METHODES LOGICIELLES (ET JAVA)

Cours de Centrale Pékin — 5A

Olivier ROUX

olivier.roux@ec-nantes.fr

2019/2020

Cours

Je me présente

Olivier ROUX

- olivier.roux@ec-nantes.fr

- Ecole Centrale Nantes & Institut Universitaire de France
  Professeur au département Informatique & Mathématiques
  Responsable de l'équipe MeForBio (informatique pour la biologie des systèmes) au laboratoire LS2N
  (unité associée au CNRS)

- Ecole Centrale Pékin
  En 2013: Dans l'option ITR: P13_ITR
  (En 2017: Méthodes logicielles)
  En 2018: Méthodes logicielles
  En 2019: Méthodes logicielles

Je présente MELOG

~ 32 heures
Yu Lei & Olivier Roux

- Une partie de **cours** *(~14h)*

- 1 **Travail Dirigé** illustrant une application *(TD ~2h)*

  Bibliothèque

- 4 **Travaux d'Applications** en salle machine *(TA ~14h)*

  Figures géométriques

  Lexique

  Expressions mathématiques

  Personne

- 1 **examen** *(2h)*

Structure du module « METHODES LOGICIELLES »

- **Objectifs**: ne pas faire de vous des informaticiens mais des ingénieurs « *avancés en informatique* »
- Trois parties :
  - La programmation à objets —
  - Le langage JAVA —
    - Les structures et méthodes pour les logiciels —
      (où les objets sont bien utiles…)

Je passerai souvent de l'une à l'autre

SOMMAIRE

- INTRODUCTION AU COURS « Méthodes logicielles et JAVA »
- LA NOTION D'OBJET et de CLASSE
- L'HERITAGE, le polymorphisme et la liaison dynamique
- INTRODUCTION au LANGAGE JAVA
- Le TRAITEMENT d'une APPLICATION COMPLETE
- Les STRUCTURES DE DONNEES

Je donnerai beaucoup d'exemples

Posez-moi beaucoup de questions

Partie I du cours

Programmation Objet, Classes et Héritage

- Pourquoi Objets et Java ?
  - Les objets parce que ça facilite la programmation
  - Java parce que:

Début de l'ouvrage de référence, The Java Language Specification par J. Gosling, B. Joy et G. Steele [GJS96] :
"***Java is a general purpose, concurrent, class-based, object-oriented language. It is designed to be simple enough so that many programmers can achieve fluency in the language. Java is related to C and C++ but is organized rather differently, with a number of aspects of C and C++ omitted and a few ideas from other languages included. Java is intended to be a production language, not a research language, and so, as C.A.R. Hoare suggested in his classic paper on language design, the design of Java has avoided including new and untested features.***"

  - Les structures parce que: on veut faire des choses complexes et solides

- [*http://webpages.lss.supelec.fr/perso/hugues.mounier/Teaching/Java_files/JCours/polyBasesJavaHM.pdf*]:
  Java est un langage : • (*1*) simple, • (*2*) orienté objet, • (*3*) réparti, • (*4*) interprété (ou compilé), • (*5*) robuste, • (*6*) sûr, • (*7*) indépendant de l'architecture, • (*8*) portable, • (*9*) efficace • (*10*) multitâches ou multi-activités (multi-thread) et • (*11*) dynamique.

- (*1*) **Java est simple**
  • plus simple que C++ :
    - Nombreux mots clés éliminés.
    - Pas de pré-processeur.
    - Bibliothèque très étendue et directement intégrée au langage.
    - Pas de surcharge d'opérateurs, de fonctions indépendantes, de goto, de structures, d'unions ni de pointeurs.
    - Pas de fichiers d'en-tête.
    - Pas d'héritage multiple ; à la place, notion d'interface, venant d'Objective C. Bien moins complexe.

  • pas de pointeurs visibles au niveau du programmeur.

- (*2*) **Java est orienté objet**
  bénéfices comme l'encapsulation et la réutilisabilité.

Introduction au cours
« Programmation Objets et introduction au langage JAVA » (suite 2)

- (*3*) **Java est réparti**
  - Java a été construit avec Internet en tête. Riche bibliothèque pour:
    - l'accès aux URL (Universal Resource Locators),
    - la programmation client/serveur via des sockets TCP et UDP,
    - l'exécution de méthodes distantes (RMI : Remote Method Invocation),
    - la gestion de serveurs Web via les Servlets,
    - la communication d'objets distants inter-langages avec des IDL (Interface Definition Language) CORBA (Common Request Broker Architecture),
    - l'administration de réseaux via SNMP (Simple Network Management Protocol) avec JMAPI.

- (*4*) **Java est interprété (ou compilé)**
  - Le source java est éventuellement transformé en un assembleur d'une machine virtuelle. Cet assembleur, ou bytecode, peut être interprété. Désavantage : lenteur d'exécution.
  - Notion de compilateur "à la volée" ou "juste à temps". La Traduction du bytecode au langage machine est effectuée juste avant l'exécution.
  - Performances avoisinant celles des langages compilés classiques.

  Puis, apparition de compilateurs natifs, avec des performances égales à celles du C.

Introduction au cours
« Programmation Objets et introduction au langage JAVA » (suite 3)

- (*5*) **Java est robuste**
  - Gestion des erreurs matérielles et logicielles, via un mécanisme d'exceptions.

(Exemples : ouverture d'un fichier inexistant, division par zéro, création d'un point de communication réseau (socket) vers une @IP inexistante, ...Le programmeur est forcé de gérer diverses exceptions.)

  - Gestion automatique de la mémoire; présence d'un "ramasse-miettes",
  - Vérification à l'exécution des compatibilités de type.

- (*6*) **Java est sûr**
  - Écriture mémoire erronée: quasi-impossible en Java, car pas de pointeurs.
  - Indices de tableau testés avant qu'ils soient référencés.
  - Test qu'une variable a été assignée avant d'être utilisée.
  - Bytecode également testé avant d'être exécuté :
    - test de bon accès aux classes,
    - tests de congestion et de famine de la pile des opérandes,
    - test de conversion illégale de données,
    - test d'accès aux ressources : fichiers,
    - ...

- (*7*) **Java est indépendant de l'architecture**
  - Bytecode indépendant de la plate-forme.
  - Bibliothèques intégrées de manière standard au langage, contrairement à C++

Introduction au cours
« Programmation Objets et introduction au langage JAVA » (suite 4)

- (*8*) **Java est portable**
  - Un même programme peut être compilé sur une machine et exécuté sur une autre, (quel que soit le processeur ou l'OS).
  - La taille des types de données est toujours la même en Java.

- (*9*) **Java est efficace**
  - Initialement, les interpréteurs rendaient l'exécution de programmes Java lente (environ 20 fois plus lente que du C).
  - Les compilateurs à la volée (JIT) la rendent presque aussi rapide que des programmes compilés classiques.
  - Des compilateurs natifs, fournissent du code machine natif pour tel ou tel processeur; performances égales à celles du C (jove; cygnus, au dessus de gcc, ...).

- (*10*) **Java est multitâches**
  - L'un des premiers langages à posséder en interne des tâches, ou activités (threads) d'exécution.
  - La coordination des activités est aisée (moniteurs de Hoare et événements).

- (*11*) **Java est dynamique**
  - Exécution coté client ⇒ dynamisme plus aisé à mettre en œuvre que dans d'autres langages.
  - Chargement des classes en cours d'exécution, lorsque nécessaire, éventuellement à travers le réseau. Chargement dynamique des classes possible grâce à des informations de typage consultables en cours d'exécution.

Introduction au cours
« Programmation Objets et introduction au langage JAVA » (suite 4)

- (*8*) **Java est portable**
  - Un même programme peut être compilé sur une machine et exécuté sur une autre, (quel que soit le processeur ou l'OS).
  - La taille des types de données est toujours la même en Java.

- (*9*) **Java est efficace**
  - Initialement, les interpréteurs rendaient l'exécution de programmes Java lente (environ 20 fois plus lente que du C).
  - Les compilateurs à la volée (JIT) la rendent presque aussi rapide que des programmes compilés classiques.
  - Des compilateurs natifs, fournissent du code machine natif pour tel ou tel processeur; performances égales à celles du C (jove; cygnus, au dessus de gcc, ...).

- (*10*) **Java est multitâches**
  - L'un des premiers langages à posséder en interne des tâches, ou activités (threads) d'exécution.
  - La coordination des activités est aisée (moniteurs de Hoare et événements).

- (*11*) **Java est dynamique**
  - Exécution coté client ⇒ dynamisme plus aisé à mettre en œuvre que dans d'autres langages.
  - Chargement des classes en cours d'exécution, lorsque nécessaire, éventuellement à travers le réseau. Chargement dynamique des classes possible grâce à des informations de typage consultables en cours d'exécution.

Notion d'objet

- A côté des langages impératifs, fonctionnels, logiques, …
- Les objets: des composants logiciels à composer

- Des entités autonomes, encapsulées, communicantes et interagissantes, réutilisables et construites suivant les besoins