

# PROGRAMMATION PAR OBJETS

Anne Etien  
Maître de Conférences  
Bureau F030  
Anne.etien@polytech-lille.fr

## Présentation

- Objectif : **Concevoir une solution objet de qualité et la réaliser en Java**
  - Présenter les concepts de base de l'approche objet
  - Adopter le “penser objet”
  - Connaître et savoir mettre en œuvre les concepts fondamentaux
  - Sensibiliser à la production d'un code de qualité en Java
- Organisation :
  - 20h de cours
  - 16h de TP
  - 24h de Projet
- Notation :
  - DS (2h)
  - Projet

Walter Rudametkin

## Avertissement

- Ce cours est très largement inspiré (voire reprend complètement certaines parties) du cours de Bernard Carré.
- Ce cours s'appuie aussi sur les cours de Manuele Kirsch, maître de conférences à Paris 1, de Jean-Christophe Routier, professeur à Lille 1

## A l'issue de ce module vous devriez...

- ... connaître les éléments de base de la programmation objet
  - ... maitriser le vocabulaire de la programmation objet :
    - classe, instance, méthode, interface, attribut, constructeur, encapsulation, polymorphisme
  - ... savoir décomposer un problème simple en classes et objets
  - ... savoir expliquer ce qui différencie la programmation objet des autres paradigmes
  - ... savoir expliquer ce qu'est le polymorphisme, en présenter les avantages et savoir expliquer ce qu'est le “late-binding”
  - ... connaître le principe ouvert-fermé, être en mesure de l'expliquer et de l'appliquer sur des exemples simples
  - ... pouvoir identifier certaines situations de mauvaises conception objet et les corriger

## A l'issue de ce module vous devriez...

- ... savoir spécifier, coder et tester un problème objet dans le langage JAVA
  - ... connaître les principaux éléments de la syntaxe du langage java
  - ... être en mesure d'écrire (et corriger) un programme dans le langage java
  - ... pouvoir expliquer clairement le rôle et la sémantique des éléments de langage suivants et savoir les utiliser :
    - new, public, private, enum this, static, final, package import, throws, throw
  - ... comprendre le transtypage (upcast/downcast)
  - ... être en mesure de choisir une structure de données appropriée et savoir utiliser les types java List, Set, Map et Iterator
  - ... savoir gérer les exceptions et connaître la différence entre capture et levée d'exception
  - ... savoir utiliser les "outils" liés à la plateforme java :
    - javac, java (et classpath), javadoc, jar

## Maintenance d'un logiciel

- Software maintenance is the modification of a software product after delivery to correct faults, to improve performance or other attributes.

ISO/IEC 14764:2006 Software Engineering – Software Life Cycle Processes — Maintenance

- Quelques chiffres :
  - 120 milliards LOC maintenues en 1990 (Ulrich, 1990)
  - 200 milliards en 2000 (Somerville, 2000)
  - Coût annuel aux US > \$70 milliard (Sutherland, 1995)
  - Nokia a dépensé \$90 milliards pour Y2K
  - Dépense du gouvernement américain > \$8 milliards

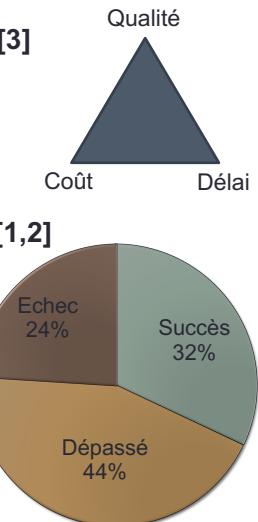
## Réussite d'un projet

### Difficile équilibre : Qualité– Coût – Délai [3]

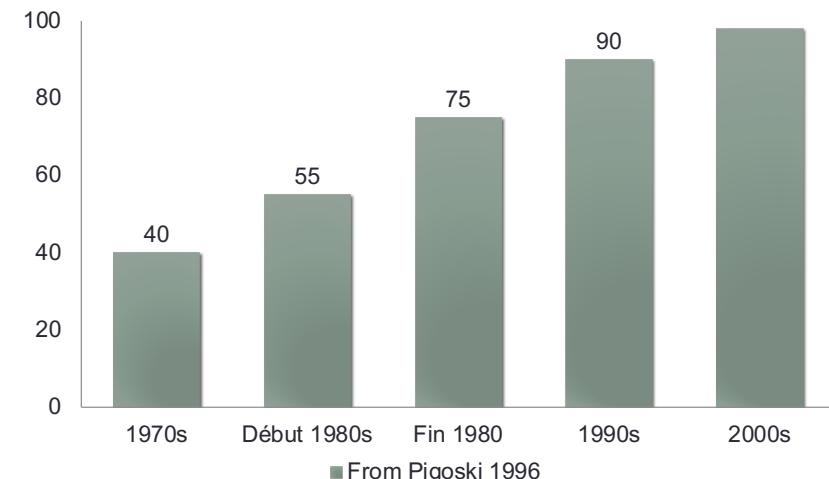
- Rapide et pas cher → Mauvaise qualité
- Rapide et de bonne qualité → Cher
- Bonne qualité et pas cher → Lent
- Lent, de mauvaise qualité et cher → Cata !!

### Chaos Report by Standish Group 2009 [1,2]

- Seulement 32% des projets réussissent (temps, budget et features)
- 44% ne respectent pas les délais, les coûts et/ou les besoins énoncés
- 24% des projets n'aboutissent pas
- Facteurs de réussite [1,2]
  - Implication de l'utilisateur
  - Exigences et spécifications claires

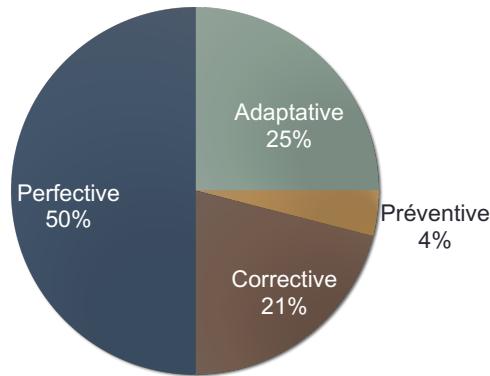


## Maintenance d'un logiciel



## Maintenance d'un logiciel

Catégories de maintenance



## Bienfaits de l'abstraction

- Tous les langages de programmation fournissent des abstractions.
- Qu'est-ce qu'on tente d'abstraire ?
  - Le langage assembleur : petite abstraction de la machine sous-jacente.
  - Les langages « impératifs » : abstractions du langage assembleur.
    - nettes améliorations par rapport à l'assembleur,
    - Mais réflexion en termes de structure ordinateur
  - Alternative : modéliser le problème qu'on tente de résoudre.
    - LISP : une vue particulière du monde (« Tous les problèmes se ramènent à des listes »)
    - PROLOG convertit tous les problèmes en chaînes de décisions.
  - Adapté mais ... pas adapté

## Leçons

Le logiciel évolue même après avoir été livré au client

→ Il faut anticiper cette évolution

Mettre en place des mécanismes facilitant l'évolution c'est à dire :

- La réutilisation
- La modularisation
- L'ajout de nouvelles fonctionnalités
- ...
- Mais aussi la documentation ;o)

## Bienfaits de l'abstraction

- L'approche orientée objet va un cran plus loin
  - Description du problème avec ses termes plutôt qu'avec ceux de la machine
  - Représentation assez générale → pas de restriction à un type particulier de problèmes.
  - Utilisation des « objets » dans l'espace problème et dans l'espace solution.
    - Plus autres objets qui n'ont pas leur analogue dans l'espace problème.
- Abstraction plus flexible et puissante que précédemment.
- Chaque objet ressemble à un mini-ordinateur ;
  - un état,
  - des opérations qu'on peut lui demander d'exécuter.
- Attention : la PPO n'est pas adéquate pour résoudre facilement tous les problèmes de programmation

## Caractéristiques des langages purs objet

- Toute chose est un **objet**.
- Un programme est un ensemble d'objets se disant les uns aux autres quoi faire en s'envoyant des **messages**.
- Chaque objet est d'un **type** précis.
  - chaque objet est une *instance* d'une *classe*,
- Tous les objets d'un type particulier peuvent recevoir le même message.

## Exemples

- Un thermomètre mesure une température (un nombre)
- Cette température mesurée est une donnée, ou caractéristique, de ce thermomètre . Elle définit son **état**.
- Sur ce thermomètre, on peut envisager certaines manipulations ou opérations = le **comportement** du thermomètre :
  - obtenir la température en degrés Celsius ou en Fahrenheit
  - modifier la température mesurée
  - associer à la température une couleur (bleu, vert, rouge, ...) ou un mot (froid, normal, chaud,...)
- A chaque opération correspond un **traitement**.

## Ces objets qui nous entourent

- des voitures, des livres, des portes, des chaises, des ordinateurs, des thermomètres, des téléviseurs,...
- des chats, des personnes, des facteurs, des éléphants,...
- des comptes en banque, des dossiers étudiant, des connexions réseau,...
- Ces objets
  - ont des **caractéristiques** : la couleur d'une voiture, l'âge ou le nom d'une personne, le solde d'un compte en banque,...
  - ont un **comportement** : ouvrir la porte, le chat miaule, créditer le compte en banque,...

## Un premier thermomètre

- Un objet thermomètre → soit thermo1 son **identité**.
- Son **état** est défini par une température mesurée temp (22.5°C).
- On peut exploiter son **comportement** :
 

```
temperatureEnCelsius, temperatureEnhrenheit,
modifierTemperature, couleurTemperature
thermo1.temperatureEnCelsius() → 22.5
thermo1.temperatureFarenheit() → 72.5
thermo1.modifierTemperature(25.8) → -
thermo1.temperatureEnCelsius() → 25.8
thermo1.couleurTemperature() → Couleur.VERT
```

Le comportement dépend de l'état et agit sur l'état.

Le comportement correspond à un ensemble de traitements programmés en “style” impératif.

#### temperatureEnCelsius

données : Ø  
résultat : un flottant  
traitement :  
    renvoyer temp

#### temperatureEnFahrenheit

données : Ø  
résultat : un flottant  
traitement :  
    renvoyer  $(9/5) * \underline{\text{temp}} + 32$

#### modifierTemperature

données : flottant : nouvelleTemp  
résultat : Ø  
traitement :  
    temp = nouvelleTemp

#### couleurTemperature

données : Ø  
résultat : une couleur  
traitement :  
    si ( $\text{temp} < 0$ ) alors  
        renvoyer Couleur.BLEU  
    sinon si ( $\text{temp} < 30$ ) alors  
        renvoyer Couleur.VERT  
    sinon  
        renvoyer Couleur.ROUGE

## Un autre thermomètre

- Un **autre** thermomètre → thermo2 son **identité**.
- Son état est une température mesurée : -4°C.
- Même “structure” de l’état que thermo1 mais valeur **diférente**.

- Il obéit au même **comportement**.

thermo2.temperatureEnCelsius() → -4  
thermo2.temperatureEnFahrenheit() → 24.8  
thermo2.couleurTemperature() → Couleur.bleu

- Résultats différents mais les **traitements** exécutés **sont les mêmes**.
- Dans traitement temp signifie “*la température du thermomètre impliquée*”.
- Il en serait de même pour **tous** les autres thermomètres.

## Chaudière

- Une chaudière est un objet dont
  - l’état est défini par le statut allumée/éteinte (un booléen)
  - le comportement est défini par
    - connaître l’état de la chaudière : `estAllumee()`, `estEteinte()`
    - allumer la chaudière : `allumer()`
    - éteindre la chaudière : `eteindre()`

## Thermostat

- L’**état** d’un thermostat est défini par
  - la chaudière qu’il contrôle, `chaudiere`
  - la température de déclenchement, `tempDeclenchement`
  - un thermomètre pour mesurer la température ambiante, `thermo`
- Cet état **est composé d’autres objets** : `chaudiere`, `thermo`  
(et `tempDeclenchement`)
- Son **comportement** est :
  - mettre en marche ou arrêter la chaudière selon la température de déclenchement et celle mesurée par le thermomètre
  - modifier la température de déclenchement,
  - etc.

Par exemple

### changerTempDeclenchement

données : flottant : nouvelleTemp

résultat : Ø

traitement :

```
tempDeclenchement = nouvelleTemp
si (chaudiere.estAllumee()) et
    tempDeclenchement < thermo.temperatureEnCelsius() alors
        chaudiere.eteindre()
sinon si (chaudiere.estEteinte()) et
    tempDeclenchement ≥ thermo.temperatureEnCelsius() alors
        chaudiere.allumer()
```

Le traitement de changerTempDeclenchement fait appel aux comportements des objets chaudiere et thermo.

L'objet thermostat **envoie des messages** à ces objets.

**C'est ainsi que se crée la dynamique du programme.**

## Langage à objets (purs)

- Tout est objet
- Chaque objet a sa propre mémoire constituée d'autres objets
- Chaque objet a (au moins) un type
- Tous les objets d'un type donné peuvent recevoir le même type de messages
- Un programme est un regroupement d'objets qui se disent quoi faire par envois de messages

Alan Kay - SmallTalk

## Java

- langage orienté objet (pas 100% objet), langage de classes
- langage compilé, fortement typé
- indépendance OS/architecture : multi plate-forme
  - utilisation d'une machine virtuelle (la JVM) - bytecode Java
  - "compile once, run everywhere"
- gestion dynamique de la mémoire
  - utilisation d'un GC (garbage collector = ramasse-miettes)
- gestion des erreurs par exceptions
- nombreuses bibliothèques/API (gratuites) (réseau, RMI, JDBC, etc.)
- existe depuis 1995, libre depuis ~ 2007... – JDK, JRE, SDK

## En Java

```
public class Thermometre {
    private float temp;
    public Thermometre(float tempInit) {
        this.temp = tempInit;
    }
    public float temperatureEnCelsius() {
        return this.temp;
    }
    public float temperatureEnFahrenheit() {
        return (9.0/5.0)*this.temp+32;
    }
    public void modifierTemperature(float
        nouvelleTemp) {
        this.temp = nouvelleTemp;
    }
}
```

```
...
public Color couleurTemperature() {
    if (this.temp < 0) {
        return Color.BLUE;
    }
    else if (this.temp < 30) {
        return Color.GREEN;
    }
    else return Color.RED;
}
```

## Objet

Objet = identité + état + comportement  
attributs méthodes

avec

L'identité permet d'exploiter le comportement d'un objet. Le comportement agit sur l'état et l'état influence le comportement

## Une identité

- Une identité permet de s'adresser à l'objet
- chaque identité est unique
  - deux objets différents ont des identités différentes
- on peut faire référence à l'objet (à son identité), la nommer
- on peut avoir plusieurs références pour une seule identité (un seul objet)

## Un état

- ensemble de propriétés ou caractéristiques définies par des valeurs
- valeurs propres (personnelles) à chaque objet
- l'état d'un objet (les valeurs des propriétés) peut évoluer dans le temps

## Un comportement

- ensemble des traitements que peut accomplir un objet (ou que l'on peut lui faire accomplir)
- On dit que l'on **invoque une méthode sur un objet**.
- On ne peut utiliser une méthode qu'en l'invoquant sur un objet.

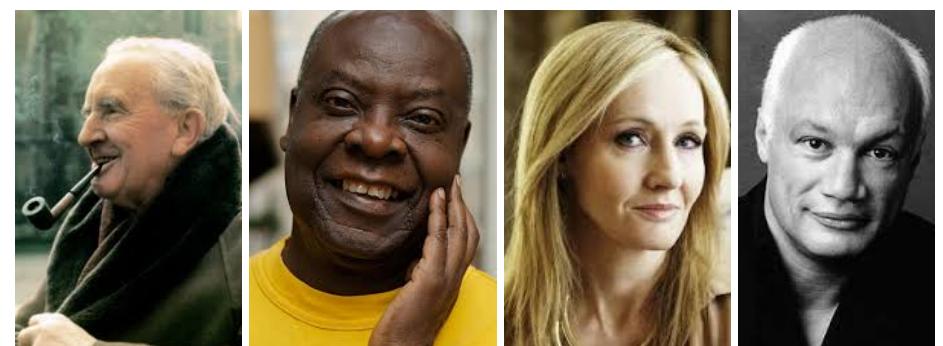
## Envoi de message

- on s'adresse à l'objet par **envoi de messages**
- on "demande" à l'objet de faire quelque chose  
envoi de message = invocation de méthode
- le comportement définit l'ensemble des messages qu'un objet peut recevoir
- interface** de l'objet
  - "ensemble" des manières que l'on a pour interagir avec l'objet
  - ensemble des messages reconnus par l'objet
  - "interface de comportement"

## 4 images, 1 concept



## 4 images, 1 concept



JRR Tolkien

E Dongala

JK Rowling

EE Schmitt

- Quelles caractéristiques ? Quels comportements ?

- Quelles caractéristiques ? Quels comportements ?

## 4 images, 1 concept



- Quelles caractéristiques ? Quels comportements ?

Tous les livres obéissent à un même schéma

→ on peut en abstraire un "moule", un patron, un modèle, etc

- Le moule définit
  - les **attributs** qui caractérisent l'état
  - l'interface et sa réaction = le comportement → les **méthodes** de tous les moulages qui en seront issus

"moulages = objets"

- certains objets présentent les mêmes caractéristiques :
  - identités différentes mais
  - états définis par les mêmes attributs
  - même interface de comportement

- exemples :

- les thermomètres thermo1 et thermo2
- des livres "Le Seigneur des Anneaux" et de John Ronald Reuel Tolkien paru en 1954

sont caractérisés par les mêmes attributs

- auteur, titre, année, texte

et ont la même interface de comportement

- on peut leur faire accomplir les mêmes actions :
- on peut les lire, les imprimer, etc.

il en serait de même pour (~) tous les livres

"Dune"  
de Frank Herbert  
paru en 1965



# Classes et instances

Un moule est appelé **classe**. Une classe est un **type**.

Les moulages sont les **objets** appelés **instances** de la classe

- Une fois qu'on a la **classe**, on peut potentiellement **créer** autant d'**objets / instances** conformes à la classe que l'on veut :
  - Ils ont des identités différentes
  - Ils ont des états définis par les **mêmes attributs**, mais avec des **valeurs différentes**
  - Ils auront le **même comportement** (mêmes méthodes)

- classe** = modèle
  - décrit la structure de l'état (les attributs et leurs types)
  - définit les envois de messages acceptés par l'objet → "interface"
  - abstrait**
    - la notion/le type "Chien" (personne n'a jamais vu "le type Chien")
- instance** = objet obéissant à un modèle
  - état correspond à la structure
    - association de valeurs aux attributs
  - n'accepte que les envois de messages autorisés par la classe
  - interface = ensemble des messages acceptés par l'objet
  - concret**
    - "ce chien noir que je vois dans la rue", "le chien de mon voisin"

- programmation** définition des classes → abstraction
- à **l'exécution** travail sur des objets/instances → concrétisation
- La classe définit le comportement de **toutes** ses instances
- Les instances ont des identités différentes et des valeurs d'attribut différentes.

Interface d'une classe

= ensemble des messages acceptés par les instances de la classe  
 ≈ ensemble des signatures des méthodes publiques (généralement)

```
public class Livre {
    // les attributs de la classe livre
    private String auteur;
    private String titre;
    private int annee;
    private String texte;
    // constructeur
    public Livre(String unAuteur, String titre, int
        annee, String texte) {
        this.auteur = unAuteur;
        this.titre = titre;
        this.annee = annee;
        this.texte = texte;
    }
}
```

```
// les méthodes de la classe Livre
public String getAuteur() {
    return this.auteur;
}
public void affiche(String msg) {
    System.out.println(msg + " -> " +this.titre+ " de "
        "+this.auteur+" paru en "+this.annee);
}
public void lit() {
    System.out.println(this.texte);
}
public void litEtAffiche() {
    this.lit(); // invocation de lit() sur l'objet
    // lui-même
    this.affiche("info :");
}
```

## Analyse (Objet) d'un problème

- Quels sont les objets nécessaires à la résolution du problème ?
  - → décomposition du problème en objets
- A quels modèles ces objets correspondent ils ? et donc quelles sont les classes ?
- Quelles sont les fonctionnalités dont on veut pouvoir disposer pour les objets de ces classes ?
  - → Quel comportement ? C à d quels messages doit/veut on pouvoir envoyer aux objets ?
- Quelle est la structure de l'état des objets ? qui est nécessaire pour la réalisation des comportements désirés.