Mathématiques 1
Expressions booléennes en Java

Institut Paul Lambin

1er octobre 2021

Nombres machines

Univers des nombres machines

Pour la suite nous allons nous placer dans l'univers U suivant

$$\begin{aligned} U &= \text{espace des nombres } \textbf{représentable par une machine} \\ &= \text{espace des nombres } \textbf{représentable sur "}\boldsymbol{x}\textbf{" bits} \end{aligned}$$

Nombres entiers en Java

En Java

- les nombres entiers sont des variables de type "`int` "
- les nombres entiers sont codés sur 32 bits dont 1 bits de signe

Donc

- le plus grand entier représentable est

$$\texttt{Integer.MAX\_VALUE} = 2^{31} - 1 = 2147483647$$

- le plus petit entier représentable est

$$\texttt{Integer.MIN\_VALUE} = -2^{31} = -2147483648$$

Conclusion : L'ensemble des entiers représentable en Java sont les entiers de l'intervalle $[-2147483648, 2147483647]$

Nombres entiers en Java

Remarques :

1) Si on ajoute 1 à `Integer.MAX_VALUE`, en faisant le calcul en binaire, on obtient

$$
\begin{aligned}
\texttt{Integer.MAX\_VALUE}_{10} + 1_{10} &= 01111111111111111111111111111111_2 + 1_2 \\
&= 10000000000000000000000000000000_2 \\
&= -2^{31}_{10} \\
&= \texttt{Integer.MIN\_VALUE}
\end{aligned}
$$

2) De même on a $\texttt{Integer.MIN\_VALUE} - 1 = \texttt{Integer.MAX\_VALUE}$
$\rightarrow$ les nombres entiers représentable en Java sont cycliques !

3) Il existe d'autres types en Java pour représenter les entiers :
   - `long` : entiers de l'intervalle $[-2^{63}, 2^{63} - 1]$
   - `short` : entiers de l'intervalle $[-2^{15}, 2^{15} - 1]$
   - `byte` : entiers de l'intervalle $[-2^{7}, 2^{7} - 1]$

Nombres réels en Java

Représentation en virgule flottante

- Sur machine, la précision est limitée $\rightarrow$ les seuls réels représentables sont les rationnels.
- Sur machine, les nombres décimaux sont codés en représentation en virgule flottante :

$$\boxed{s \cdot m \cdot b^e}$$

où

- s est le signe
- m est la mantisse
- b est la base
- e est l'exposant

Nombres réels en Java

Représentation en virgule flottante

Exemples :

1. Le nombre -423.54 se représente $-4.2354 \cdot 10^2 \rightarrow \begin{cases} \text{signe} = -1 \\ \text{mantisse} = 4.2354 \\ \text{base} = 10 \\ \text{exposant} = 2 \end{cases}$
2. Le nombre 0.00054 se représente $5.4 \cdot 10^{-4}$
3. Le nombre 15 se représente $1.5 \cdot 10^1$
4. Le nombre -7 se représente $-7 \cdot 10^0$

Types de nombres réels en Java

Il y a deux types de réels en Java : `float` et `double`

1) Le type `float` :

Les réels de types `float`, appelés *nombres décimaux simple précision*, sont codés sur 32 bits :

$\rightarrow$ 23 bits pour la mantisse. 8 bits pour l'exposant. 1 bit pour le signe.

Ceci implique que

- le plus grand décimal simple précision est
$$\texttt{Float.MAX\_VALUE} = \left(2-2^{-23}\right)\cdot 2^{127} \approx 3.4028235\cdot 10^{38}$$
- le plus petit décimal simple précision est
$$-\texttt{Float.MAX\_VALUE} = -\left(2-2^{-23}\right)\cdot 2^{127}$$
- La distance minimale entre deux décimaux simple précision est
$$\texttt{Float.MIN\_VALUE} = 2^{-149} \approx 1.4\cdot 10^{-45}$$

$\rightarrow$ Le premier décimal simple précision non nul est 2^{-149}

$\rightarrow$ Il n'y a pas de réel représentable en simple précision entre 0 et 2^{-149}

Types de nombres réels en Java

Il y a deux types de réels en Java : `float` et `double`

2) Le type `double` :

Les réels de types `double`, appélés nombres décimaux double précision, sont codés sur 64 bits

$\rightarrow$ 52 bits pour la mantisse. 11 bits pour l'exposant. 1 bit pour le signe.

Ceci implique que

- le plus grand décimal double précision est
  $\texttt{Double.MAX\_VALUE} = \left(2-2^{-32}\right)\cdot 2^{1023} \approx 1.7976931348623157\cdot 10^{308}$
- le plus petit décimal double précision est
  $$-\texttt{Double.MAX\_VALUE} = -\left(2-2^{-32}\right)\cdot 2^{1023}$$
- la distance minimale entre deux décimaux double précision est
  $$\texttt{Double.MIN\_VALUE} = 2^{-1074} \approx 4.9\cdot 10^{-324}$$

  $\rightarrow$ Le premier décimal double précision non nul est 2^{-1074}

  $\rightarrow$ Il n'y a pas de réel représentable en double précision entre 0 et 2^{-1074}

Quelques Opérateurs en Java

Opérateur	Signification	Type des opérandes
`!`	"négation"	**booléens** ($\neg$)
`&&`	"then and"	**booléens** ($\wedge$)
`||`	"else or"	**booléens** ($\vee$)
`==`	"égal"	**booléens** ($\Leftrightarrow$) **nombres** **références**
`!=`	"différent"	**booléens** ($\oplus$) **nombres** **références**
`<`	"strictement inférieur"	**nombres**
`<=`	"inférieur ou égal"	**nombres**
`>`	"strictement supérieur"	**nombres**
`>=`	"supérieur ou égal"	**nombres**
`/`	"division entière" "division"	`int, long` `float, double`
`%`	"reste de la division entière"	`int, long`

Rappel sur la division

La division d'un entier a par un entier b, notée $b|a$, s'écrit

$$a = b \cdot q + r$$

où

- a est le dividende
- b est le diviseur
- q est le quotient
- r est le reste

En Java, le quotient et le reste s'obtiennent par

- `q = a/b`
- `r = a%b`

Exemples :

1) `22/6 = 3` et `22%6 = 4`
2) `25/4 = 6` et `25%4 = 1`

Rappel sur la division

Remarques :

1) Les opérateurs `/` et `%` sont définis sur les entiers (positifs et négatifs) :

   - $10 = 3 \cdot 3 + 1 \rightarrow$ `10/3=3` et `10%3=1`
   - `(-10)/3=-3` et `(-10)%3=-1` car $-10 = 3 \cdot (-3) - 1$
   - `10/(-3)=-3` et `10%(-3)=1` car $10 = (-3) \cdot (-3) + 1$
   - `(-10)/(-3)=3` et `(-10)%(-3)=-1` car $-10 = (-3) \cdot 3 - 1$

2) Attention ! Ces opérateurs sont aussi définis sur les nombres décimaux !

Valeurs de vérité en Java

- Le type de variable pouvant contenir des valeurs de vérité est `boolean`

- Les valeurs de vérité en Java sont `true` et `false`
  - → L'évaluation d'une expression booléenne en Java (d'une proposition) ne pourra avoir que deux résultats :

    `true` ou `false`

Exemples d'expression booléenne

Voici quelques exemples d'expression

Expression	Traduction	Évaluation
`(a==4)`	"a est égal à 4"	$a = 4 \rightarrow$ true $a = 5 \rightarrow$ false
`(b!=5)`	"b n'est pas égal à 5"	$b = 5 \rightarrow$ false $b = 4 \rightarrow$ true
`(a%3==0)`	"a est divisible par 3"	$a = 6 \rightarrow$ true $a = 7 \rightarrow$ false
`(b%3!=0)`	"b n'est pas divisible par 3"	$b = 6 \rightarrow$ false $b = 7 \rightarrow$ true
`((a>10)||(b<=3))`	a strictement supérieur à 10 ou b inférieur ou égal à 3	$a = 11$ et $b = 6 \rightarrow$ true $a = 10$ et $b = 6 \rightarrow$ false
`((a%2==0)&&(b%2!=0))`	a pair et b impair	$a = 6$ et $b = 11 \rightarrow$ true $a = 6$ et $b = 4 \rightarrow$ false

Exemples d'expression booléenne

Remarques :

1. En Java, la condition `a` strictement compris entre 4 et 10 s'écrit `((a>4) && (a<10))` et surtout pas `(4<a<10)`
2. En Java, l'opérateur `&&` est prioritaire sur l'opérateur `||`.
   S'il y a plusieurs opérateurs de même priorité, ils sont évalués **de gauche à droite** :
   Exemple :
   L'expression

   `((a<2)||(a>7)&&(b==2)||(b%2==1))`

   est équivalente à l'expression

   `(((a<2)||((a>7)&&(b==2)))||(b%2==1))`

Exemples d'expression booléenne

3. L'opérateur `&&` n'est pas toujours commutatif :

$$\begin{cases} \texttt{((a>0)\&\&(24/a)>2)} & \rightarrow \quad \text{OK car si } \texttt{a} = 0 \text{ alors} \\ & \qquad \text{la première expression donne } \texttt{false} \\ & \qquad \text{et on n'évalue pas la seconde} \\ \texttt{((24/a)>2)\&\&(a>0)} & \rightarrow \quad \textcolor{red}{\text{KO}} \text{ car si } \texttt{a=0} \text{ alors} \\ & \qquad \text{on va faire une division par 0} \end{cases}$$

4. L'opérateur `||` n'est pas toujours commutatif :

$$\begin{cases} \texttt{((a>-1)||(24/a)>2)} & \rightarrow \quad \text{OK car si } \texttt{a} = 0 \text{ alors} \\ & \qquad \text{la première expression donne } \texttt{true} \\ & \qquad \text{et on n'évalue pas la seconde} \\ \texttt{((24/a)>2)||(a>-1)} & \rightarrow \quad \textcolor{red}{\text{KO}} \text{ car on si } \texttt{a=0} \text{ alors} \\ & \qquad \text{on va faire une division par 0} \end{cases}$$

Quantificateur universel et Java

Technique pour évaluer l'expression $\forall x\, p(x)$

Soit un univers U.

Pour évaluer la valeur de vérité de la formule $\forall x\, p(x)$, la technique est la suivante :

Parcourir tous les x de l'univers U :

- Si je trouve un x tel que $p(x)$ est fausse alors $\forall x\, p(x)$ est **fausse**
- Si j'ai parcouru tous les x sans avoir trouvé de x tel que $p(x)$ est fausse alors $\forall x\, p(x)$ est **vraie**

Quantificateur universel et Java

Pseudo-code pour évaluer l'expression $\forall x\, p(x)$

Soient

- l'univers $U = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$
- le prédicat $p(x) =$ "le cube de x est strictement inférieur à 1000"

Alors le pseudo-code permettant d'évaluer $\forall x\, p(x)$ est le suivant

$$\boxed{\begin{array}{l} x = 0 \\ \textit{tant que } (x <= 10) \textit{ et } (x * x * x < 1000) \textit{ répéter} \\ \qquad x = x + 1 \\ \textit{si } (x > 10) \textit{ alors résultat} = \textit{vrai} \\ \textit{sinon résultat} = \textit{faux} \end{array}}$$

Quantificateur universel et Java

Pseudo-code pour évaluer l'expression $\forall x\, p(x)$

Analyse du pseudo-code

1) La condition pour rester dans la boucle est un "et"
   $\rightarrow$ on reste dans la boucle tant que les deux conditions sont vérifiées
2) Quand on sort de la boucle cela veut dire qu'une des deux conditions n'est plus vérifiée :

   Soit la condition $x <= 10$ n'est plus vérifiée

   - $\rightarrow$ $x > 10$ est vraie
   - $\rightarrow$ on a parcouru tous les x de l'univers sans trouver de x tel que $p(x)$ soit fausse
   - $\rightarrow$ $\forall x\, p(x)$ est **vraie**.

   Soit la condition $x <= 10$ est vérifiée

   - $\rightarrow$ la condition $x * x * x < 1000$ n'est plus vérifiée
   - $\rightarrow$ on a trouvé un x tel que $p(x)$ soit fausse
   - $\rightarrow$ $\forall x\, p(x)$ est **fausse**.

Quantificateur universel et Java

Code Java pour évaluer l'expression $\forall x\, p(x)$

Le code Java correspondant au pseudo-code précédent est le suivant

```
int x = 0 ;
while ((x<=10)&&(x*x*x<1000)){
     x=x+1 ;
}
boolean result = (x>10) ;
```

Analyse de ce code Java :

1) Boucle *tant que (condition) répéter* $\rightarrow$ en Java : boucle `while`.
2) Dernière ligne du code : Java va évaluer la condition `(x>10)`.

   Si elle est vérifiée alors `result=true` sinon `result=false`.
3) Le résultat de l'exécution de ce code sera `result=false`.
   En effet, quand on arrivera à `x=10`, on aura `x*x*x=1000>=1000`.
   - $\rightarrow$ la seconde condition ne sera plus vérifiée.
   - $\rightarrow$ on sort de la boucle avec `x=10<=10`.
   - $\rightarrow$ `result=false`.

Quantificateur existentiel et Java

Technique pour évaluer l'expression $\exists x \, p(x)$

Soit un univers U.

Pour évaluer la valeur de vérité de la formule $\exists x \, p(x)$, la technique est la suivante :

Parcourir tous les x de l'univers U :

- Si je trouve un x tel que $p(x)$ est vraie alors $\exists x \, p(x)$ est **vraie**
- Si j'ai parcouru tous les x sans avoir trouvé de x tel que $p(x)$ est vraie alors $\exists x \, p(x)$ est **fausse**

Quantificateur existentiel et Java

Pseudo-code pour évaluer l'expression $\exists x\, p(x)$

Soient

- l'univers $U = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$
- le prédicat $p(x) =$ "le cube de x est strictement inférieur à 1000"

Alors le pseudo-code permettant d'évaluer $\exists x\, p(x)$ est le suivant

$$
\begin{array}{|l|}
\hline
x = 0 \\
\textit{tant que } (x <= 10) \textit{ et } (x * x * x >= 1000) \textit{ répéter} \\
\qquad x = x + 1 \\
\textit{si } (x <= 10) \textit{ alors résultat} = \textit{vrai} \\
\textit{sinon résultat} = \textit{faux} \\
\hline
\end{array}
$$

Quantificateur existentiel et Java

Pseudo-code pour évaluer l'expression $\exists x\, p(x)$

Analyse du pseudo-code

1) La condition pour rester dans la boucle est un "et"
$\rightarrow$ on reste dans la boucle tant que les deux conditions sont vérifiées

2) Quand on sort de la boucle cela veut dire qu'une des deux conditions n'est plus vérifiée :

Soit la condition $x <= 10$ n'est plus vérifiée

- $\rightarrow$ $x > 10$ est vraie
- $\rightarrow$ on a parcouru tous les x de l'univers sans trouver de x tel que $p(x)$ soit vraie
- $\rightarrow$ $\exists x\, p(x)$ est **fausse**.

Soit la condition $x <= 10$ est vérifiée

- $\rightarrow$ la condition $x * x * x * >= 1000$ n'est plus vérifiée
- $\rightarrow$ la condition `x*x*x<1000` est vérifiée
- $\rightarrow$ on a trouvé un x tel que $p(x)$ soit vraie
- $\rightarrow$ $\exists x\, p(x)$ est **vraie**.

Quantificateur existentiel et Java

Code Java pour évaluer l'expression $\exists x\, p(x)$

Le code Java correspondant au pseudo-code précédent est le suivant

```
int x = 0 ;
while ((x<=10)&&(x*x*x>=1000)){
     x=x+1 ;
}
boolean result = (x<=10) ;
```

Analyse de ce code Java :

1) Boucle *tant que (condition) répéter* $\rightarrow$ en Java : boucle `while`.
2) Dernière ligne du code : Java va évaluer la condition `(x<=10)`.

   Si elle est vérifiée alors `result=true` sinon `result=false`.
3) Le résultat de l'exécution de ce code sera `result=true`.
   En effet, quand on arrivera à `x=0`, on aura `x*x*x=0<1000`.
   - $\rightarrow$ la seconde condition ne sera plus vérifiée.
   - $\rightarrow$ on sort de la boucle avec `x=0<=10`.
   - $\rightarrow$ `result=true`.

Classes utilitaires

La classe `Math`

- Classe Java d'outils mathématiques
- Directement disponible $\rightarrow$ Pas de besoin de l'importer

La classe `Math`

Opérations classiques :

Méthode	Opération
`Math.abs(x)`	Valeur absolue de x
`Math.max(x,y)`	Maximum de x et y
`Math.min(x,y)`	Minimum de x et y
`Math.sqrt(x)`	Racine carrée de x
`Math.pow(x,y)`	x à la puissance y (x^y)

Opérations trigonométriques :

Méthode	Opération
`Math.sin(x)`	Sinus de x
`Math.cos(x)`	Cosinus de x
`Math.tan(x)`	Tangente de x

La classe Math

Opérations logarithmiques et exponentielles :

Méthode	Opération
`Math.log(x)`	Logarithme népérien (ln) de x
`Math.log10(x)`	Logarithme en base 10 de x
`Math.exp(x)`	Exponentielle de x (e^x)
`Math.pow(b,x)`	Exponentielle en base b de x (b^x)

La classe `Math`

Opérations d'arrondis :

Méthode	**Opération**
`Math.ceil(x)`	Plus petit entier $\geq x$ (arrondi à l'entier supérieur)
`Math.floor(x)`	Plus grand entier $\leq x$ (arrondi à l'entier inférieur)
`Math.round(x)`	Arrondi à l'entier le plus proche de x

Exemples :

1) `Math.ceil(12.369)` $\rightarrow$ 13.0
2) `Math.ceil(-5.46)` $\rightarrow$ -5.0
3) `Math.floor(12.369)` $\rightarrow$ 12.0
4) `Math.floor(-5.46)` $\rightarrow$ -6.0
5) `Math.round(12.369)` $\rightarrow$ 12.0
6) `Math.round(-5.46)` $\rightarrow$ -5.0
7) `Math.round(7.5)` $\rightarrow$ 8.0
8) `Math.round(-7.5)` $\rightarrow$ -7.0

Génération de nombres (pseudo-)alétoires

Avec la classe `Math`

`Math.random` génère un **`double`** pseudo-aléatoire dans $[0.0\,,\,1.0[$

Exemple : générer un nombre "aléatoire" entre 1 et 10 :

```
int x;
double y = 10*Math.random();
y = Math.floor(y) + 1;
x = (int)y;
```

1) On génère un `double` dans $[0.0\,,\,1.0[$
2) On le multiplie par 10 $\rightarrow$ on obtient un `double` dans $[0.0\,,\,10.0[$
3) On arrondi vers le bas $\rightarrow$ on obtient un entier contenu dans une variable de type `double` dans $[0.0\,,\,9.0]$
4) On ajoute 1 $\rightarrow$ on obtient un `double` dans $[1.0\,,\,10.0]$
5) On fait un transtypage explicite pour obtenir un `int` dans $[1, 10]$

Génération de nombres (pseudo-)alétoires

Avec la classe `Random`

Instanciation d'un objet de la classe `Random`

$\rightarrow$ `Random generateur = new Random()`

- La méthode `generateur.nextDouble()` génère un **`double`** pseudo-aléatoire dans $[0.0\,,\,1.0[$
- La méthode `generateur.nextInt()` génère un **`int`** pseudo-aléatoire dans [`Integer.MIN_VALUE` , `Integer.MAX_VALUE`]
- La méthode `generateur.nextInt(int n)` génère un **`int`** pseudo-aléatoire dans $[0\,,\,\mathtt{n}[$

Attention ! il faut importer cette classe : `import java.util.Random`.

Mise en pratique en Java

Séance d'exercices → `TestEgalite`, `TestDivisibilite` et `TestDate`.

Extrait de la classe `TestDate` :

```
public class TestDate {
    public static java.util.Scanner scanner = new java.util.Scanner(System.in);

    // Horoscope chinois :
    // Années du chien : 1934, 1946, 1958, 1970, 1982, 1994, 2006, 2018, ...
    // Cette methode renvoie true si annee est une annee du chien et false sinon
    public static boolean estAnneeDuChien(int annee){
        return annee%12==2 ; //A MODIFIER
    }
        ⋮
    public static void main(String [] args){
        ⋮
            choix=scanner.nextInt();
        ⋮
    }
}
```

Mise en pratique en Java

Commentaires :

1) En orange : instructions pour déclarer et utiliser le scanner qui permet les entrées au clavier
2) En vert : commentaires permettant de savoir ce que la méthode à implémenter doit calculer et renvoyer
3) En bleu : en-tête, la définition, de la méthode à implémenter.
4) En violet : commentaire indiquant les endroits où il faut écrire le code.
5) En pourpre : méthode permettant de tester les méthodes que vous avez implémentées.
6) Méthode `estAnneeDuChien` : il faut renvoyer `true` si `année` est une année du chien selon l'horoscope chinois et `false` sinon.
   Or $1934 \mod 12 = 2$ et il y a toujours 12 ans entre 2 années du chien.
   Donc toutes les années du chien auront un reste égal à 2 si on les divise par 12.
   $\rightarrow$ condition à mettre dans le code : `annee%12==2`.