Université de Nice – Sophia Antipolis — Deug MASS MIAS SM 1re année
UEF1 : Informatique & Programmation — 2000–2001

TP N° 5

Expressions booléennes – L'instruction `if` – Comparaisons

Ça va mieux en le révisant . . . Attention en JAVA, il ne faut pas confondre le signe d'affectation `=` et le signe d'égalité `==` :

```
n = 7;  //   affectation : n reçoit 7
if (n == 7)
    System.out.println("n égal 7.");
else
    System.out.println("n est différent de 7.");
```

Une variable de type `boolean` peut prendre seulement deux valeurs : `true` ou `false`. Les expressions booléennes peuvent contenir des opérateurs relationnels (<,>,==,!=,. . .) et des opérateurs logiques `&&`, `||` et `!`. Ces derniers représentent respectivement la conjonction (*et logique*), la disjonction (*ou logique*) et la négation (*non logique*) .

L'instruction `if` permet de faire exécuter à l'ordinateur une suite d'instructions ou bien une autre selon que le résultat d'un test est vrai ou faux.

Les opérateurs `==` ou `!=` testent l'égalité (ou la non-égalité) de deux données primitives de type comparables. Une variable déclarée du type d'un objet reçoit l'adresse d'un ensemble de cases mémoire qui contiennent les données privées de l'objet. On parle de *référence*. Du coup, ces mêmes symboles `==` ou `!=` utilisés pour comparer deux objets se contentent de comparer leurs références.

1 Un peu de logique

Dans l'ordre, du plus prioritaire au moins prioritaire, les opérateurs *logiques* ou *booléens*[1] sont :

`!  &&  ||`

Notez bien que les opérateurs logiques `&&` et `||` n'évaluent leur deuxième opérande que si elle peut influer sur le résultat final de l'expression.

Exercice 1) Complétez la table de vérité suivante en trouvant les valeurs des expressions booléennes pour toutes les combinaisons possibles des variables `p`, `q` et `r` :

p	q	r	p && q || !r
false	false	false	...
false	false	true	...
false	true	false	...
false	true	true	...
true	false	false	...
true	false	true	...
true	true	false	...
true	true	true	...

◇

1. Georges Boole (1815-1864), mathématicien.

Exercice 2) Les lois De Morgan[2] décrivent des identités booléennes.

`!(A && B)` identique à `!A || !B`
`!(A || B)` identique à `!A && !B`

Utilisez-les afin simplifier les expressions booléennes suivantes :

```
!(x > 0 && y >0)
!(x != 0 || y !=0)
```
◇

L'écriture des conditions sur les nombres reposent sur l'usage de six *opérateurs relationnels* ou *de comparaison* :

JAVA	Math.	Nom
`>`	$>$	Supérieur
`>=`	$\geq$	Supérieur ou égal
`<`	$<$	Inférieur
`<=`	$\leq$	Inférieur ou égal
`==`	$=$	Egal
`!=`	$\neq$	Différent

2 L'instruction `if`

Exercice 3) Trouvez un couple de nombres approchés `x` et `y` pour lesquels les deux ensembles d'instructions suivants produisent le même résultat. Idem pour un résultat différent.

```
s=0;                          |  s=0;
if (x > 0) s = s+1 ;          |  if (x > 0) s = s+1 ;
if (y > 0) s = s+1 ;          |  else if (y > 0) s = s+1 ;
```
◇

Conseil : quand, dans une méthode qui retourne un résultat (i.e. une fonction), le calcul du résultat dépend d'une ou plusieurs conditions, surtout, n'oubliez pas de faire figurer le mot-clé `return` à la fin de chacun des cas. Il ne doit pas y avoir de cas où la fonction n'aurait pas de résultat, c'est-à-dire où elle ne serait pas définie, sous peine d'engendrer une erreur à la compilation.

Exercice 4) Ajoutez à votre classe `Numerik` une méthode statique `max3`(...) qui retourne le maximum de trois nombres approchés, et une autre `min3`(...) qui retourne leur minimum. On vous demande de n'utiliser aucune autre méthode, que ce soit de la classe `Math` ou du reste de la classe `Numerik`. Toujours dans votre classe `Numerik`, ajoutez une méthode `compareApprochés`(...) qui aura 3 arguments de type `double` : les deux nombres approchés x et y à comparer et le seuil ε. Cette méthode statique retournera un résultat *booléen* dont le type de retour est : `boolean`. En fait, les nombres approchés seront considérés égaux si l'inégalité suivante est vérifiée :

$$| \texttt{x - y} | \leq \varepsilon$$

Utilisez ces trois nouvelles méthodes statiques dans un petit programme-test. ◇

Exercice 5) Ecrivez un programme qui lit trois coefficients réels a, b et c. Il devra afficher le ou les

2. De Morgan (1806-1871), logicien.

racines réelles de l'équation

$$ax^2 + bx + c$$

si elle(s) existe(nt) ou sinon un message d'explication. N'oubliez aucun cas particulier ($a = 0$, discriminant nul, négatif ...). ◇

3 Comparaison des objets

Nous avons vu que l'opérateur `==` permet de comparer si les références de deux objets sont identiques ; il teste si deux variables de type `Employe` par exemple désignent exactement le même objet c'est-à-dire si elles référencent le même emplacement mémoire. Le test d'égalité ne porte pas simplement sur le contenu d'un objet, comme l'exercice suivant en témoigne :

Exercice 6) Ecrivez un programme qui crée deux employés `lePremier` et `leSecond` ayant exactement le même nom mais dont les salaires diffèrent. Effectuez le test d'égalité puis faites afficher les résultats.
Affectez au salaire du premier celui du second et recommencez l'exercice. Que se passe-t-il? ◇

Cas des chaînes de caractères

La semaine dernière, vous avez appris qu'il y a deux façons bien distinctes de créer une chaîne de caractères :

```
String ch1 = "Bonjour" ;
String ch2 = new String("Aurevoir") ;
```

Par souci de simplicité, nous vous demandons de créer vos chaînes de caractères sans utiliser le constructeur de la classe `String`, c'est-à-dire de la première façon employée ci-dessus pour la chaîne `"Bonjour"`.

Pour comparer deux chaÎnes de caractères, nous utiliserons[3] la méthode d'instance

`equals`(...)

Elle teste si la chaîne de caractères à laquelle on l'applique est semblable à celle passée en argument, au sens que les deux chaînes ont les mêmes lettres (qu'elles soient deux objets distincts ou non). Rien de tel qu'un petit exemple pour comprendre :

Exercice 7) Tapez un programme qui englobe les instructions suivantes, de façon à vérifier la valeur de `b1`, `b2` et `b3` :

```
String ch1 = "Bonjour", ch2 = "Aurevoir", ch3 = "Bonjour" ;
String ch4 = ch1 ;
boolean b1 = ch1.equals(ch2) ;
boolean b2 = ch1.equals(ch3) ;
boolean b3 = ch1.equals(ch4) ;
```
◇

3. Dans le cas où les chaînes de caractères sont construites sans appel au constructeur, et seulement dans ce cas-là, JAVA autorise la comparaison des chaînes de caractères avec le seul opérateur `==`. Nous vous demandons de ne pas utiliser ce moyen de comparaison. En pratique, on n'a pas forcément à l'esprit la manière dont telle ou telle chaîne a été créée et du coup, l'usage de ce mode de comparaison peut s'avérer scabreux.

4 Approfondissement

Exercice 8) Dans la formule de fin de repas "*Fromage ou dessert ?*", avez-vous l'impression que la signification du *ou* est la même que celle du *ou* vu en cours ? Ecrivez la table de vérité de ce deuxième *ou*. Il porte le nom de *ou exclusif* et est celui de la langue courante (il signifie *ou bien*).
Donnez ensuite une formule booléenne qui permet de l'écrire en fonction des opérateurs usuels. ◇

Exercice 9) Munissez votre classe `Point` d'une méthode d'instance `identiqueA`(...) qui teste si le point auquel elle s'applique a les mêmes coordonnées que celui passé en paramètre.
Demandez à l'ordinateur de choisir au hasard un point aux coordonnées entières dans le cadran délimité par les points $\binom{1}{1}$ et $\binom{5}{5}$. Ensuite, vous donnerez 2 (ou 3 même) essais à l'utilisateur pour qu'il devine le point en question. Voici un exemple d'exécution :
Les réponses de l'utilisateur sont écrites en gras.

```
Premier essai :
devinez un point de coordonnées entières entre 1 et 5.
Entrez une abcisse :
1
Entrez une ordonnée :
4
Perdu ! Vous avez un deuxième essai :
Entrez une abcisse :
5
Entrez une ordonnée :
2
Encore perdu ! Le point à trouver était le point <2,4>.
Au revoir !
```
◇