Informatique 1
L1 Portail IE
L1 Portail MA

Responsables:

pierre-alain.fouque@univ-rennes1.fr

Patrick.derbez@univ-rennes1.fr

Variables

Une variable est représentée par un identificateur, elle a un type (c'est-à-dire qu'on doit savoir dans quel ensemble elle prend ses valeurs).

On peut utiliser sa valeur, lui en affecter une (nouvelle).

Une variable doit toujours être déclarée :

```
int x;
double u;
```

Affectation

```
u = e;
```

- `u` est une variable et `e` une expression ;
- les deux ont même type ;
- l'expression `e` est évaluée, puis la variable `u` prend pour valeur le résultat de cette évaluation : on a affecté la valeur de `e` à `u`.

```
x = -155;
z = x + 3;
```

Une variable doit toujours être initialisée.

Affectation

On peut condenser déclaration et initialisation :

```
int x = -155;
```

Une instruction idiomatique : avec le mécanisme d'affection, les instructions :

```
int i = 3;
i = i + 1;
```

sont valides, car on calcule d'abord **`i+1`** et le résultat 4 est mis dans **`i`** à la place de 3.

Instructions compactes

`i = i+1;` ou `i += 1;` (idem pour `-`, `*`, `/`).
`i++;` ou `++i;` (idem avec `-`).

Post-incrémentation

`n = 5;`	
`m = n++;`	`m = n;`
	`n += 1;`

Pré-incrémentation

`n = 5;`	
`m = ++n;`	`n += 1;`
	`m = n;`

Expressions logiques et arithmétiques

- Priorité des opérateurs

Niveau	Opérateur	Description	Exemple
1	++	incrémentation postfixe	x++
	--	décrémentation postfixe	x--
2	++	incrémentation préfixe	++x
	--	décrémentation préfixe	--x
	-	changement de signe	-x
	+	signe +	+x
	!	NON logique	! x
4	*	multiplication	x * y
	/	division	x / y
	%	modulo	x % y
5	+	addition	x + y
	-	soustraction	x - y
7	<	plus petit que	x < y
	<=	plus petit ou égal à	x <= y
	>	plus grand que	x > y
	>=	plus grand ou égal à	x >= y
8	==	égal	x == y
	!=	différent	x != y
10	^	OU logique exclusif	x ^ y
12	&&	ET logique	x && y
13	||	OU logique inclusif	x || y
14	? :	opérateur conditionnel	x ? y : z
15	=	affectation	x = y
	+=, -=, *=, /=, %=	affectations composées	x += y

Figure 15. Priorité de quelques opérateurs Java.

Evaluation d'expressions de gauche à droite:

Int x = 1 - 2 - 3; // donc x= (1-2)-3

(sauf pour les opérateurs 2,3, 14 et 15)

Int x = y = 2 + 3; // donc x= (y = (2+3))

Exemples d'expressions

- Int x = 1 + 2 * 3 / 6; // x=?
- Int x = 3 * 5 - 2 * 1 - 12 / 2 * 3; // x=?
- b=-9 >= 6 && true || 7*3 < 2; // b=?

- b=2 * 7 + 12 >= 17 - 1 / 2.0 || !true; // b=?

(parenthésez les expressions pour vous aider)

Expressions logiques

Règles d'évaluation: soient `C1` et `C2` deux expressions booléennes, dans l'instruction

```
boolean b = (C1 || C2);
```

Java évalue `C1`; si `C1` est vraie, `C2` n'est pas évaluée et `b` vaut `true`.

De même, dans:

```
boolean b = (C1 && C2);
```

si `C1` est faux, `C2` n'est pas évaluée, et `b` vaut `false`.

Perte de l'associativité

```
double x=1.0E35, y=1.0E35, z=1.0, f1, f2;
f1 = x - (y + z);
f2 = (x - y) - z;
System.out.print("x-(y+z)=");
System.out.print(f1);
System.out.print(" (x-y)-z=");
System.out.println(f2);
```

```
x-(y+z)=0.0 (x-y)-z=-1.0
```

Perte de précision

```
public static void main(String[] args){
   double a = 0.3, b = 2.1, c = 3.675, d;
   d = b*b-4*a*c;
   System.out.print("d=");
   System.out.println(d);
   return;
}
```

```
d=8.881784197001252E-16
```

et pourtant, le discriminant vaut mathématiquement 0.

Conclusion sur les flottants

- Marche quand même souvent, mais avec prudence (Ariane 5).
- Autres méthodes: précision arbitraire, arithmétique d'intervalles, etc.

Lire des données

- Méthode 1: données passées en argument du programme
- Méthode 2: données demandées à l'utilisateur
- Méthode 3: données stockées dans un fichier

Méthode 1

```
1
2  public class parse_ex {
3      public static void main(String[] args){
4          int nombre = Integer.parseInt(args[0]);
5          System.out.println("le nombre passé en argument est " + nombre);
6      }
7  }
8
```

java parse_ex 20

> le nombre passé en argument est 20

java parse_ex "pas un nombre"

> Exception in thread "main" ...

parseInt

```
public class Test {

   public static void main(String args[]) {
      int x =Integer.parseInt("9");
      double c = Double.parseDouble("5");
      int b = Integer.parseInt("444",16);

      System.out.println(x);
      System.out.println(c);
      System.out.println(b);
   }
}
```

```
9

5.0

1092
```

Méthode 2

```java
import java.util.Scanner; // on importe la bibliothèque
public class parse_ex {
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in); // on ouvre un scanner
        System.out.print("Entrez un nombre: ");
        int nombre = sc.nextInt(); // on récupère l'entier
        System.out.println("Vous avez entré le nombre: " + nombre);
        sc.close(); // on ferme le scanner
    }
}
```

```
Entrez un nombre: 3
Vous avez entré le nombre: 3
```

Instructions conditionnelles

But: faire prendre des décisions simples par l'ordinateur.

Syntaxe:

```
if(E)
   I
else
   J
```

`I` et `J` sont des blocs d'instructions, `E` une expression logique.

`J` peut ne pas être présente, on écrit alors simplement:

```
if(E)
   I
```

Exemple

```
import java.util.Scanner;
public class Ifelse {
    public static void main(String[] args){
        int devinette = 3; // nombre à trouver
        Scanner sc = new Scanner(System.in);
        System.out.print("Entrez un nombre: ");
        int nombre = sc.nextInt(); // on récupère l'entier
        if (nombre == devinette) { // on teste l'égalité
            System.out.println("Bravo!"); // si oui on affiche "Bravo!"
        }
        else {
            System.out.println("Perdu!"); // sinon on affiche "Perdu!"
        }
        sc.close();
    }
}
```

Exemple

```
import java.util.Scanner;
public class Ifelse {
    public static void main(String[] args){
        int devinette = 3;
        Scanner sc = new Scanner(System.in);
        System.out.print("Entrez un nombre: ");
        int nombre = sc.nextInt();
        // une seule instruction --> pas besoin d'accolades
        if (nombre == devinette) System.out.println("Bravo!");
        else System.out.println("Perdu!");
        sc.close();
    }
}
```

Exemple 2

```java
import java.util.Scanner;
public class Ifelse {
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        System.out.print("Entrez un chiffre: ");
        int chiffre = sc.nextInt();
        if (chiffre < 0 || chiffre > 9) // deux tests
            System.out.println("Un chiffre doit être compris entre 0 et 9 inclus.");
        sc.close();
    }
}
```

& | &&?

```
boolean a, b;

Operation     Meaning                          Note
---------     -------                          ----
   a && b     logical AND                       short-circuiting
   a || b     logical OR                        short-circuiting
   a &  b     boolean logical AND               not short-circuiting
   a |  b     boolean logical OR                not short-circuiting
   a ^  b     boolean logical exclusive OR
  !a          logical NOT

short-circuiting        (x != 0) && (1/x > 1)   SAFE
not short-circuiting    (x != 0) &  (1/x > 1)   NOT SAFE
```

If-Else imbriqués

```
if (A > B) {
    if (A > 10) System.out.println("premier choix");
    else if (B < 10) System.out.println("deuxième choix");
}
else {
    if (A == B) System.out.println("troisième choix");
    else System.out.println("quatrième choix");
}
```

Pour quelles valeurs de A et B ce programme affiche *premier choix*, *deuxième choix*, … ?

If-Else imbriqués

```
if (A > B) {
    if (A > 10) System.out.println("premier choix");
    else if (B < 10) System.out.println("deuxième choix");
}
else {
    if (A == B) System.out.println("troisième choix");
    else System.out.println("quatrième choix");
}
```

Existe-t-il des valeurs de A et B pour lesquelles le programme n'affiche rien?

Itérations

But: écrire de manière générique des calculs répétitifs.

Exemple: calculer les chiffres binaires d'un entier $\geqslant 0$:

$$n = \sum_{i=0}^{p} b_i 2^i, p \geqslant 0, \quad b_i \in \{0,1\}.$$

Ex. $13 = ??$
À la main: 13 est impair $\rightarrow b_0 = 1$; $(13-1)/2 = 6$ est pair $\rightarrow b_1 = 0$; $6/2 = 3$ est impair $\rightarrow b_2 = 1$; $(3-1)/2 = 1$ est impair $\rightarrow b_3 = 1$, et on s'arrête.

$$13 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2 + 1.$$

Boucle while

Principe du calcul: on regarde la parité du nombre n et on en déduit le chiffre binaire correspondant; on recommence avec $n/2$, jusqu'à ce que n soit nul.

Reformulation: tant que n n'est pas nul, on calcule la parité de n et on en déduit le chiffre binaire; on divise n par 2.

On utilise

```
while(E)
   I
```

qui, tant que `E` est vraie, répète l'instruction `I`.

Décomposition binaire

```
while (n != 0) {
    System.out.println(n%2);
    n /= 2;
}
```

Pour $n = 13$:

1
0
1
1

Dans le processeur

```
→  012  si n = 0 aller à la ligne 16
   013  afficher n mod 2
   014  n = n/2
   015  aller à la ligne 12
   016  suite des instructions
```

```
   012  si n = 0 aller à la ligne 16
   013  afficher n mod 2
   014  n = n/2
→  015  aller à la ligne 12
   016  suite des instructions
```