Programmation Orientée Objet

Marc Champesme
mailto:Marc.Champesme@univ-paris13.fr

7 septembre 2022

Vos enseignants

Cours	Marc Champesme
TD	Marc Champesme, Carlos Olarte, Ali Akhavi
TP	Marc Champesme, Carlos Olarte, Ali Akhavi, Boris Eng, ?, ?
DL (CM/TD/TP)	Pierre Rousselin (Cours/TD/TP)
Contact	prenom.nom@univ-paris13.fr

Situation COVID / Accès à distance

Cours/ENT support de cours, sujets de TD et TP (Moodle 3 : nouvelle version)

Communications avec enseignants sur Mattermost, canaux POO-Annonces et POO-Discussions

Salles TP `https://si-galilee.univ-paris13.fr/guacamole/`

Organisation du cours

- 1,5h Cours / 1,5h TD / 1,5h TP par semaine
- Programme de la semaine affiché sur l'ENT
- Lire le support de cours avant de venir en cours, préparer des questions, poser vos questions en amphi ou sur le canal POO-Discussions
- Faire chaque TP seul sur son PC (avec guacamole si besoin)

Contrôle des connaissances

- Devoir n°1 (D1) à rendre à mi-semestre sur l'ENT
- Devoir n°2 (D2) à rendre en fin de semestre sur l'ENT, avant le partiel
- Deux partiels (Pa1 et Pa2) en milieu de semestre (semaine du 25/10) et fin de semestre (semaine du 3/01)
- Formule CC : $SUP((D + Pa1 + 2Pa)/4, Pa2)$ avec $D = (D1 + D2)/2$

Objectif de ce cours

- Objectif : Maîtriser les principaux concepts de la POO (classe/instance/méthodes, contrat, héritage. . .)
- Utilisation du langage JAVA pour la mise en œuvre des concepts, en particulier en TP
- Mais : L'objectif **n'est pas** de maîtriser complètement le langage JAVA ! !

Le projet Zork

- Définir des classes n'a de sens que dans un contexte bien défini
- (Re)définir un nouveau contexte pour chaque exercice, TD, TP est long
- Zork : contexte commun dans une grande partie des TD/TP/devoirs et partiels
- Recommandation (facultatif, non noté) : faites le projet !

Plan de ce cours

1. Introduction / historique
2. Classes, instances, objets, méthodes...
3. JAVA : présentation du langage et de sa syntaxe
4. Programmation par contrat (lien avec le cours de Spécification Algébrique)
5. Tests unitaires (Junit)
6. Mi-semestre : devoir n°1 + PA1
7. Héritage
8. Gestion des erreurs (exceptions)
9. Généricité
10. Fin du semestre : devoir n°2 + PA2

Introduction

La Programmation Orientée Objet (POO) est un paradigme de programmation = une famille de langages de programmation

Autres paradigmes de programmation :

Programmation Impérative : langage C, PASCAL, ...

Programmation fonctionnelle : CAML, Lisp, ...

Programmation logique : PROLOG

Langages Orientés Objets (LOO) :

SMALLTALK, JAVA, Objective-C, C++, EIFFEL, ... et beaucoup d'autres...

Historique

- 1980 : SMALLTALK (premier LOO)
- années 80 : forte diffusion de l'informatique, premiers PC
- industrialisation du développement logiciel, prise de conscience de nombreux problèmes :
  - délais de fabrication jamais tenus
  - mauvaise qualité (bugs)
  - difficultés à réutiliser le logiciel
  - difficulté à faire évoluer le logiciel et à assurer la maintenance
- 1995 : première version de JAVA

Objectifs de la POO

- pouvoir répartir le travail de programmation entre un nombre important de programmeurs
- logiciel correct par construction
- réutilisation
- évolution et maintenance du logiciel

Comment ?

- répartition du travail : diviser le logiciel en modules $\rightarrow$ classes en POO
- logiciel correct par construction : contrat, spécifier avant de programmer, puis programmer pour satisfaire la spécification (i.e le contrat)
- réutilisation : héritage, généricité
- évolution et maintenance du logiciel : séparation interface/implémentation, masquage d'information (niveaux de visibilité)

Programmation Orientée Objet vs Programmation Impérative

Programmation Impérative

Données

```
struct Pioche { Carte[]
contenu; int hauteur; }
```

Fonctions

```
Carte piocher(Pioche p) {
... }
empiler(Pioche p, Carte c)
{ ... }
```

POO

Classe = données + fonctions (fonctions → méthodes)

```
class Pioche {
Carte[] contenu;
int hauteur;
Carte piocher() { ... }
empiler( Carte c) { ... }
}
```

Programmation Orientée Objet vs Programmation Impérative (suite)

Un peu de vocabulaire :

Programmation Impérative	POO
• Types →	• Toute classe définit un type
• Fonctions →	• Méthodes
• Appels de fonctions →	• Envois de messages entre objets
• ? ? →	• instances d'une classe ≈ objets

Le langage JAVA - Histoire

- Première version en 1995 (Sun Microsystems)

Le langage JAVA - Histoire

- Première version en 1995 (Sun Microsystems)
- Mise en place du *Java Community Process* (JCP) en 1998, fonctionnement par examen/adoption de *Java Specification Request* (JSR)

Le langage JAVA - Histoire

- Première version en 1995 (Sun Microsystems)
- Mise en place du *Java Community Process* (JCP) en 1998, fonctionnement par examen/adoption de *Java Specification Request* (JSR)
- Version 5 en 2004 : introduction de la généricité

Le langage JAVA - Histoire

- Première version en 1995 (Sun Microsystems)
- Mise en place du *Java Community Process* (JCP) en 1998, fonctionnement par examen/adoption de *Java Specification Request* (JSR)
- Version 5 en 2004 : introduction de la généricité
- Passage en GPL en 2006 $\rightarrow$ OpenJDK

Le langage JAVA - Histoire

- Première version en 1995 (Sun Microsystems)
- Mise en place du *Java Community Process* (JCP) en 1998, fonctionnement par examen/adoption de *Java Specification Request* (JSR)
- Version 5 en 2004 : introduction de la généricité
- Passage en GPL en 2006 $\rightarrow$ OpenJDK
- Version 8 en 2014 : Lambdas et Stream (non abordés dans ce cours)

Le langage JAVA - Histoire

- Première version en 1995 (Sun Microsystems)
- Mise en place du *Java Community Process* (JCP) en 1998, fonctionnement par examen/adoption de *Java Specification Request* (JSR)
- Version 5 en 2004 : introduction de la généricité
- Passage en GPL en 2006 $\rightarrow$ OpenJDK
- Version 8 en 2014 : Lambdas et Stream (non abordés dans ce cours)
- Version 18 aujourd'hui (en 2022)

Le langage JAVA - Histoire

- Première version en 1995 (Sun Microsystems)
- Mise en place du *Java Community Process* (JCP) en 1998, fonctionnement par examen/adoption de *Java Specification Request* (JSR)
- Version 5 en 2004 : introduction de la généricité
- Passage en GPL en 2006 $\rightarrow$ OpenJDK
- Version 8 en 2014 : Lambdas et Stream (non abordés dans ce cours)
- Version 18 aujourd'hui (en 2022)
- Java est un des langages de programmation les plus utilisés aujourd'hui

Principales caractéristiques de JAVA

- Syntaxe proche du C

Principales caractéristiques de JAVA

- Syntaxe proche du C
- Langage compilé s'exécutant sur une machine virtuelle :
  - compilation $\rightarrow$ *bytecode*
  - la machine virtuelle exécute le bytecode
  - une machine virtuelle par architecture/système d'exploitation
  - un même code compilé s'exécute de manière identique sur toutes les machines possédant une machine virtuelle $\rightarrow$ langage très portable

Principales caractéristiques de JAVA

- Syntaxe proche du C
- Langage compilé s'exécutant sur une machine virtuelle :
  - compilation $\rightarrow$ *bytecode*
  - la machine virtuelle exécute le bytecode
  - une machine virtuelle par architecture/système d'exploitation
  - un même code compilé s'exécute de manière identique sur toutes les machines possédant une machine virtuelle $\rightarrow$ langage très portable
- Langage fortement typé $\rightarrow$ + de contrôles par le compilateur $\rightarrow$ moins d'erreurs à l'exécution

Principales caractéristiques de JAVA

- Syntaxe proche du C
- Langage compilé s'exécutant sur une machine virtuelle :
  - compilation $\rightarrow$ *bytecode*
  - la machine virtuelle exécute le bytecode
  - une machine virtuelle par architecture/système d'exploitation
  - un même code compilé s'exécute de manière identique sur toutes les machines possédant une machine virtuelle $\rightarrow$ langage très portable
- Langage fortement typé $\rightarrow$ + de contrôles par le compilateur $\rightarrow$ moins d'erreurs à l'exécution
- Gestion mémoire par Garbage Collector (i.e. ramasse miettes)

Principales caractéristiques de JAVA (suite)

- Une librairie standard de plusieurs milliers de classes
- Java Runtime Environment (JRE) : uniquement pour exécuter des programmes compilés
- Java Development Kit (JDK) : pour compiler (et exécuter) des programmes. JDK = JRE + compilateur et outils de développement
- La machine virtuelle JAVA est aussi utilisée par d'autres langages de programmation dont le compilateur produit du bytecode (Groovy, Scala, Kotlin, Clojure, ...)