Licence Pro DevOps

Java avancé

2021-2022

Rémi Watrigant

remi.watrigant@univ-lyon1.fr

http://perso.univ-lyon1.fr/remi.watrigant/

IUT Informatique – Université Claude Bernard Lyon 1

Présentation du module

- 17 séances de 2h = 34h

  (dernière séance le 4 février)

- Évaluation :
  - 1 projet, sujet plus ou moins imposé :
    - présentation et démo
    - livraison du code
    - livret et documentation

Présentation du module

- Premières séances : remise à niveau
  - Bases de la Programmation Orientée Objet
  - Java
- Puis, concepts plus avancés
  - Réseau
  - Programmation parallèle
  - Interfaces graphiques
  - Bases de données
  - "nouveautés" Java 8 & 9
  - Utilisation d'autres API
- Début du projet ≈ fin novembre/début décembre

Questionnaire

Classe :

Méthode :

Élément statique :

Visibilité des données :

Héritage :

Surcharge :

Classe abstraite :

Généricité :

Garbage collector :

Machine virtuelle :

Interface :

Exception :

Getter/setter :

Polymorphisme :

Librairie standard :

Agrégation :

Annotations :

Javadoc :

Design pattern :

Constructeur :

Bibliographie

- Site de référence :

http://www.oracle.com/fr/java/index.html

- Documentation en ligne (développement) :

http://docs.oracle.com/javase/8/docs/api/

- Livres :
  - La programmation orientée objet, H. Bersini, Eyrolles
  - Programmer en java, 9ème édition, C. Delannoy, Eyrolles
  - En ligne : Penser en java, B. Eckel
    http://bruce-eckel.developpez.com/livres/java/traduction/tij2/

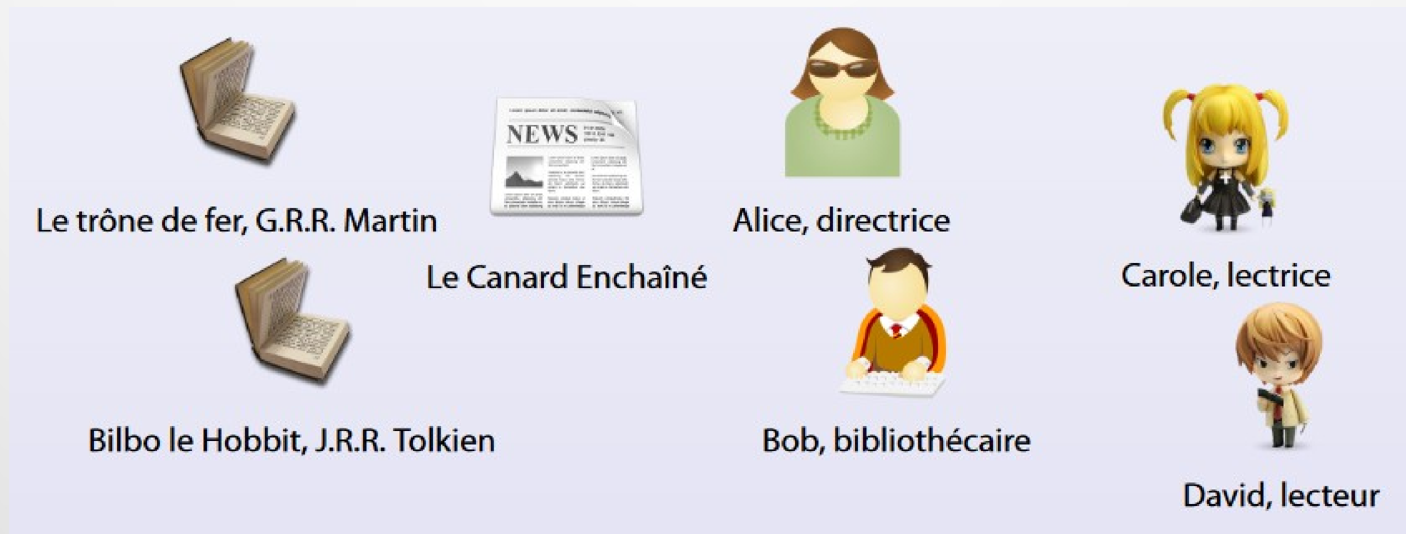
Programmation orientée objet

Différents paradigmes :

	IMPERATIF	FONCTIONNEL	OBJET	DESCRIPTIF
DEMARCHE	Procédurale Série d'instructions, sauts conditionnels	Flots de données Diagramme de structure	Objets, classes, composition, réseau de messages	Besoin, expressif, léger
CONCEPTS	Itération, structures de contrôles Exécution d'instructions qui modifient l'état de la mémoire	Evaluation d'expressions qui ne dépendent que de la valeur des arguments, et non de l'état de la mémoire	Objet, classes, méthodes, encapsulation, héritage, relations de composition, d'utilisation, …	Description des buts à atteindre à l'aide d'une syntaxe légère
LANGAGES	Fortran, C, Pascal	Lisp, Scheme, Caml	SmallTalk, C++, Java, Python	HTML, XML, LaTeX

Programmation orientée objet

- Approche procédurale : “que doit faire mon programme ?
- Approche objet : “de quoi doit être composé mon programme ?”



Programmation orientée objet

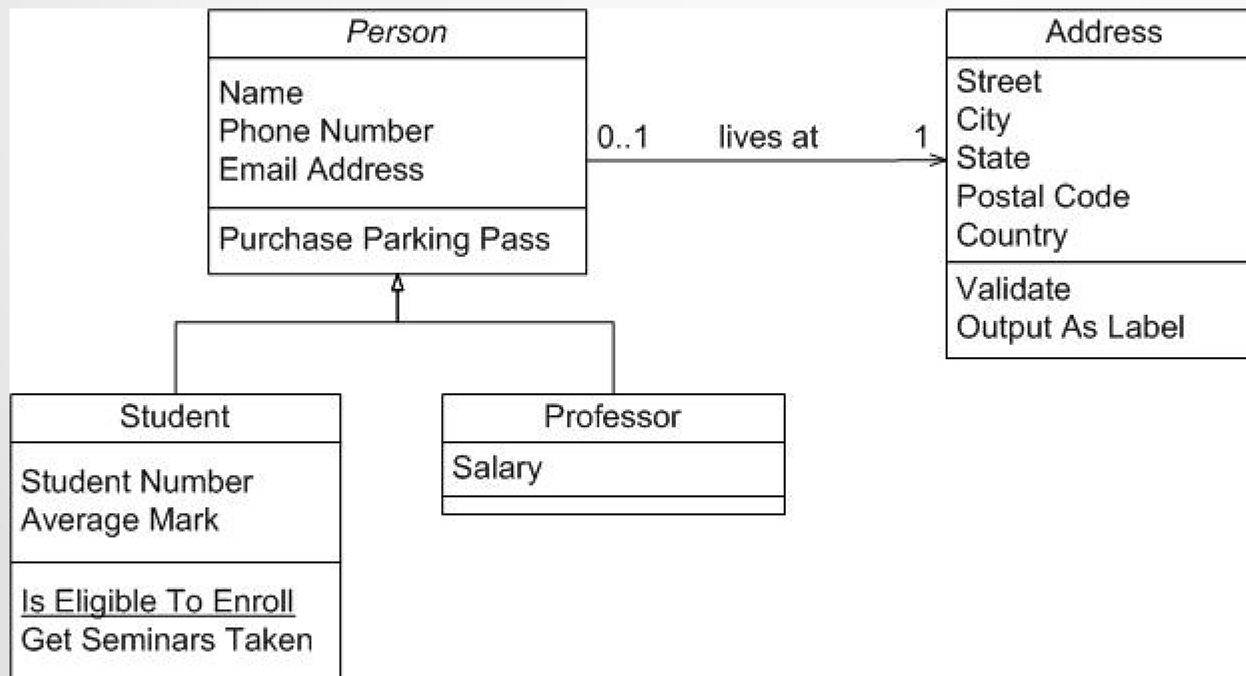
- Approche procédurale : “que doit faire mon programme ?

- Approche objet : “de quoi doit être composé mon programme ?”

Programmation orientée objet

- Avant la programmation…

...La conception



Programmation orientée objet

- Les objets représentent des données modélisées par des **classes** qui définissent des types

  Un peu comme `typedef struct` en C

- Les classes définissent les actions que les objets peuvent prendre en charge et la manière dont les actions affectent leur état.

  Ces traitements sont des **méthodes**

- Les données d’un objet sont appelées ses **attributs**

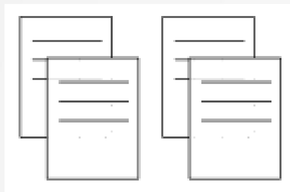
Le langage Java

- En quelques mots :
  - Orienté Objet
  - Simple, Robuste, Dynamique et Sécurisé
  - Indépendant de la Plateforme (VM)
  - Semi Compilé/Semi Interprété
  - Fortement typé
  - Bibliothèque Importante (JDK API)

Le langage Java

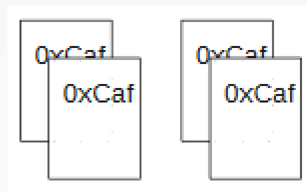
→ Compilé puis interprété

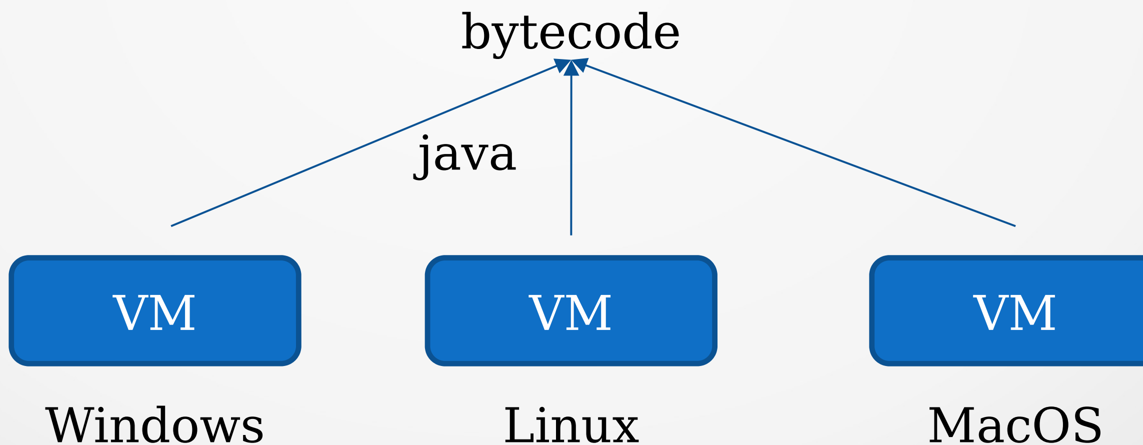
Fichiers .java



javac

Fichiers .class





Classe

```java
public class Livre {

    private String titre;

    private Lecteur empreunteur;

    public Livre(String titre, Lecteur empreunteur) {
        this.titre = titre;
        this.empreunteur = empreunteur;
    }

    public Livre(String titre) {
        this.titre = titre;
        empreunteur = null;
    }

    public void setTitre(String titre) {
        this.titre = titre;
    }

    public String getTitre() {
        return titre;
    }

    public Date empreunte(Lecteur lec) {
        if (empreunteur != null) {
            empreunteur = lec;
            return new Date();
        } else {
            return null;
        }
    }

    public String toString() {
        return "Livre "+titre+" empreunté par "+empreunteur.toString();
    }
}
```

Attributs

Constructeurs

Mutateur (*Setter*)

Accesseur (*Getter*)

Méthodes

Bonnes pratiques de programmation

Les règles de nommage en Java :

- les noms de classes commencent toujours par une majuscule
- les noms de méthodes et variables commencent toujours par une minuscucle
- on utilise la règle dîte du "CamelCase" :

  `VoiciUnNomDeClasse`

  ~~**`voici_un_nom_de_classe`**~~

- exception : constantes (ex : variables statiques) en majuscules, séparées par des _

  ex: **`Integer`**`.MAX_VALUE`

Instanciation

- Mot-clé *new* pour instancier un nouvel objet de la classe :
```
Livre monLivre = new Livre("Germinal");
```

- Invocation d'un constructeur existant.

- Sans instanciation, on ne peut pas utiliser les données et méthodes de la classe ! (sauf cas *static*...)

Invocation de méthode

- une méthode appartient toujours à un objet
  → on appelle la méthode d'un objet donné
- un point . sépare le nom de la méthode de l'objet :
  `String leTitre = monLivre.getTitre();`
- le mot clé **this** désigne, à l'intérieur d'une classe, l'objet courant sur lequel est appliqué le code

```
public Livre(String titre, Lecteur empreunteur) {
    this.titre = titre;
    this.empreunteur = empreunteur;
}
```

```
public String toString() {
    return "Livre "+this.titre+" empreunté par "+empreunteur.toString();
}
```

Héritage

- Un des fondements de la POO
- Idée :
  - organiser les classes de manière hiérarchique
  - définir des sous-types
- Favorise :
  - la réutilisation de code (factorisation)
  - la conception par spécialisation progressive
- Modélise la relation de spécialisation "is a"

Héritage

- Vocabulaire :
  - une classe **B** hérite d'une classe **A**
  - **B** est une sous-classe/classe dérivée/spécialisation de **A**
  - les objets instanciant **B** instancient également **A**

- La classe **B** hérite d'emblée des fonctionnalités de **A** :
  - attributs (public ou protected)
  - méthodes (public ou protected)

- But : développer des nouveaux outils en se fondant sur certains acquis

- Remarque : en Java, pas d'héritage multiple
  (on n'hérite que d'une seule classe)

Exemple d'héritage : la classe Point

```
public class Point {
    protected double abscisse;
    protected double ordonnee;

    public Point(){
        abscisse = 0.0;
        ordonnee = 0.0;
    }

    public Point(double x, double y){
        abscisse = x;
        ordonnee = y;
    }

    public double getAbscisse(){ return abscisse; }
    public double getOrdonnee(){ return ordonnee; }

    public double distanceP(Point p){
        double x1=abscisse, x2=p.abscisse;
        double y1=ordonnee, y2=p.ordonnee;
        return Math.sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
    }

    public String toString(){
        return "("+abscisse+" , "+ordonnee+")";
    }
}
```

Exemple d'héritage : la classe Point

- Classe dérivée : `ColoredPoint`
- Possède :
  - les attributs **protected** et **public** de `Point`
  - les méthodes **protected** et **public** de `Point`
  - un attribut `color` en plus
  - une méthode `toString()` différente

Exemple d'héritage : la classe Point

```
public class ColoredPoint extends Point {
    protected Color couleur;


    public ColoredPoint(double x, double y, Color c){
        super(x, y);
        couleur = c;
    }

    public void toString(){
        return "("+abscisse+" , "+ordonnee+", "+couleur+")";
    }
}
```

- le mot clé **super** fait appel à la classe mère
  → par exemple pour appeler un constructeur déjà défini

mot clé super

- le mot clé **`super`** fait appel à la classe mère
  → par exemple pour appeler un constructeur déjà défini

- l'instruction `super` doit être la 1ère instruction du constructeur de la classe dérivée

- si aucun appel `super` n'est fait, il y a quand même un appel implicite au constructeur vide de la classe de base
  (comme si on avait quand même écrit `super()`)

  → si aucun constructeur vide dans la classe de base : erreur de compilation
  SAUF si la classe de base a 0 constructeur, auquel cas Java ajoute implicitement le constructeur vide

Visibilité d'attributs, méthodes

Modificateur du membre	`private`	*défaut*	`protected`	`public`
Accès depuis la classe	Oui	Oui	Oui	Oui
Accès depuis une classe du même package	Non	Oui	Oui	Oui
Accès depuis une sous-classe	Non	Non	Oui	Oui
Accès depuis tout autre classe	Non	Non	Non	Oui

Classes abstraites

- Méthode abstraite : méthode qui n'admet pas d'implémentation.
  - Mot clé *abstract*
- Toute classe ayant au moins une méthode abstraite doit elle-même être déclarée abstract :

```
public abstract class Maladie {

protected String nom;
protected int force;
protected int dangerosite;
protected String symptome;

public abstract boolean traiter (Medicament med);
```

Classes abstraites

- Ne peut être instanciée
- Mais peut avoir des constructeurs (permet de mutualiser du code) qui pourront alors être utilisés dans les classes filles (notamment grâce à l'appel à **`super()`**)
- Pour toute classe héritant d'une classe fille, soit :
  - Implémente toutes les méthodes abstraites de la classe mère, elle peut alors être instanciée
  - N'implémente pas toutes les méthodes abstraites de la classe mère, elle doit alors également être abstraite
- But : mutualiser le code de certaines méthodes communes à plusieurs classes

Variables et méthodes de classe (static)

- **Attachées à une classe plutôt qu'à un objet**
  - Existe même si aucun objet n'est instancié
  - Déclarées avec le préfixe `static`
  - Appel à partir du nom de la classe

- Variable :
  - valeur propre à la classe
  - Même valeur pour tous les objets de la classe

- Méthode :
  - Peut être exécutée même si aucun objet n'existe

Variables et méthodes de classe

```
public final class Math{
    ...
    public static final double PI = 3.14159265358979323846;
    ...
    public static double toRadians(double angdeg){
        return angdeg / 180.0 * PI;
    }
    ...
}


public class MathMain{
    public static void main(String[] args){
        System.out.println("pi = "+Math.PI);
        System.out.println("90° = "+Math.toRadians(90));
    }
}
```

Méthode main

- Méthode d'entrée de l'exécution du programme java
- Peut être située dans une classe définissant un objet (par exemple *Livre.java*)
- Généralement se trouve dans une classe à part (i.e. *TestLivre.java*)
- Signature formelle de la méthode unique et imposée :

```
public static void main(String[] args)
```

- args comporte les arguments du programme