Introduction à la programmation (Java)

(D’après des slides de Pierre Poulin); A ne pas distribuer

Condition et test

- Une condition correspond à vrai ou faux
- E.g. (age < 50)
- Tester une condition:

  if *condition A;* else *B*;

  – si *condition* est satisfaite, alors on fait *A*;
  – sinon, on fait *B*
  – E.g.

```
if (age < 65)
    System.out.println("jeune");
else
    System.out.println("vieux");
```

Tests

- Pour les valeurs primitives (int, double, …)
  - x == y : x et y ont la même valeur?
  - x > y, x >= y, x != y, …
  - Attention: (== != =)
- Pour les références à un objet
  - x == y : x et y pointent vers le même objet?
  - x.compareTo(y): retourne -1, 0 ou 1 selon l'ordre entre le contenu des objets référés par x et y

Un exemple de test

```
public class Salutation
{
   public static void main(String[] args)
   {
      int age;
      age = Integer.parseInt(args[0]);
      // afficher une salutation selon l'age
      System.out.print("Salut, le ");
      if (age < 65)
         System.out.println("jeune!");
      else
         System.out.println("vieux!");
   }
}
```

- Utilisation:
  - java Salutation 20 // ici, args[0] = "20"
    Salut le jeune!
  - java Salutation 70

    Salut le vieux!

args[0]: premier argument après le nom

Integer.parseInt(args[0]): reconnaître et transmettre la valeur entière qu'il représente

print: sans retour à la ligne
println: avec retour à la ligne

Exercices

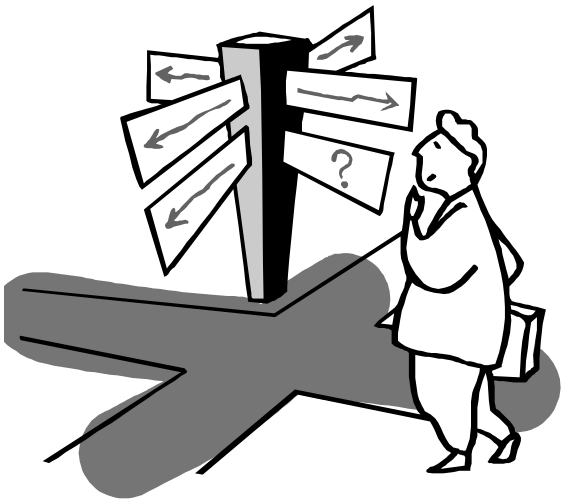
1) Recopiez le programme et testez le
2) Faite afficher, « vieux », « adultes » et « mineurs » (<18) et « bebe » (<2)
3) && en Java permet de retourner vrai (le boolean true) si les deux conditions valent vrais (ou false sinon). Faite afficher en plus (pour adultes) que si on a entre 30 et 65 alors on est « ». Et

Recursivité

1) Un peu de maths pour bcp de facilité
2) Changement de support...

Les choix

Le bloc conditionnel



Où en sommes nous ?

- Les objets Java modélisent les objets d'un problème

  Machine à tickets de parking, tickets de concert
- Les objets sont créés à partir de classes

  La classe des machines à tickets
- Le bloc de déclaration de classe déclare le nom de la classe, ses champs, ses méthodes, son constructeur.
- Les champs stockent les données utilisées par les objets.

  Prix d'un ticket, montant inséré, total encaissé
- Le constructeur permet de donner des valeurs initiales différentes aux différents objets d'une classe

  1€ pour le parking, 30 € pour le concert
- Les méthodes sont les messages acceptés par les objets

  `getPrice(), insertMoney(int), printTicket()`

Le texte d'une méthode (rappel)

- Le bloc de déclaration des méthodes est constitué de deux parties :
  - l'en-tête qui décrit le type des paramètres et de la valeur de retour

    `void printTicket()`
  - Le corps qui est une séquence d'instructions à exécuter lorsque le message est reçu par l'objet :

```
{
  System.out.println("(" + this.price + " euros)");
  this.total += this.balance;
  this.balance = 0;
}
```

- Problème : le billet est délivré même si le client n'a pas inséré la bonne somme. Aucun test ?!

L'instruction conditionnelle `if`

- Elle réalise un « aiguillage » suivant le résultat d'un test :

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

- « Il fait beau » est soit vrai, soit faux :
  C'est une expression booléenne.

- SI test ALORS instruction

```
if (<test>) <instruction>;
```

```
if (balance >= price)                  // on a payé en trop !
  System.out.println("(Ticket " + price + " euros)");
```

- SI test ALORS $instr_1$ puis $instr_2$ puis … puis $instr_k$

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

```
if (balance >= price) {              // on a payé en trop !
  System.out.println("(Ticket " + price + " euros)");
  total += balance ;
  balance -= price ;
}
```

- SI test ALORS $instr_1$ SINON $instr_2$

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

```
if (balance >= price)              // on a payé en trop
   System.out.println(”(Ticket ” + price + ” euros)”) ;
else                               // on n'a pas assez payé
   System.out.println("Insérez " + (price-balance)) ;
```

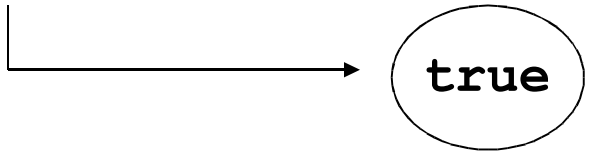
- Avec les mêmes conventions d'accolades que précédemment s'il y a plusieurs instructions dans l'une ou l'autre des parties...

- Toutes les instructions sont acceptées dans les blocs conditionnels, y compris d'autres blocs conditionnels !

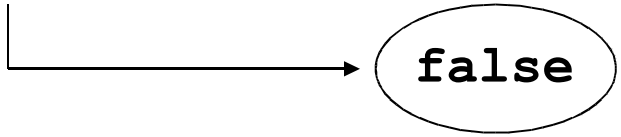
Les valeurs booléennes `true` et `false`

- Ne pas confondre l'affectation `=` et la comparaison `==` !

```
int x = 2, y = 3, z = 6;          // déclarations + initialisations

z = z + 1;          // z devient égal à z + 1, donc à 7

y == x + 1          // est-ce que y vaut x + 1, c'est à dire 3 ?
```




```
x == y + 1          // est-ce que x vaut y + 1, c'est à dire 4 ?
```



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

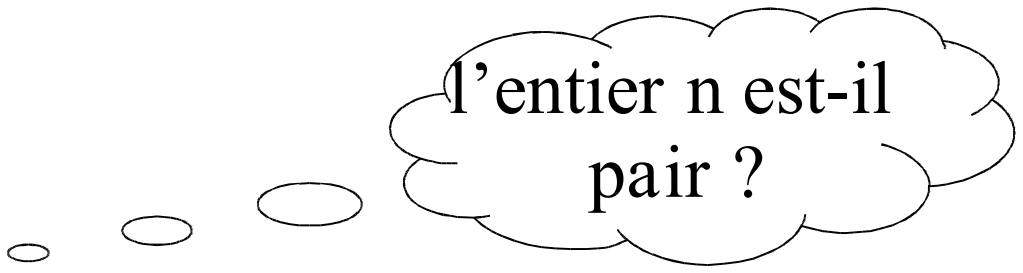
```
int x = 5;
boolean b1 = (x == 2 + 3);
boolean b2 = (x != 2 + 3);
boolean b3 = (x <= 0);
System.out.println("b1 vaut : " + b1);
System.out.println("b2 vaut : " + b2);
System.out.println("b3 vaut : " + b3);
```


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

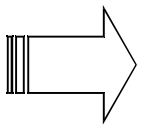
Les expressions booléennes

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

```
x < y + 5
somme >= 0
n % 2 == 0
```




```
si n est pair alors
   ......
sinon
   ......
```



```
if (n % 2 == 0)
   ......
else
   ......
```

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);
```

0.1 est un nombre compliqué en binaire... Cf. TD/TP 2 !

Exemple : la méthode **`insertMoney`**

- La fonction **`insertMoney`** ne vérifie pas que les sommes entrées sont cohérentes (valeurs négatives ?)

```
class TicketMachine {
    . . .
    void insertMoney (int amount) {
       if (amount > 0)
         this.balance += amount ;
       else
         System.out.println("Introduisez une somme > 0”);
    }
}
```

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) ...`

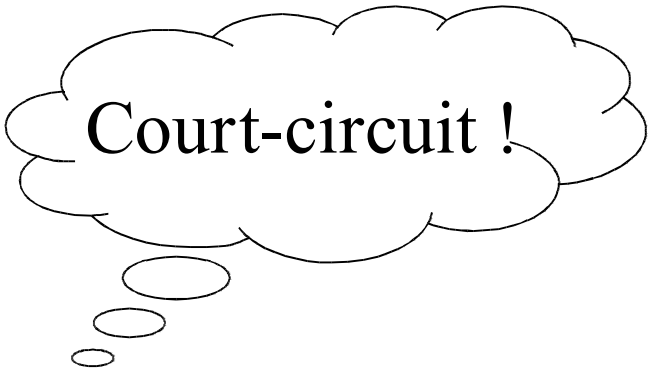
La conjonction logique &&

ET

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

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 !

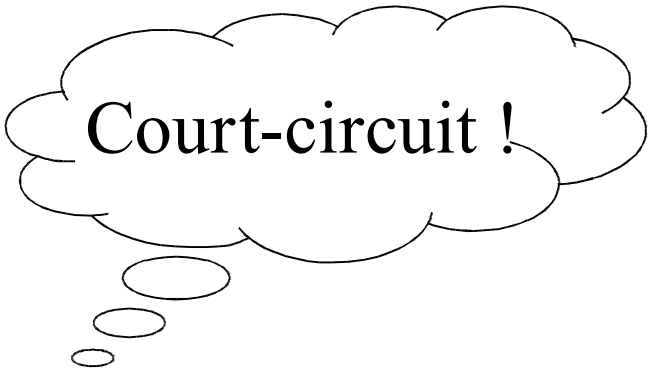
```
if ((x >= 0) && (ticketMa1.getBalance() == 0))

   ......
```

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`**



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

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

Les opérateurs de comparaison

- En résumé, Java a 6 opérateurs de comparaison sur les expressions numériques (types *int*, *long*, *float*, *double*) :

>	plus grand que
<	plus petit que
>=	plus grand ou égal à
<=	plus petit ou égal à
!=	différent de
==	égal à

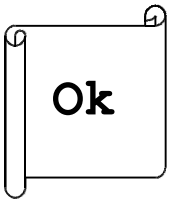
- Java a 3 opérateurs logiques pour manipuler les expressions booléennes (type *boolean*) :

&&	et logique	(non équivalent à **&** hors-programme)
||	ou logique	(non équivalent à **|** hors-programme)
!	non logique	(non équivalent à **~** hors-programme)

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)
   System.out.println("Ok");
```



➢ Encore une fois : éviter de questionner l'égalité de nombres approchés ! Demander plutôt si leur distance est inférieure à une précision ε donnée...

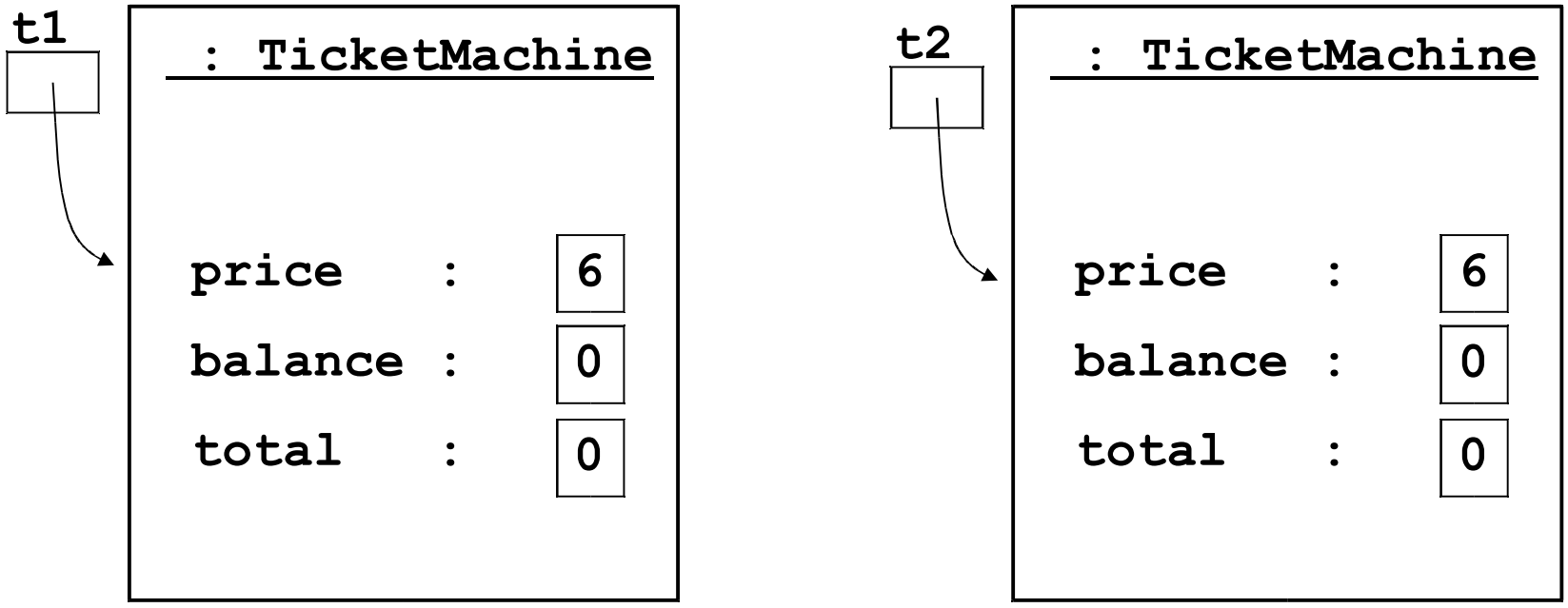
```
double x = Math.log(3), y = Math.sqrt(0.8);

if (x == y)                          bad

if ( Math.abs(x - y) < 1e-6 )        good
```

Quid de l'égalité == entre objets ?…

ATTENTION : le signe == entre objets est une comparaison entre références et non entre champs !

```
TicketMachine t1 = new TicketMachine(6);
TicketMachine t2 = new TicketMachine(6);
```

`t1 == t2` ⟹ `false`

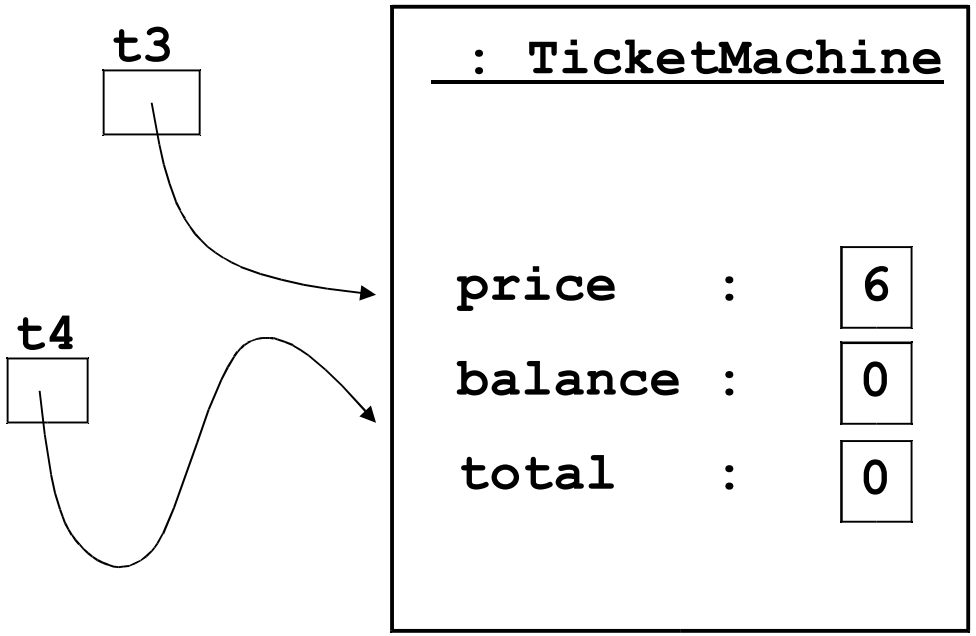


Quid de l'affectation = entre objets ?…

ATTENTION : le signe **=** entre objets est une affectation entre références et non entre champs !

```
TicketMachine t3 = new TicketMachine(6);
TicketMachine t4 = t3;
```

t3 == t4 ⟹ **true**



Comparer tous les champs : **egalA**

```
boolean egalA(TicketMachine t)
{
   if ( (t.price == this.price) &&
        (t.balance == this.balance) &&
        (t.total == this.total) )
      return true ;
  else
      return false ;
}
```

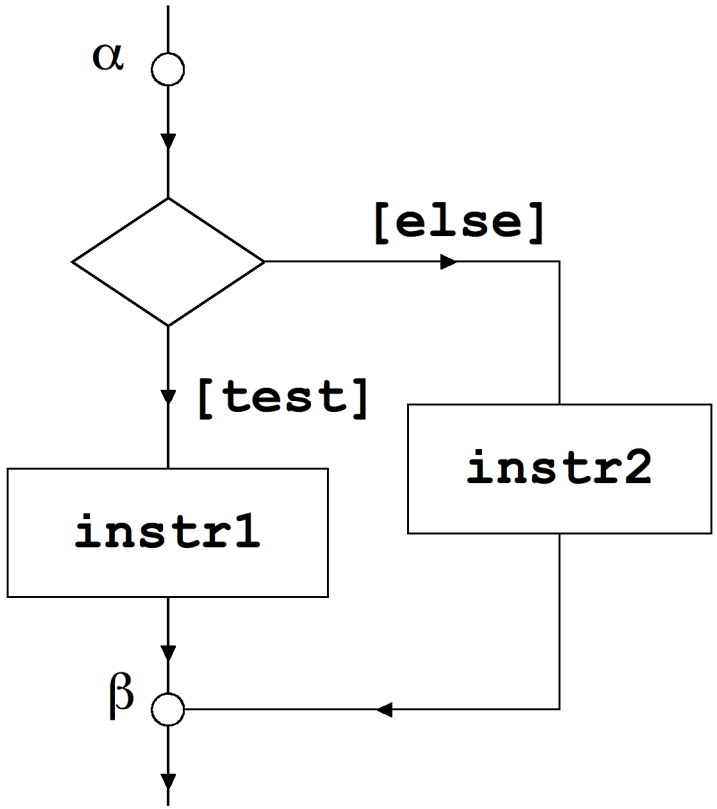
```
t1.egalA(t2) ?
t3.egalA(t4) ?
```

ou mieux :

```
boolean egalA(TicketMachine t)
{

   return (t.price == this.price)
          && (t.balance == this.balance)
          && (t.total == this.total) ;
}
```

Schématisation avec un ordinogramme
organigramme

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

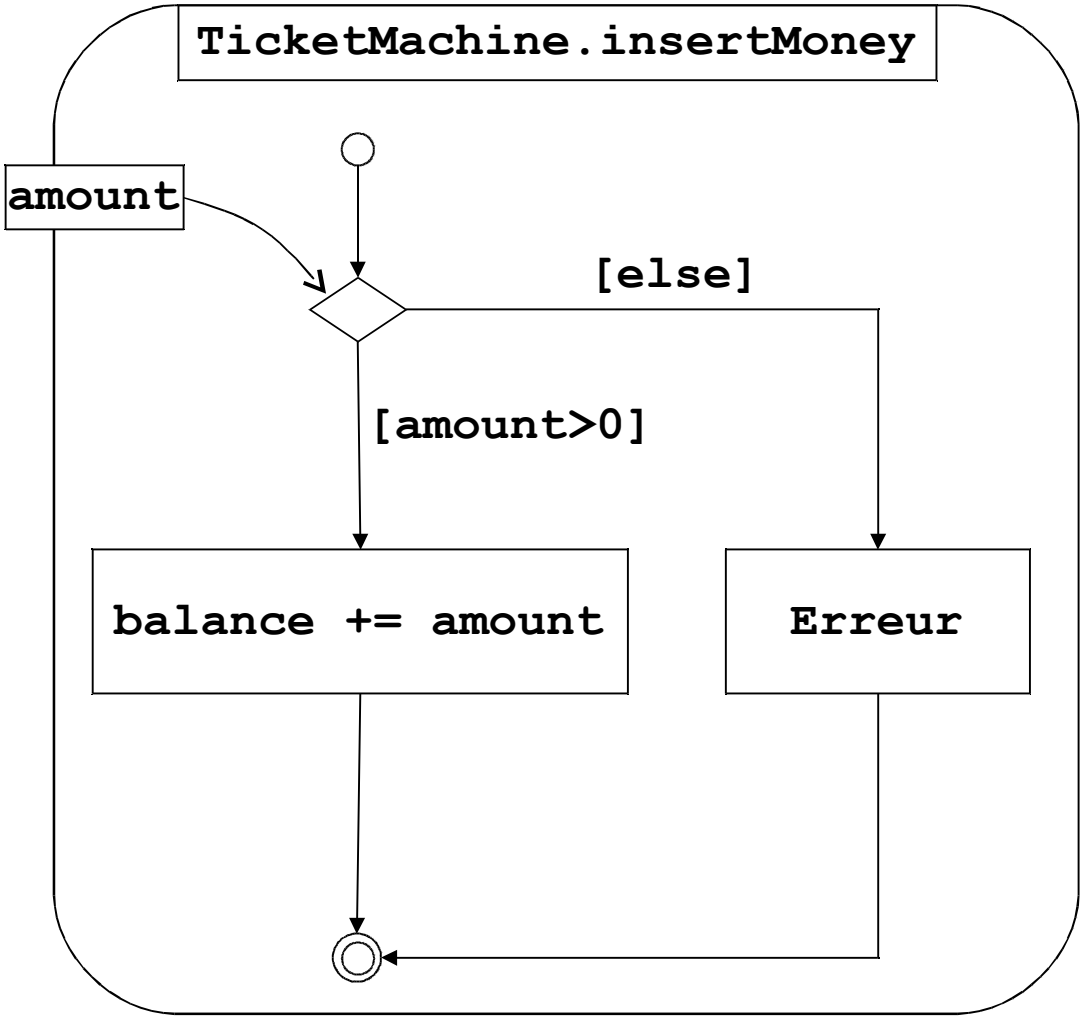


UML : diagrammes d'activités

Diagrammes d'activités

- Les **rectangles** contiennent les actions à effectuer
  - Instruction élémentaire
  - Appel à une méthode (un autre diagramme d'activités)
- On suit les **flèches** pour trouver l'ordre d'exécution des actions
- Les **losanges** représentent les décisions
  - 1 entrée
  - des sorties gardées : une et une seule garde doit être valide.
- Avec Java, un diagramme d'activité a : ○
  - Un **seul point d'entrée** symbolisé par◎
  - Un **seul point de sortie** symbolisé par
- On peut associer **un diagramme par méthode** complexe.

Exemple : **`insertMoney`**



Les variables locales

Rendre la monnaie

- **`int refundBalance ()`**
  - Il faut mettre à zéro le solde et rendre l'argent qu'il reste:

```
this.balance = 0 ;
return this.balance ;
```

```
return this.balance ;
this.balance = 0 ;
```

```
int refundBalance ()
{
    int refund ;
    refund = this.balance ;
    this.balance = 0 ;
    return refund ;
}
```

variable locale

champ

3 sortes de variables : Champs, paramètres et variables locales

- Les variables permettent toutes de stocker des valeurs compatibles avec leur type, par exemple int pour une valeur entière.
- On modifie leur valeur avec l'opérateur d'affectation **=**
- LES CHAMPS. Exemples : **`price`**, **`balance`**, **`total`**
  - Ils sont définis en dehors des méthodes et constructeurs.

    ```
    int balance ;
    ```
  - Ils mémorisent les données qui persistent tout au long de la vie d'un objet : ils représentent son état.
  - Ils ont la portée de la classe : leur valeur est accessible à partir de n'importe quelle méthode de la classe.
  - Leur valeur est initialisée par le constructeur et peut être modifiée par les méthodes (« modificateurs »).

- LES PARAMETRES. Exemples : **`ticketCost`**, **`amount`**
  - Ils sont définis dans l'en-tête des méthodes ou des constructeurs

    ```
    void insertMoney(int amount)
    ```

  - Ils reçoivent leur valeur de l'extérieur en fonction de l'évaluation de l'argument d'appel

    ```
    ticketMa1.insertMoney(5);
    ```

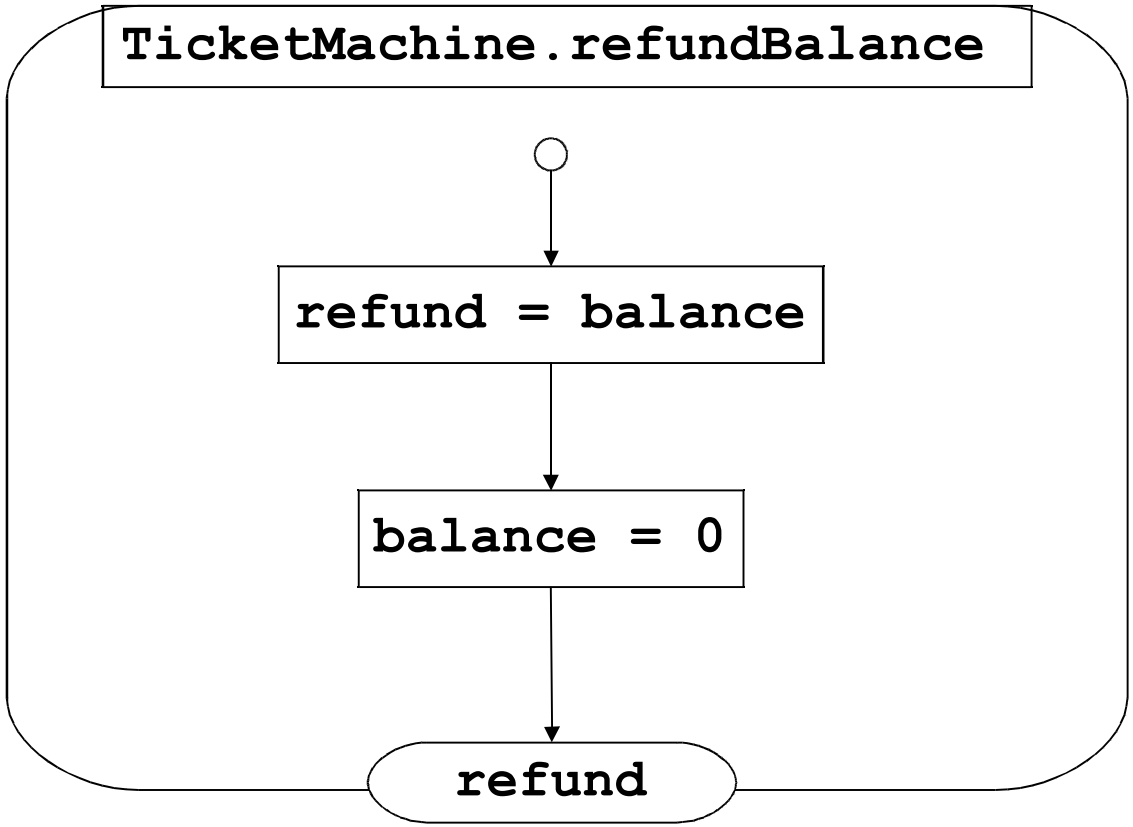
  - Ils ont la portée de la méthode : ils existent pendant toute la durée de l'appel de la méthode ou du constructeur, mais ni avant, ni après !

- LES VARIABLES LOCALES. Exemple : **`refund`**
  - Elles sont définies dans le corps des méthodes ou des constructeurs.

    ```
    { int refund ; … }
    ```
  - Elles DOIVENT être initialisées à l'intérieur du bloc

    ```
    refund = this.balance ;
    ```
  - Elles ont la portée de la méthode : elles existent à partir de leur déclaration, jusqu'à la fin de l'exécution du bloc.
  - Contrairement aux champs, elles n'ont PAS de valeur par défaut !
  - Elles n'apparaissent pas lors de l'inspection avec BlueJ car elles ne sont pas accessibles en dehors de leur bloc.

Activité avec valeur de retour



Bien commenter les programmes !

avec ***Javadoc***

- Outil pour générer automatiquement la documentation HTML sur le modèle de l'API de Java.

Trois formes de commentaires :

- Demi-ligne de commentaires

```
if (balance > price)  // on a trop payé
```

- Plusieurs lignes de commentaires

```
/* ceci est un bloc de commentaires
   qui peut s'étaler sur plusieurs lignes */
```

- Bloc de documentation Javadoc **`/** ... */`**
  - Zone de description en HTML suivie de tags @
  - Utilisé pour la javadoc …

Bloc de documentation

- Pour les classes
  - Le bloc de commentaire doit précéder la ligne **`class`** …
  - La zone de description est en HTML, attention aux accents
  - Respecter l'ordre des tags

```
/**
 * Un cercle que l'on peut manipuler (taille, couleur,
 * position), et qui se dessine sur un Canvas.
 *
 * @author Michael Kolling and David J. Barnes
 * @version Sept 2005
 * @see Canvas, Triangle
 */
```

- Pour les constructeurs : **`@param, @see`**

```
/** Constructeur de la classe TicketMachine
 * @param ticketCost prix unitaire d'un ticket
 */
TicketMachine(int ticketCost) { … }
```

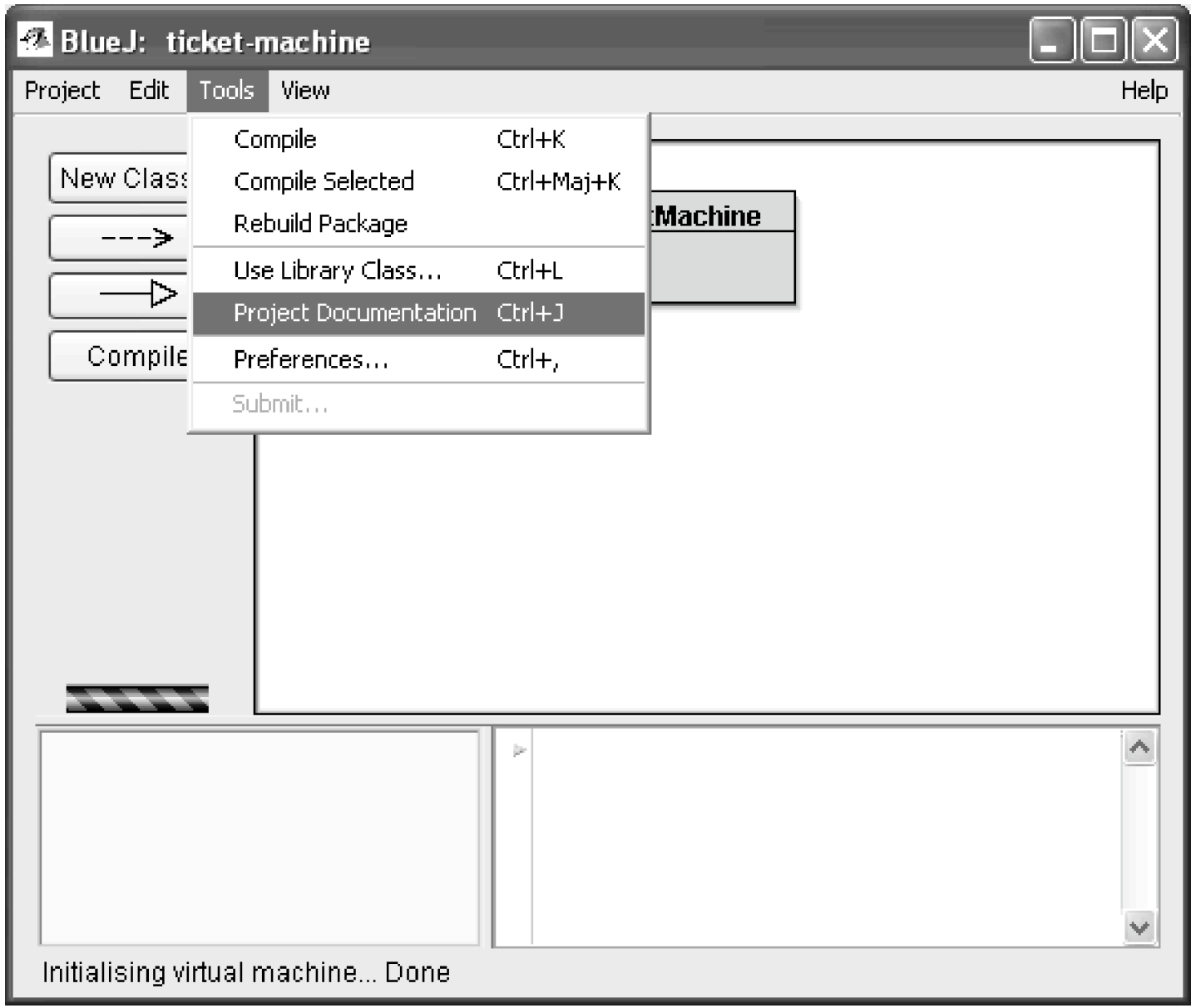
- Pour les méthodes : **`@param, @return, @see`**

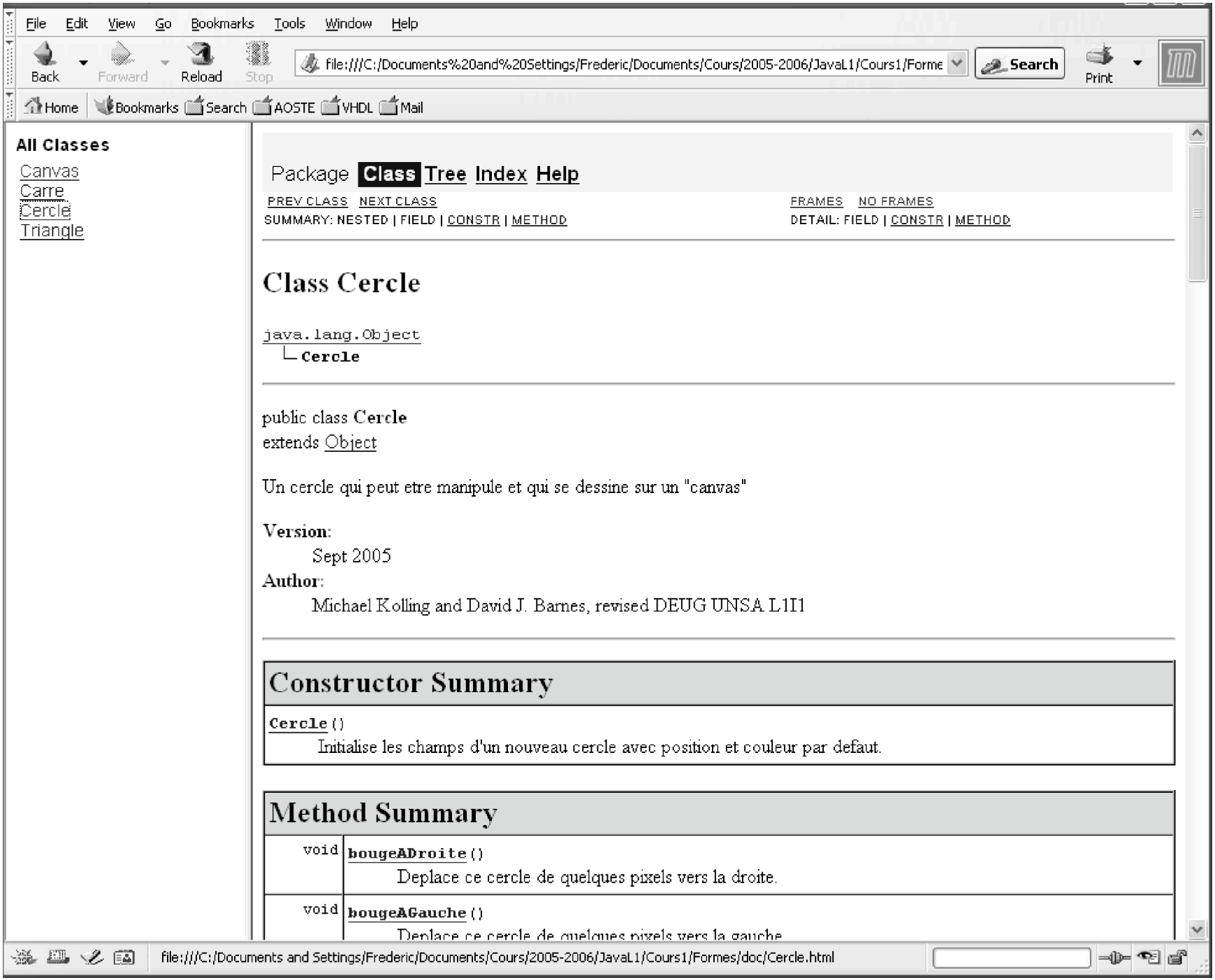
```
/**
 * Annule l'opération et restitue l'argent introduit
 * @return   montant introduit par l'utilisateur
 */
int refundBalance () { … }
```

- Pour les champs

```
/** la somme totale contenue dans la machine */
int total;
```

Voir la documentation Javadoc avec BlueJ




File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
file:///C:/Documents%20and%20Settings/Frederic/Documents/Cours/2005-2006/JavaL1/Cours1/Forme
Search
Print
Home Bookmarks Search AOSTE VHDL Mail
All Classes
Canvas
Carre
Cercle
Triangle
Package Class Tree Index Help
PREV CLASS NEXT CLASS
FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD
Class Cercle
java.lang.Object
└ Cercle
public class Cercle
extends Object
Un cercle qui peut etre manipule et qui se dessine sur un "canvas"
Version:
Sept 2005
Author:
Michael Kolling and David J. Barnes, revised DEUG UNSA L1I1
Constructor Summary
Cercle()
Initialise les champs d'un nouveau cercle avec position et couleur par defaut.
Method Summary
void bougeADroite()
Deplace ce cercle de quelques pixels vers la droite.
void bougeAGauche()
Deplace ce cercle de quelques pixels vers la gauche
file:///C:/Documents and Settings/Frederic/Documents/Cours/2005-2006/JavaL1/Cours1/Formes/doc/Cercle.html