Introduction à Java

Matthieu Herrb
CNRS-LAAS
http://homepages.laas.fr/matthieu/cours/java/

Février 2018

Plan

Histoire et motivations

Langage développé par James Gosling chez Sun Microsystems (Oracle) à partir de 1990

Pour des environnements embarqués :

- appareils spécifiques ("Appliances")
- applications Web

Contraintes :

- indépendant du matériel
- sécurité

La machine virtuelle Java

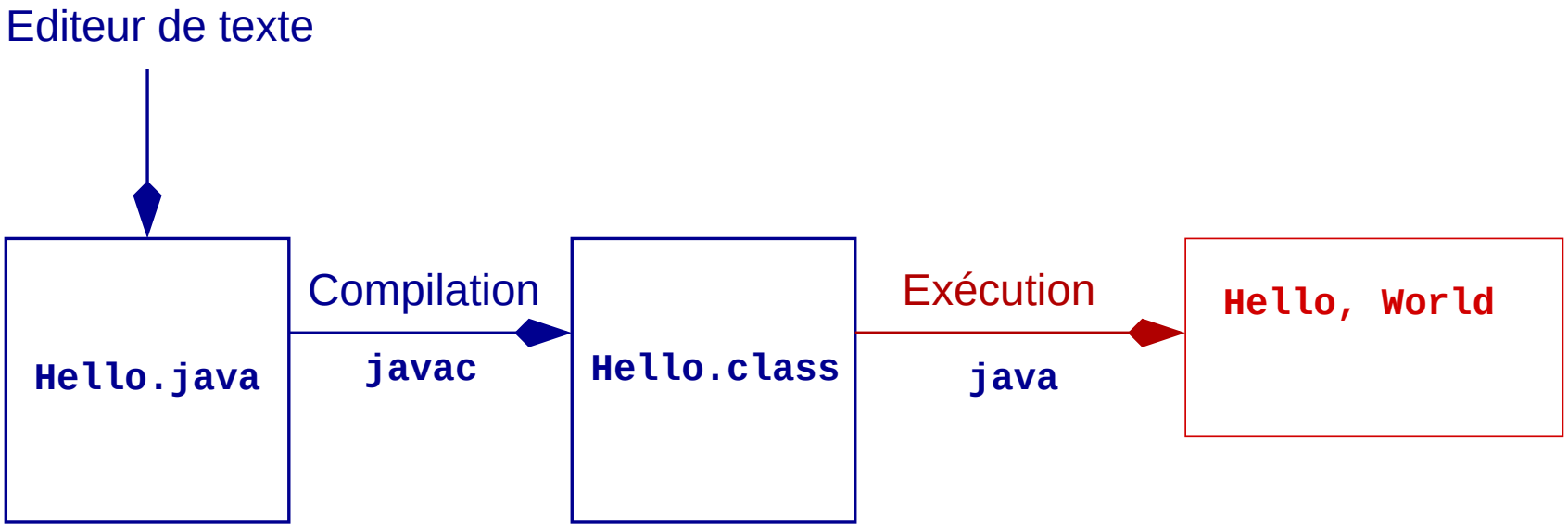
Le langage Java utilise une *machine virtuelle* :

- le compilateur produit un **bytecode**
- ce code est indépendant de l'architecture matérielle sur laquelle il est exécuté
- la **machine virtuelle** interprète ce bytecode et le transforme en code machine natif à la volée (*Just in time compilation*)

Machines virtuelles Java

HotSpot	(Oracle JRE/JDK)
OpenJDK, IcedTea	Implémentations sous licence libre
ART	Android Runtime (Google) (Dalvik)
Jazelle	extension du jeu d'instruction de certains processeurs ARMv5 pour exécution directe de byte-code java.
ThumbEE	généralisation de Jazelle sur processeurs ARMv7. Pas spécifique de Java (supporte aussi C#, Perl, Python)

Cycle de développement



Premier programme java

```java
/**
 ** Premier programme java
 **/
public class Hello {
    public static void main(String argv[]) {
        System.out.println("Hello, World!");
    }
}
```

Classes et objets

Classe définit un type de données, et des fonctions (*méthodes*) qui le manipulent.

Objet une *instance* de classe : une variable du type définit par la classe et ses méthodes.

Héritage permet de créer des classes qui reprennent les données de la classe parente.

Interface définition des prototypes des méthodes d'une classe sans préciser leur implémentation.

Gestion de la mémoire

Java simplifie la gestion de la mémoire

- pas de pointeurs : les références aux objets sont implicites, copies automatiques lorsque nécessaire.
- pas besoin de détruire ou libérer explicitement les objets : ceux qui ne sont plus utilisés sont recyclés par le processus **ramasse-miettes**

Classes standard

Environnement de classes standardisées et prédéfinies :

- types étendus (java.lang)
- entrées/sorties (java.io)
- réseau (java.net)
- ...

Documentation standardisée : javadoc

Documentation des classes et API standard :
`http://docs.oracle.com/javase/8/docs/api/`

Plan

Variables

Quatre catégories de variables :

variables d'instance : données d'un objet

variables de classe : variable globale d'une classe,
existe en un seul exemplaire pour tous les objets de la classe.
déclarée dans une classe avec `static`.
`final` déclare une variable de classe constante.

variables locales : déclarées dans le corps d'une méthode,
locales à la méthode qui les déclare.

paramètres : pseudo-variables utilisées pour décrire les paramètres d'une méthode

Types de données de base

byte	1 octet ($-128..+127$)
short	entiers sur 16 bits ($-32768..+32767$)
int	entiers sur 32 bits
long	entiers sur 64 bits
float	nombres réels sur 32 bits
double	nombre réels sur 64 bits
boolean	variable binaire (`true` ou `false`)
char	caractère Unicode

Il existe aussi des classes pour les types de base (**Number** et **String**).

Tableaux

Collections d'objets dont la taille est fixée à la création.

```
int[] anArray;
anArray = new int[10];
int[] anotherArray = {100, 200, 300, 400};
anArray[0] = anotherArray[0];
```

Opérateurs

opérateur	précédence
postfix	*expr*`++` *expr*`--`
unaire	`++`*expr* `--`*expr* `+`*expr* `-`*expr* `~` `!`
multiplicatif	`*` `/` `%`
additif	`+` `-`
décalages	`«` `»` `»>`
relations	`<` `>` `<=` `>=` `instanceof`
égalités	`==` `!=`
et bit à bit	`&`
ou exclusif bit à bit	`^`
ou inclusif bit à bit	`|`
et logique	`&&`
ou logique	`||`
ternaire	`?` `:`
affectation	`=` `+=` `-=` `*=` `/=` `%=` `&=` `^=` `|=` `«=` `»=` `»>=`

Instructions

- `if (`*expr*`) { ... }`
- `if (`*expr*`) { ... } else { ... }`
- `switch case`
- `while (`*expr*`) { ... }`
- `do { ... } while (`*expr*`)`
- `for (`*initialisation*`;` *terminaison*`;` *incrément*`) { ... }`
- `break`

Plan

Déclaration de classes

```
class MaClasse {
    // variables, constructeurs, méthodes...
}
```

Classe dérivée :

```
class MaClasse extends MaSuperClasse {
    // variables, constructeurs, méthodes...
}
```

Implémentation d'une interface :

```
class MaClasse implements MonInterface {
    // variables, constructeurs, méthodes...
}
```

Constructeurs

Le constructeur porte le nom de la classe, ne retourne rien.
Appelé par l'opérateur **new**.
Exemple :

```
public class Cube {
        float dimension;
        String couleur;

        public Cube(float d, String c) {
                dimension = d;
                couleur = c;
        }
}
..
        Cube c = new Cube(1.0, "rouge");
```

Objets

- Créer les objets avec l'opérateur **new**
- Appel des méthodes de l'objet avec .

```
public class Cube {
        float dimension;
        ...

        public float volume() {
                return dimension*dimension*dimension;
        }
}

Cube c = new Cube(0.75, "vert");
System.out.println("le volume de c est "
                   + c.volume + "m3");
```

Méthodes et variables statiques

Méthodes et variables qui ne sont pas attachés à une instance particulière

$\rightarrow$ utilisé pour le ń `main` ż d'une classe

$\rightarrow$ variables ou constantes globales

```
class MonAppli {
  public static void main(String argv[]) {
    ....
  }
  public static int MAX_OBJ = 100;
}
```

Exceptions

Méthode de traitement des erreurs.

```
try {
      code
} catch (ExceptionType1 nom1) {
      .... traitement exception 1
} catch (ExceptionType2 nom2) {
      .... traitement exception 2
}
```

En Java il est **interdit** d'ignorer une exception !

Génération d'une exception

- Opérateur **throw**
- Classe **Exception**

Exemple :

```
try {
    ... code
    if (erreur)
        throw new MyException("bug de logique");
    ... code
} catch (MyNewException e) {
    writeln("exception: " + e);
}
```

Threads

- Classe `Thread` / Interface `Runnable`.
- La méthode `run` est la fonction principale du thread.

```
class MyThread extends Thread {
  int data;
  MyThread(int data) {
    this.data = data;
  }

  @Override public void run() {
    // code du thread
  }
}
```

Thread(2)

```
MyThread p = new MyThread(42);
p.start();
writeln("new thread: " + p.toString());
// Augmente la priorité
p.setPriority(Thread.MAX_PRIORITY);
...
```

Synchronisation

Moniteurs

- mot clé **`synchronized`** dans la déclaration d'une méthode.
- condition anonyme : **`wait()`**
- signal sur une condition : **`notify()`** ou **`notifyAll()`**.

```
public synchronized void P(Semaphore s) {
     while (s.valeur == 0) {
         try {
             wait();
         } catch (InterruptedException e) {}
     }
     s.valeur--;
}
```

Plan

Packages

Mécanisme pour grouper un ensemble de classes et les rendre accessibles aux applications.
Un environnement Java (JRE) fourni un ensemble de packages standard.
Exemples :

`System`	fonctions système, en particulier entrées/sorties
`java.lang.Math`	fonctions mathématiques
`java.util.Random`	nombre aléatoires
`javax.swig`	boite à outils pour applications graphiques

Utilisation : `import java.util.Random;`

Exemple: philosophes et spaghettis

```
public synchronized void arriver(int i) {
    philo[i].etat = DEMANDEE;
    while (! accessible(i)) {
        try {
            wait();
        } catch (InterruptedException e) {}
    }
    philo[i].etat = OCCUPEE;
}
public synchronized void quitter(int i) {
    philo[i].etat = LIBRE;
    notifyAll();
}
```

Exemple: philosophes et spaghettis (2)

```
// voisins
private Philosophe gauche(int i) {
        return philo[i == 0 ? 4 : i - 1] ;
}

private Philosophe droite(int i) {
        return philo[(i + 1) % 5];
}

// Accessibilite
private boolean accessible(int i) {
        return (philo[i].etat == DEMANDEE
            && gauche(i).etat != OCCUPEE
            && droite(i).etat != OCCUPEE);
}
```

Questions ?