

# Base de la Programmation Orientée Objet

(Ceci N'est PAS un cours de Java !!!)

IUT de Clermont-Ferrand  
Université d'Auvergne

25 janvier 2016

# Organisation

## 7 semaines

- 2h cours amphi : paradigme Objet + UML + Java
- 2h TD : des exercices pour travailler les notions
- 2 x 2h TP : mise en pratique avec Eclipse

## Des évaluations

- 1 à mi-parcours sur le cours en amphi
- 1 note de TP
- 1 DS final

Attention pas de support de cours pour l'amphi : *prenez des notes !*

# Objectifs

- Maîtriser les concepts objets de base
- Savoir programmer (proprement) en Java
- S'initier aux outils classiques de développement  
(IDE, debugger, ...)
- Comprendre les diagrammes de conception et pouvoir les implémenter

# Méthodologie

## Des pistes pour y arriver

- Soyez actifs (allez chercher l'information, posez des questions, participez en cours)
- Soyez concentrés en cours
- Travaillez régulièrement (faites le point sur vos connaissances)
- Faites des erreurs ! (o\_O)
- Ne croyez pas savoir avant d'avoir cherché à ré-expliquer
- Diversifiez vos sources d'informations (lisez !)



Hugues Bersini

*La programmation orientée objet : Cours et exercices en UML2 avec Java 6, C# 4, C++, Python, PHP 5 et LinQ*

Eyrolles (mars 2013)

# À la recherche de la POO

## Question

La Programmation Orienté Objet, qu'est-ce que c'est ?

- ... et d'ailleurs, pourquoi la POO ?

## Activité

Dites-moi ce que c'est pour vous que de *développer un logiciel*.

# Étapes d'un projet

## ■ Construisez une maison !



# Étapes d'un projet

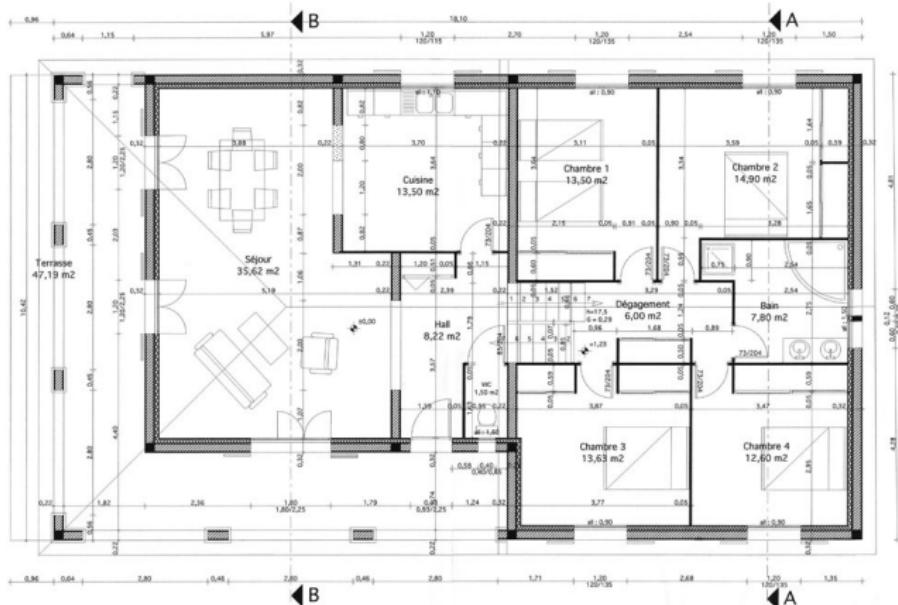


?



- Choisir le type de construction

## Étapes d'un projet



## Projet de M. et Me.

Rez-de-chaussée - Ech : 1/50

#### ■ Préparer les plans

# Étapes d'un projet



## ■ Construire

# Étapes d'un projet



■ Entretenir

# Étapes d'un projet

Processus de développement logiciel similaire

- 1** Définir les besoins (cahier des charges)
- 2** Modéliser (analyse et conception)
- 3** Développer (programmation, test, débogage)
- 4** Maintenir

# Découpage fonctionnel vs approche OO

- Problèmes rencontrés
- Avantages / Inconvénients

Il faut penser le plus tôt possible à :

- la réutilisation
- l'évolution
- la maintenance

# Notion d'objet

## Objet (*Grady Booch*)

Un objet a un état, un comportement et une identité.

- L'*identité* permet d'exploiter le *comportement* d'un objet.
- Le *comportement* agit sur l'*état* de l'objet.
- Et l'*état* de l'objet influence le *comportement*.

# Notion d'objet

## Identité

- Un objet forme un tout
- Deux objets différents ont des identités différentes
- On peut désigner un objet (y faire référence)

## État

## Comportement

# Notion d'objet

## Identité

## État

- Un ensemble de caractéristiques, ou propriétés, définies par des valeurs
- Permet de personnaliser l'objet, de le distinguer des autres objets
- Peut évoluer dans le temps

On parle d'**attributs** (*data member*)

## Comportement

# Notion d'objet

Identité

État

Comportement

- Ensemble des traitements que l'on peut effectuer sur l'objet (ou que l'objet peut accomplir)
- Fonctions décrivant les actions possibles sur/de l'objet

On parle de méthodes (*member functions*)

# Mécanique de fonctionnement

- En POO, on s'adresse à un objet par *envoi de messages*  
    → on « demande » à l'objet de faire ceci ou cela
- Envoi de message = accès à un attribut ou invocation d'une méthode

## Interface (de comportement) de l'objet

- Ensemble des manières dont on dispose pour interagir avec l'objet
- Ensemble des messages reconnus par l'objet

# Notion de classe

Certains objets présentent les mêmes caractéristiques

- Ils ont des identités différentes mais
  - ↪ un état défini sur les mêmes propriétés
  - ↪ une interface de comportement similaire

## Exemple

*Le Seigneur des Anneaux*,  
de John Ronald Reuel Tolkien,  
paru en 1954

et

*Dune*,  
de Franck Herbert,  
paru en 1965

# Notion de classe

Certains objets présentent les mêmes caractéristiques

## Exemple

*Le Seigneur des Anneaux*,  
de John Ronald Reuel Tolkien,  
paru en 1954

et

*Dune*,  
de Franck Herbert,  
paru en 1965

sont caractérisés par les mêmes propriétés  
(titre, auteur, année, texte) mais associées à des valeurs différentes

et ont la même interface de comportement (on peut les lire, imprimer, ...)

mais ont des identités différentes (ce ne sont pas les mêmes livres)

Et il en serait de même pour (presque) tous les livres

# Notion de classe

- Tous les livres obéissent à un même schéma  
⇒ on peut en abstraire un patron (abstrait), une sorte moule, un modèle

Ce moule définit :

- les attributs qui caractérisent l'état
- l'interface et sa réaction = son comportement

Les moulages qui en seront issus sont les *objets*

# Notion de classe

## Classe

En POO un tel « moule » est appelé une **classe** (= **type**)

## Instance

Les moulages issus de ce moule, c'est-à-dire les objets, sont appelés des **instances** de la classe

- Moulage ne signifie pas copie, on peut les « décorer » différemment
- Ainsi, une fois qu'on a la classe, on peut potentiellement créer autant d'instances que l'on veut
  - elles auront des identités différentes
  - elles seront composées des mêmes attributs, mais avec des valeurs différentes possibles
  - elles auront toutes le même comportement (même interface)

# Notion de classe

Représentation UML :

<b>Livre</b>
auteur : String
titre : String
annee : int
texte : String
lire() : void
imprimer() : void

<b>Auteur</b>
nom : String
prenom : String
naissance : int
deces : int
biblio : String
ajouterBiblio(livre : String)
afficher()
fixerDeces(dateMort : int)

# Construction d'objets

**Programmation** = définition des classes : processus d'abstraction

**À l'exécution** = travail sur les objets/instances : concrétisation

- La classe définit le comportement de **toutes** ses instances
- Les instances ont des identités différentes et des valeurs d'attributs différentes
- Par contre l'interface est la même pour toutes les instances

Pour créer les instances selon le modèle de la classe, c'est-à-dire pour concrétiser les entités permettant la résolution du problème, on utilise ce qu'on appelle un **constructeur**.

# Les constructeurs

Chaque appel à un constructeur crée un nouvel objet (instance) qui obéit au patron défini par la classe :

- l'instance créée aura les attributs et le comportement définis dans la classe
  - ↪ réservation d'un espace mémoire pour la mémorisation de l'état
  
- le constructeur est généralement l'occasion d'initialiser les attributs (« personnaliser » l'état de l'instance)
- il peut y avoir plusieurs constructeurs pour une même classe
  - ↪ reflète le fait qu'il peut y avoir différentes manière de construire un objet

## Construction en Java

```
new NomDeLaClasse (parametre1, parametre2, ...);
```

## Exemple avec la classe Livre

```
new Livre();  
new Livre("JRR Tolkien","Le Seigneur des Anneaux", 1954);
```

- Si on ne définit pas explicitement de constructeur, il y a un toujours un constructeur par défaut (constructeur sans argument) défini implicitement (attention, **que** s'il n'y a pas d'autres constructeurs explicites)

# OK... et concrètement

```
class Livre {  
    // les attributs de la classe livre  
    String auteur;  
    String titre;  
    int annee;  
    String texte;  
  
    // constructeur  
    Livre(String unAuteur, String unTitre, int uneAnnee, String unTexte) {  
        auteur = unAuteur;  
        titre = unTitre;  
        annee = uneAnnee;  
        texte = unTexte;  
    }  
  
    // les méthodes de la classe Livre  
    String retournerAuteur() { return auteur; }  
    void affiche(String msg) {  
        System.out.println(msg + " -> " + titre + " de " +  
                           auteur + " paru en " + annee);  
    }  
    void lit() { System.out.println(texte); }  
    void litEtAffiche() {  
        lit();                      // invocation de lit() sur l'objet lui-même  
        affiche("info :");  
    }  
}
```

## Définition de la classe

```
class NomDeLaClasse {  
    ...  
    déclaration et définition des constructeurs,  
    attributs et méthodes  
    ...  
}
```

- Un fichier une classe
- Le nom de la classe et du fichier doivent correspondre  
    → classe MachinChose ==> fichier MachinChose.java  
dans l'exemple : Livre.java
- On définit la classe (= le moule)
- L'ordre des déclarations dans le fichier importe peu
- Convention de nommage et d'écriture de code
- Différenciation majuscules/minuscules

## Définition du constructeur

```
NomDeLaClasse (parametres) {  
    ... initialisations et traitements ...  
}
```

- Une classe, plusieurs constructeurs possibles
- Le nom du constructeur et de la classe doivent correspondre  
    → classe MachinChose ==> constructeur MachinChose(...)  
        Livre (String unAuteur, String titre, int annee, String texte)
- On définit comment instancier un objet

## Définition de méthodes

```
type nom (paramètres) {  
    ... traitements ...  
    return expression;           // si type ≠ void  
}
```

- type = type primitif ou classe de la valeur de retour
- La valeur de retour est précisée par return (fin de traitement)

## Signature

Signature de la méthode = nom + type des paramètres

- Impossible d'avoir deux méthodes de même signature au sein d'une même classe

# Comment on manipule l'objet

- Il est possible de nommer un objet crée pour pouvoir y faire référence par la suite.
  - On précise le type (classe) de la référence (donc de l'objet référencé)
  - On nomme la référence
  - On affecte une valeur (existante ou résultante d'une construction) = l'objet
- Destructeur d'objet ?
  - But : libérer l'espace mémoire occupé par l'objet
  - En Java pas de destructeur d'objet explicite
  - Pas nécessaire de libérer la mémoire « à la main » le GC s'en charge (GC = Garbage Collector = « ramasse-miettes »)

# Identificateur d'objets

- On fait référence à un objet en utilisant un identificateur  
on dit alors que l'identificateur est une *référence* sur l'objet
- Tout identificateur doit être initialisé avant d'être utilisé  
→ il faut lier l'identificateur à l'objet
- Un identificateur non initialisé a la valeur null

## en Java

```
Auteur unAuteur = new Auteur();
Livre unLivre = new Livre("JRR Tolkien", "Le Seigneur des Anneaux", 1954);
Livre memeLivre;
memeLivre = unLivre;
```

- Un identificateur est unique (correspond à un seul objet à un certain moment)
- Deux identificateurs peuvent faire référence au même objet

# Communiquer avec l'objet

## Envoi de messages

La communication avec l'objet se fait grâce à l'opérateur « . »  
    objet.message

Plusieurs possibilités :

- `objet.attribut` : envoi du message « accès à l'attribut attribut » à objet – modification directe
- `objet.methode(parametres)` : envoi du message « exécute la méthode méthode avec les paramètre parametres » à objet  
    → le traitement décrit dans la méthode est alors exécuté
- Il faut évidemment que ce message soit reconnu par l'objet  
    → il doit faire partie de l'interface de la classe de l'objet

## Communiquer avec l'objet

```
//A travers un identificateur  
Livre unLivre = new Livre();  
unLivre.auteur = "Tolkien";  
unLivre.affiche();  
  
// Directement à la création  
new Livre("JRR Tolkien", "Le Seigneur des Anneaux",  
1954).affiche();
```

## Les cascades sont possibles

```
unLivre.auteur.nom          unLivre.auteur.fixeDeces(1907)
```

## Attention au références null

```
Livre unLivre;  
unLivre.auteur = "Jules Verne";           // erreur
```

# Auto-référence

- Dans le traitement de l'une de ses méthodes un objet peut avoir à s'envoyer un message (pour accéder à un de ses attributs ou invoquer une des ses méthodes)
- utilisation de l'**auto-référence**, en Java : **this**

## Exemple

On se place dans le corps d'une méthode de la classe Livre

↪ lors du traitement, l'objet invoquant est une instance de Livre

`this.imprime()` signifie « envoyer à moi-même le message `imprime()` »

- Si pas d'ambiguïté, `this` peut être omis :

`imprime() ≡ this.imprime()`

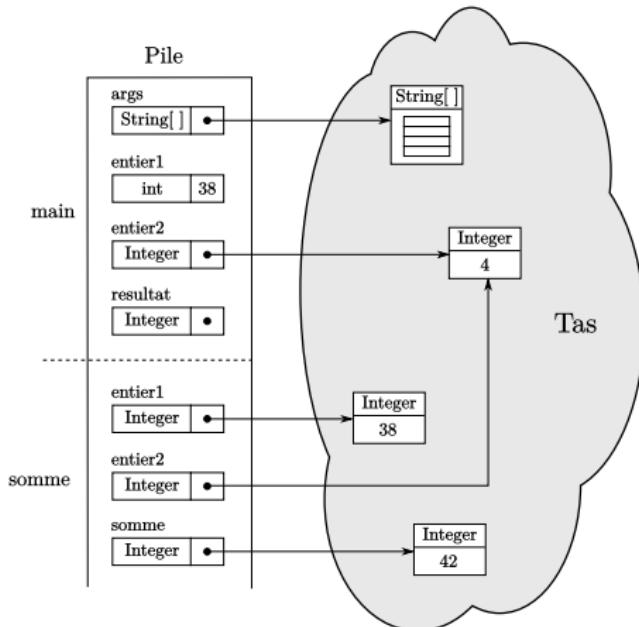
`auteur ≡ this.auteur`

# Schéma mémoire

Fait au tableau

# Schéma mémoire

```
public class Memory1 {  
  
    static Integer somme(Integer entier1, Integer entier2) {  
        Integer somme;  
        somme = entier1 + entier2;  
        return somme;  
    }  
  
    public static void main(String[] args) {  
        int entier1 = 38;  
        Integer entier2 = new Integer(4);  
  
        Integer resultat;  
        resultat = somme(entier1, entier2);  
  
        System.out.println("Résultat: " + resultat);  
    }  
}
```



# Schéma mémoire

```
public class Memory1 {  
    static Integer somme(Integer entier1, Integer entier2) {  
        Integer somme;  
        somme = entier1 + entier2;  
        return somme;  
    }  
  
    public static void main(String[] args) {  
        int entier1 = 38;  
        Integer entier2 = new Integer(4);  
  
        Integer resultat;  
        resultat = somme(entier1, entier2);  
        System.out.println("Résultat: " + resultat);  
    }  
}
```

