Introduction à la programmation objet

Objectifs du module

Approche objet de la programmation

- Connaître et maîtriser les concepts de la programmation objet
  - Classe, instance/objet, attribut, méthode, constructeur, encapsulation, héritage, interface, polymorphisme, liaison tardive
- Adopter le "penser objet"
  - Savoir décomposer un problème en classes et objets
  - Savoir expliquer ce qui différencie le paradigme objet des autres paradigmes
- Connaître les principes ouvert-fermé, de substitution de Liskov et *KISS* (*Keep It Simple, Stupid*), et savoir les appliquer

Objectifs du module

Le langage Java

- Connaître les principaux éléments de la syntaxe du langage Java et pouvoir expliquer clairement leur rôle et leur sémantique
  - `new`, `this`, `super`, `public`, `private`, `protected`, `static`, `final`, `extends`, `implements`, `package`, `import`, `enum`, `throws`, `throw`
- Savoir écrire (et corriger) un programme dans le langage Java
  - Maîtriser les "outils" pour développer en Java : `javac`, `java` (et `classpath`), `javadoc`, `jar`, *IDE*, débogueur
  - 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 la capture et la levée d'une exception

Au menu

Au menu

Paradigme de programmation

- Est un style fondamental de programmation informatique qui traite de la manière dont les solutions aux problèmes doivent être formulées dans un langage de programmation (source : `Wikipédia`)
- Exemples de paradigme de programmation
  - Paradigme impératif (*e.g.*, Pascal, C)
  - Paradigme objet (*e.g.*, Java)
  - Paradigme fonctionnel (*e.g.*, Lisp)
  - Paradigme logique (*e.g.*, Prolog)
  - *etc.*

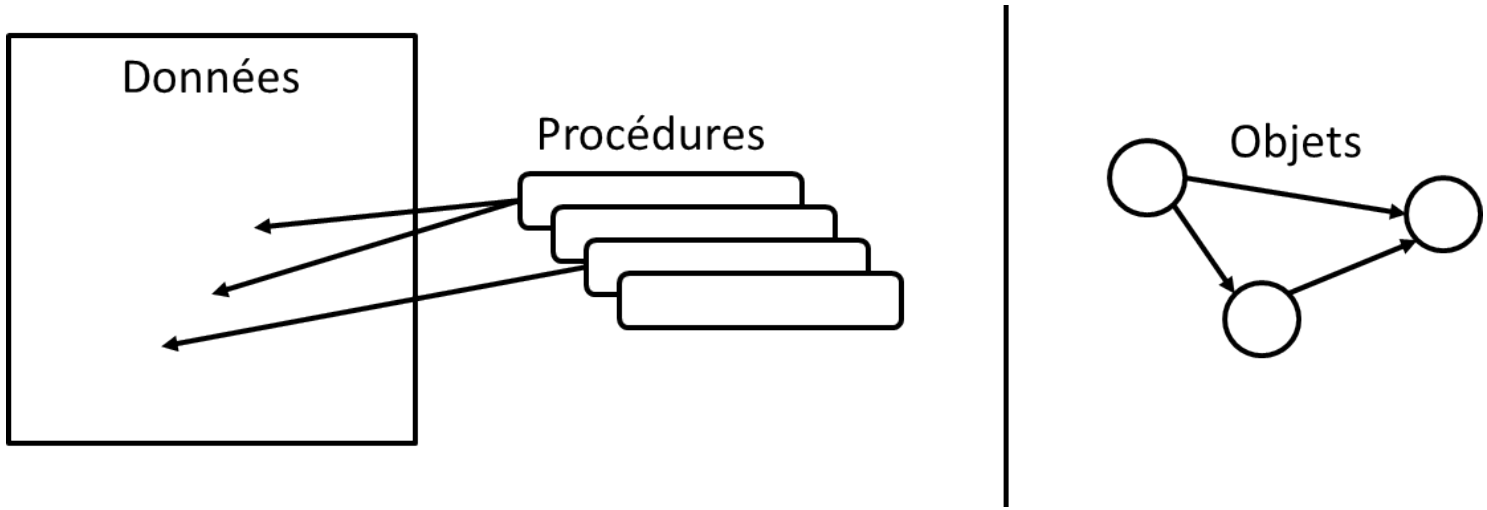
Programmation impérative et modulaire

- Programmation impérative
  - Un programme est une séquence d'instructions exécutées par un ordinateur pour modifier son état
  - Instructions : affectations, séquences, structures conditionnelles et itératives
- Programmation modulaire
  - Un programme est décomposé en éléments plus simples afin de faciliter son développement et permettre la réutilisation
    - ★ Principe diviser pour mieux régner
  - Éléments : procédures, fonctions, modules, unités
- Exemples de langage : Pascal, C...

Programmation objet

- Programmation objet
  - Un programme est un ensemble d'objets qui interagissent
  - Reprend et prolonge la démarche modulaire : décomposition d'un problème en parties simples
  - La programmation des traitements reste impérative
  - Plus intuitive car s'inspire du monde réel pour une modélisation plus naturelle
  - Facilite la réutilisation et la conception de grandes applications
- Exemples de langage : Java, C++, C#, Python, PHP5...

Programmation impérative vs programmation objet



Historique des langages de programmation objet

- 1967 : Simula
  - Langage à classes
- 1972 : Smalltalk
  - Langage "pur" objet
- 1983 : C++
- 1986 : Eiffel
- 1988 : CLOS (*Common Lisp Object System*)
- 1991 : Python

Historique de Java

- Début des années 90 : langage Oak (chêne)
  - Créé par James Gosling (*Sun Microsystems*)
  - Destiné à la programmation des systèmes embarqués
  - Objectif principal : améliorer le C++
  - Rebaptisé Java en 1994
- 1995 : Java 1.0
  - Lien avec le Web (applet)
  - Versions 1.1 (JavaBeans, RMI, JDBC)
- 1998 : Sun appelle Java 2 les versions de Java
  - 1.2 (Swing, optimisation JVM, collections), 1.3, 1.4 (assertions, regexp)
- 2004 : Java 5.0 (annotations, types génériques, `enum`, `foreach`)
  - Changement dans le système de numérotation (mais encore JDK 1.5)
- 2006 : Java 6 (amélioration de l'API, intégration SGBD)
- 2010 : Oracle rachète Sun
- 2011 : Java 7
- 2014 : Java 8 (version courante)

Caractéristiques de Java (1)

- Simple et familier
  - Basé sur C/C++, sans certaines caractéristiques compliquées ou mal utilisées (*e.g.*, pas de pointeur, pas de gestion explicite de la mémoire)
- Orienté objet
  - Modèle objet propre tout en fournissant un accès à des types primitifs (`int`, `float`, *etc.*)
    - ★ Approche hybride adoptée pour des raisons de performance qui sont aujourd'hui largement obsolètes
  - Héritage simple + interfaces
  - Vaste bibliothèque standard (réutilisation)
- Portabilité du code source et des fichiers binaires (*bytecode*)
  - *Write once, run anywhere*
    - ★ Un code Java peut s'exécuter partout (*i.e.*, quels que soient le matériel et le système d'exploitation) où il existe une machine virtuelle Java (du moins en théorie)
  - Bibliothèque standard indépendante
  - Définition (sémantique) précise du langage

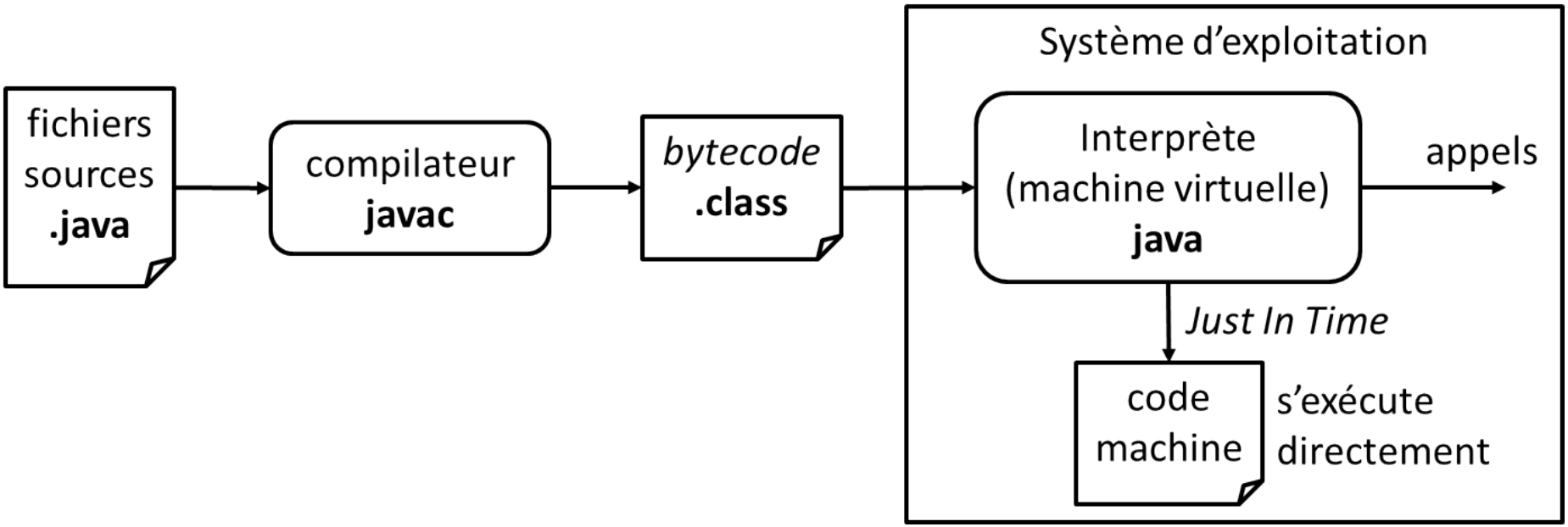
Caractéristiques de Java (2)

- Sûr
  - Fortement typé
    - ★ Vérification de type statique
  - Transtypage contrôlé
  - Contrôle de l'accès à la mémoire
    - ★ Pas de risque d'écrasement, pas de dépassement de tampon
    - ★ Pas d'arithmétique des pointeurs
    - ★ Vérification des bornes d'un tableau
  - Gestion automatique de la mémoire (ramasse-miettes)
    - ★ Pas de fuite mémoire
- Sécurisé
  - Interprété, s'exécute sur une machine virtuelle protégée
  - Bac à sable
    - ★ Par exemple, le code d'un applet ne peut pas accéder à la machine (sauf par des moyens clairement définis)
  - Vérification du *bytecode*, signature de code, autorisation d'accès

Caractéristiques de Java (3)

- Dynamique
  - Chargement de classes et édition des liens dynamiques
    - ★ Permet d'étendre des systèmes à l'exécution
    - ★ Minimise les re-compilations et facilite la modularité
  - Introspection
    - ★ Capacité d'un programme à examiner et modifier sa structure et son comportement à l'exécution
- Distribué
  - Applets, servlets, RMI, Corba
- *Multi-threadé*
  - Exécution parallèle dans le même espace d'adressage

Comment marche Java ?



- Java est un langage compilé et interprété !
- *Bytecode* est un code intermédiaire pour la JVM, indépendant de la plate-forme, qui ne peut pas être directement exécuté par la machine
  - Quelle que soit la plate-forme (Windows, Linux, MacOS, *etc.*) :
    - ★ Est obtenu par compilation identique
    - ★ S'exécute à l'identique
  - Moins performant que le code natif (*e.g.*, `.exe`) ?

Performances de Java

- Java a souffert des problèmes de performance pendant de nombreuses années par rapport à d'autres langages qui ont été directement compilé pour une plate-forme/machine particulière
  - Par exemple, C/C++
- Aujourd'hui, l'utilisation de la compilation à la volée (*Just In Time*) a largement éliminé ces problèmes
- La JVM est continuellement améliorée avec de nouvelles techniques
  - Interfaces de code natif (accès à des bibliothèques C) pour gagner en vitesse si nécessaire
  - Cache mémoire pour éviter le chargement (et la vérification) multiple d'une même classe
  - Ramasse-miettes : processus indépendant de faible priorité
- Java fournit d'excellentes performances pour de nombreux *frameworks* dans de nombreux domaines
- Minecraft est développé en Java + OpenGL

Que faut-il pour faire du Java ?

- Un éditeur de texte : emacs, vi, Notepad++, *etc.*
- Un kit de développement : JDK (*Java Development Kit*)
  - `javac` : compilateur
  - `java` : machine virtuelle pour une plate-forme particulière
  - `javadoc` : générateur de documentation HTML
  - `jar` : constructeur d'archives
  - `jdb` : débogueur
  - *etc.*
- Des outils d'automatisation : Ant, Makefile, *etc.*
- Un environnement de développement (IDE - *Integrated Development Environment*) : Eclipse, IntelliJ, Netbeans, *etc.*

Différentes plate-formes

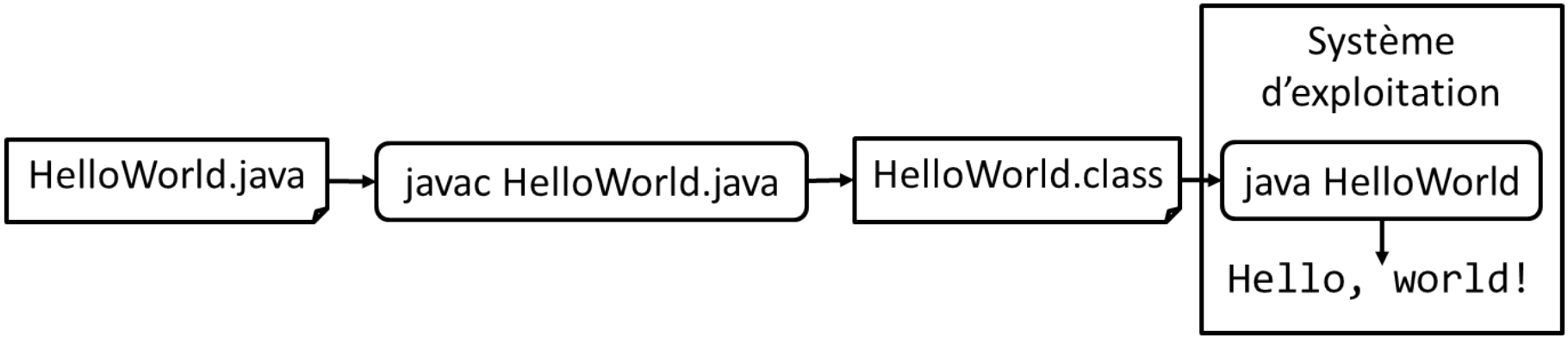
- *Java Platform, Standard Edition* (Java SE)
  - *Java Runtime Environment* (JRE) : environnement d'exécution
    - ★ Java API, JVM, *etc.* pour exécuter une application/applet Java
  - *Java Development Kit*(JDK) : kit de développement
    - ★ JRE + outils de développement (compilateur, *etc.*)
- *Java Platform, Enterprise Edtion* (Java EE)
  - Développement d'applications d'entreprise multi-couches (client/serveur) orientées composants (JavaBeans), services Web (servlet, JSP, XML), *etc.*
  - Inclus Java SE
- *Java Platform, Micro Edition* (Java ME)
  - Développement d'applications pour les téléphones mobiles, PDA et autres systèmes embarqués
  - Optimisé pour la mémoire, la puissance de traitement et les entrées/sorties

Premier programme : HelloWorld.java

```
public class HelloWorld {
   public static void main(String[] args) {
      System.out.println("Hello, world!");
   }
}
```

- Une classe par fichier
- Le nom de la classe est le même que celui du fichier

Compilation et exécution



- Compilation : `> javac HelloWorld.java`
  - Détermine les dépendances et compile tous les fichiers nécessaires
    - ★ Il suffit donc de compiler la classe principale
  - Produit autant de fichiers `.class` qu'il y a de classes (ici, `HelloWorld.class`)
- Exécution : `> java HelloWorld`
  - Lance la JVM en exécutant la méthode `main` de la classe `HelloWorld`
    - ★ Attention à ne pas mettre d'extension derrière le nom de la classe !
    - ★ Peut être suivie d'arguments
  - Affiche dans la console : `Hello, world!`

Classpath

- Par défaut, les outils du JDK cherchent les classes dans le répertoire courant
- Si les classes sont dans plusieurs répertoires, utiliser le `classpath` :
  - Soit avec l'option `-classpath` des outils du JDK
  - Soit avec la variable d'environnement `CLASSPATH`
- Nécessaire dès que des bibliothèques (*e.g.*, JUnit, Log4J) qui sont dans des répertoires ou fichiers d'archive (.jar) propres sont utilisées
- Exemples :
  - Unix : `javac -classpath /foo/junit.jar:. HelloWorld.java`
  - Windows : `java -classpath \foo\junit.jar;. HelloWorld`
  - Les classes sont cherchées dans `junit.jar`, puis dans le répertoire courant (`.`)

La méthode principale `main`

```
public static void main(String[] args) {
   ...
}
```

- Est publique et statique
- Ne retourne pas de valeur (`void`)
- Prend un seul paramètre : un tableau de chaîne de caractères correspondant aux arguments de la ligne de commande
- Est le point de départ de l'exécution du programme
- Chaque classe peut ou non définir sa méthode principale

Un peu de lecture

- James Gosling, Bill Joy, Guy Steele, and Gilad Bracha, *The Java Language Specification*, Addison-Wesley, 3rd Edition, 2005
- Kathy Sierra and Bert Bates, *Head First Java*, O'Reilly Media, 2nd Edition, 2005
- Bruce Eckel, *Thinking in Java*, Prentice-Hall, 4th Edition, 2006
- Joshua Bloch, *Effective Java*, Addison-Wesley, 2nd Edition, 2008
- Ben Evans and David Flanagan, *Java in a Nutshell*, O'Reilly Media, 6th Edition, 2014
- Java API : `http://docs.oracle.com/javase/7/docs/api/`