Un petit air de Java
Ca fait du bien

Jean-Jacques.Levy@inria.fr

http://www.jeanjacques-levy.com/

INRIA – Rocquencourt

tel: 01 39 63 56 89

Catherine Bensoussan

cb@lix.polytechnique.fr

Aile 00, LIX

tel: 34 67

http://w3.edu.polytechnique.fr/informatique/

Plan

1. Premier programme
2. Types primitifs
3. Conversions de type
4. Instruction d'affectation
5. Instruction composée
6. Instruction conditionnelle
7. Calcul de la date de Pâques

Premier programme

Programme qui affiche un texte à l'écran:

```
import java.io.*;

class PremierProg {
    public static void main (String args[ ]){
        System.out.println ("Bonjour");
        System.out.println ("les débutants!");
    }
}
```

Compilation, Exécution

A mettre dans un fichier PremierProg.java, ensuite compiler

```
javac PremierProg.java
```

et exécuter

```
java PremierProg
```

En fait le code généré par `javac` **est du** *byte-code*; **il est interprété par la machine virtuelle Java (JVM) par** `java`.

Le *byte-code* **est généré dans le fichier PremierProg.class**

Variables

Une variable a un nom, un type et une valeur.

- Le nom est une séquence de lettres ou chiffres commençant par une lettre.
- Le type décrit l'ensemble de valeurs que peut prendre la variable.
- On doit toujours déclarer une variable.
- Le type d'une variable est donné à sa déclaration.

```
int x1;
String phrase;
float pi;
```

Types primitifs

1. **Octets (8 bits)** `byte`
2. **Entiers courts (16 bits)** `short`
3. **Entiers (32 bits)** `int`
4. **Entiers longs (64 bits)** `long`
5. **Réels (32 bits)** `float`
6. **Réels longs (64 bits)** `double`
7. **Caractères (16 bits)** `char`
8. **Booléens** `boolean`

Valeur d'une variable

- **Une variable doit toujours être initialisée**

```
int x1 = -155;
String phrase = "Bonjour";
float pi = 3.14159;
```

- **Une variable peut être affectée, c'est à dire que sa valeur est modifiée.**
- **Une variable peut être évaluée, le résultat de l'évaluation est alors la dernière valeur qui lui a été affectée.**
- **Une variable peut faire partie d'une expression**

```
float r = 4.5;
float surface = 4 * pi * r * r ;
```

Déclarations – types – valeurs

- Toute variable a un type qu'il faut déclarer.
- Toute variable a une valeur par défaut.
- On peut initialiser cette valeur à la déclaration.

```
long u, v;
int x = 1, y = 3, z = -9;
boolean a = true, b = false;
float r = 4.5;
double surface = 4 * pi * r * r ;
```

Conversions implicites de types

- **Conversions implicites:**

  `byte` → `short` → `int` → `long` → `float` → `double`

  On a aussi:

  `char` → `int`

Exemples:

```
short i = 23;
int j = i;
long u = j;
float v = u;
```

Instruction d'affectation

- `u =` e`;`
- `u` **est une variable,** e **est une expression**
- **Les deux ont même type**
- **L'expression** e **est évaluée**
- **La valeur de** `u` **devient la valeur de l'expression** e**.**

Exemples:

```
s = 4 * pi * r * r ;
s = s / 2;
i = i + 1;
```

Conversions explicites

- $(t)\ e$
- t est un type, e est une expression
- la valeur de e est convertie dans le type t
- une telle conversion peut faire perdre de la précision
- dans le cas où e est réel et t est entier, on prend la partie entière.

Exemples:

```
long i = 23;
int j = (int) i;
short u = (short) j;

float pi = 3.14159;
int x = (int) pi;
```

Expressions arithmétiques

Calculer la différence des surfaces d'un cercle de rayon 2.75 et d'un carré de coté 5:

```
int a = 5;
float r = 2.75;
float pi = 3.14159;
float delta = pi * r * r - a * a;
System.out.println (delta);
```

Priorité des opérations!

Calculer le reste de la division de 3125 par 713:

```
int a = 3125 % 713;
System.out.println (a);
```

Java

- Fortement typé. Toute expression a un type.
- Orienté objet.
- Tout programme contient au moins une classe, contenant une fonction `main`
- Pour afficher sur l'écran

```
System.out.println ("Coucou!!");
```

Les types permettent d'éviter des erreurs uniquement détectables à l'exécution.

Les avantages ou désavantages de la programmation orientée-objet sont impossibles à expliquer dans ce cours. Grosso modo: orienté-objet $\rightarrow$ modularité $+$ développement incrémental des programmes.

Instruction composée – Instructions conditionnelles

- { *inst*$_1$ *inst*$_2$ ... *inst*$_n$ } **où** *inst*$_1$**,** *inst*$_2$**, ...** *inst*$_n$ **sont des instructions quelconques (**$n \geq 0$**).**
- `if` (e) *instruction*
- `if` (e) *instruction*$_1$ `else` *instruction*$_2$

Exemple:

```
if (x > 110)
  x = x - 10;
else {
  x = x + 11;
  z = x * z;
}
```

Expressions booléennes

- valeurs `true` ou `false`
- Opérations de comparaison

```
x > 8
x == 7
x != 9
```

- Expressions booléennes composites

```
(a > 8) && (b < 16)
!(a == 9)
(c != 0) && (b/c == 1)
(b/c == 1) || (a = 8)
```

Remarques:

- = pour l'affectation, == pour la comparaison
- les expressions booléennes sont évaluées de gauche à droite

Un programme complet

```
class Additionneur {
  public static void main (String[ ] args) {
    if (args.length != 2)
      System.out.println ("Mauvais nombre d'arguments.");
    else {
      int n1 = Integer.parseInt (args[0]);
      int n2 = Integer.parseInt (args[1]);
      System.out.println (n1 + n2);
    }
  }
}
```

Terminaison des programmes

- **Jusqu'à présent, on n'a vu que des programmes sans boucles (*straight line programs*)**

  Théorème 1 *Les programmes sans boucles terminent toujours.*

- **Si on a des instructions d'itérations (cf prochain cours), il faut plus réfléchir. Par exemple, un programme itératif simple peut calculer la suite:**

$$u_{n+1} = \begin{cases} 3u_n + 1 & \text{si } u_n \text{ impair} \\ u_n/2 & \text{si } u_n \text{ pair} \end{cases}$$

  à partir d'un u_0 quelconque, et personne ne sait si on termine toujours par 1.

Exemple:

- **4, 2, 1**
- **5, 16, 8, 4, 2, 1**
- **7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1**

Un autre programme complet

```
class Syracuse {
  public static void main (String[ ] args) {
    int a = Integer.parseInt (args[0]);
    System.out.println (a);
    while (a != 1) {
      if (a % 2 == 0)
        a = a / 2;
      else
        a = 3 * a + 1;
      System.out.println (a);
    }
  }
}
```

Le calcul de la date de Pâques

Premier dimanche après la 1ère lune qui suit l'équinoxe de printemps. Soit Y l'année, dont on cherche la date de Paques.

1. **Golden number** $G = (Y \bmod 19) + 1$

2. **Century** $C = \lfloor Y/100 \rfloor + 1$

3. **Corrections** $X = \lfloor 3C/4 \rfloor - 12$, $Z = \lfloor (8C+5)/25 \rfloor - 5$

4. **Find Sunday** $D = \lfloor 5Y/4 \rfloor - X - 10$

5. **Epact** $E = (11G + 20 + Z - X) \bmod 30$.
Si $E = 25$ **et** $G > 11$**, ou si** $E = 24$**, alors** $E \leftarrow E + 1$

6. **Find full moon** $N = 44 - E$. **Si** $N < 21$**, alors** $N \leftarrow N + 30$

7. **Advance to Sunday** $N \leftarrow N + 7 - ((D + N) \bmod 7)$

8. **Get month Si** $N > 31$**, la date est le** $(N - 31)$ **AVRIL**
Sinon, la date est le N **MARS.**