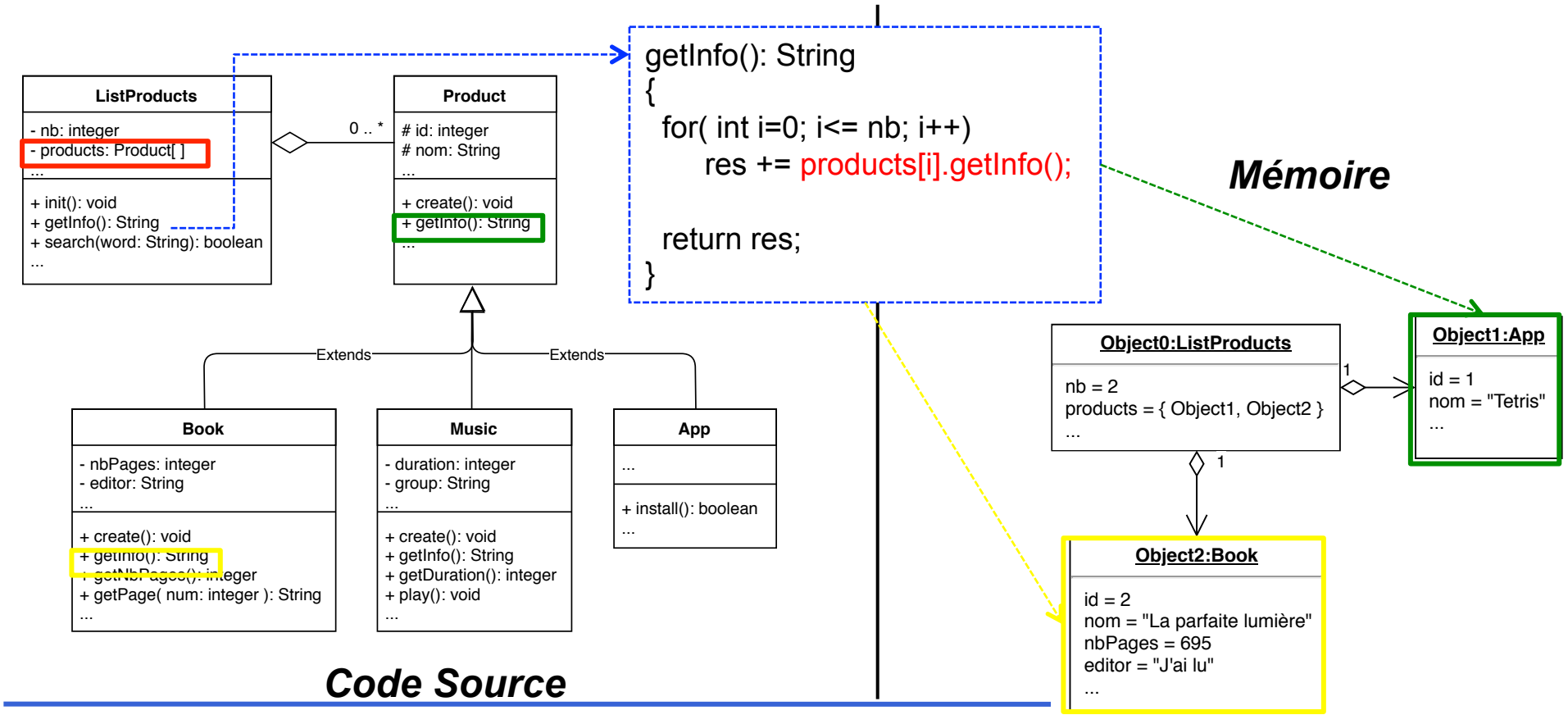
```
+ addition( a: integer, b: integer ): integer
+ addition( a: float, b: float ): float
```

    - Les méthodes et/ou classes **génériques** (*templates*)**:** un même code peut être appliqué à des données de n'importe quel type



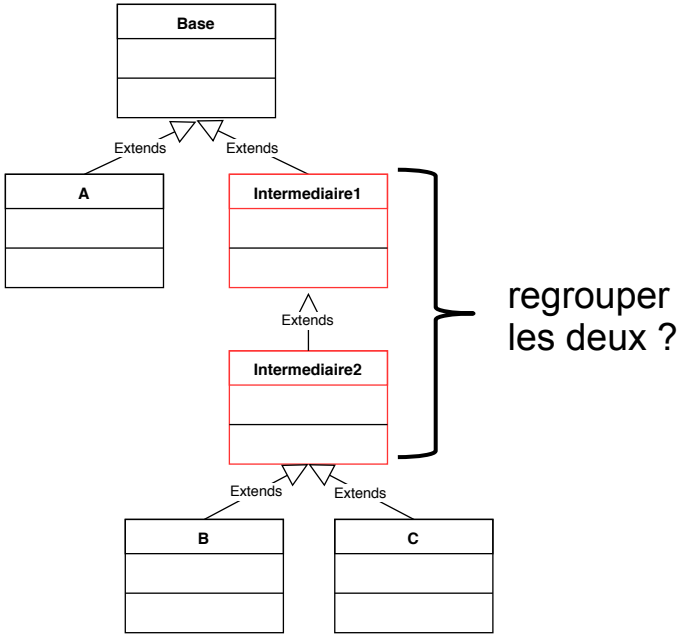
Les principes fondamentaux en POO

- **Polymorphisme** (suite)
  - Le polymorphisme dynamique (i.e. résolu à l'exécution)
    - La **redéfinition** de méthode (*method overridden*): un même code peut être appliqué à tout objet appartenant à la même hiérarchie de classes

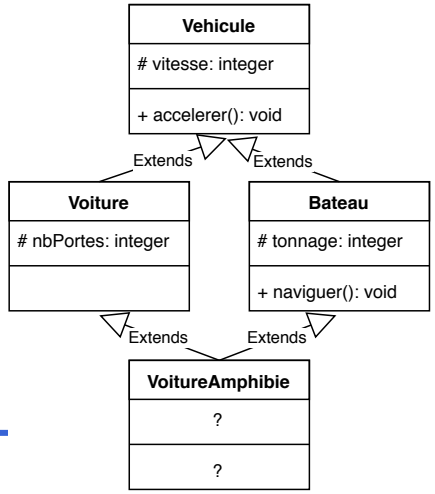


Les dangers de l'héritage

- **Erreurs de conception**
  - Création de classes superflues
  - Héritage d'attributs ou de méthodes inadaptés
    - p.ex. pingouin dérive d'oiseau mais ne vole pas
  - Faire une classe dérivée là où un attribut suffit
    - p.ex. classes *ChienNoir* et *ChienJaune*

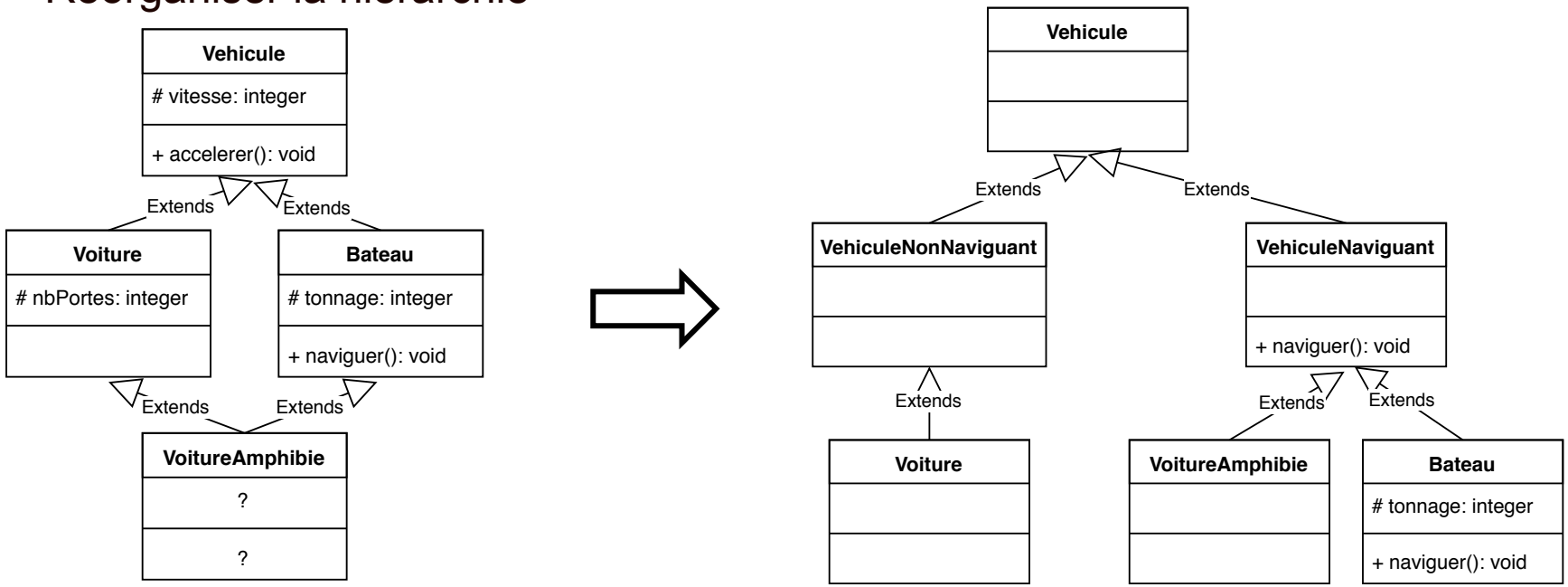


- **Problème de l'héritage multiple**
  - Une classe hérite de plusieurs classes mères
  - ➢ Attention à l'héritage à répétition
    - p.ex. héritage de deux attributs *vitesse* et de deux méthodes *accelerer()*



Les solutions aux problèmes liés à l'héritage

- Réorganiser la hiérarchie

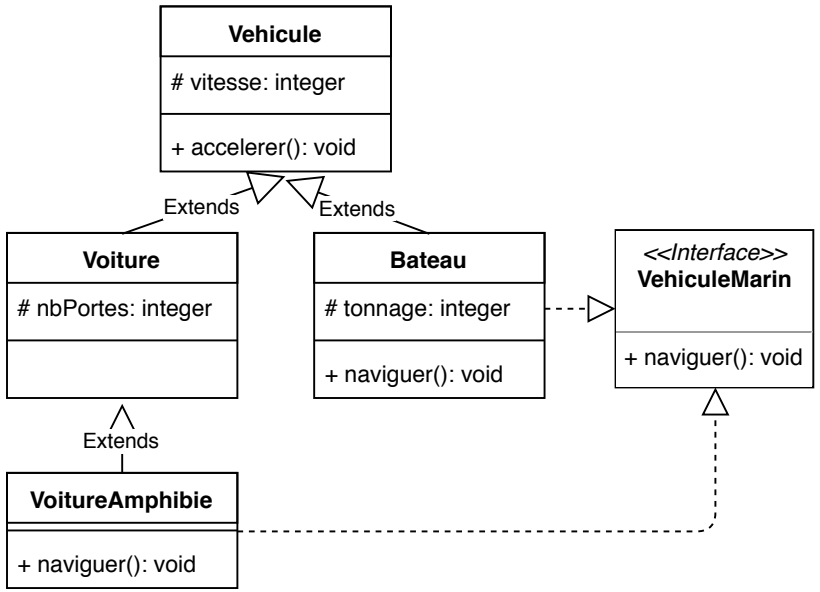


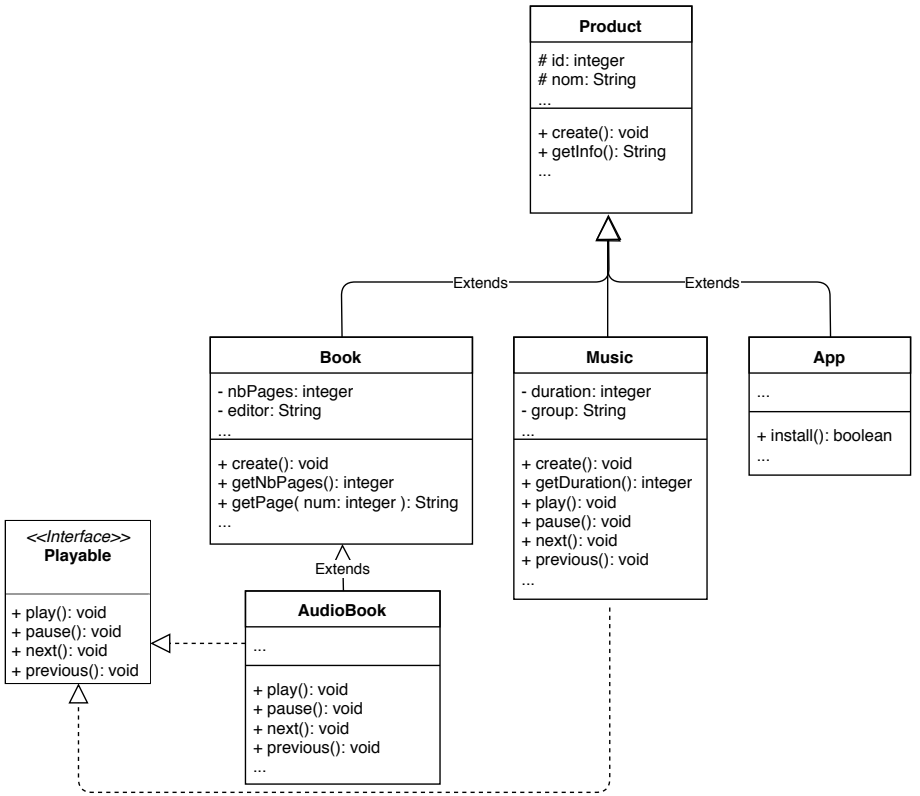
- Ajouter/modifier des méthodes
  - p.ex. ajouter une méthode à *Vehicule* indiquant si le véhicule peut naviguer
  - p.ex. modifier la méthode *naviguer()* de façon à ce qu'elle retourne faux pour les voitures

Les solutions aux problèmes liés à l'héritage

- Utiliser la notion d'interface de classes
  - **Interface** = un "contrat" indiquant un ensemble de spécifications minimales (des méthodes) que **devront** implémenter des classes
    - important pour des groupes de développeurs différents de se mettre d'accord sur comment leur code va interagir (sans forcément en connaître les détails)

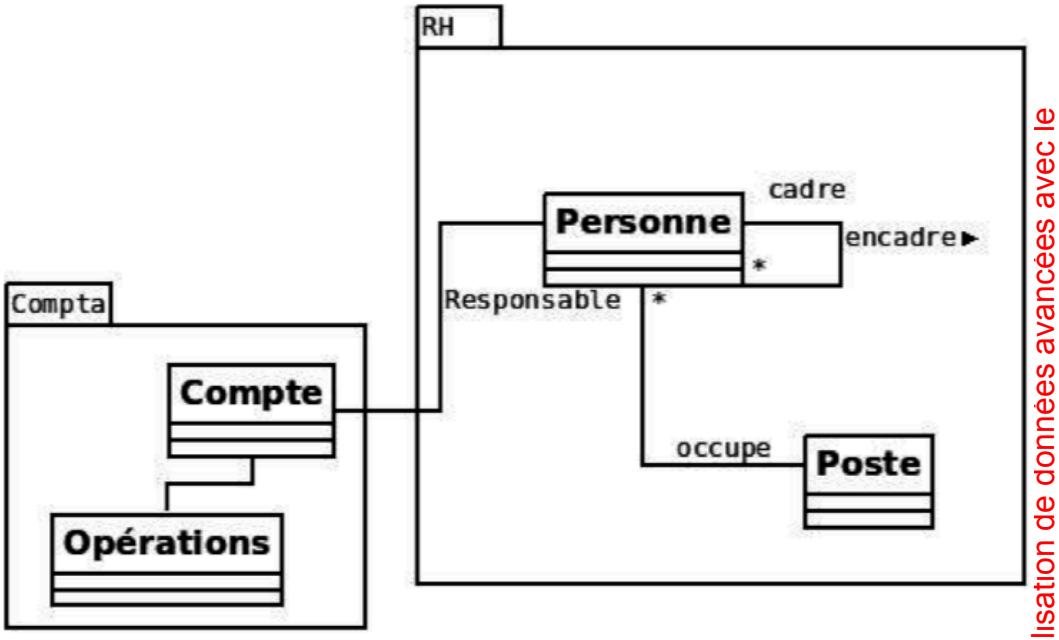
➢ Pas d'instanciation possible





Les packages

- Un espace de nommage servant à organiser un ensemble de classes et d'interfaces reliées entre elles
  - Concept similaire à celui de répertoire
  - Ex. la libraire Java (API ou *Application Programming Interface*) composée de plus de 6000 classes organisées en packages
    - p.ex. java.util (ArrayList, Calendar, Scanner, ⋯), java.math (BigDecimal,⋯), …
    - https://docs.oracle.com/javase/8/docs/api/index.html



Modélisation de données avancées avec le digramme de classes UML, Stéphane Crozat, UTC.

Exercice: modélisation d'un logiciel de gestion des prêts dans des bibliothèques

- L'objectif est de développer un système de gestion des documents de bibliothèques (pour une utilisation dans le terminal).
- Une bibliothèque est identifiée par un nom, une adresse, un mail de contact et un numéro de téléphone. Elle contient un certain nombre de documents disponibles à l'emprunt et/ou à la consultation en ligne. Ces documents sont des livres, des CD audio ou des vidéos. Les livres sont uniquement empruntables. Les CD audio sont empruntables et consultables en ligne. Les vidéos sont uniquement consultables en ligne. Par ailleurs, les personnes désirant emprunter ou consulter des médias doivent s'inscrire en tant que client.
- L'inscription des clients consiste à saisir dans l'application les informations suivantes : nom et prénom du client ainsi que son adresse. Le nombre de documents empruntables et la durée de prêt sont identiques pour toutes les bibliothèques. Tout client qui n'a pas restitué un document avant sa date limite de restitution ne pourra plus faire de nouvel emprunt tant qu'il n'aura pas régularisé sa situation.
- Les documents sont associés à une référence (identifiant), un titre, un auteur, et une année de sortie. Afin de disposer de statistiques d'utilisation, on souhaite connaître le nombre d'emprunts effectués par document. De manière plus spécifique, chaque livre est caractérisé par un nombre de pages, et repéré par une localisation (salle/rayon) dans la bibliothèque. Un CD audio est composé de différentes pistes avec chacune un titre, une durée, et un compositeur. Une vidéo par une durée et une langue.
- Chaque sortie de document (livre ou CD audio) entraîne la constitution d'une fiche d'emprunt. Sur cette fiche sont notés, le client emprunteur, la date de début du prêt, les documents empruntés et la date limite de restitution. Le système de gestion doit permettre l'ajout et la suppression de clients et de documents. Il doit aussi permettre de consulter les documents disponibles, ainsi que les informations d'un client.

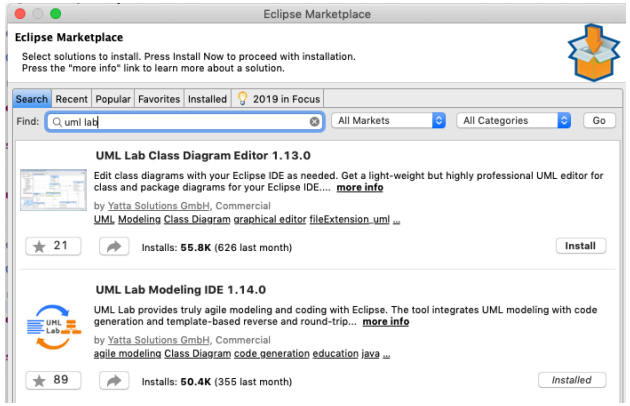
Exercice: modélisation d'un logiciel de gestion des prêts dans des bibliothèques

- Intégrer une partie de ce modèle dans Eclipse (se focaliser sur les classes *Document*, *Bibliotheque*, *Client*, et *FicheEmprunt*) en utilisant le plugin UML Lab

1. Installer UML Lab:
   - *Help > Eclipse Marketplace > Search*, écrire "UML Lab" et installer "Uml Lab Modeling"

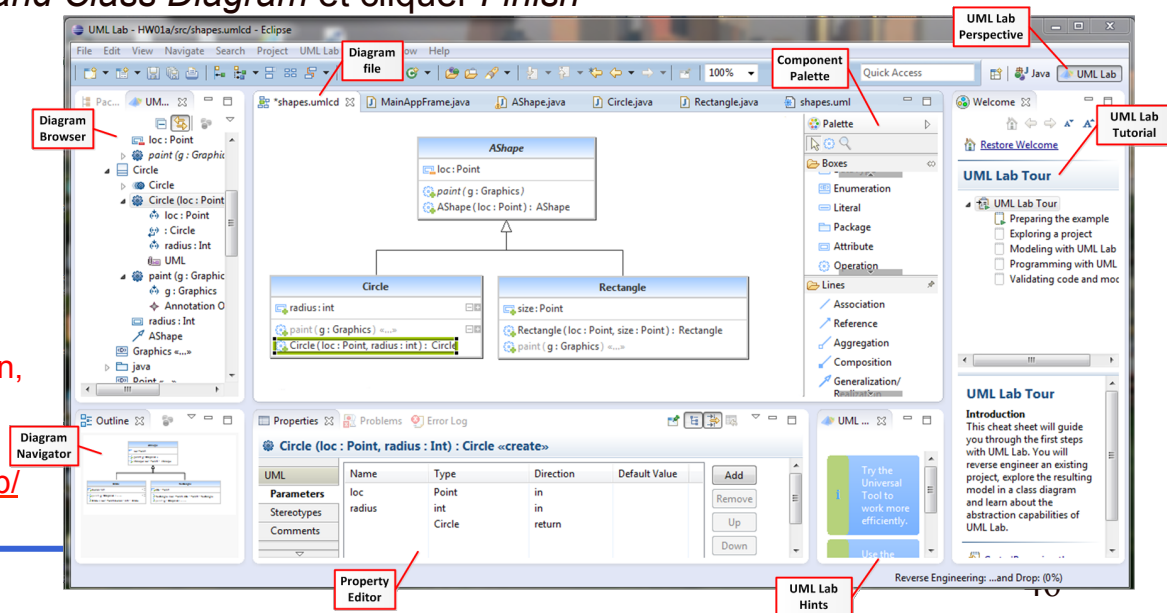
   OU
   - *Help > Install New Software*, entrer le site http://update.uml-lab.com et sélectionner "UML Lab"

   OU
   - Télécharger et installer *UML LAB INSTALLER* https://www.uml-lab.com/en/download/



2. Créer un diagramme de classe
   - *File > New > UML Model and Class Diagram* et cliquer *Finish*

3. Dessiner les classes, les attributs et les relations



Pour plus de détails cf.
Advanced Object-Oriented Programming and Design,
Dr Stephen Wong, Rice University.

https://www.clear.rice.edu/comp310/Eclipse/UMLLab/