# IFT 1015 - Conditions

Professeur:

Stefan Monnier

B. Kégl, S. Roy, F. Duranleau, S. Monnier

Département d'informatique et de recherche opérationnelle

Université de Montréal

hiver 2006

# *Au programme*

- Instructions conditionnelles: `if - else` et `switch`
- Blocs d'instruction
- Expressions relationnelles et booléennes
- [Tasso:3], [Niño:6]

# Les instructions de sélection

Objectif

- permettre au programme de choisir entre les directions
- l'ordre d'exécution peut dépendre de l'information qui n'existe que pendant l'exécution

Instructions

- `if` et `switch`

Type primitif logique

- `boolean`: `true` ou `false`

# Exemple de `if`

Retirer l'argent seulement s'il y en a assez sur le compte

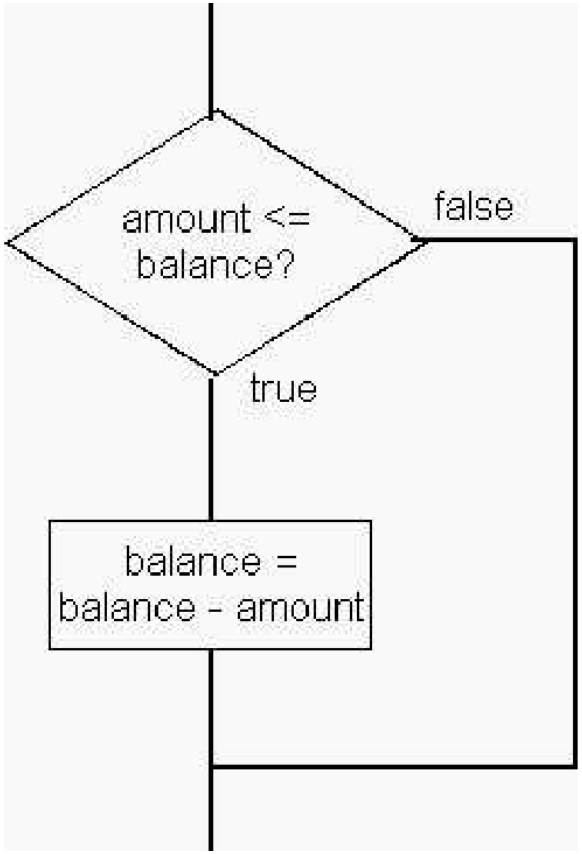
```
double balance = 1345.89;
System.out.println("How much to withdraw?");
double amountToWithdraw = Keyboard.readDouble();
if (balance > amountToWithdraw)
    balance = balance - amountToWithdraw;
System.out.println("New balance is " + balance);
```

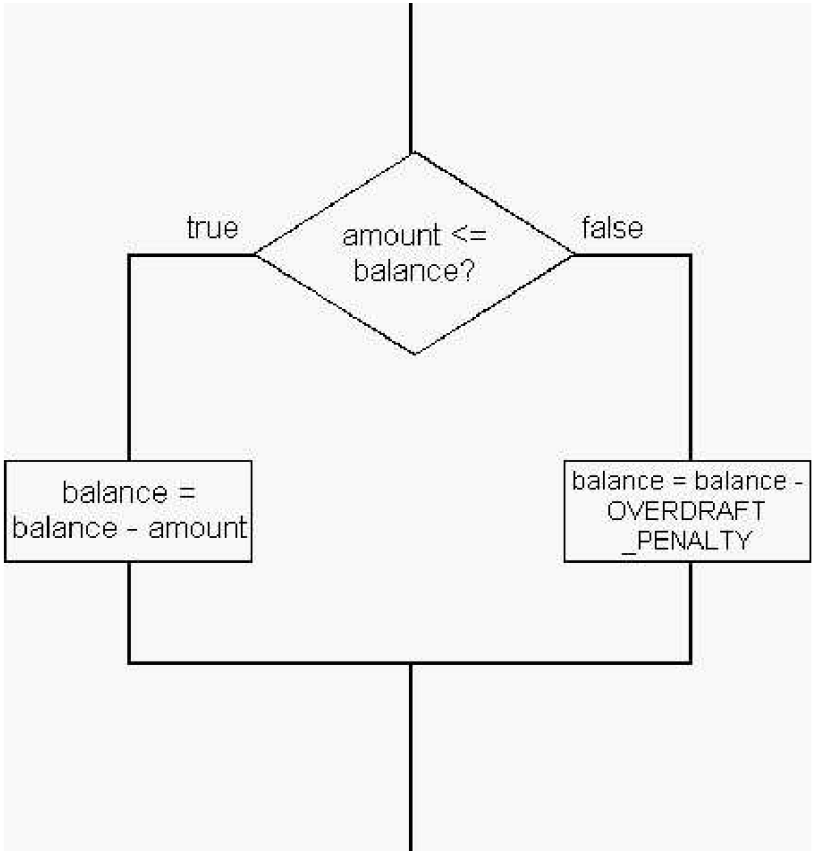
# *Exemple de* `if ... else`

Sinon, ajouter une pénalité

```
double balance = 1345.89;
final double OVERDRAFT_PENALTY = 25;
System.out.println("How much to withdraw?");
double amountToWithdraw = Keyboard.readDouble();
if (balance > amountToWithdraw)
    balance = balance - amountToWithdraw;
else
    balance = balance - OVERDRAFT_PENALTY;
```

# Organigrammes de `if ... else`



# *Syntaxe de* `if` *et* `if ... else`

Syntaxe de `if`

```
if (<condition>)
   <instruction>
```

Syntaxe de `if ... else`

```
if (<condition>)
   <instruction1>
else
   <instruction2>
```

# *Sémantique de* `if ... else`

Exécution de `if ... else` (la sémantique)

1. `<condition>` est évaluée
2. si elle est vraie, `<instruction1>` est exécutée
3. si elle est fausse, `<instruction2>` est exécutée
4. après l'instruction `if ... else`, l'exécution se poursuit à partir de la prochaine instruction

# L'`else` suspendu

L'`else` appartient au dernier `if` sans `else`

```
if (true)
   if (false)
      System.out.println(2);
else
   System.out.println(3);
```

- faites attention à l'indentation
- utilisez les $\{\}$ si nécessaire

# Le bloc d'instructions

```
if (balance > amountToWithdraw) {
    balance = balance - amountToWithdraw;
    System.out.println(
        "Succesful withdraw, new balance is "
         + balance);
} else {
    balance = balance - OVERDRAFT_PENALTY;
    System.out.println(
        "Withdraw refused, penalty applied, "
        + "new balance is " + balance);
}
```

# Syntaxe des blocs d'instructions

```
{
    <instruction_1>
    . . .
    <instruction_n>
}
```

- alignez les accolades
- utilisez toujours les blocs, même s'il n'y a qu'une instruction

# Comparaisons

Opérateurs relationnels; pour comparer des nombres

| opérateur de Java | notation en math | description |
|---|---|---|
| `>` | $>$ | supérieur à |
| `>=` | $\geq$ | supérieur ou égal à |
| `<` | $<$ | inférieur à |
| `<=` | $\leq$ | inférieur ou égal à |
| `==` | $=$ | égal à |
| `!=` | $\neq$ | n'est pas égal à |

# Comparaisons en virgules flottantes

erreurs d'arrondi

```
double r = Math.sqrt(2);
double d = r * r - 2;
if (d == 0)
   System.out.println("sqrt(2)^2 minus 2 is 0");
else
   System.out.println(
      "sqrt(2)^2 minus 2 is not 0 but " + d);
```

résultat:

```
sqrt(2)^2 minus 2 is not 0 but 4.440892098500626E-16}
```

# Comparaisons approximatives

Nombres en point flottant

- tester s'ils sont proches

comparaison absolue: $|x - y| \leq \epsilon$

comparaison relative: $\frac{|x - y|}{\max(|x|, |y|)} \leq \epsilon$

```
final double EPSILON = 1E-14;
if (Math.abs(x - y) <= EPSILON)
    . . .
if (Math.abs(x - y)/Math.max(Math.abs(x),Math.abs(y)) <=
    EPSILON)
    . . .
```

# Comparaisons de chaînes

- `if (str1.equals(str2)) ...`,
  PAS `if (str1 == str2) ...`!!!
- ignorer les minuscules/majuscules:
  `if (str1.equalsIgnoreCase(str2)) ...`
- comparer en ordre alphabétique: `str1.compareTo(str2)`

# `boolean` *et les opérateurs conditionnels*

Le type `boolean`

- stocker les valeurs logiques
- littéraux: `true`, `false`

Opérateurs conditionnels

| opérateur de Java | notation en math | vrai si |
|---|---|---|
| `a && b` | $a \wedge b$ | les deux, $a$ et $b$ sont vrais |
| `a \|\| b` | $a \vee b$ | soit $a$ soit $b$ est vrai |
| `!a` | $\overline{a}$ ou $\neg a$ | $a$ est faux |

# Tables de vérité

`a && b`

| | b est `true` | b est `false` |
|---|---|---|
| a est `true` | `true` | `false` |
| a est `false` | `false` | `false` |

`a || b`

| | b est `true` | b est `false` |
|---|---|---|
| a est `true` | `true` | `true` |
| a est `false` | `true` | `false` |

# *Exemples de* `boolean` *et conditions*

- `if (0 < amount && amount < 1000) ...`
- `if (in.equals("S") || in.equals("M")) ...`
-

```
boolean married;
. . .
if (married) // PAS if (married == true)
   . . .
else
   . . .
```

# La préséance des opérateurs

| Nom de l'opérateur | Genre | Opérateur |
|---|---|---|
| in/décrémentation | numérique | `++ --` |
| négation | booléen | `!` |
| signe unaire | numérique | `+ -` |
| cast | numérique | `(<type>)` |
| multiplication, division, reste | numérique | `* / %` |
| add/concaténation),soustraction | numérique | `+ -` |
| comparaison | relationnel | `== != < > <= >=` |
| ET logique | booléen | `&&` |
| OU logique | booléen | `\|\|` |
| affectation | numérique | `=` |
| affectation-opérateur | numérique | `+= -= /= *=` |

# Un switch sans `switch`

```
int noteNumerique = 78; // 0 <= noteNumerique <= 100
char note = ' ';
if(noteNumerique/10 == 10)
   note = 'A';
else if(noteNumerique/10 == 9)
   note = 'A';
else if(noteNumerique/10 == 8)
   note = 'B';
else if(noteNumerique/10 == 7)
   note = 'C';
else if(noteNumerique/10 == 6)
   note = 'D';
else
   note = 'F';
```

# Un switch avec `switch`

```
int noteNumerique = 78; // 0 <= noteNumerique <= 100
char note = ' ';
switch (noteNumerique/10) {
   case 10: note = 'A'; break;
   case 9:  note = 'A'; break;
   case 8:  note = 'B'; break;
   case 7:  note = 'C'; break;
   case 6:  note = 'D'; break;
   default: note = 'F'; break;
}
```

# *Syntaxe de* `switch`

[Tasso:3], [Niño:6]

```
switch (<expression NUMÉRIQUE>) {
   case <constante>:
      <séquence d'instructions>
         ...
   case <constante>:
      <séquence d'instructions>
   default:
      <séquence d'instructions>
}
```

# Sémantique de `switch`

1. `<expression NUMÉRIQUE>` est évaluée
2. si elle est égale à n'importe quelle `<constante>`, l'exécution se poursuit à partir de l'étiquette `<constant>:`
3. sinon, l'exécution se poursuit à partir de l'étiquette `default:`

Remarques

- `<constante>` est soit une variable `final`, soit un littéral
- `break` termine le bloc d'instructions