UEF 1 : Informatique & Programmation

Faculté des Sciences de Nice

DEUG 2000-2001

Jérôme DURAND-LOSE

Sandrine JULIA

Jean-Paul ROY

COURS 5

La logique et les tests

Les valeurs *booléennes* `true` et `false`

- Ne pas confondre *affectation* et *test* !

```
int x = 2, y = 3, z = 6;

z = z + 1;      /* z devient 7 */

y == x + 1     /* est-ce que y vaut x+1 ? */
```

→ **true**

```
x == y + 1     /* est-ce que x vaut y+1 ? */
```

→ **false**

- Le type *boolean* comporte donc deux constantes :
  true [le « vrai »] et **false** [le « faux »]

```
int x = 5;
boolean b1 = (x == 2 + 3);
boolean b2 = (x != 2 + 3);
boolean b3 = (x <= Math.PI);
System.out.println("b1 vaut : " + b1);
System.out.println("b2 vaut : " + b2);
System.out.println("b3 vaut : " + b3);
```

converti en String

```
b1 vaut : true
b2 vaut : false
b3 vaut : false
```

Les expressions booléennes

♦ *Poser une question* en Java revient donc à *évaluer une expression booléenne*, comme :

```
x < y + 5
somme >= 0
omega + 1 > phi + Math.sin(theta/r)
n % 2 == 0
```

l'entier n est-il pair ?

Attention aux calculs approchés !!!

♦ Méfiez-vous des calculs approchés avec les nombres flottants [le type *double*]. Ils sont bien **APPROCHES** :

```
double x = 0.1;
boolean b = (x+x+x+x+x+x+x+x == 0.8);
System.out.println("b vaut : " + b);
```

```
b vaut : false
```

0.1 est un nombre compliqué : écrivez-le donc en binaire...

L'instruction conditionnelle if

♦ Elle réalise un « aiguillage » suivant le résultat d'un test représenté par une expression booléenne :

SI il fait beau *ALORS* je vais à la plage

SI il fait beau *ALORS* je vais à la plage
SINON je vais au cinéma

♦ SI *test* ALORS *instruction*

```
if (test) instruction;
```

```
if (r * r < 1)
  compteur = compteur + 1;
```

♦ SI *test* ALORS $instr_1$ *puis* $instr_2$ *puis … puis* $instr_k$

```
if (test) {instr1; …… ;instrk;}
```

```
if (r * r < 1)
  { compteur = compteur + 1;
    System.out.println("Râté !");
  }
```

♦ SI *test* ALORS $instr_1$ *SINON* $instr_2$

```
if (test) instr1; else instr2;
```

```
if (r * r < 1)
  compteur = compteur+1;
else
  System.out.println("Râté !");
```

♦ Avec les mêmes conventions d'accolades que précedemment s'il y a plusieurs instructions dans l'une ou l'autre des parties...

Exemple : une fonction « valeur absolue »

♦ Supposons que **Math.abs**(...) n'existe pas ! Nous la rajouterions dans notre classe Numerik (cf TP) :

```
class Numerik
{  . . .
   static double valAbs (double x)
   {  if (x < 0)
         return -x;
      else
         return x;
   }
}
```

if (Numerik.valAbs(x) > 5) . . . .

La *négation* logique

NON

♦ L'opérateur de *négation* est une fonction notée ! :

! : *boolean --> boolean*

P	!P
true	*false*
false	*true*

♦ Exemples :

	équivaut à :
`if (! (x+y > 5)) ...`	`if (x+y <= 5) ...`
`if (! (x == 5)) ...`	`if (x != 5) ...`

La *conjonction* logique &&

ET

♦ P && Q sera *vraie* si et seulement si P et Q sont *vraies* toutes les deux.

Court-circuit !

P \ Q	*true*	*false*
true	*true*	*false*
false	*false*	*false*

table de vérité P && Q

♦ Attention, si P est faux, *Q ne sera pas évalué* puisque le résultat sera faux quel que soit Q !...

```
if ((x >= 0) && (Math.sqrt(x) < 1)) ...
```

La *disjonction* logique ||

OU

♦ P || Q sera *fausse* si et seulement si P et Q sont *fausses* toutes les deux.

P \ Q	*true*	*false*
true	*true*	*true*
false	*true*	*false*

table de vérité P || Q

Court-circuit !

♦ Attention, si P est vrai, *Q ne sera pas évalué* puisque le résultat sera vrai quel que soit Q !...

```
if ((x < -1) || (x > 1)) ...
```


Sur la notion d'égalité...

♦ L'opérateur d'égalité pour les types primitifs est **==** [deux signes **=** accolés] :

```
int x = 1, y = x + 2;
if (x == y - 2) Console.println("Ok");
```

Ok

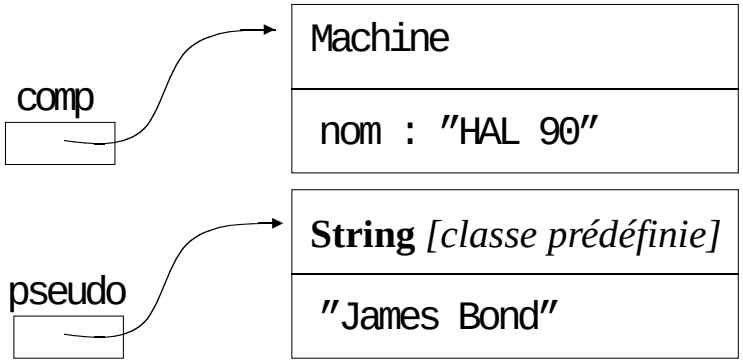
♦ Encore une fois : **éviter de questionner l'égalité de nombres approchés !** Demander plutôt si leur différence est inférieure à une précision donnée...

```
double x = ..., y = ...;
if (x == y)                    // mal
if (Math.abs(x-y) < 1e-6)      // bien
```


Et pour les *objets* ?...

♦ Rappel : on accède à un objet par une *référence* [en quelque sorte l'adresse de l'objet en mémoire].

```
String pseudo = new String("James Bond");
Machine comp = new Machine("HAL 90");
```



♦ L'opérateur == entre *objets* teste **l'égalité des références aux objets**, donc leur *identité* en mémoire et non seulement leurs contenus :

```
Machine pc = new Machine("HAL 90");
if (pc == comp)
   System.out.println("oups");
else
   System.out.println("distincts");
```

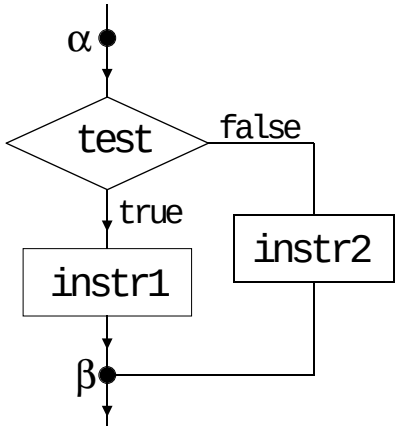
deux machines de même nom mais distinctes !

distincts

if (pc != comp) ...

Schématisation avec un *ordinogramme* *organigramme*

```
α
if (test)
  instr1;
else
  instr2;
β
```




```
class Test
{   public static void main(String[] args)
    {   int x = 2;
        System.out.println("alpha");
        if (x == 2)
           System.out.println("Oui");
        else
           System.out.println("Non");
        System.out.println("beta");
    }
}
```



```
2:getstatic       #10    // System.out
5:ldc1            #3     // "alpha"
7:invokevirtual   #11    // println
-------------------------------------
10:iload_1               // charger x
11:iconst_2              // 2
12:icmpne         26     // si ≠, aller en 26
-------------------------------------
15:getstatic      #10    // System.out
18:ldc1           #2     // "Oui"
20:invokevirtual  #11    // println
23:goto           34     // aller en 34
-------------------------------------
26:getstatic      #10    // System.out
29:ldc1           #1     // "Non"
31:invokevirtual  #11    // println
-------------------------------------
34:getstatic      #10    // System.out
37:ldc1           #4     // "beta"
39:invokevirtual  #11    // println
-------------------------------------
42:return
```