Cours de Java

Paul Bedaride

Formation Continue

15 mai 2008

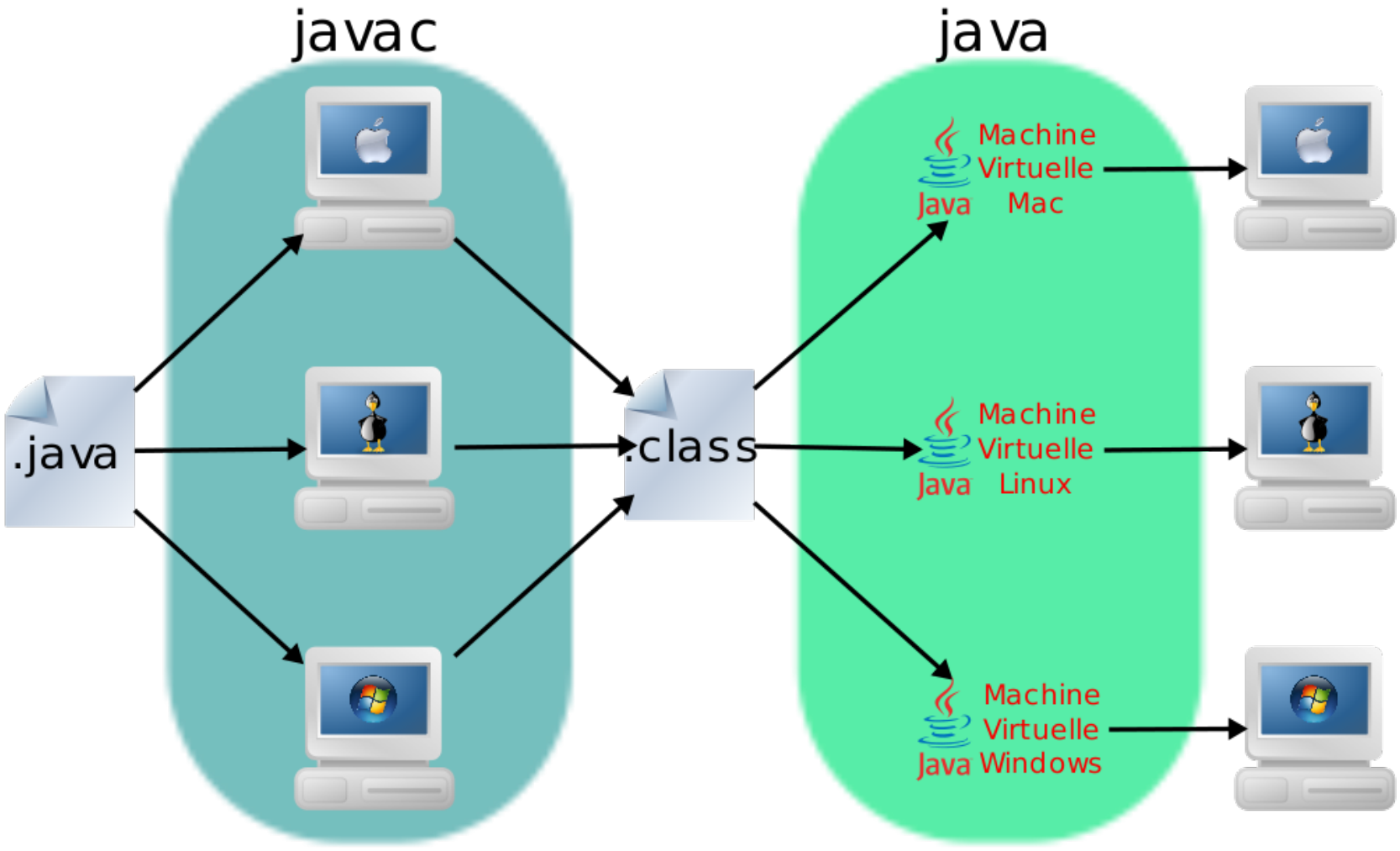
Qu'est-ce que Java ?

- Langage de programmation
  - syntaxe proche du C
  - orienté objet
- Plateforme
  - Compilateur

    `javac fichier.java -> fichier.class`
  - Machine Virtuelle

    `java fichier`

Un langage multi-plateformes



Quelques exemples de codes en C et Java

Code minimal

Java

```
public class Minimum{
  public static void main(String[] →
      ↪ args){
  }
}
```

C

```
main(){
  return 0;
}
```

HelloWorld

Java

```
public class hello_world {
  public static void main(String[] →
      ↪ args) {
    System.out.println("Hello →
        ↪ World");
  }
}
```

C

```
#include <stdio.h>
main() {
    puts("Hello World");
    return 0;
}
```

- Commentaires
- Classes
  - constructeur
- Package
- Fonction main
- Variable
  - typé
  - déclaration
  - instanciation

Fichier DateDuJour.java

```
/**
 * Cette application \'ecrit la date du jour
 * en toutes lettres (e.g. lundi 1 janvier
      ↪ 2007)
 */
import java.util.Date;
import java.text.SimpleDateFormat;

class DateDuJour {
  public static void main(String[] args) {
    // creer la date du jour
    Date dateDuJour;
    dateDuJour = new Date();
    // creer son modele de presentation
    SimpleDateFormat fd;
    fd = new SimpleDateFormat("EEEE dd MMMM
         ↪ yyyy");
    // afficher la date du jour selon ce
         ↪ modele
    System.out.println(fd.format(dateDuJour))
         ↪ ;
  }
}
```

Les types élémentaires

- Les entiers :
  - `byte` (8 bits), `short` (16 bits), `int` (32 bits), `long` (64 bits)
  - Exemples : `3 125 0 3456 234`
  - Opérateurs : `+ - * / % < <= == != > >=`
- Les réels :
  - `float` (32 bits), `double` (64 bits)
  - Exemples : `12.45 34.0 12. 56e34 4E-5 45.67e2 67`
  - Opérateurs : `+ - * / < <= == != > >=`
- Les booléens :
  - `boolean`
  - Exemples : `true false`
  - Opérateurs : `! | ^ & || &&`
- Les caractères :
  - `char`
  - Exemples : `'a' 'c' '\n' '\t' '\''`
  - Opérateurs : `< <= == = > >=` !

Les identificateurs

- Les variables :

```
int age;
char voyelle, consonne;
double delta;
boolean majeur;
age = 10;
voyelle = 'e';
delta = 13.15;
majeur = age >= 18;
age = 20;
int taille = 210;
char voyelle, consonne = 'z';
```

- Les constantes (une seule affectation possible) :

```
final double pi = 3.141562653;
final double deuxpi;
deuxpi = 2 * pi;
```

Les énoncés I

- Les affectations :

```
age = 25;
q = false;
age += 3; // age = age + 3;
q |= true; // q = q | true;
// age = 28
x = age++;
// age = 29 et x = 28
x = ++age;
// age = 30 et x = 30
```

- Les blocs :

```
// x est defini
{
  x = x+1;
  double y = Math.sin(1.34);
  // y est defini
  y *= x;
}
// y n'existe plus
```

Les énoncés II

- Le si-alors-sinon :

```
if (a>b) max = a; else max = b;
// ou
max = b;
if (a>b) max = a;
// ou
max = (a<b) ? a : b;

if (x>0) {
  System.out.println("x est inferieur a 0");
} else if (x>0) {
  System.out.println("x est superieur a 0");
} else {
  System.out.println("x vaut 0");
}
```

Les énoncés III

- Le switch :

```
// calculer c = a op b, avec op = + - x ou /
switch (op) {
  case '+' : c = a + b; break;
  case '-' : c = a - b; break;
  case 'x' : c = a * b; break;
  case '/' : c = a / b; break;
  default  : System.err.println("operateur inconnu");
}
switch (op) {
  case '+' :
  case '-' : System.out.println("op additif"); break;
  case 'x' :
  case '/' : System.out.println("op multiplicatif");  →
      ↪ break;
  default  : System.err.println("operateur inconnu");
}
```

Les énoncés IV

- Le while :

```
n = 5; i = 0; fact = 1;
while (i<n) {
  i++;
  fact *= i;
}
// i = n et fact = n!
```

- Le do-while :

```
int somme = 0; x = 0;
do {
  x = StdInput.readlnInt();
  somme += x;
} while (x >0);
```

Les énoncés V

- Le for :

```
int somme = 0; int impairs = 1;
for (int i = 0; i < n; i++) {
  somme += impais;
  impairs += 2;
}
// somme = 1 + 3 + 5 + ... (n fois)
```

- Le foreach :

```
somme = 0; int[] a = {4, -5, 12};
for (int i : a)
        somme += i;
```

Les chaînes de caractères

- package `java.lang` contient :
  - `String` (chaîne constante)
  - `StringBuilder` (chaîne modifiable dynamiquement)
- opérateur + concatène des chaînes, et de façon plus générale des objets munis de la méthode `toString()`

```
String s = "Bonjour";
System.out.println(s.length() + s.charAt(3)); // "7j"
StringBuilder sb = new StringBuilder("Bonjour");
sb.setLength(8);
sb.setCharAt(7,'!');
System.out.println(sb); // "Bonjour!"

System.out.println(s.equals("Bonjour"); // "true"
System.out.println(" " + s.equals("zoo"); // " false"
System.out.println(" " + s.compareTo("zoo"); // " -25"
System.out.println(" " + s.compareTo("Bonjour"); // " 0"
System.out.println(" " + s.compareTo("abraca"); // " 17"
```

Les conteneurs

- **Déf :** classe représentant la version objet d'un type élémentaire
  Ex : `Byte` pour `byte`, `Character` pour `char`
- Convertion implicite entre un type élémentaire et son conteneur
- Les conteneurs possèdent différentes méthodes permettant d'effectuer des traitements divers et variés.
  - Par exemple ils ont tous les méthodes `equals(Object o)` et `toString()`
  - Chaque conteneur possède aussi des méthodes propres :
    - `Integer` : `static Integer valueOf(int i)`, `static Integer valueOf(String s)`, `int intValue()`, `static int parseInt(String s)`, `static String toString(int i)`
    - `Character` : `static boolean isDigit(char c)`, `static boolean isLetter(char c)`, `static boolean isLowerCase(char c)`, `static char toLowerCase(char c)`

Les tableaux I

- Déf : agrégat de composants (objet élémentaire ou non) de même type.
- instanciation et déclaration :

```
int [] ti; // tableau d'entiers
Date [] td; // tableau de dates
ti = new int [5];
td = new Date [5];
int ti2 = { 4, -5, 12};
Date td2 = {null, new Date() };
ti = new int [] {4, -5, 12};
td = new Date [] { null, new Date() };
```

- duplication et test d'égalité

```
import java.util.Arrays;
... // t1 et t2 tableaux de int
t1 = t2; // un seul tableau en memoire
t2 = (int []) t1.clone();
System.out.println(Arrays.equals(t1,t2));
```

Les tableaux II

- tableaux à plusieurs dimmensions

```
double [][] matrice;
matrice = new double[m][n];

double [][] matrice2 = {{1,2,3}, {4,5,6}};

boolean [][][] tabBool;
tabBool = new boolean [][][] {
        {{true, true},{true,false}},
        {{false,true},{false,false}} };

double [][] matrice 3 = new double [4][]:
matrice3[0] = new double[5];
...
matrice3[3] = new double[10];
```

Les énumérations

```
enum Couleur { vert, bleu, gris, rouge, jaune }
Couleur.values()
// ["vert","bleu","gris","rouge","jaune"]
Couleur.jaune.name()
// "jaune"

Couleur c = Couleur.valueOf("jaune")
// c = Couleurs.jaune
c = Couleur.valueOf("noir");
// erreur

System.out.println(Couleur.gris.ordinal());
// 2
```

Le fil rouge

- Objectif : implémentation d'un petit jeu qui consiste en l'utilisation d'un robot ("Azertyuiop") manipulateur de cubes posés sur une table.
- Environement :
  - table de taille infinie
  - ensemble de cubes posés sur la table
  - main pour manipuler les cubes
- Requêtes possibles au robot :
  - raconte : pour décrire l'état du monde
  - aide : aide pour possibilités d'action
  - pose cube sur objet : pour deplacer un cube
  - quitte : pour arrêter le jeu
- Régles à suivre :
  - la table ne peut pas bouger
  - un cube est soit sur un cube soit sur la table
  - on ne peut déplacer que les cubes libre (sans cube dessus)
  - l'ordre donné doit avoir un sens (deplace 2 sur 2)

Les concepts

- Classe : moule à partir duquel on obtient des objets par un mécanisme d'instanciation
- Structure informatique en mémoire crée lors de l'instanciation
- Héritage : possibilité de récuperer les caractéristiques d'une classe pour définir une nouvelle classe

La classe

- `class` introduit la définition d'une classe qui pourra contenir des variables, des constructeurs et des méthodes
- `static` lie la variable ou la méthode à la classe
- `public`, `protected`, → ↪ `private` donne la visibilité de l'élément
- `extends` permet d'hériter d'une classe

```
class Cube{
  /* VARIABLES */
  static int nbPieces = 1;
  protected int numero;
  private Cube sur;
  private Cube porte;

  /* CONSTRUCTEUR */
  Cube {
    ...
  }

  /* METHODES */
  public Cube quiJePorte() {
    ...
  }
  public Cube quiMePorte() {
    ...
  }
  public int quiJeSuis() {
    ...
  }
  public void vaEn(Cube dest) {
    ...
  }
  ...
}
```

Les variables et les méthodes

```
public int quiJeSuis() {
  System.out.println("Je suis le cube " + numero + ", ");
  if (this.sur == null)
    System.out.println("Je suis sur la table");
  else System.out.println("Je suis le cube " + this.sur. →
      ↪ numero);
  System.out.println((this.porte == null?".":",");
  if (this.porte != null)
    System.out.println(" et je porte le cube " + this.sur →
        ↪ .numero + ".");
  return numero;
}
// lecture
public Cube quiJePorte() {
  return porte;
}
// ecriture
public Cube quiJePorte(Cube porte) {
  return this.porte = porte;
}
```

Le constructeur d'instance

Un constructeur :

- est invoqué par `new`
  Ex : `Cube c = new Cube()`
- doit avoir le même nom que la classe
- n'as pas de type de retour
- peut être surchargé
- n'est pas hérité par les sous-classes

```
Cube() {
  porte = null;
  this.numero = Cube.nbPiece++;
}
```

Les paquetages

- Moyen de structuration du code
- Unité de compilation : le fichier
- `package` défini l'appartenance à un paquet
- `import` défini la dépendance à d'autres fichiers

```
package Azertyuiop;
import java.util.List;
import staticjava.lang.Math.*;
...
double uncercle = PI * rayon;
```

L'accessibilité des éléments

- Permet de limiter l'accessibilité de certaines méthodes, variables
- Le `main` doit être `public` et `static`

Qualificateur	Sous-classe	Paquetage	Monde
`public`	accessible	accessible	accessible
`protected`	accessible	accessible	
`private`			