« Je crois qu'il y a un marché mondial pour environ cinq ordinateurs. »
- T.J. Watson (1943), fondateur d'IBM

ITI 1520
Introduction à l'informatique I
Notes de cours, automne 2010

D. Inkpen

(Contributeurs: G. Arbez, D. Amyot, S. Boyd, M. Eid, A. Felty, R. Holte, W. Li, S. Somé, et A. Williams)

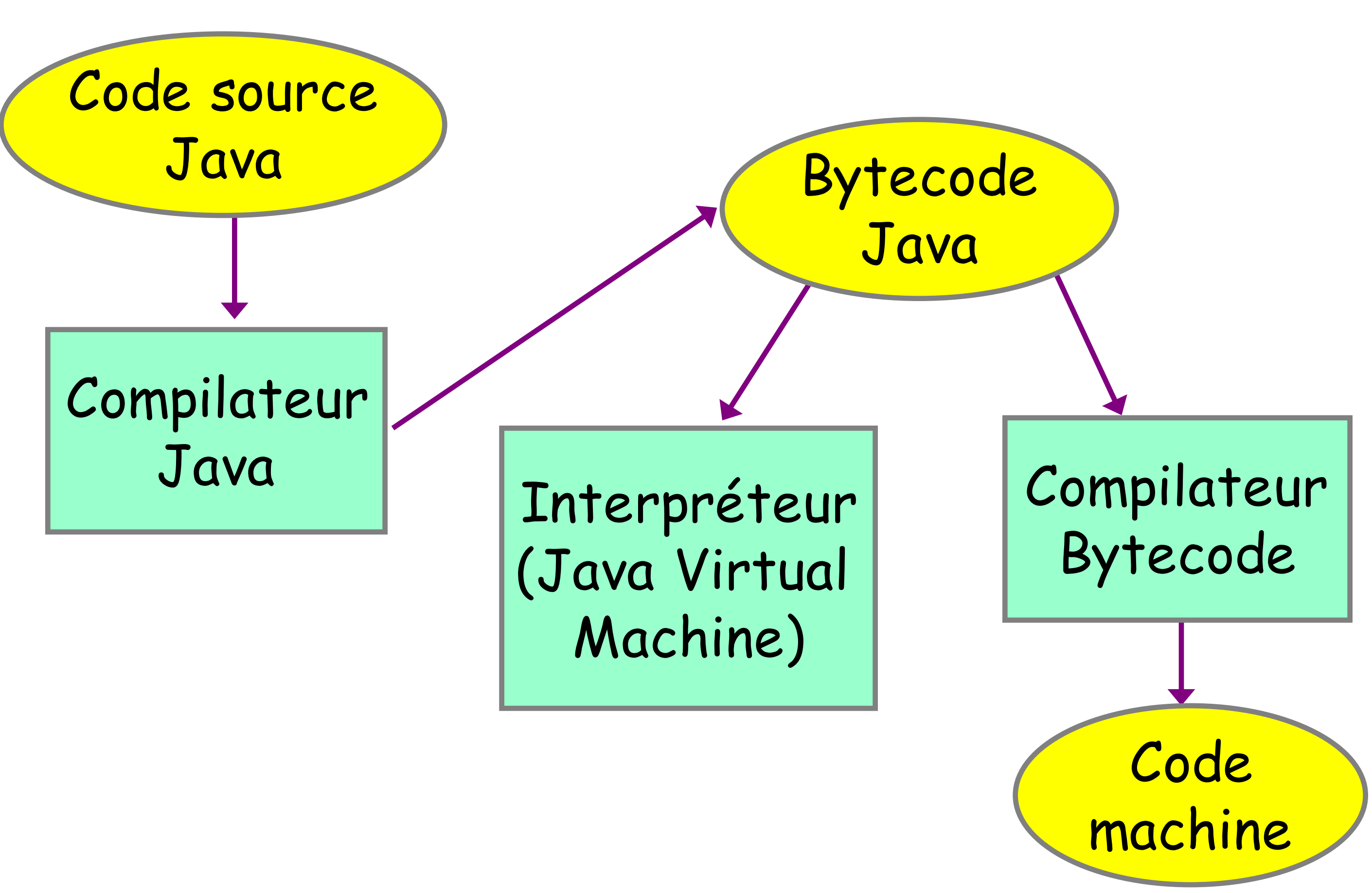
© 2010, ÉITI, Université d'Ottawa

Il est interdit d'utiliser ou de reproduire ces notes sans la permission des auteurs.

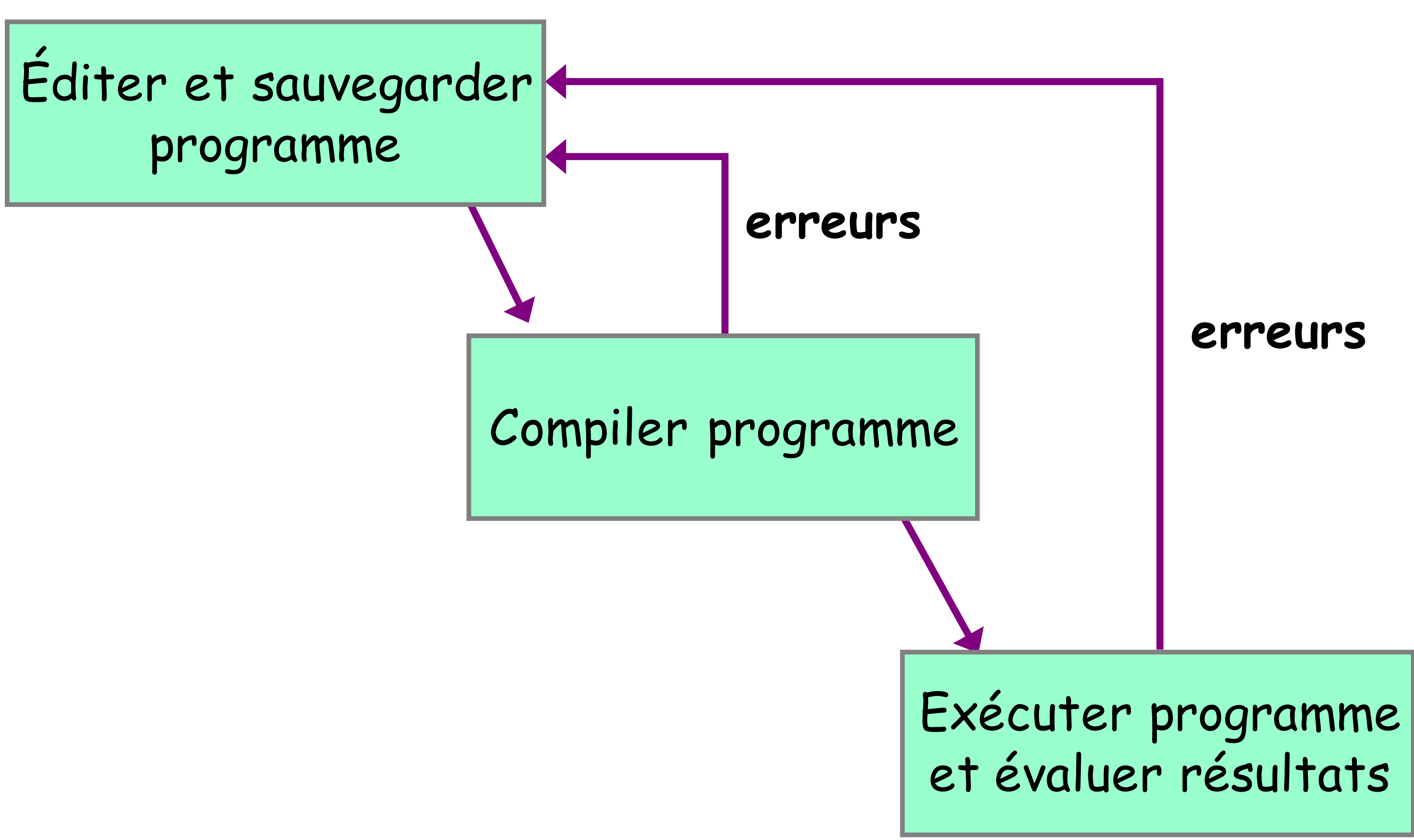
Langage de programmation et compilation

- Langage « compréhensible » par un compilateur.
  - Règles précises de syntaxe et de sémantique.
- Les programmes dans des langages tels que Pascal, C ou Java doivent être convertis dans une forme directement exécutable par l'ordinateur (langage machine).
- D'autres sont seulement interprétés (Perl, Javascript)
- Compilation: processus de conversion d'un programme écrit dans un langage de haut niveau (ex: Pascal, C, Java) vers un langage de bas niveau (langage machine).
- Cette conversion est effectuée automatiquement par un compilateur (ex: javac).

Traduction en Java



Développement de programmes



Test, débogage et maintenance

- Test = recherche d'erreurs (bogues) dans un algorithme ou un programme en lui fournissant des données en entrée et en vérifiant l'exactitude des résultats.
  - N.B. Il est habituellement impossible de tout tester, et de cette façon de prouver qu'un programme est correct.
- Débogage = processus de localisation et de correction d'une erreur dans un algorithme ou un programme.
- Maintenance = concerne la modification d'un algorithme ou d'un programme pour mettre à jour ses fonctions ou corriger des erreurs.

Trois types d'erreurs

1. Erreurs de syntaxe: combinaisons illégales de symboles qui ne respectent pas les règles d'un langages de programmation. Ces erreurs sont détectées pour vous par le compilateur.

2. Erreurs d'exécution (*run-time*): erreurs qui résultent des valeurs fournies en entrée.

3. Erreurs logiques: erreurs qui résultent d'un raisonnement incorrect du programme
   - Probablement causées par un algorithme incorrect.
   - Aussi appelées erreurs « sémantiques »

“La seule façon d’apprendre un nouveau langage de programmation est de programmer avec lui.”
- B. Kernighan & D. Ritchie, inventeurs de Unix et du langages C

ITI 1520
Section 2: Introduction à Java

Objectifs:
- Types de données
- Caractères et chaînes
- Opérateurs et précédence
- Lecture et affichage de valeurs
- Gabarit de classe

Conversion vers le code source

- Notre approche à la programmation consiste à développer et tester en premier un algorithme en un modèle logiciel et puis à le TRADUIRE (convertir, ou coder) en code source dans un langage de programmation.

> "Quand un nouveau langage permet de programmer simplement en anglais, nous découvrons que les programmeurs ne peuvent pas parler anglais…"
> -- Anonyme

- Traduire des algorithmes en code source est un processus plutôt mécanique qui n'implique que peu de décisions.

- Pour chaque bloc du modèle logiciel, nous verrons une façon de le traduire fidèlement en code source. Les algorithmes peuvent être traduits bloc par bloc.

Java

- Java est le langage de programmation utilisé dans ce cours
- Un programme Java contient un certain nombre de classes. Chaque classe est sauvegardée dans son propre fichier, du même nom mais avec l'extension **`.java`**.
- Une méthode (ex: **`main`**) peut aussi invoquer d'autres méthodes, qui peuvent être ajoutées à la classe. Les méthodes Java sont des *sous-prgrammes* conçus par la décomposition de problème.
- Nous créerons des logiciels avec UNE seule classe qui ne contient que **deux** méthodes. La méthode **`main`** permet d'implémenter l'interaction avec l'utilisateur et la deuxième pour résoudre un problème.
  - Vous aurez donc à développer deux l'algorithmes, une pour chaque méthode du logiciel.
  - La méthode main est la première méthode exécuté lorsque le logiciel est démarré. Elle appelle la méthode qui résout le problème.

Données en Java

- Les programmes Java utilisent des LITTÉRAUX (*literal* en anglais) et des VARIABLES.
  - Un littéral représente une constante, telle que
    - **`123`** (un entier),
    - **`12.3`** (un nombre réel),
    - **`'a'`** (un caractère),
    - **`true`** (une valeur Booléenne).
  - Une variable utilise un identifiant (ex: **`x`**) pour représenter une donnée.
    - L'identifiant représente l'adresse de l'emplacement en mémoire où la valeur est stockée
    - Seulement après sont initialisation (affectation d'une valeur), devez-vous considérez que sa valeur est « connue ».

Types de données

- Les variables et valeurs ont des TYPES.

- Un type de données indique:

1. Quelle valeurs une variable peut prendre (l'ensemble des valeurs légales).

2. Quelles opérations sont disponibles pour manipuler ces valeurs, et ce que font ces opérations.

3. La façon dont ces valeurs sont emmagasinées en mémoire dans l'ordinateur.
   - Représentation sujet d'un autre cours…

Type de données primitifs en Java

- Une variable ou valeur a un type de donnée PRIMITIF si elle représente une valeur simple qui ne peut pas être décomposée.
- Java possède certains type de données primitifs . Nous utiliserons dans ce cours::

`int`	représente des entiers (ex: `3`)
`double`	représente des nombres réels (ex: `3.0`)
`char`	représente des caractères simples (ex: 'A')
`boolean`	représente des valeurs Booléennes (ex: true)

Type `int`

- Une variable de type `int` accepte des valeurs de **-2147483648** à **2147483647**.
  - Dépasser cet intervalle mènera à un dépassement de capacité (OVERFLOW), une erreur d'exécution.

- Les opérateurs suivants sont disponibles pour le type `int` :
  - `+` (addition)
  - `-` (soustraction, négation)
  - `*` (multiplication)
  - `/` (division entière, où la fraction est perdue; retourne un `int`)
    Exemple: `7 / 3 donne 2`
  - `%` (modulo; reste de la division)
    Exemple: `7 % 3 donne 1`
  - `== != > < >= <=` (comparaisons: prennent deux valeurs de type `int` et retournent une valeur `boolean`)

Type double (littéraux)

- Le type **double** représente des nombres “réels” de -1.7×10^{308} à 1.7×10^{308} avec 15 chiffres significatifs.
  - Même s'il y a beaucoup de valeurs acceptées par **double**, leur ensemble est fini, et donc ces valeurs ne sont que des approximation de nombres réels.
  - Après un calcul, l'ordinateur choisi la valeur **double** la plus près du vrai résultat; ceci peut introduire des erreurs d'arrondissement.

- Formats des grandes et petites valeurs:
  $12345600000000000.0 = 0.123456 \times 10^{17}$ est `0.123456e17`
  $0.00000123456 = 0.123456 \times 10^{-5}$ est `0.123456e-5`

- Si la valeur de l'exposant **e** est plus négative qu'environ -308, alors le résultat est à nouveau un dépassement de capacité (UNDERFLOW), et la valeur sera remplacée par zéro. Ceci ne génère **pas** d'erreur d'exécution.

Type **double** (opérateurs)

- Les opérateurs suivants sont disponibles pour le type **double** :
  - **+** (addition)
  - **–** (soustraction, négation)
  - ***** (multiplication)
  - **/** (division, où le résultat est un **double**)
  - **> <** (comparaisons: prennent deux **double** et retournent une valeur **boolean**)

- ATTENTION: L'utilisation de **==** (ou **!= >= <=**) pour comparer deux variables de type **double** est permis, mais NON recommandé, à cause des erreurs d'arrondissement potentielles.
  - Au lieu de `(a == b)`, utilisez
    `(Math.abs(a - b) < 0.001)`
    où `Math.abs(x)` produit |x|

Type `boolean`

- Seulement deux valeurs: `true` et `false`
- Ces opérateurs sont disponibles pour le type `boolean`:

  `!` NON (opérateur unaire, a une seule opérande)

  `&&` ET

  `||` OU

  comparaison : `==` `!=`

Type `char` (valeurs)

- Les caractères sont des symboles individuels, entourés de guillemets (apostrophes)
  - Lettre de l'alphabet (majuscules différentes des minuscules) `'A'`, `'a'`
  - Ponctuation (virgule, point d'interrogation, …) `','`, `'?'`
  - Espace blanc `' '`
  - Parenthèses `'('`,`')'`; crochets `'['`,`']'`; accolades `'{'`,`'}'`
  - Chiffres (`'0'`,`'1'`,… `'9'`)
  - Caractères spéciaux tels `'@'`,`'$'`,`'*'`,etc.

Type `char` (codes)

- Chaque caractère se voit assigné un code, selon la norme utilisée:
  - Code ASCII
    - 128 caractères (sans accent)
    - Le plus connu (surtout si vous parlez un anglais américanisé!)
  - Code UNICODE
    - Plus de 64,000 caractères (international)
    - inclut ASCII
    - utilisé en Java
  - Code ISO Latin 1
    - 256 caractères (avec accents)
    - Les 128 premiers caractères sont ceux d'ASCII
    - Utilisé par défaut dans nos navigateurs Web
    - Couvre plusieurs langues européennes
      - incluant le français, sauf pour 2-3 caractères ésotériques (ex: Y tréma)
    - Caractères de Windows: extension de ISO Latin 1

Ordre des caractères

- Dans un ordinateur, un caractère est représenté et emmagasiné comme un code numérique.
- Par exemple, le code ASCII indique quels sont les caractères représentés par les valeurs de 0 à 127 (mêmes que ISO Latin 1 et Unicode pour Java).
- Exemples:

Caractère	`'a'`	`'A'`	`' '`	`'0'`	`'?'`
Valeur Unicode	97	65	32	48	63

- **Exercice 2-1** - L'ordre numérique des caractères permet de comparer ces derniers:

`'A' < 'a'` est

alors que

`'?' < ' '` est

Codes pour lettres et chiffres

- Points importantes sur les codes ASCII/UNICODE:
  - Les **chiffres** sont codés par des valeurs consécutives (les codes pour `'0'` à `'9'` sont 48-57 respectivement). Ainsi

    `'2' < '7'` donne `true`

    `'7' - '0'` donne `7`
  - Ce raisonnement est aussi valide pour les lettres **minuscules** (`'a'` à `'z'` → 97-122). Ainsi

    `'r' < 't'` donne `true`
  - Ce raisonnement est aussi valide pour les lettres **majuscules** (`'A'` à `'Z'` → 65-90).
    - Notez que leurs codes sont plus petits que ceux des lettres minuscules: `'b' - 'A'` donne `33`

Exercice 2-2 - Test pour majuscule ?

- Soit la variable **x** contenant une valeur de type **char**.
- Écrivez une expression Booléenne qui est VRAIE si la valeur de **x** est une lettre majuscule et FAUSSE sinon.
  - Notez que nous n'avons pas besoin de connaître les valeurs codées de ces caractères!

Caractères spéciaux

- Quelques caractères sont traités spécialement parce qu'ils ne peuvent pas être tapés facilement ou parce qu'ils ont une interprétation particulière en Java.
  - Nouvelle ligne `'\n'`
  - Tabulateur `'\t'`
  - Apostrophe `'\''`
  - Guillemet `'\"'`
  - Barre oblique arrière `'\\'`

- Les caractères ci-haut commencent par un code d'échappement spécial (`\`) mais ne représentent qu'un seul caractère.

Conversion de types

- En général, on ne peut PAS convertir une valeur d'un type en un autre type en Java, sauf pour certains cas spéciaux.
- Quand une conversion est permise, vous pouvez faire un forçage de type (*type casting* en anglais):

  `(double) 3` retourne `3.0`

  `(int) 3.5` retourne `3` (notez la perte de précision!)

  `(int) 'A'` retourne `65` (Valeur UNICODE)

  `(char) 65` retourne `'A'`

- ATTENTION: La conversion de types avec des résultats inattendus a causé de sérieux problèmes logiciels dans le passé:
  - Une telle erreur a causé l'autodestruction de la fusée Ariane 501 en 1996.
- La meilleure stratégie: ne pas mélanger les types, à moins de nécessité absolue!

Chaînes de caractères (String)

- Une chaîne de caractères est un ensemble ordonné de caractères.
  - Il n'y a PAS de type primitif pour les chaînes en Java.
    - Nous verrons plus tard comment gérer ces chaînes.
- Des valeurs de chaînes de caractères (constantes) peuvent être utilisées pour rendre les résultats de votre programme plus lisibles (valeurs entre doubles guillemets)
  - **`"Voici une chaîne de caractères"`**
- ATTENTION:
  - **`"a"`** (une chaîne) versus **`'a'`** (un caractère)
  - `" "` (une chaîne de longueur 1 contenant un espace) versus `""` (une chaîne vide de longueur 0)
  - **`"257"`** (une chaîne) versus **`257`** (un entier)
  - “ ” « » ne sont pas des guillemets valides en Java!

Concaténation de chaînes (String)

- Les chaînes peuvent être CONCATÉNÉES (fusionnées) en utilisant l'opérateur **+** :

  "Mon nom est" + "Daniel" donne

  "Mon nom estDaniel"

- La concaténation peut aussi se faire entre une chaîne et une valeur d'un autre type avec le même opérateur **+** :

  "X contient " + 1.5 donne

  "X contient 1.5"

  //1.5 converti en String!

Précédence des opérateurs

- Les opérateurs sont évalués de gauche à droite, avec la précédence suivante (les opérateurs sur une même ligne on la même précédence):

  () (expression) **[]** (indice de tableau) **.** (membre d'objet)

  + - (unaire – indiquant positif ou négatif) **!** (non)

  *** / %**

  + - (pour addition ou soustraction de deux valeurs)

  < > >= <=

  == !=

  &&

  ||

  = (affectation d'une valeur à une variable)

Exercice 2-3 - Précédence des opérateurs ?

- Quelle est l'ordre d'évaluation des expressions suivantes?

```
a + b + c + d + e          a + b * c - d / e
```

```
a / (b + c) - d % e
```

```
a / (b * (c + (d - e)))
```

Variables en Java

- Les données, résultats et intermédiaires d'un algorithme sont représentées par des variables dans un programme Java.

- Pour utiliser une variable, il faut d'abord la déclarer.
  - Une déclaration de variable spécifie son type, son nom et, optionnellement, sa valeur initiale.
  - La déclaration réserve de la mémoire pour la variable.
  - Exemples

```
int x = 0;            // Une variable int x avec 0 comme valeur initiale
double x;             // Une variable double non initialisée
char c = ' ';         // Une variable char initialisée avec un espace
boolean b1 = false;   // Une variable boolean initialisée à false
```

- Les déclarations de variables se terminent par un point-virgule.

Exercice 2-4 - Un peu de math

- Pour affecter une valeur à une variable, l'opérateur « = » est utilisé, ex: **x=2**;
- Déclarez les variables suivantes:
  - Trois variables réelles: **cout1, cout2, cout3**
  - Affectez des valeurs aux 3 variables
  - Une variable réelle **moy**
  - Une expression qui calcule la moyenne des variables **cout1**, **cout2**, et **cout3** et affecte le résultat à **moy**.
- Déclarez les variables entières **varEnt1**, **varEnt2**, **quotient**, et **restant**. (affectez des valeurs à varEnt1 et varEnt2)
  - Donnez l'expression qui affecte le quotient de **varEnt1** divisé par **varEnt2** à **quotient**
  - Donnez l'expression qui affecte le restant de **varEnt1** divisé par **varEnt2** à **restant**

Exercice 2-4 - Un peu de math

?

```
// Déclarations de variables




// Calcul de la moyenne

// Déclarations de variables



// Calcule quotient et restant
```

Variables primitives

- Lorsque qu'une variable est déclarée avec un type primitif,
  - Une espace en mémoire est réservée pour contenir la valeur de la variable.
  - Le nom de la variable (représente l'adresse de la variable) est associé de façon *permanente* à cet espace mémoire.

Variables de référence

- Seulement un nom pour une variable de type tableau ou un type défini par le programmeur (un « objet », à voir plus tard).
- La variable est utilisée pour « référer » à un tableau ou objet.
- À la déclaration de la variable de référence, il n'y a pas d'espace mémoire réservé par défaut pour le tableau ou l'objet. Vous devez utiliser l'opérateur **`new`** pour créer l'objet ou le tableau, et ensuite leur associer un nom. Une telle variable peut être par la suite associée à ce tableau ou cet objet (*ou des différents*!)
  - La variable de référence est une variable qui contient une adresse.
  - Associé un tableau ou objet à une variable de référence veut dire affecter l'adresse du tableau/objet dans la variable de référence
  - Détails et exemples à venir…

Commentaires

- Un commentaire explique le sens de votre variable ou instruction.
- Les commentaires Java ont principalement deux formes:
  - Commence par `//` jusqu'à la fin de la ligne

```
int taille = 30; // nombre d'étudiants
```

  - Un commentaire entre `/*` et `*/`

```
/* Ce programme calcule la somme des éléments
 * d'un tableau et affiche le résultat.
 */
```

- Les commentaires sont utilisés pour aider les gens à lire le programme, et sont ignorés par le compilateur.

> "Être forcé d'écrire des commentaires améliore le code, car il est plus simple de corriger une erreur que de l'expliquer." -- G. Steele

Javadoc

- Quelques outils (p.e.: javadoc, fourni avec le compilateur Java et avec DrJava) peuvent être utilisés pour générer automatiquement des pages Web de documentation à partir du code et des commentaires, si ces derniers sont d'un format particulier.

- Exemple de commentaire javadoc:

```
/**
 * Ce texte sera inclus dans un document HTML.
 *
 * @author Daniel Amyot
 * @version 2009
 */
```

- Voir http://java.sun.com/j2se/javadoc et la documentation en ligne de la classe **iti1520**

Méthodes Java

- Tel que déjà mentionné, la méthode Java est un sous-programme
- Chaque méthode Java a
  - Un type de retour: Spécifie le type du *résultat* de la méthode. S'il n'y a pas de résultat, ce type est mis à **`void`**.
    - Pour le moment, toujours ajouté les mots clefs « public static » avant le type de retour.
  - Un nom: Même nom que l'algorithme, qui sert de référence à la méthode.
  - Une liste de paramètres: Noms et types des paramètres (liste ordonnée).
    - Notez que la liste de paramètres est une liste de « déclarations de variables ».
  - Un module (corps): Le bloc d'instructions Java (syntaxe et grammaire Java) traduit du module de l'algorithme
    - Notez la forme du bloc d'instruction: l'utilisation des accolades { } et de l'indentation.

Gabarit pour méthodes

```
// MÉTHODE Nom: Courte description de ce que fait la
// méthode, suivie d'une description des variables
// présentes dans la liste des paramètres
```

Type de retour | Nom de la méthode | Liste de paramètres

```
public static double moy3(int a, int b, int c)
{
   // DÉCLARATION DES VARIABLES / DICTIONNAIRE DE DONNÉES
   // intermédiaires et résultat (s'il a un nom)

   // MODULE DE L'ALGORITHME

   // RÉSULTAT RETOURNÉ
   return <valeur_retournée> ;
}
```

expression
(valeur à retourner)

La méthode Java “retourne” une valeur

- La méthode Java peut retourner aucune ou une valeur
  - Il n’est pas possible de retourner plus d’une valeur en Java comme dans nos algorithmes. Mais…
    - Cette valeur peut être de type primitif ou de type référence. Par exemple, une méthode peut retourner la référence à un tableau.
  - Pour le moment, nous allons développer des algorithmes qui utilise une seule variable comme résultat.
  - La méthode Java retourne une « valeur », il n’affecte pas une valeur à une variable
    - Un appel à une méthode Java peut être interpréter comme la « lecture d’une valeur de variable » et donc peut être placé à tout endroit ou peut se trouver un nom de variable.
    - Dans ce cours, pour correspondre à nos algorithmes, nous allons affecter la valeur de retour d’un appel à un variable et dans une méthode toujours déclarer un variable résultat
    - Plus de détails à section 3.

Méthodes avec type de retour **void**

- Si une méthode ne retourne pas de valeur (type **void**), alors son invocation ne fait qu'exécuter le module de la méthode sans obtenir de valeur en retour.

```
public static void print3(int x, int y, int z)
{
    System.out.println("Cette méthode ne retourne aucune valeur.");
    System.out.println(x);
    System.out.println(y);
    System.out.println(z);
}
```

- Quand la méthode est invoquée par

  ```
  print3(3, 5, 7);
  ```

  elle ne fait qu'afficher une phrase suivie des 3 arguments.

Méthodes « standards » en Java

- Le Java offre des classes et méthodes « standards »
  - Par exemple certaines méthodes sont disponibles pour accomplir des fonctions mathématiques
    - Utilisé **`Math.abs(a-b)`** pour traduire |a-b| de l'acétate <u>#55. Type double (opérateurs)</u>.
    - Pour traduire $\sqrt{x}$ utilisez **`Math.sqrt(x)`**
    - Voyez le chapitre 5 de Tasso ou la section 5.9 de Liang pour une liste de méthodes mathématiques.
  - D'autres méthodes existe pour lire du clavier et écrire à la console
    - On y revient à ces méthodes dans un moment.

Sorties en Java

- Java utilise la console comme sortie de base (l'affichage); pour communiquer avec l'utilisateur
  - fenêtre textuelle (DOS, Linux)
  - fenêtre spéciale dans un environnement de développement (comme DrJava).

- La plupart des applications Windows utilisent aussi des dialogues (autres fenêtres spéciales) comme sortie.

La sortie (affichage) en Java

- Pour afficher une valeur à l'écran, nous utiliserons deux méthodes Java de la classe **`System.out`**:

  `System.out.println(...)`

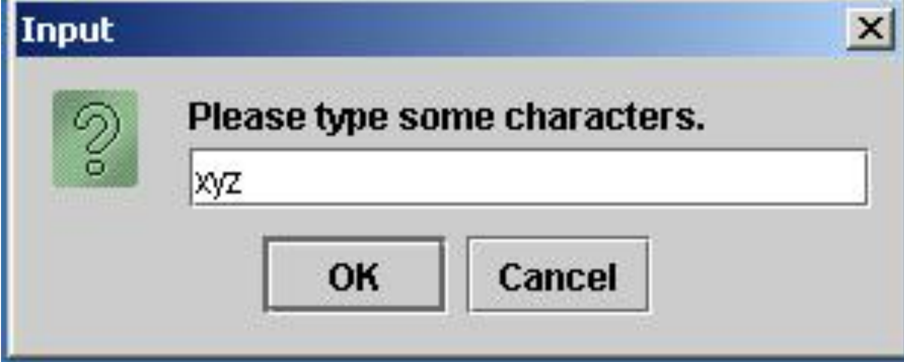
  `System.out.print(...)`

  - La méthode **`println()`** affiche la chaîne de caractères (ou le caractère) suivi d'une retour-chariot (nouvelle ligne) alors que **`print()`** affiche mais ne change pas de ligne.

- Les appels à ces méthodes sont spéciales car elle accepteront n'importe quel type d'argument ou aucun argument.
  - L'appel **`println()`** sans argument peut être utilisée pour afficher une ligne vide.

Entrées en Java

- Les entrées en Java ne sont malheureusement pas aussi simples que les sorties.
- La lecture au clavier est particulièrement difficile
  - Java 5.0 a cependant grandement amélioré la situation
- Plusieurs applications utilisent aussi des dialogues en entrée:



Gabarit pour programmes (à utiliser dans vos devoirs)

```
class NomDeVotreClasse
{
  // La méthode main interagit avec l'utilisateur
  public static void main (String[] args)
  {
   // DÉCLARATIONS DES VARIABLES ET DICTIONNAIRE DE DONNÉES
   int var;
   // LECTURE DES VALEURS DONNÉES - utiliser ITI1520 pour l'entrée
   // MODULE DE L'ALGORITHME - Appel méthode de résolution problème
   var = résoudProblème();
   // AFFICHAGE DES RÉSULTATS ET MODIFIÉES À L'ÉCRAN
  }

  // La méthode suivante résout un problème.
  public static int résoudProblème( )
  {
    // DÉCLARATION DE LA VARIABLES
    int varRésultat;
    // MODULE
    // RETOURNER LE RÉSULTAT
    return( varRésultat );
  }
}
```

Méthodes de la classe **Scanner**

`nextInt( ):`	Retourne un entier du type **`int`**.
`nextDouble( ):`	Retourne un nombre réel du type **`double`**
`nextBoolean( ):`	Retourne la valeur **`true`** ou **`false`** comme valeur du type **`boolean`**
`nextLine( ):`	Retourne un **`String`** contenant une ligne complète.

- La valeur retournée par ces méthodes doit être affectée à une variable du type correspondant.
- Suite à l'invocation de l'une de ces méthodes, votre programme sera suspendu et attendra vos données.
- Quand vous entrez une valeur à partir du clavier et appuyez sur ENTER, le programme emmagasine la valeur entrée dans la variable que vous aurez spécifiée dans le programme, qui pourra alors poursuivre son exécution.

Exemples d'utilisation de `Scanner`

- Initialisation du scanner:

```
Scanner clavier = new Scanner( System.in );
```

```
int x = clavier.nextInt( );
```

- Si vous entrez `123` et appuyez sur ENTER, `x` contiendra `123` .

```
boolean b = clavier.nextInt( );
```

- Si vous entrez `true` et appuyez sur ENTER, `b` contiendra `true`.

```
String s = clavier.nextLine( );
```

- La méthode `nextLine` place TOUS les caractères (incluant les espaces) que vous tapez sur la ligne dans la chaîne référée par `s`.

Gabarit alternatif pour programmes (Java 5.0/6.0 seulement)

```
// ITI1520 (Automne 2008), Devoir 0, Question 57
// Nom: Grace Hopper, numéro d'étudiant: 1234567
// Groupe de laboratoire: LAB (1, 2, ou 3) AE: Prénom et nom de votre AE
// Description/utilisation du programme

import java.util.Scanner ;

class NomDeVotreClasse
{
  public static void main (String[] args)
  {
   // MISE EN PLACE DE LA LECTURE AU CLAVIER
     Scanner clavier = new Scanner( System.in );

   // DÉCLARATIONS DES VARIABLES ET DICTIONNAIRE DE DONNÉES
   // AFFICHAGE DE L'INFO D'IDENTIFICATION
     System.out.println();
     System.out.println("ITI1520 (A-2008), Devoir 0, Question 57");
     System.out.println("Nom: Grace Hopper, #Etudiant: 1234567");
     System.out.println("Groupe de laboratoire: LABx, AE: VotreAE");
     System.out.println();
   // LECTURE DES VALEURS DONNÉES
   // MODULE DE L'ALGORITHME - appel à la méthode pour résoudre problème
   // AFFICHAGE DES RÉSULTATS ET MODIFIÉES À L'ÉCRAN
  }
  // DÉFINITIONS DE LA MÉTHODE INVOQUÉE PAR « main ».
}
```

« Ne demandez pas ce que cela veut dire,
mais plutôt comment l'utiliser »
- Ludwig Wittgenstein

ITI 1520
Section 3: Algorithmes et leur traduction vers Java

Objectifs:

- Invocation d'algorithmes
- Passage d'information
- Arguments modifiés
- Traduction systématique vers Java

Portée et longévité d'une variable

- Portée: Où vous pouvez utiliser le nom d'une variable.
  - Règle générale: Les variables ne peuvent être accédées qu'à l'intérieur de leur algorithme.
  - Si vous utilisez le nom X dans deux algorithmes, ces noms représentent deux valeurs différentes dans les algorithmes.

- Longévité: La « durée de vie » d'une valeur de variable.
  - Lorsqu'un algorithme termine son exécution, les valeurs de ses variables sont oubliées.
    - Les valeurs dans la section RÉSULTATS sont cependant retournées à l'algorithme invocateur.
  - Lorsque l'algorithme est invoqué à nouveau, de nouvelles valeurs sont utilisées dans les variables.
  - Fonctionnement de l'ordinateur: Lorsqu'un sous-programme est exécuté (algorithme), une parcelle de mémoire de travail est affecté pour le temps de son exécution.

Raffinement d'algorithme

- Si un algorithme contient un bloc d'instructions complexe imbriqué dans un autre bloc, vous pouvez transformer le bloc imbriqué en un algorithme séparé.
  - Il est donc possible de diviser une tâche complexe en plus petites tâches; exemple:
    - Tâche 1 – interagir avec l'utilisateur
    - Tâche 2 – résoudre un problème

- Chaque algorithme peut à son tour invoquer d'autres algorithmes.
- De cette façon, nous pouvons garder nos algorithmes simples, courts, et clairs.

Décomposition de problème

- La meilleure façon de développer des algorithmes pour les problèmes non triviaux est la décomposition de problème (aussi appelée conception de haut vers le bas, ou *top-down*).

- Un algorithme pour le problème P est développé de la façon suivante:
  1. Identifiez les sous-problèmes (P1, P2, …, Pn), plus simples que P, dont les résultats seraient utiles pour résoudre P.
  2. Déterminez l'information d'en-tête (données, résultats, en-tête) pour P1…Pn, mais ne concevez pas leurs algorithmes tout de suite.
  3. Écrivez l'algorithme de P en supposant l'existence d'algorithmes pour P1…Pn.
  4. Développez les algorithmes de P1…Pn.

“It's OK to figure out murder mysteries,
but you shouldn't need to figure out code.
You should be able to read it.”
- *Steve McConnell*

Traduction systématique vers Java

Traduire nos algorithmes à un Programme Java

- Note approche de programmation est d'abords de développer et tester nos algorithmes (modèle de notre programme) et par la suite TRADUIRE ceux-ci à du code d'un langage de programmation.
- La traduction d'algorithmes est un procédé mécanique, qui ne demande que quelques décisions.
- Chaque algorithme est traduit à une méthode Java.
- Pour la première moitié du cours, nos programmes ne contiendront qu'une seule classe Java ayant deux méthodes:
  - **`main`** qui interagit avec l'utilisateur
  - Une méthode de résolution de problème pour implémenter l'algorithme de résolution de problème.

Traduction à un programme (2 méthodes)

```
import java.util.Scanner;
class CalculeMoy
{
   /**
    *  Méthode: main
    *  Description: Interface avec l'utilisateur
    */
   public static void main (String[] args)
   {
      // Instructions
   }
   /**
    *  Méthode: resMoy
    *  Description: Calcule la moyenne (pourcentage)
    *               de trois notes sur 25
    *  Paramètres (DONNÉES):
    *      n1, n2, n3    Les trois notes
    */
   public static double resMoy(double n1,
                                double n2,
                                double n3 )
   {
      // Instructions
   }
}
```

Bloc d'instructions

Bloc d'instructions

Traduction de l'algorithme Principal à une méthode Java

- Les INTERMÉDIAIRES sont traduites à des VARAIBLES Java
  - Elles doivent être déclarées et données un type. Leur description sont traduit à des commentaires (ce qui forme le DICTIONNAIRE DE DONNÉES)
- Les invocations des algorithmes AfficheLigne(…) et Affiche(…) sont traduites à des invocations correspondantes de méthodes Java "System.out.println(…)" et "System.out.print(…)"
- Les invocations des algorithmes LireRéel(), LireEntier(), LireLigne() et LireBooléen() sont traduites à des invocations correspondantes des méthodes de la classe ITI1520 ou de la classe Scanner.
- Logique de base:
  - Obtenir des valeurs d'entrées de l'utilisateur
  - Invoquer l'algorithme de résolution de problème pour produire les résultats
  - Afficher les résultats.

Traduction des instructions à Java

- Instruction d'affectation
  - Algorithme: X ← expression
  - Java: `x = expression ;`

- Expressions

  Somme ← N1 + N2 + N3

```
somme = n1 + n2 + n3;
```

  hypoténuse ← $\sqrt{x^2 + y^2}$

```
hypoténuse = Math.sqrt(Math.pow(x,2) + Math.pow(y,2));
```

- Instruction d'invocation

  Moyenne ← RésMoy(Premier, Deuxième, Troisième)

```
moyenne = resMoy(premier, deuxieme, troisieme);
```

  Premier ← LireRéel()

```
premier = ITI1520.readDouble( );
```

Invocation en Java

Modèle logiciel: X ← Moy3(10, J, K)

Java: `x = moy3(10, J, K) ;`

- Ici, `moy3(10, J, K)` est une instruction d'*appel* ou d'*invocation* de méthode. Cette invocation représente la valeur retournée par la méthode, qui est affectée à une variable
- L'appel de méthode peut aussi être utilisé dans une expression d'un type approprié; la valeur de retour est utilisée pour évaluer l'expression.
  - Exemple:

    ```
    y = 10.2 * moy3( A, B, C ) + moy3( P, Q, -11 )
    ```
  - L'utilisation d'invocations multiples dans une même expression est difficile à tracer! À éviter si possible dans ce cours.

Lors de l'invocation …

1. L'exécution de la méthode invocatrice est suspendue.
2. Un espace mémoire est réservé pour les variables locales de types primitifs (`int`, `double`, `char`, `boolean`) dans la méthode invoquée.
3. Les valeurs initiales des paramètres de types primitifs sont copiées à partir des arguments de l'appel correspondants.
4. Les paramètres de type de référence sont associés aux tableaux ou objets auxquels se réfèrent les arguments correspondants.
5. L'exécution du module de la méthode commence.

- Quand le module de la méthode termine, la valeur de retour est retournée (s'il y a lieu) et la méthode invocatrice continue alors son exécution. Toutes les autres variables dans la méthode invoquée sont effacées et oubliées.

Exercice 3-3 - Traduction de l'algorithme Principal à une méthode Java

?

```
// PROGRAM Moyenne -lit 3 notes et calcule leur moyenne.
import java.util.Scanner;
class CalculeMoy
{
   public static void main (String[] args)
   {
      // MISE EN PLACE DE LA LECTURE AU CLAVIER
      Scanner clavier = new Scanner( System.in );
      // DÉCLARATIONS DES VARIABLES ET DICTIONNAIRE DE DONNÉES


      // LECTURE DES VALEURS DE L'UTILISATEUR




      // INVOCATION DE resMoy

      // AFFICHAGE DESRÉSUTATS ET MODIFIÉES À L'ÉCRAN

  }
  // la méthode résNotes() est placé ici
}
```

Traduction de l'algorithme de résolution de problème à une méthode Java (exemple)

- Les DONNÉES, RÉSULTATS et INTERMÉDIARES sont tous traduit à des VARIABLES de Java.
  - Elles doivent être déclarées et données un type. Leur description sont traduit à des commentaires (ce qui forme le DICTIONNAIRE DE DONNÉES)
  - Les valeurs initiales pour les DONNÉES sont reçues de `main` via l'invocation de la méthode.
- La valeur du résultat est retournée à la méthode main pour affichage.
  - IMPORTANT:
    - en Java, la valeur d'une variable est retournée avec l'instruction « `return(uneVariable);` ».
    - Cette instruction termine l'exécution de la méthode, et donc ne doit être placée à la fin de la méthode (dernière instruction).
    - Il est interdit de placé cette instruction à toute autre endroit.
  - Malgré que les algorithmes permettent le retour de plusieurs résultats, la méthode Java ne peut que retourné une valeur.
  - Pour le moment, nos algorithmes ne doivent retourner qu'une valeur.

Exercice 3-3 - Traduction de l'algorithme de résolution de problème à une méthode Java

```
// PROGRAM Moyenne -lit 3 notes et calcule leur moyenne.

class CalculeMoy
{
   // méthode main est placé içi
   // DONNÉES: n1, n2, n3    notes sur 25
   public static double résNotes(double n1, double n2, double n3)
   {
      //DÉCLARATIONS DES VARIABLES ET DICTIONNAIRE DE DONNÉES
      //INTERMÉDIAIRES


      // RÉSULTAT

      // MODULE DE L'ALGORITHME



      // RETOURNER RÉSULTAT

  }
}
```

"Lorsque vous arriverez au branchement sur la route, prenez-le."
- *Yogi Berra*

ITI 1520
Section 5: Branchements

Objectifs:

- Diagrammes pour modèles logiciels
- Instruction de branchement
- Traçage d'un branchement et traduction en Java
- Expressions Booléennes complexes

Instruction de branchement

- Jusqu'ici, les modules de nos algorithmes ont contenu:
  - une instruction simple
  - une séquence d'instructions simples

- Nous avons souvent besoin de structures plus complexes dans nos solutions, par exemple lorsque des calculs différents dépendent de certaines conditions.
  - Exemple: trouver la valeur absolue d'un nombre X

- → Instruction de branchement (condition)!

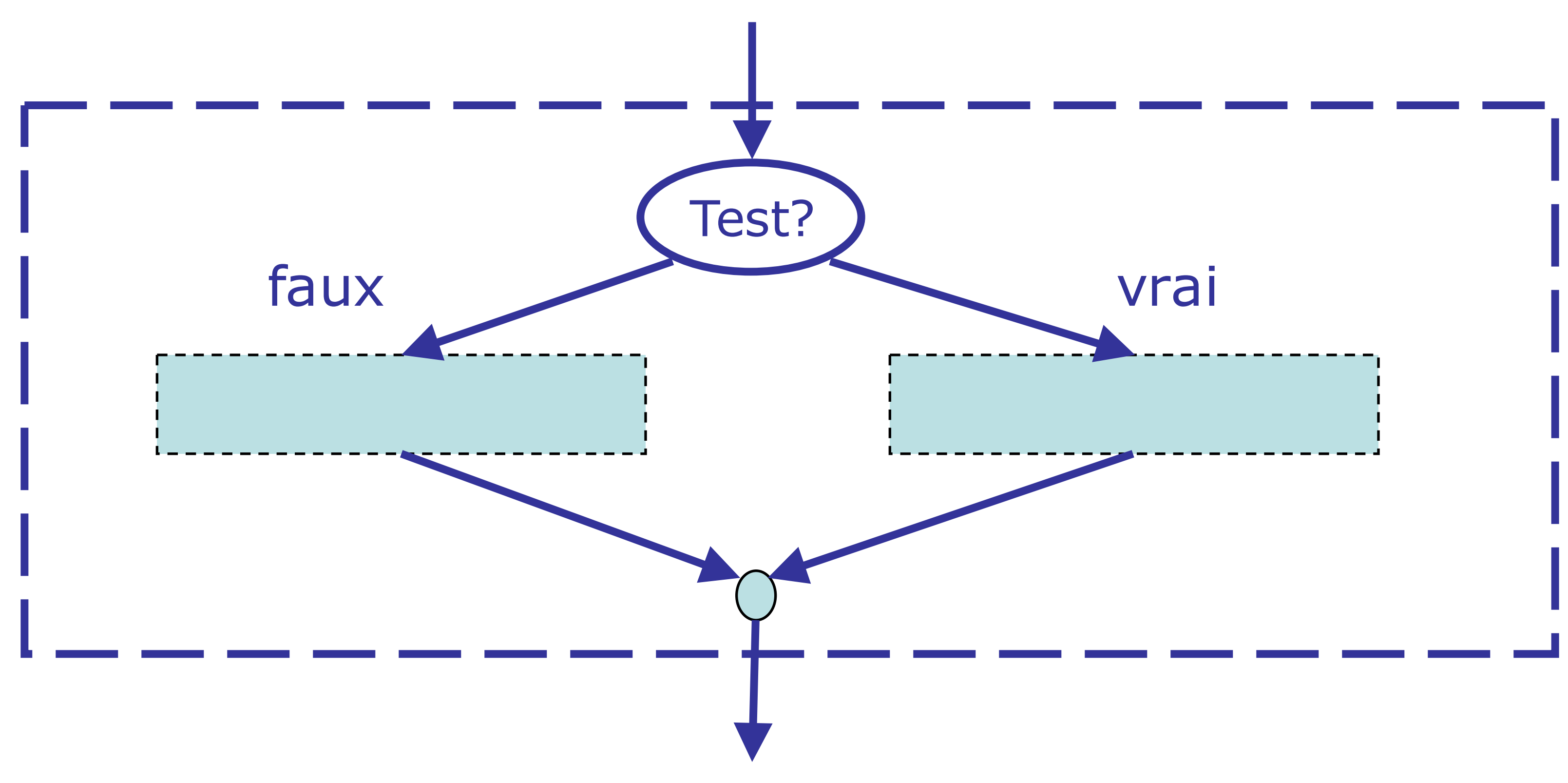
Problème: Nombre le plus grand

- Écrivez un algorithme qui retourne la plus grande valeur entre deux nombres donnés.

DONNÉES:	X, Y	*(deux nombres)*
RÉSULTAT:	M	*(maximum entre X et Y)*
EN-TÊTE:	M ← Max2(X, Y)	

- Nous représenterons la solution avec une instruction de branchement

Instruction de branchement



- Les boîtes pointillées sont des BLOCS D'INSTRUCTIONS.
- Notez que l'instruction de branchement est complexe et contient plusieurs autres instructions dans les blocs d'instructions
  - Représentation graphique des modèles logiciels!

Diagrammes de modèles logiciels

- Permet une description visuelle des algorithmes.
- Composé de nœuds connectés par des flèches.

- Nœud de test:
  - Représente la vérification d'une condition (expression Booléenne, avec point d'interrogation).

Test?

- Nœud de bloc d'instruction:
  - Indique où un autre bloc d'instruction peut être inséré. Ce bloc peut contenir des instructions simples ou complexes (et même aucune instruction)

Bloc

Contenu d'un bloc d'instruction

- Instruction simple (invocation, affectation)
- Instruction vide (Ø = "ne rien faire")
- Instruction de branchement
- Instruction de répétition/boucle (à venir…)

- Important: Chaque bloc a exactement une entrée (une flèche entrante) et une sortie (une flèche sortante).

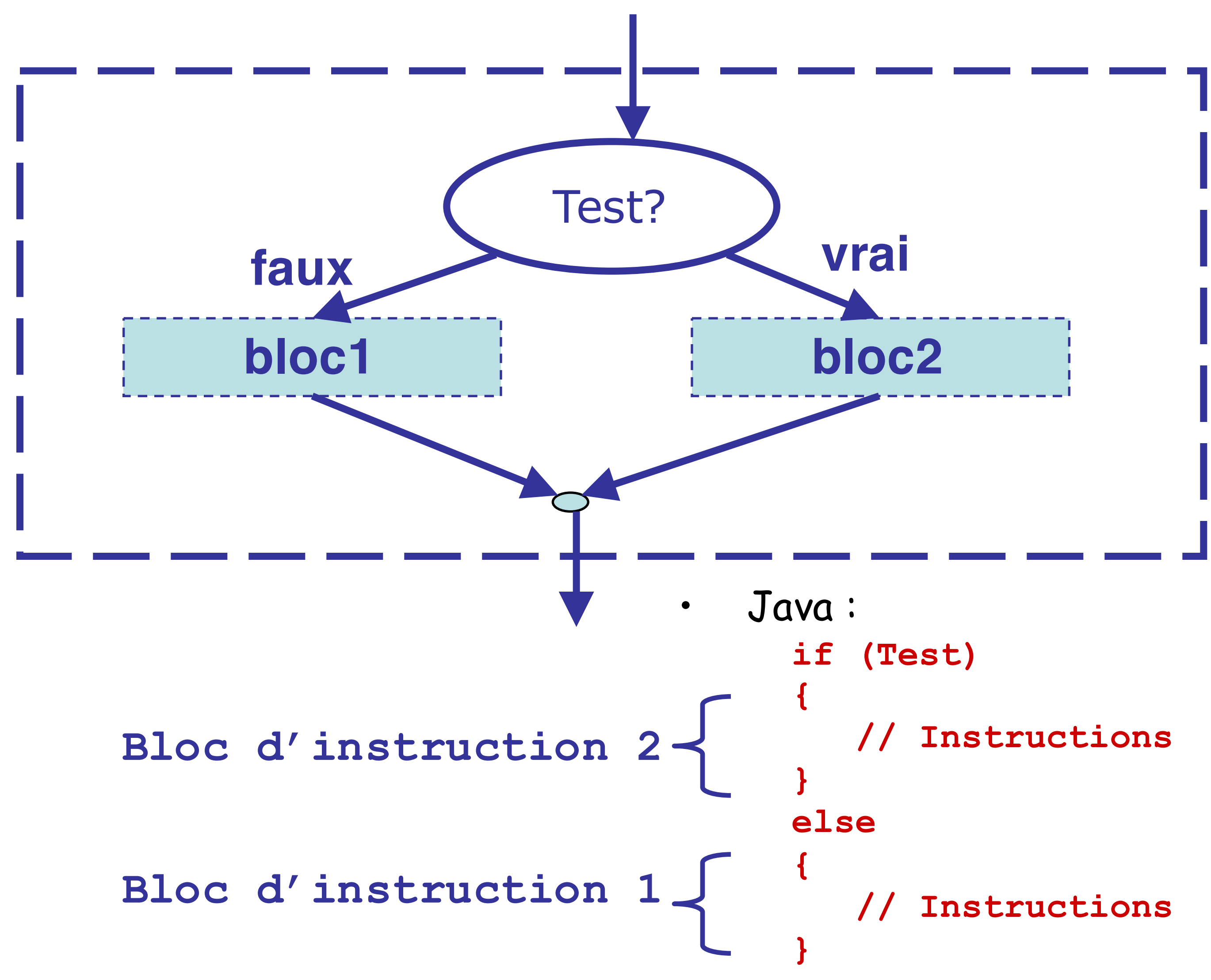
Exercice 5-1 - De retour au problème du plus grand nombre

- Comment écrire l'algorithme trouvant le maximum (M) entre deux valeur (X et Y) dans notre modèle logiciel?

Exercice 5-2 - Maximum entre 3 nombres ?

- Trouvez le maximum entre trois nombres donnés X, Y, et Z.
  - Version 1: avec tests imbriquées
  - Version 2: séquence de tests

Traduction de branchements au Java



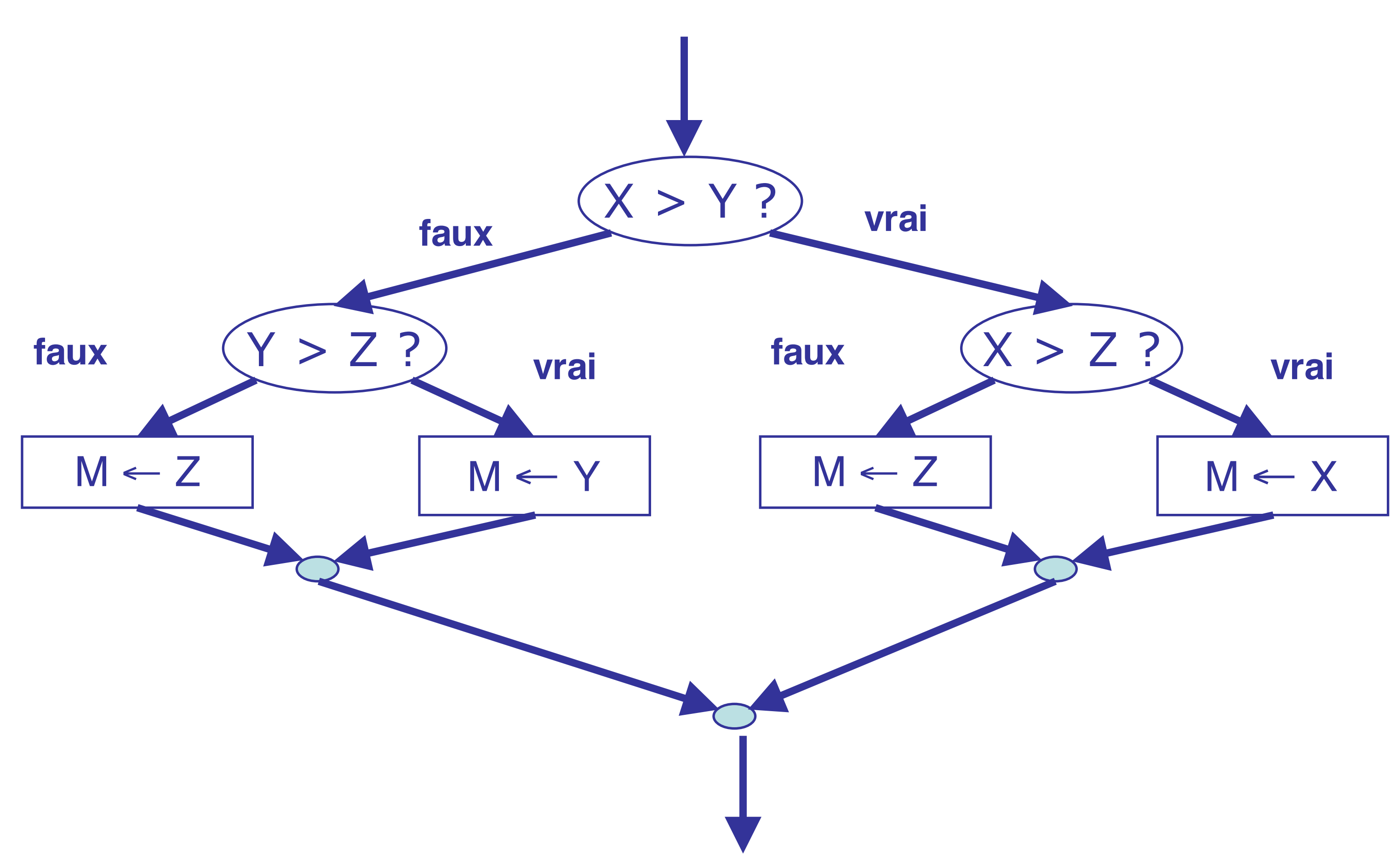
Exercice 5-3 – Traduction de branchements 1

?

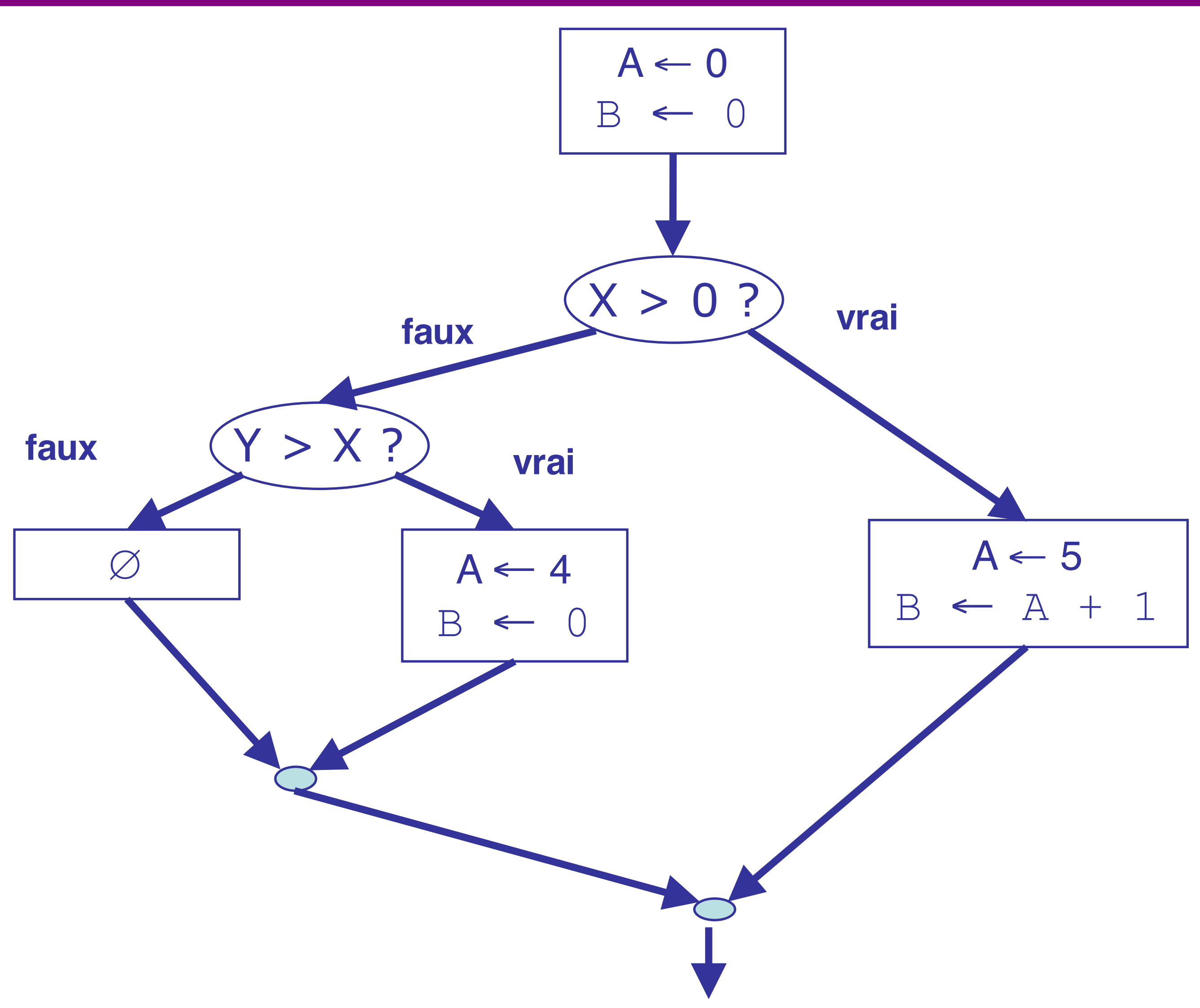
Données:	X, Y, Z	*(trois nombres)*
Résultat:	M	(*la plus grande des données*)
En-tête:	M←Max3(X, Y, Z)	

- Deux solutions:
  - séquence d'instructions de branchement
  - instructions de branchement imbriquées
- Traduisez-les en Java:

Solution avec branchements imbriqués



Exemple d'un bloc d'instruction avec instruction de branchement



Exercice 5-7 - Billet de cinéma

- Écrivez un algorithme pour calculer le prix d'un billet pour une personne lorsque le coût est de: 7$ pour les personnes de 16 ans ou moins, 5$ pour celles de 65 ans ou plus, et 10$ pour les autres
  - Version 1: tests imbriqués
  - Version 2: séquence de tests

Variables Booléennes

- Une variable Booléenne ne peut avoir que 2 valeurs possibles: VRAI ou FAUX
  - En réalité représenté par deux valeurs (ex: 0 et 1) mais dans le langage de programmation seulement ces deux mots clefs sont permis!

- Les affectations de valeurs peuvent être utilisées

  X ← VRAI

  Y ← FAUX

- Le résultat d'un test (expression Booléenne) peut aussi être affecté à une variable Booléenne:

  X ← (A < 0)

Exercice 5-8 - Valeur positive

- Écrivez un algorithme qui vérifie si un nombre donné X est strictement positif.

DONNÉE:
RÉSULTAT:

EN-TÊTE:

MODULE

Expressions Booléennes composées

- Une expression Booléenne composée (aussi appelée condition multiple) contient deux ou plusieurs expressions Booléennes simples connectées par des opérateurs logiques (ET/*AND* et OU/*OR*).

- Exercice 5-9 - Construisez une expression Booléenne qui retourne vrai si un âge donné est entre 16 et 65 (16 et 65 exclus) et faux autrement.

Tables de vérité

?

- Une table de vérité pour une expression Booléenne composée montre les résultats pour toute les combinaisons de valeurs possibles:

X	Y	X ET Y	X OU Y
VRAI	VRAI		
VRAI	FAUX		
FAUX	VRAI		
FAUX	FAUX		

Opérateur NON (*NOT*)

X	NON X
VRAI	FAUX
FAUX	VRAI

- NON est un opérateur servant à trouver le complément d'une valeur simple ou d'une expression Booléenne complexe:
- Exemple. Si Âge = 15, alors:
  - L'expression Âge > 16 sera évaluée à FAUX, et NON (Âge > 16) aura comme valeur VRAI.
  - L'expression Âge < 65 sera évaluée à VRAI, et NON (Âge < 65) aura comme valeur FAUX.

Exercice 5-10 - Encore des expressions Booléennes complexes

Supposons que X = 5 et que Y = 10.

Expression	Valeur
(X > 0) ET (NON Y = 0)	
(X > 0) ET ((X < Y) OU (Y = 0)	
(NON X > 0) OU ((X < Y) ET (Y = 0))	
NON ((X > 0) OU ((X < Y) ET (Y = 0)))	

Expressions dans les tests

- Le TEST dans un branchement ou une boucle peut être n'importe quelle expression Booléenne:
  - Variable Booléenne
  - Négation d'une variable Booléenne
    - NON (Java: `!`)
  - Comparaison entre deux valeurs de types compatibles
    - Opérateurs Java: `==` `!=` `<` `>` `<=` `>=`
    - Les données comparées ne sont pas nécessairement des `boolean`, mais le résultat de la comparaison est `boolean`
  - Expressions Booléennes composées
    - ET (Java: `&&`)
    - OU (Java: `||`)
- ATTENTION
  - Ne pas confondre `=` avec `==`
  - Ne pas confondre ET avec OU
- ex: tester si `X` est dans l'intervalle 12..20:

```
X >= 12 && X <= 20
```

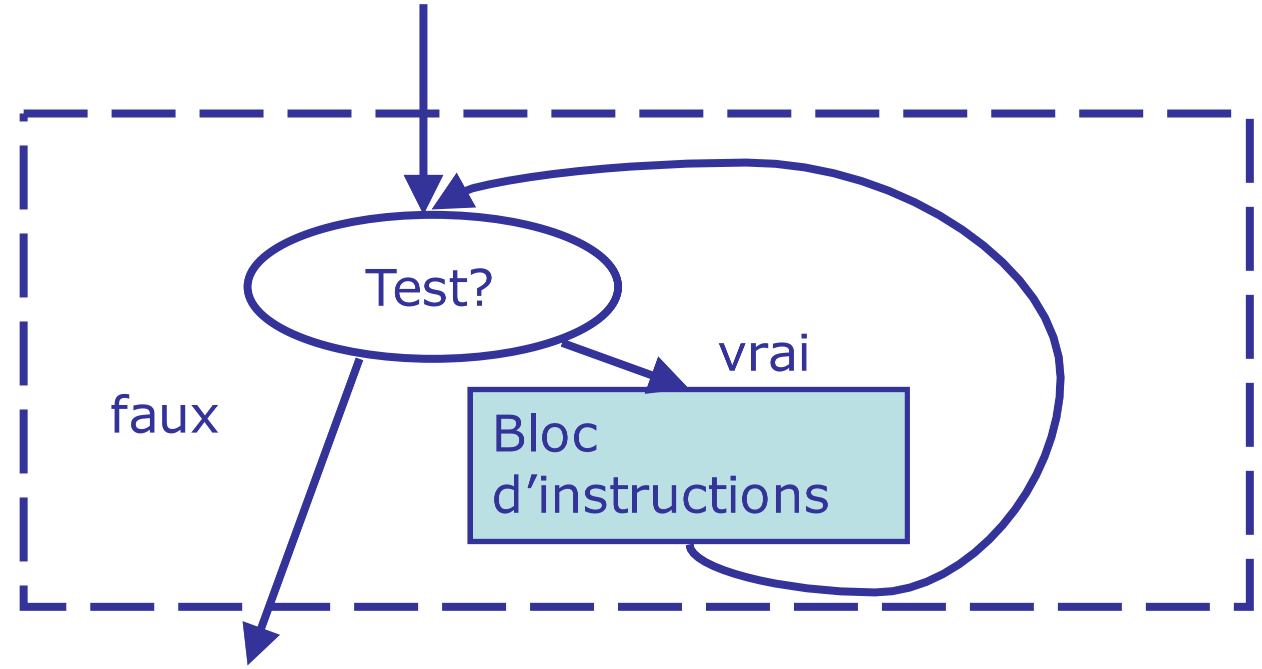
"Un programme sans boucle… ne vaut pas la peine d'être écrit."
- *A. Perlis*

ITI 1520
Section 6: Tableaux et boucles

Objectifs:
- Instructions de boucles
- Tableaux de variables
- Traduction vers Java
- Traçage de tableaux et de boucles
- Chaîne de type *String* en Java
- Plusieurs exemples!

Instruction de boucle

- Nous avons parfois besoin de répéter un bloc d'instruction. Dans un modèle logiciel, nous utilisons une instruction de boucle:



- Le bloc d'instruction dans la boucle est répété jusqu'à ce que le test devienne faux.

Conception d'une instruction de boucle

1. Initialisation
   - Y a-t-il des variables à initialiser?
   - Ces variables seront mises à jour dans la boucle.
2. Condition à tester
   - Une condition pour déterminer s'il faut répéter le bloc d'instruction de la boucle ou non
3. Bloc d'instruction de la boucle
   - Quelles sont les étapes à répéter?
   - Boucle déterminé – le nombre de fois que la boucle est répétée est connu – utilise normalement une variable compteur
   - Boucle indéterminé – le nombre de fois que la boucle est répétée n'est pas connu – utilise normalement une variable drapeau

Exercice 6-1: Somme de 1 à N

DONNÉE:
INTERMÉDIAIRE:
RÉSULTAT:
EN-TÊTE:
MODULE:

Exercice 6-2: Boucle “déterminée”

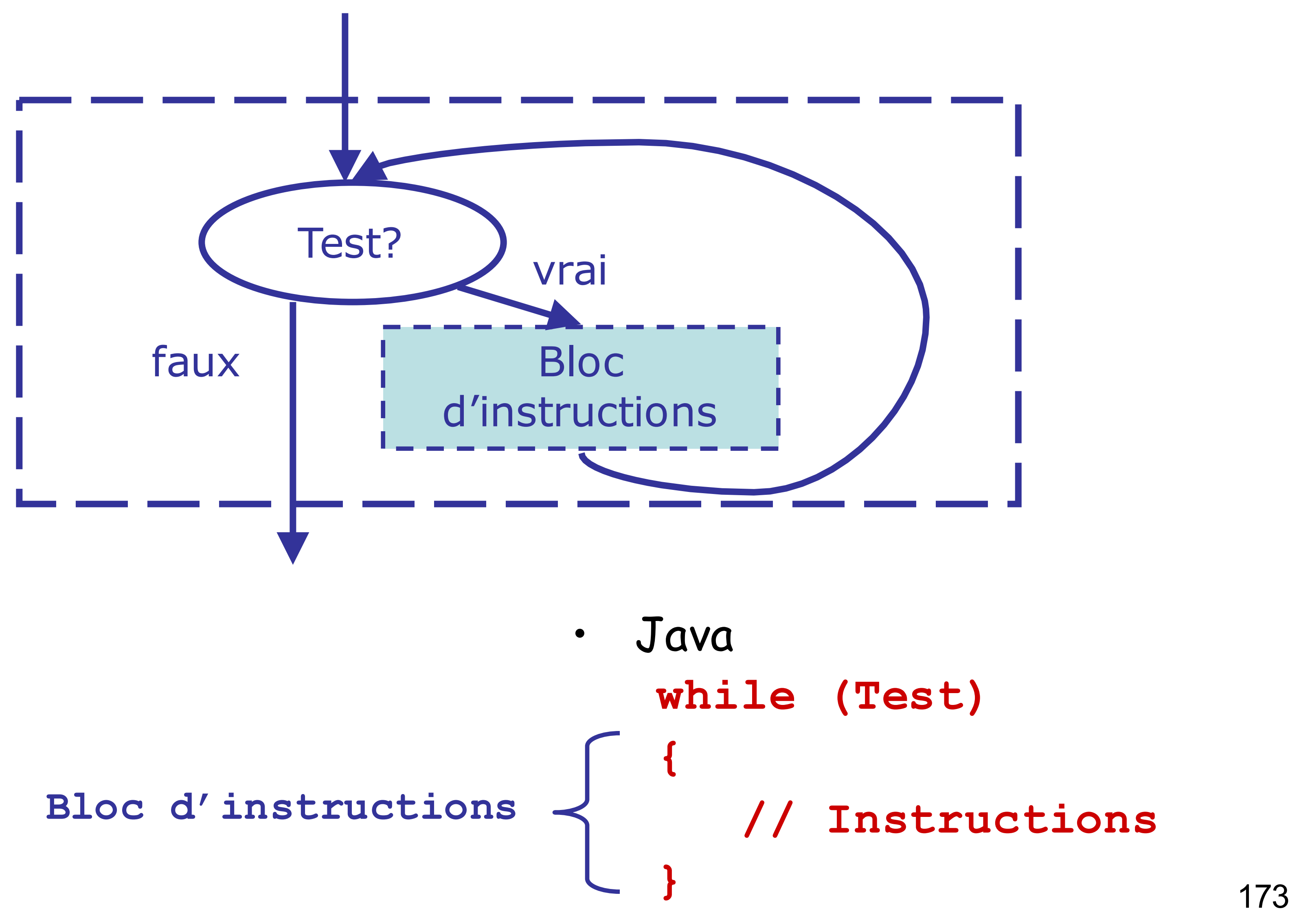
- Développez un algorithme pour trouver le factoriel d’un entier N, représenté par N!
- Définition du factoriel (produit de 1, 2, 3 jusqu’à N):

  $N! = 1 \times 2 \times \ldots \times N$

Exercice 6-3: Boucle “indéterminée”

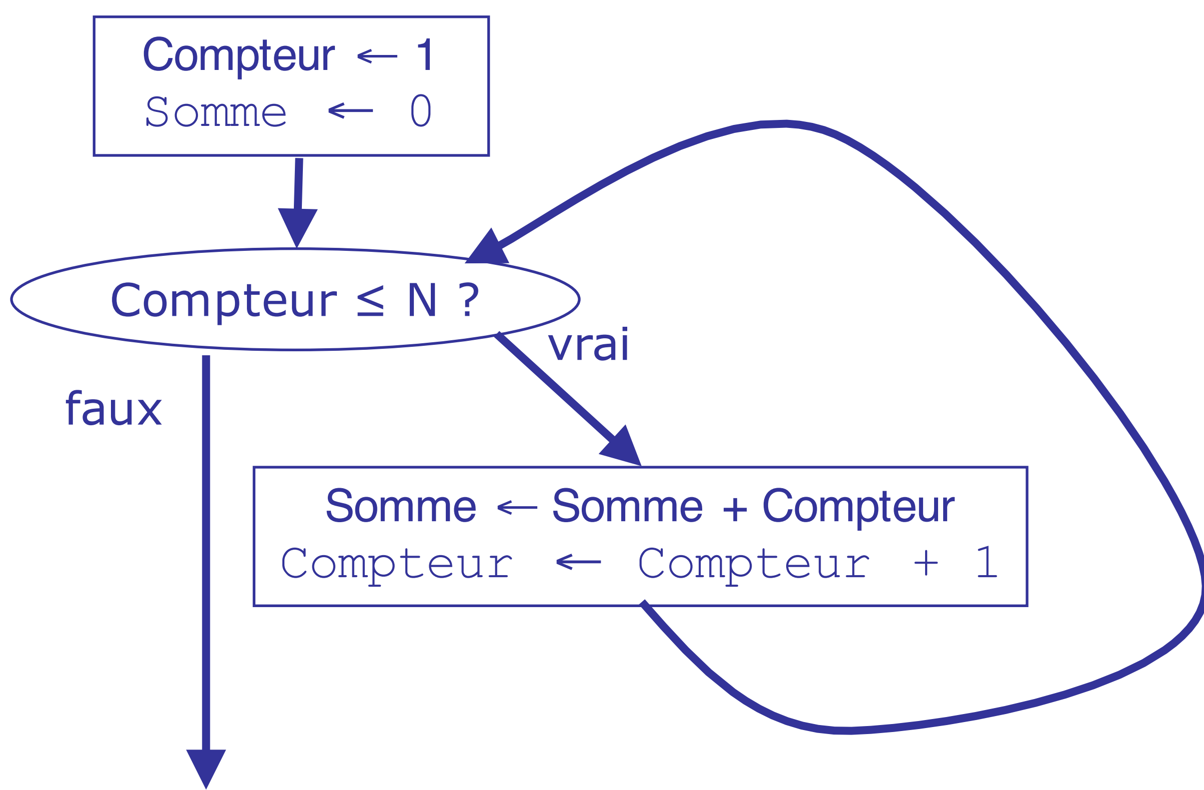
- Développez un algorithme pour déterminer le nombre de fois qu’un entier UnNombre peut être divisé par un autre entier Diviseur, jusqu’à ce que le résultat est plus petit que Diviseur
  - Ceci donne la partie entière de la fonction logarithmique.

Traduction de boucles au Java



Algorithme: Somme de 1 à N

DONNÉE: N *(un entier positif)*
INTERMÉDIAIRE: Compteur *(index allant de 1 à N)*
RÉSULTAT: Somme *(somme des entiers de 1 à N)*
EN-TÊTE: Somme ← Somme1àN(N)
MODULE:



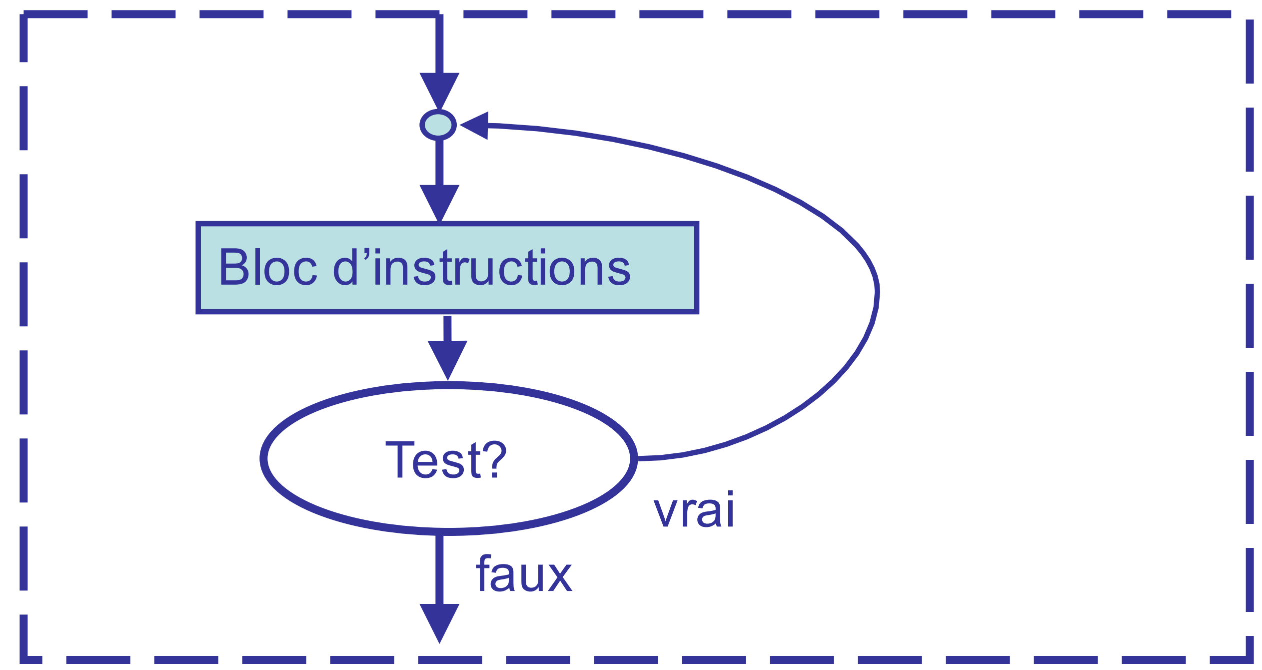
Exercice 6-4: Traduire la boucle au Java

```
import java.io.* ;

class [          ]
{
   public static void main (String args[ ])
   {
   [                                        ]
   }
}
```

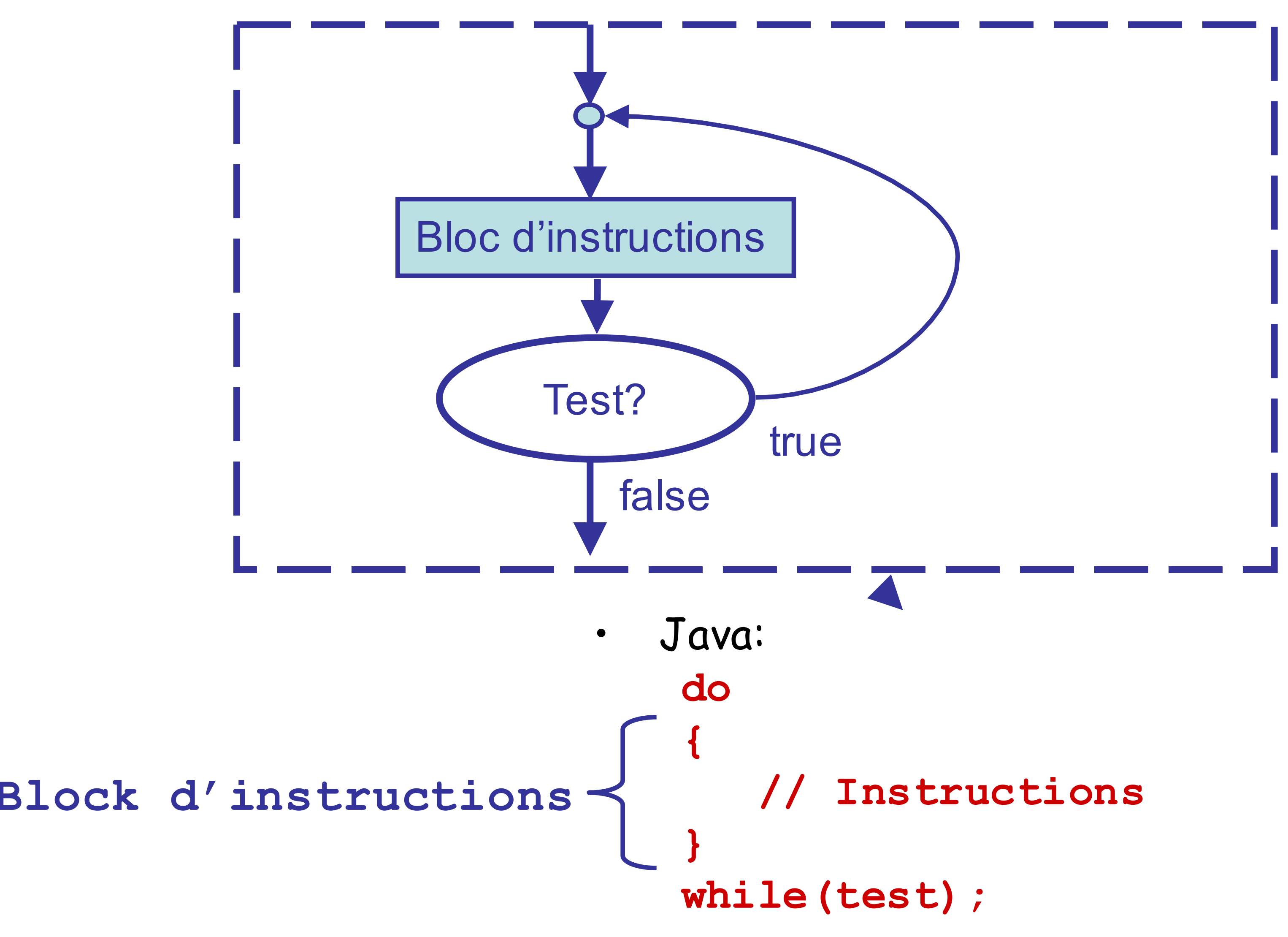
Instruction de boucle post-test

- Dans l'instruction de boucle post-test, le test se fait après l'exécution du bloc d'instructions.



- Le bloc d'instruction de la boucle est exécute au moins une fois et répété jusqu'à ce que le test devienne faux.

L'instruction de boucle post-test
Traduction au Java



Exercice 6-5: Exemple de boucle post-test ?

- Utilisez une boucle post-test pour développer un algorithme « principal » pour calculer le factoriel et traduire en Java

DONNÉES:

RÉSULTATS:

INTERMÉDIAIRES:

CONSTRAINTES:

EN-TÊTE:

MODULE:

Exercice 6-5: Traduire au Java

```
public static void main(String args)
{
   // Variables


   // Module







}
```

Boucle FOR

- Java offre une autre forme de boucle, qui peut remplacer une boucle WHILE lorsque nous savons à l'avance combien de fois une boucle sera exécutée (boucle déterminée).

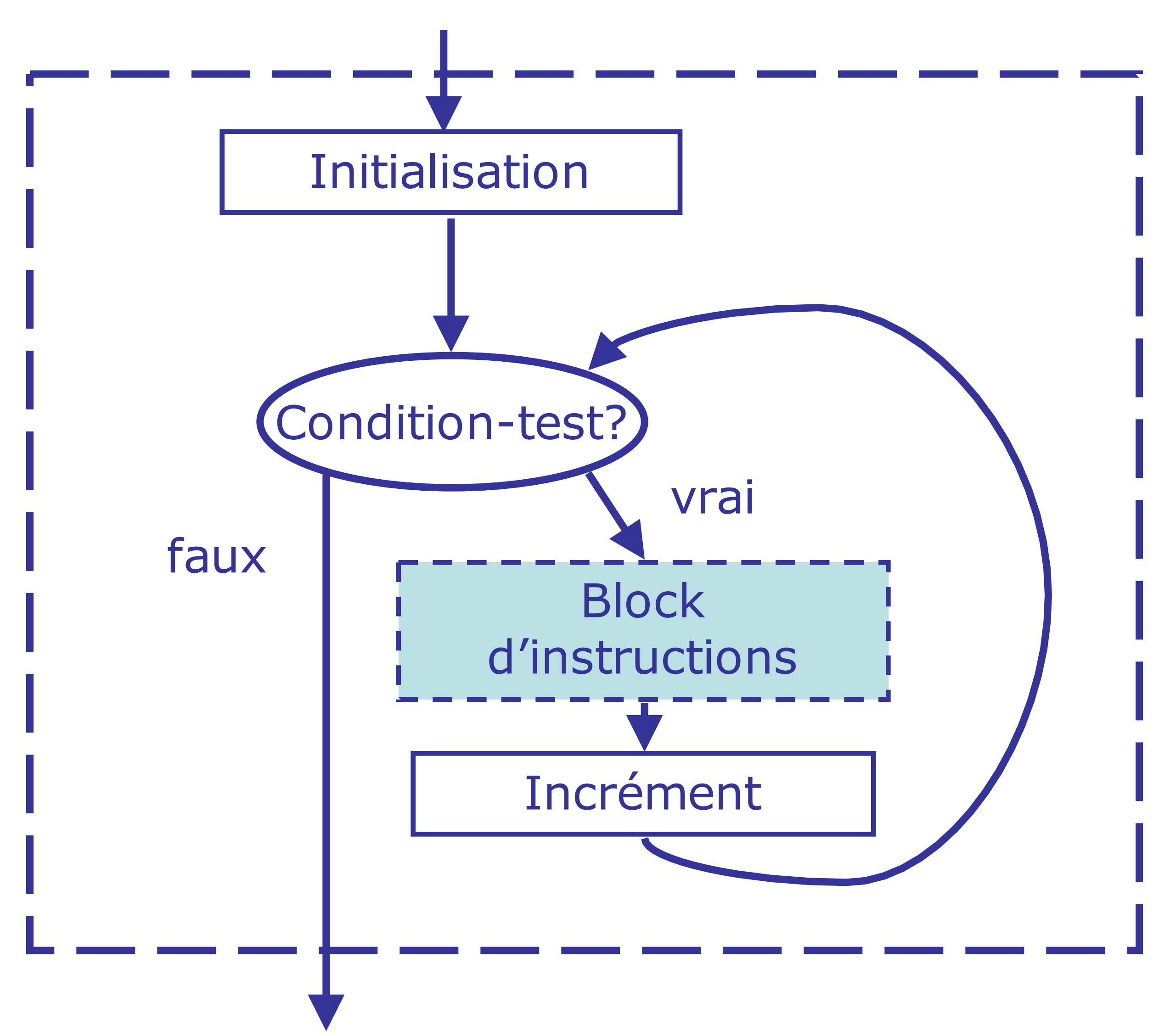
- La boucle FOR a comme format:

```
for (<initialisation>; <condition-test>; <incrément>)
{
    // instructions
}
```

- Dans la plupart des cas:
  - **Initialisation** initialise un compteur
  - **condition-test** teste si un compteur a atteint une certaine limite
  - **Incrément** incrémente le compteur (+ 1, - 1, + 5, + Var, …)

- Toute boucle FOR peut être reformulée en boucle WHILE.

Diagramme de la boucle FOR



Exercice 6-6: Somme1àN avec boucle FOR

Somme ← 0

Compteur ← 1

Compteur ≤ N ?

vrai

Somme ← Somme + Compteur

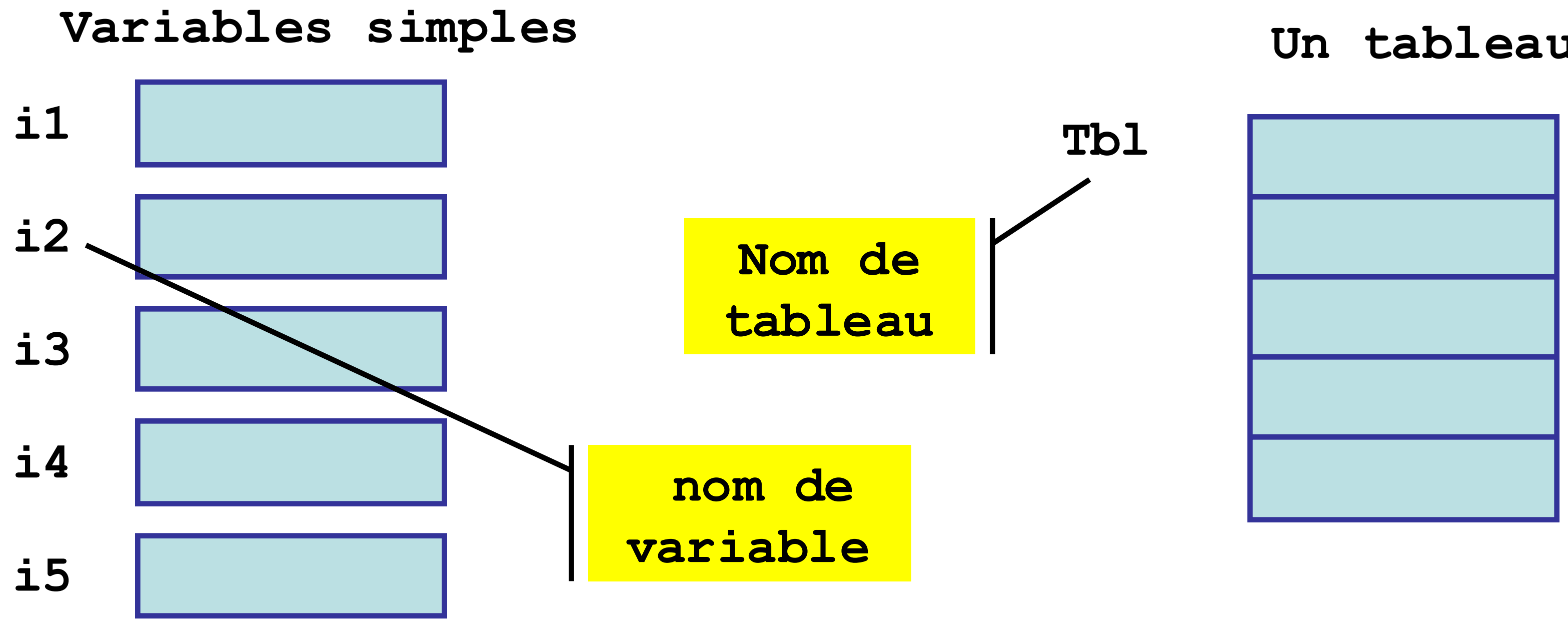
Compteur ← Compteur + 1

faux

- Traduction à Java:

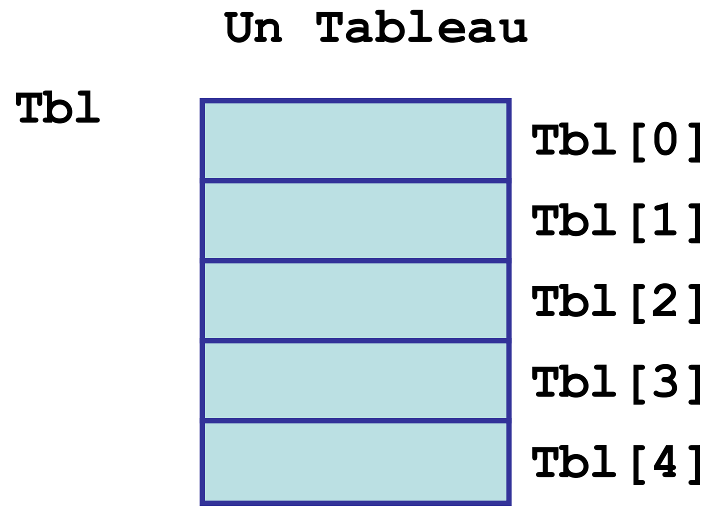
Tableaux informatiques

- "Simples" variables ne contiennent qu'une valeur.
- Un tableau informatique (computer array en anglais) est composé de plusieurs positions, chacune capable de contenir une valeur.
- Le tableau est essentiellement une collection de variables du même type.



Tableaux informatiques (suite)

- Si un tableau Tbl a 5 positions, nous pouvons les accéder en utilisant les entiers 0-5 comme indices (ou index).
  - Ex: Tbl[2] est la troisième position avec index 2.
  - Notez que Tbl[2] est équivalent à un nom de variable est peut être utilisé à tout endroit où un nom de variable est utilisé, ex: dans des expressions.

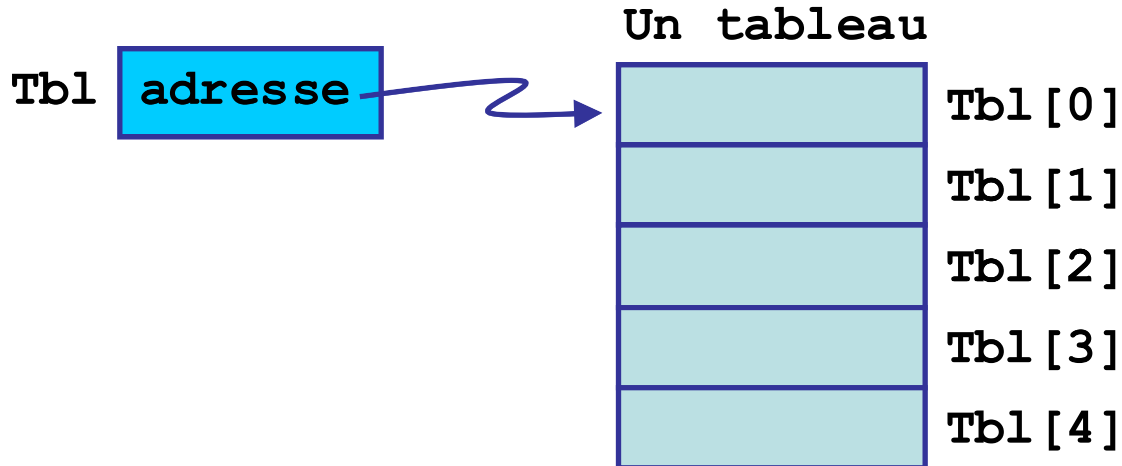


Tableaux informatiques (*suite*)

- Deux informations peuvent être nécessaires pour composer avec les tableaux:
  - La première information est la longueur du tableau, i.e. le nombre de positions disponibles.
  - La deuxième information est le nombre de positions qui contiennent des « valeurs », i.e. le positions qui ont été initialisées.
  - Quand nous passons des tableaux à des algorithmes, on peut y passer une ou les deux valeurs
  - Ex: si les valeurs sont dans les 3 premières positions, nous pourrions passer 3 à la DONNÉE LongueurT
  - Ex: si l'algorithme a besoin de voir toutes les 5 positions, alors la DONNÉE LongueurT recevra la valeur 5.
  - Quels seraient les indices valides?

Le nom du tableau

- Que représente le nom d'un tableau?
- Il peut représenter l'adresse en mémoire où se trouve le tableaux.
  - Ceci est semblable à un nom de variable, mais se fait traité différemment d'un nom de variable
  - Cette approche est utilisée dans des langages tel que le C
  - L'image dans les acétates précédentes reflète cette représentation.
- Le nom de tableau peux aussi être le nom d'une variable de référence – et donc notre image devient:



- Ceci sera le modèle de nos tableaux.

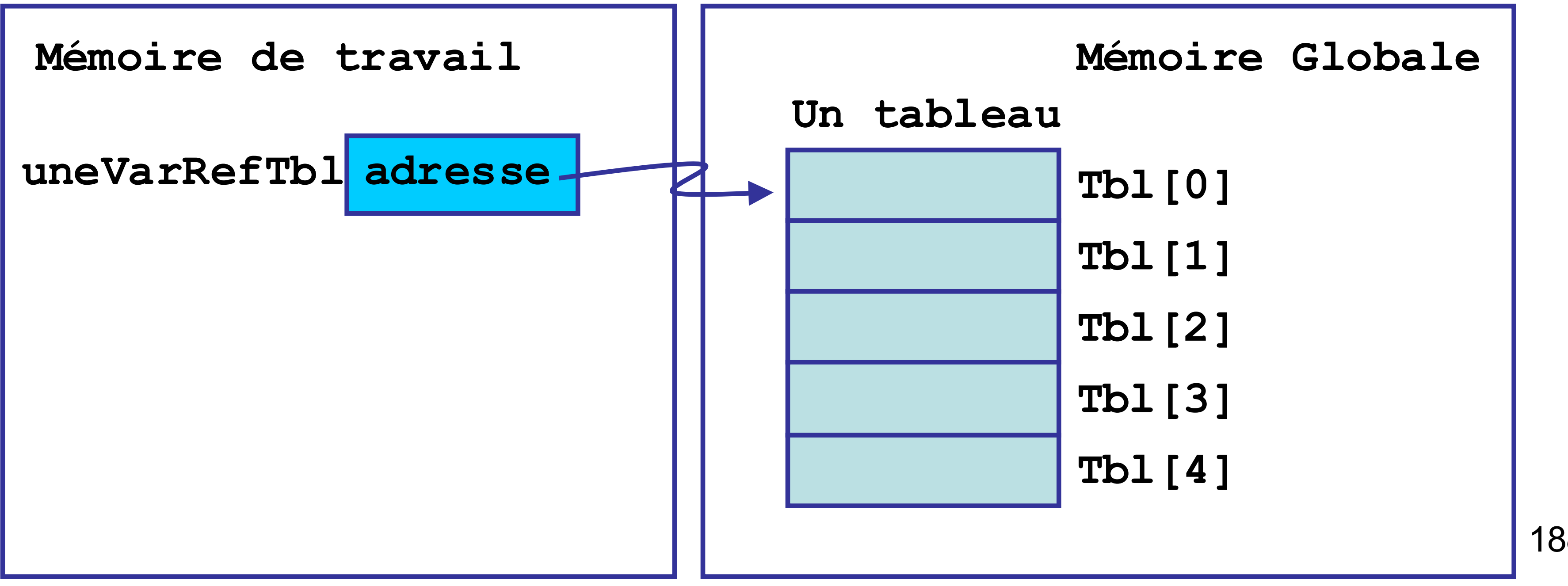
La création de tableaux

- L'algorithme "standard" suivant sera utilisé pour créer des tableaux

  uneVarRefTbl ← CréerTableau(L)

  - Crée un tableau ayant L positions ayant des valeurs inconnues.
  - uneVarRefTbl est une variable de référence à laquelle est affectée l'adresse du tableau.

  E.g. Tbl ← CréerTableau(5)



Exercice 6-7: Utilisation des indices

- L'indice d'un tableau T de taille L peut être n'importe quelle expression mathématique retournant un entier entre 0 et L-1.

> "L'indice d'un tableau devrait-il commencer à 0 ou à 1? Mon compromis De 0,5 a été rejeté, à mon avis, sans considération suffisante."
> -- *Stan Kelly-Bootle*

  – Soit K =2, et T réfère à

2	-1	5	7

(de longueur 4)

T[2] =
T[K] =
T[2*K-1] =
T[T[0]+1] =

- T[expression] peut être utilisé comme toute autre variable dans des expressions plus complexes.
- Rappelez-vous T est une variable de référence

Exercice 6-8: Valeur au milieu d'un tableau

- Écrivez un algorithme qui retourne la valeur au milieu d'un tableau T contenant N nombres (avec N impair).
- Notez que la DONNÉE T reçoit la référence (adresse) au tableau.

DONNÉES:

RÉSULTAT:

INTERMÉDIAIRES:

EN-TÊTE:

MODULE:

Exercice 6-9: Échanger les valeurs d'un tableau

- Écrivez un algorithme qui échange les valeurs aux positions I et J d'un tableau T.
- Ceci est possible car l'adresse du tableau est passée à l'algorithme (sous-programme).

DONNÉES:

MODIFIÉES:
INTERMÉDIAIRES:

RÉSULTAT:
EN-TÊTE:
MODULE:

Exercice 6-10: Création d'un nouveau tableau

- Créez un tableau contenant les entiers de 1 à N en ordre inverse.
- Rappelez-vous de l'algorithme standard

  uneVarRefTbl ← CréerTableau(L)

  qui crée un tableau référé par uneVarRefTbl; le tableau a L positions (sans valeurs).

Exercices de boucles (I)

6-1 Trouvez la somme des nombres de 1 à N (1+2+…+N).

? 6-11 Trouvez la somme des valeurs d'un tableau de N valeurs.

6-12 Soit une valeur V et un tableau X de N valeurs, vérifiez si la somme des valeurs de X excède V.

? a) Utilisez l'algorithme de l'exercice 6-11.

? b) Faites une version efficace qui sort de la boucle aussitôt que la somme excède V.

? 6-13 Comptez combien de fois K apparaît dans un tableau de taille N.

6-14 Soit un tableau T de N valeurs et un nombre K, vérifiez si K apparaît dans T ou non.

? a) Utilisez l'algorithme de l'exemple 4.

? b) Faites une version efficace qui sort de la boucle aussitôt que K est trouvé.

Exercices de boucles (II)

? 6-15 Soit un tableau X de N valeurs et un nombre K, trouvez la position de la première occurrence de K. (Si K n'est pas dans X, retournez –1 comme position.)

? 6-16 Trouvez la valeur maximale dans un tableau de taille N.

6-17 Trouvez la position de la première occurrence de la valeur maximale d'un tableau de taille N.

? a) Utilisez l'algorithme de l'exemple 7.

? b) Utilisez des algorithmes de n'importe quels exemples.

? c) Version avec une boucle sans autres algorithmes.

6-18 Vérifiez si un tableau de N valeurs contient ou non des valeurs dupliquées.

? – Stratégies?

Tableaux Java

- Une variable tableau est déclarée avec le type de ses éléments (membres)
  - Par exemple, voici une déclaration d'une variable tableau dont les éléments sont des nombres réels:

```
double [] unTableau; // forme préférée
```

ou encore:

```
double unTableau[];
```

- Lorsqu'une variable tableau est déclarée, le tableau lui-même n'est PAS créé. Nous avons une variable de référence qui peut pointer au tableau.
  - **unTableau** contient la valeur spéciale **null** jusqu'à ce qu'une référence valide y est affectée.

Création d'un tableau

- Comment traduire l'algorithme CréerTableau?
- Pour créer un tableau en Java, l'opérateur **new** est utilisé.
- Nous devons fournir le nombre d'éléments du tableau, de même que leur type (qui est le même pour tous les éléments):
  e.g. **new double[5]**
  - Le nombre et le type ne peuvent plus être changés par la suite.
  - L'opérateur `new` retourne un référence (adresse) qui peut être affecté à une variable de référence, par exemple:

```
double[] unTableau;
unTableau = new double[5];
```

- Notez bien: La création d'un tableau initialise tous les éléments avec des zéros (ce qui se traduit à **0**, **null** , ou **'\0'** dépendant du type utilisé).
- La taille d'un tableau peut être obtenue à l'aide de l'attribut .length: **unTableau.length**
- Les tableaux sont créés en mémoire globale.
- Les variables de références sont crées dans la mémoire de travail.

Mémoire pour tableaux

```
double[] unTableau ;
```

unTableau: `null`

> "Thou shalt not follow
> the NULL pointer,
> for chaos and madness
> await thee at its end."
> - H. Spencer

```
unTableau = new double[3]
```

unTableau: `adresse` →

0	1	2
?	?	?

`length` 3

Accès aux éléments d'un tableau

- Les éléments d'un tableau sont accédés à l'aide d'indices en utilisant l'opérateur `[]`. Les indices sont des entiers débutant à 0.
- Par exemple, si `unTableau` est un tableau de trois entiers, alors :
  - Le premier élément est `unTableau[0]`,
  - Le deuxième élément est `unTableau[1]`,
  - Le troisième élément est `unTableau[2]`.
- Un index peut être représenté par n'importe quelle expression qui retourne un entier.
- Si un index est trop petit (< `0`) ou trop grand (> `length-1`) alors une erreur d'exécution surviendra.

Initialisation de tableaux

- Les éléments d'un tableau peuvent être initialisés individuellement:

```
int [] intTableau = new int[3];
intTableau[0] = 3;
intTableau[1] = 5;
intTableau[2] = 4;
```

- … ou lors de la création du tableau:

```
int [] intTableau;
intTableau = new int [] { 3, 5, 4 };
```

- … aussi possible, mais déconseillé (*new* implicite):

```
int [] intTableau = { 3, 5, 4 };
```

Initialisation partielle d'un tableau

- Un tableau peut être initialisé partiellement.

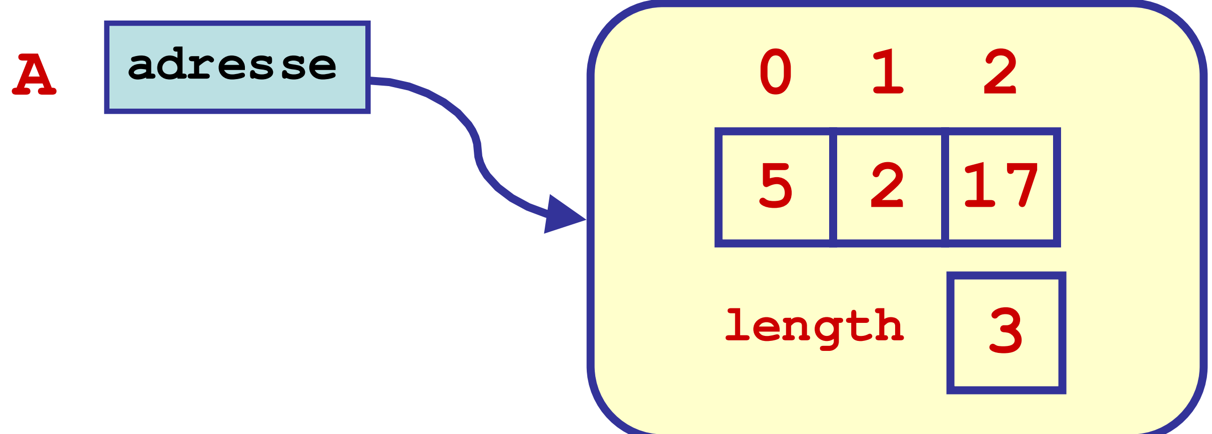
```
int [] intTableau = new int [5];
intTableau[0] = 3;
intTableau[1] = 5;
intTableau[2] = 4;
```

  - Dans ce cas, `intTableau[3]` et `intTableau[4]` valent 0. Dans des langages autres que Java, ces valeurs peuvent être indéfinies et ainsi causer des problèmes lorsqu'elles sont utilisées en lecture.

- Quand un tableau est utilisé, il est parfois utile d'avoir une autre variable (ou plusieurs) qui permettra de savoir quelles sont les indices pour lesquelles une valeur a été affectée.

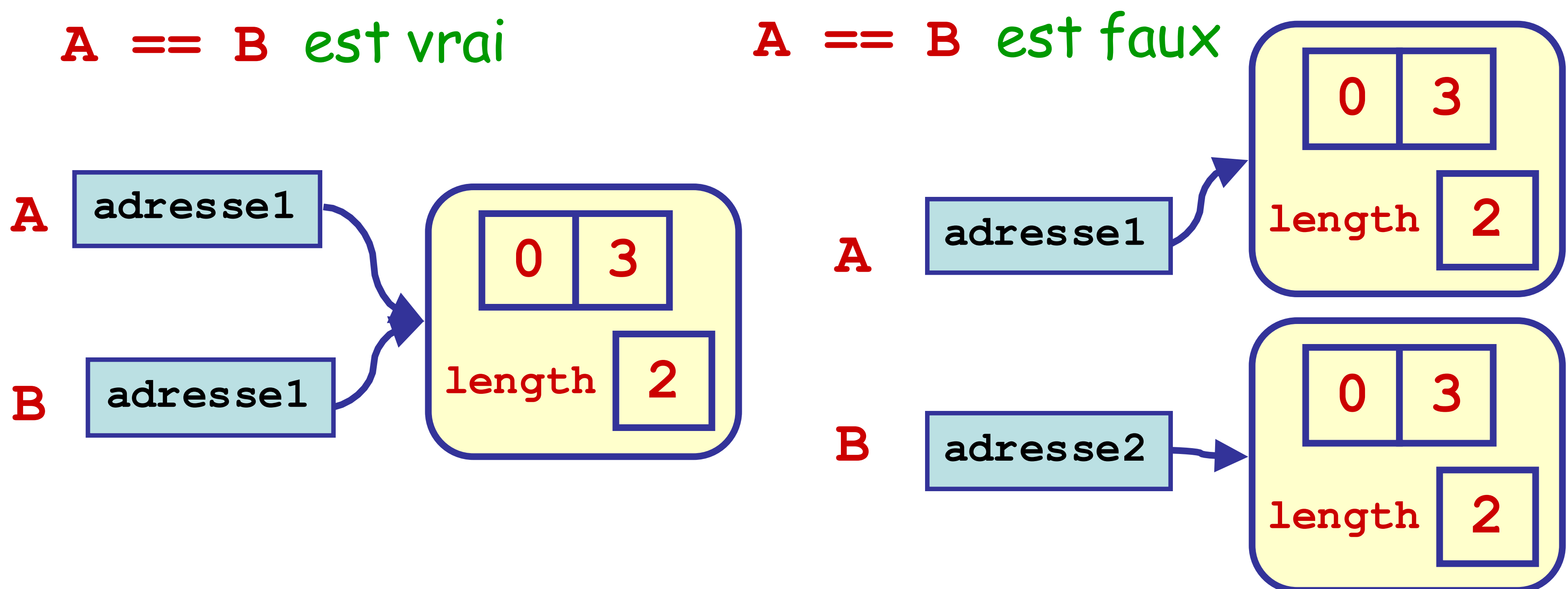
Types de référence

- Un type tableau est un type de référence, à cause du « *pointeur* » vers le tableau.
- Il est important de distinguer la référence (pointeur) de l'item référencé.
  - Dans le diagramme suivant, **A** est la référence, et le tableau est ce qui est référencé.
    - Java ne nous permet pas de regarder la valeur de **A** pour voir le pointeur.



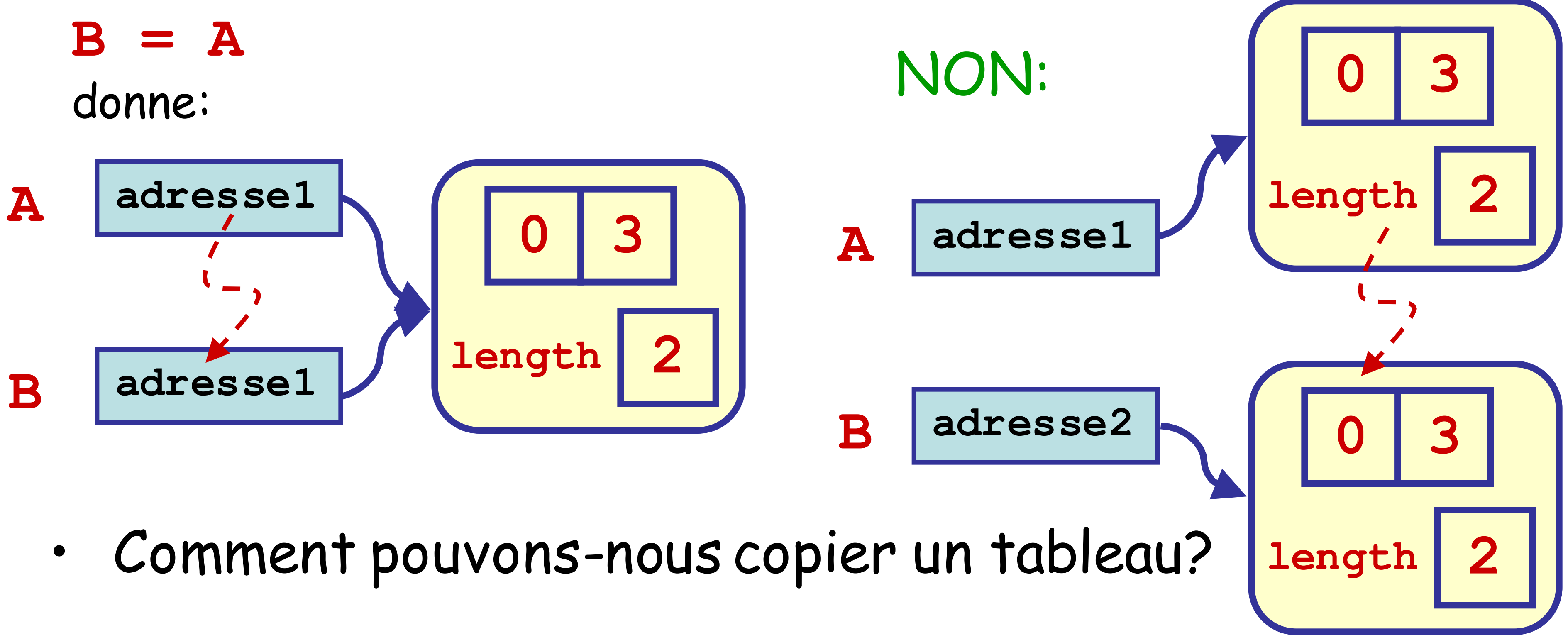
Types de référence

- Que dire des affectations et des comparaisons pour les types de références?
  - Ce sont les références qui sont comparées ou affectées, pas les tableaux!.



Types de référence

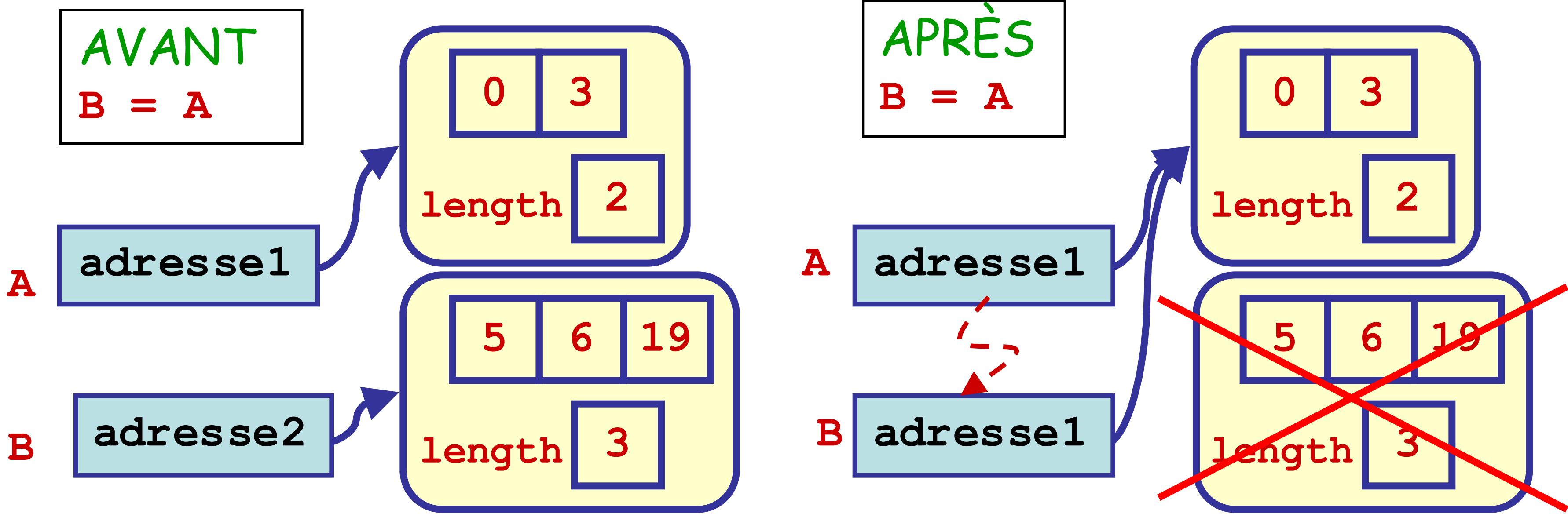
- L'affectation ne copie que la référence, pas l'objet référencé.



- Comment pouvons-nous copier un tableau?

Références perdues

- Avec les types de référence, il faut faire attention de ne pas « perdre » un objet référencé.



- Après l'affectation, il n'y a plus de référence vers le deuxième tableau! Celui-ci sera oublié par Java et ne pourra pas être récupéré.
- Voir exemple: AssignationTableaux.ppt

> "Les objets peuvent être classifiés scientifiquement en trois catégories: ceux qui fonctionnent, ceux qui se brisent, et ceux qui se perdent." – R. Baker

Trouver l'élément maximal dans un tableau

- Problème (rappelez-vous de l'exercice 6-16):
  - Étant donné un tableau de nombres, comment trouver la valeur maximale parmi les éléments de ce tableau?
- Idée:
  - Stratégie « parcours et mise à jour ». Premièrement, utiliser le premier élément comme candidat à la valeur maximal. Ensuite, parcourir les autres éléments du tableau et mettre à jour le maximum vu jusqu'ici lorsqu'un nombre plus grand est rencontré.
- Nous utiliserons une boucle pour parcourir les éléments du tableau.

Le modèle logiciel

DONNÉES:	N	*(un entier positif)*
	T	*(tableau de N valeurs)*
INTERMÉDIAIRE:	Index	*(index pour T)*
RÉSULTAT:	Max	*(élément maximal de T)*
EN-TÊTE	Max ←	MaxDansTableau(T, N)

MODULE

Traduction de 6-16 en Java

Variables de type *String*

- Les variables de type chaîne de caractères (*String*) ont toujours représenté un défi pour les langages de programmation.

  - Elles ont des tailles variables et, pour les emmagasiner, l'ordinateur préfèrerait pouvoir prédire la quantité d'espace mémoire nécessaire.

- Ainsi, les chaînes de caractères sont souvent des « cas spéciaux » dans un langage de programmation.

Type `String` en Java

- Les chaînes de caractères en Java sont aussi accéder avec des variables *de référence*.
  - Elles sont semblables à des tableaux de caractères
  - SAUF QUE:
    - Vous n'avez pas à utiliser **new** pour créer une chaîne
    - Vous n'utilisez pas **[]** pour accéder aux caractères de la chaîne.
- Exemple:

```
String message = "Bonjour!";
System.out.println( message );
```

- La classe (type de donnée) **String** offre plusieurs méthodes utiles pour la gestion de ces chaînes.
  - Ceci veut dire que les Strings sont des objets (on étudiera les objets dans la deuxième moitié du cours)

Méthodes utiles de String

- Soit la chaîne de caractères

```
String message = "Bonjour!";
```

alors:

  - Pour trouver la longueur d'une chaîne:

```
int laLongueur = message.length();
```

  - Pour accéder au (i+1)[ème] caractère:

```
int i = 4;
char leCaractère = message.charAt( i );
```

- Pour convertir un type primitif en String :

```
int unEntier = 17;
String uneChaîne = String.valueOf( unEntier );
// fonctionne pour int, double, boolean, char
```

- Pour joindre une chaîne à une autre (concaténation):

```
String chaîneJointe = chaîne1 + chaîne2;
```

- Voir autres méthodes pour l'API Java

Comparaisons de chaînes `String`

- Les chaînes `String` sont des types référence, et donc elles ne peuvent pas être comparées à l'aide de `==`.
- La classe `String` offre la méthode `compareTo()` pour comparer deux chaînes.
  - Les caractères de chaque chaîne sont comparés un à un de gauche à droite
    - La comparaison s'arrête dès que deux caractères sont différents ou quand une chaîne se termine.
      - Si str1 < str2, alors `compareTo()` retourne un `int` < 0
      - Si str1 > str2, alors `compareTo()` retourne un `int` > 0
    - Si les caractères sont les mêmes pour chaque position et que les deux chaînes on la même taille, alors la méthode retourne `0`.

Exercice 6-19: Comparaisons de chaînes String ?

- Quelle est la valeur de **résultat** pour ces exemples?
  - Exemple 1:

```
String str1 = "abcde" ;
String str2 = "abcfg" ;
int résultat = str1.compareTo(str2);
```

  - Exemple 2:

```
String str1 = "abcde" ;
String str2 = "ab" ;
int résultat = str1.compareTo(str2);
```

"Everything should be made as simple as possible,
but not one bit simpler ."
-- A. Einstein

ITI 1520
Section 7: Structure de programme

Objectifs:

- Programmes composés de plusieurs sous-programmes
- Tableaux en paramètres
- Programmes multi-classes

Utilisation de plusieurs sous-programmes

- Nous avons utilisé jusqu'à présent des programmes composés de 2 sous-programmes (2 algorithmes traduits à 2 méthodes Java)
  - Si un algorithme contient un bloc d'instructions complexe imbriqué dans un autre bloc, vous pouvez transformer le bloc imbriqué en un algorithme séparé.
  - Donc chaque algorithme peut à son tour invoquer d'autres algorithmes. De cette façon, nous pouvons garder nos algorithmes simples, courts, et clairs.
  - Donc il est possible de divisé un problème complexe en plusieurs tâches qui à leurs tours peuvent être subdivisées (conception descendante, i.e. top-down design)

Utilisation de plusieurs sous-programmes (suite)

- Un programme débute avec un algorithme **principal** (traduit à la méthode `main`) qui peut lire des données et afficher des résultats. L'algorithme **principal** invoque d'autres algorithmes, qui sont traduites chacun à une méthode Java,
  - De cette façon l'algorithme **principal** (méthode **main**) agit un peu à la façon d'un répartiteur.
  - Tentez de garder l'algorithme **principal** (et ainsi la méthode **main**) le plus simple possible – vous devez pouvoir déduire le fonctionnement général du programme en examinant l'algorithme **principal**.

Accessibilité des méthodes

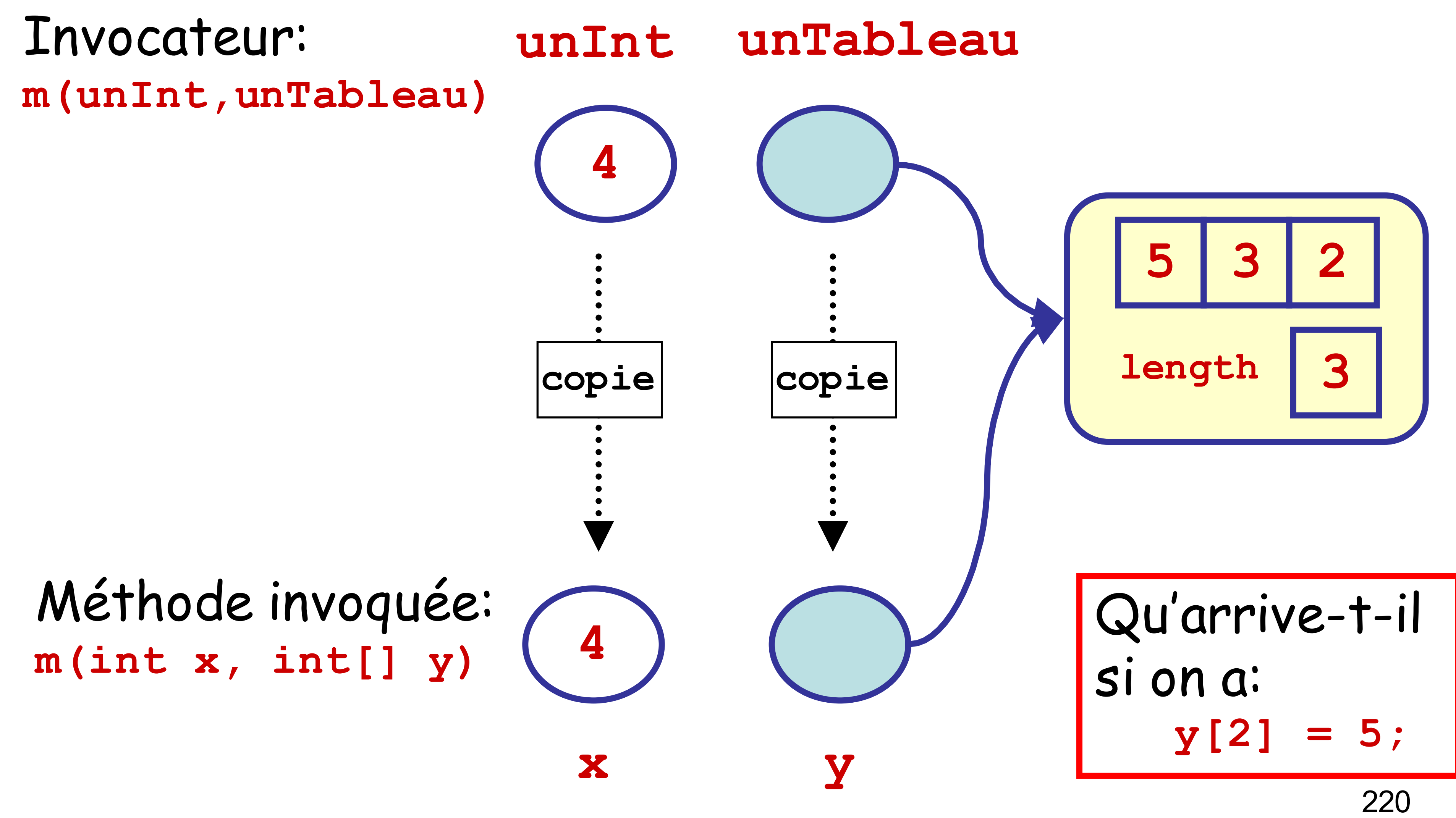
- En Java
  - Les méthodes sont amassés dans une « classe ».
  - Un programme Java peut être composé de plusieurs classes
- Si une méthode est **`public`**, elle peut être invoquée partout dans un programme, qui peut contenir plusieurs classes et fichiers dans divers répertoires.
- Si une méthode est **`private`**, elle peut être invoquée seulement dans la classe où elle est définie.
- Une méthode **`protected`** est invocable par la classe ainsi que par les sous-classes
- S'il n'y a rien de spécifié pour la méthode, elle ne peut alors être invoquée que par les classe qui se trouvent dans le même répertoire (*package methods*).

- Notez bien: seules les méthodes publiques et privées seront utilisées dans ce cours.

Tableaux en paramètres

- Un tableau est un type de référence; i.e. est accédé via une variable de référence.
- Un tableau n'est pas passé d'une méthode à une autre, c'est sa référence (i.e. le contenu d'une variable de référence) qui est passée à la méthode (ou retourné par une méthode).
- Ainsi, nous avons (temporairement) deux références vers le même tableau.
- Bien que la méthode invoquée ne puisse pas modifier la variable de référence originale, elle peut modifier le contenu du tableau. Les changements apportés au tableau demeureront même après le retour de l'invocation.
  - La copie d'une variable d'un type primitive est détruite lorsque la méthode retourne.
  - Pour un tableau, c'est la copie de la variable de référence qui est détruite lors du retour.

Passage de types primitifs et de référence



Exercice 7-1: Tracez ce programme

```
import java.io.* ;
class ÉchangeComplet
{
   public static void main (String args[ ])
   {
      int i = 0;
      int[ ] t = { 2, 4, 6, 8, 10, 12 } ;
      while (i <= 2)
      {
         échangeTbl(t, i, 5-i ) ;
         i = i+1;
      }
      for (i = 0 ; i <= 5 ; i = i+1)
      {
         System.out.println("t[" + i + "] vaut " + t[i]);
      }
   }
   // échangeTbl : échange les valeurs de x aux positions i,j
   // Données: x, référence à un tableau; et i,j, 2 indices de x
   public static void échangeTbl(int[ ] x, int i, int j)
   {
      // DECLARATION DES VARIABLES/DICTIONNAIRE DE DONNÉES
      int temp ; // Intermédiaire, contient x[i]
      // MODULE DE L'ALGORITHME
      temp = x[i] ;
      x[i] = x[j] ;
      x[j] = temp;
   }
}
```

Exercice 7-2: Validation de numéros

- Quelques cartes de crédit utilisent la méthode suivante pour déterminer la validité d'un numéro de carte. Un tel numéro est valide si son dernier chiffre est égal au dernier chiffre de la somme des autres chiffres du numéro. Par exemple:
  - 5792 est invalide (5+7+9 = 21)
  - 4231628 est valide (4+2+3+1+6+2 = 18)
- Problème: Écrivez un programme qui vérifie qu'un numéro de carte fourni par un utilisateur est valide. Ce programme devrait utiliser une boucle pour vérifier plus d'une carte, jusqu'à ce que l'utilisateur entre le nombre zéro.
- Note: Les numéros de carte de crédit ont habituellement 16 chiffres, et le type `int` n'est pas suffisant pour représenter de tels nombres. Un entier peut cependant représenter 4 chiffres.
- Hypothèse: Les quatre premiers chiffres du numéro de carte de crédit ne sont pas tous à zéro.

Exercice 7-2: Structure de données pour numéros

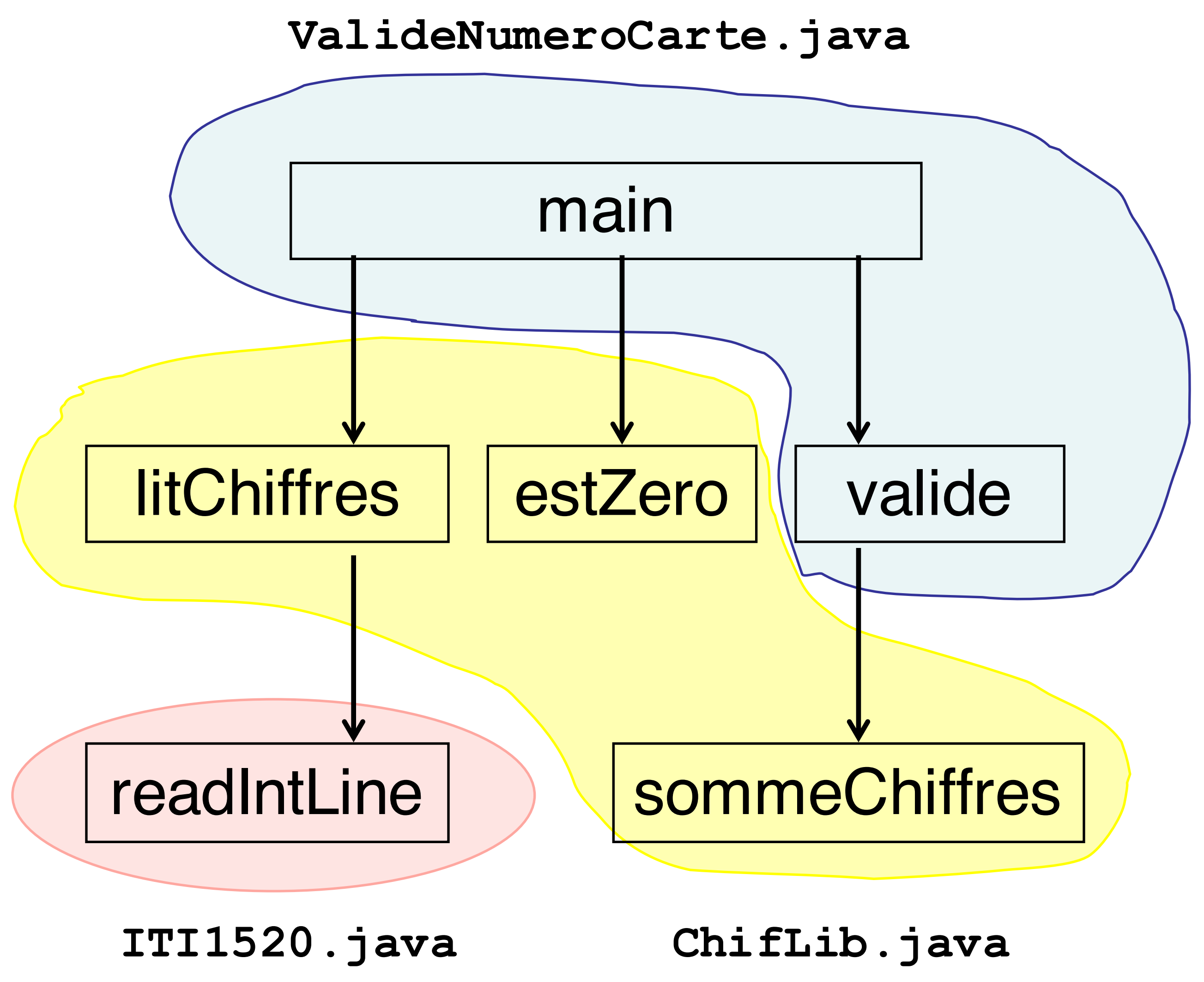
- Une **structure de données** est une façon d'organiser des données utilisées dans un programme.

- Pour ce problème, nous utilisons un tableau de quatre entiers (*int*) pour représenter le numéro de carte de crédit.
  - Les nombres réels n'ont pas la précision voulue.
  - Chaque entier dans le tableau représente 4 numéros de la carte de crédit

Exercice 7-2: Conception du programme

- **Organisation du programme:** Un algorithme possible qui résoudrait ce problème aurait besoin de faire les choses suivantes (détails non fournis):
  1. Lire un numéro de carte fourni par l'utilisateur. Ceci sera fait dans une méthode séparée.
  2. Vérifier si le numéro entré est 0 (premier ensemble de 4 chiffres).
  3. Si le numéro n'est pas 0, vérifier si le nombre est valide ou non. Ceci sera fait dans une autre méthode séparée.
  4. Afin de vérifier la validité du nombre, nous appellerons une méthode qui trouve la somme de ses chiffres.
  5. Afficher le résultat.
  6. Lire un autre numéro fourni par l'utilisateur.

Exercice 7-2: Diagramme de structure



Exercice 7-2: Méthode main

```
/* La méthode main, tel un chef d'orchestre, invoque les autres
   méthodes afin de compléter les tâches individuelles. */

public static void main (String [ ] args)
{
  // invoque litChiffres( ) pour saisir la donnée
  int [ ] chiffres = ChifLib.litChiffres( );
  while ((chiffres.length == 4) && (!ChifLib.estZero(chiffres)))
  {
    // envoie ce nombre à la méthode valide
    boolean testValide = valide(chiffres);
    // affiche le résultat
    if (testValide)
      { System.out.println("Ce numéro est valide."); }
    else
      { System.out.println("Ce numéro est invalide."); }

  }
}
```

Exercice 7-2: Méthode estZero( )

```
// première version: seulement les 4 premiers chiffres doivent
   être à 0
public static boolean estZero(int [ ] chiffres)
{
   boolean drapeau;
   drapeau =  chiffres[0] == 0;
   return(drapeau);
}

// deuxième version: tous les 16 chiffres doivent être à 0
public static boolean estZero(int [ ] chiffres)
{
   boolean drapeau;
   drapeau =



   return(drapeau);
}
```

Exercice 7-2: Méthode litChiffres( )

```
/* Cette méthode demande à l'utilisateur d'entrer son
   numéro de carte de crédit à l'aide de 4 nombres, qui
   seront placés dans un tableau. Cette méthode fait appel
   à readIntLine( ) de la classe ITI1520 afin de lire le
   tableau d'entiers. */

public static int [ ] litChiffres( )
{
  int [ ] tableauEntiers;
  System.out.println
    ("Entrez le numéro de carte à l'aide de quatre ");
  System.out.println
    ("nombres de quatre chiffres, séparés d'espaces");
  System.out.println
    ("blancs; ou appuyez sur 0 pour terminer.");
  tableauEntiers = ITI1520.readIntLine( );

}
```

Exercice 7-2: Méthode valide( )

```
/* Cette méthode invoque sommeChiffres( ) pour trouver la
   somme des chiffres d'un entier.  Elle compare ensuite
   les derniers  chiffres de la somme des 15 premiers
   chiffres et du numéro lui-même pour déterminer la
   validité de la carte. NOTE: Peut être privée!!! */
private static boolean valide(int [ ] chiffres)
{
  // trouve les 3 premiers chiffres du dernier groupe
  int troisPremiers =
  // trouve le tout dernier chiffre du numéro
  int dernierChiffre =
  // trouve la somme des 15 premiers chiffres
  int somme =



  // détermine la valididé
  boolean estValide =
  return estValide;
}
```

Exercice 7-2: Méthode sommeChiffres(?

```
// Retourne la somme des chiffres d'un nombre x

public static int sommeChiffres(int x)
{
  int somme = 0;
  while (x != 0)
  {

  }
  return somme;
}
```

Exercice 7-2: Combinez le tout!

```
import java.io.* ;
class ValideNumeroCarte
{
  public static void main (String [] args) { … }
  private static boolean valide (int [ ] chiffres)  { … }
}

import java.io.* ;
class ChifLib
{
  public static boolean estZero (int [ ] chiffres)  { … }
  public static int [ ] litChiffres( ) { … }
  public static int sommeChiffres (int x)  { … }
}
```

- Remplacez les `{ … }` par les modules vus aux pages précédentes, et sauvegardez dans deux fichiers (`ValideNumeroCarte.java` et `ChifLib.java`)
- Placez aussi `ITI1520.class` dans le même répertoire.
- Vous pouvez maintenant valider des cartes de crédit ☺

> "You take the red pill and you stay in Wonderland and I show you how deep the rabbit-hole goes. Remember: all I am offering is the truth, nothing more ."
> -- Morpheus, *The Matrix*

ITI 1520
Section 9: Matrices

Objectifs:

- Matrices = tableaux de tableaux
- Déclaration, accès, modification
- Modèle logiciel et Java

Matrices

- Une matrice L x C a R rangées et C colonnes.
- Exemple. Une matrice 4 x 6 d'entiers (0-100)

$$M = \begin{bmatrix} 71 & 62 & 33 & 89 & 85 & 74 \\ 68 & 65 & 75 & 88 & 70 & 72 \\ 87 & 0 & 0 & 90 & 92 & 88 \\ 58 & 72 & 66 & 57 & 74 & 74 \end{bmatrix}$$

M[r][c] représente l'entrée à l'intersection de la rangée r et la colonne c. Cette idée peut être étendue à 3, 4, … *n* dimensions.
(Note: les indices commencent à 0).

Matrices et tableaux à 2 dimensions

$$M = \begin{bmatrix} 71 & 62 & 33 & 89 & 85 & 74 \\ 68 & 65 & 75 & 88 & 70 & 72 \\ 87 & 0 & 0 & 90 & 92 & 88 \\ 58 & 72 & 66 & 57 & 74 & 74 \end{bmatrix}$$

- Une matrice est représentée dans nos algorithmes par un tableau à deux dimensions (un tableau de tableaux).

- Exercice 9-1: La matrice M est un tableau de 4 tableaux, chacun ayant 6 éléments. Ainsi:

  M[1][2] contient

  M[2][5] contient

  M[4][1] contient

  M[3] contient

?

Exercice 9-2: Valeur maximale dans une matrice (p. 1)

- Écrivez un algorithme qui trouve la valeur maximale dans une matrice:

Exercice 9-2: Valeur maximale dans une matrice (p. 2)

MODULE:

Exercice 9-2: Algorithme alternatif (utilisant MaxDansTableau, p. 206)

MODULE:

Matrices diagonales

- Une matrice carrée a le même nombre de lignes et de colonnes. Si les valeurs dans les deux triangles entourant la diagonale sont 0, alors il s'agit d'une matrice diagonale. Par exemple:

$$M1 = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad M2 = \begin{bmatrix} 2 & 4 & 0 \\ 3 & 5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

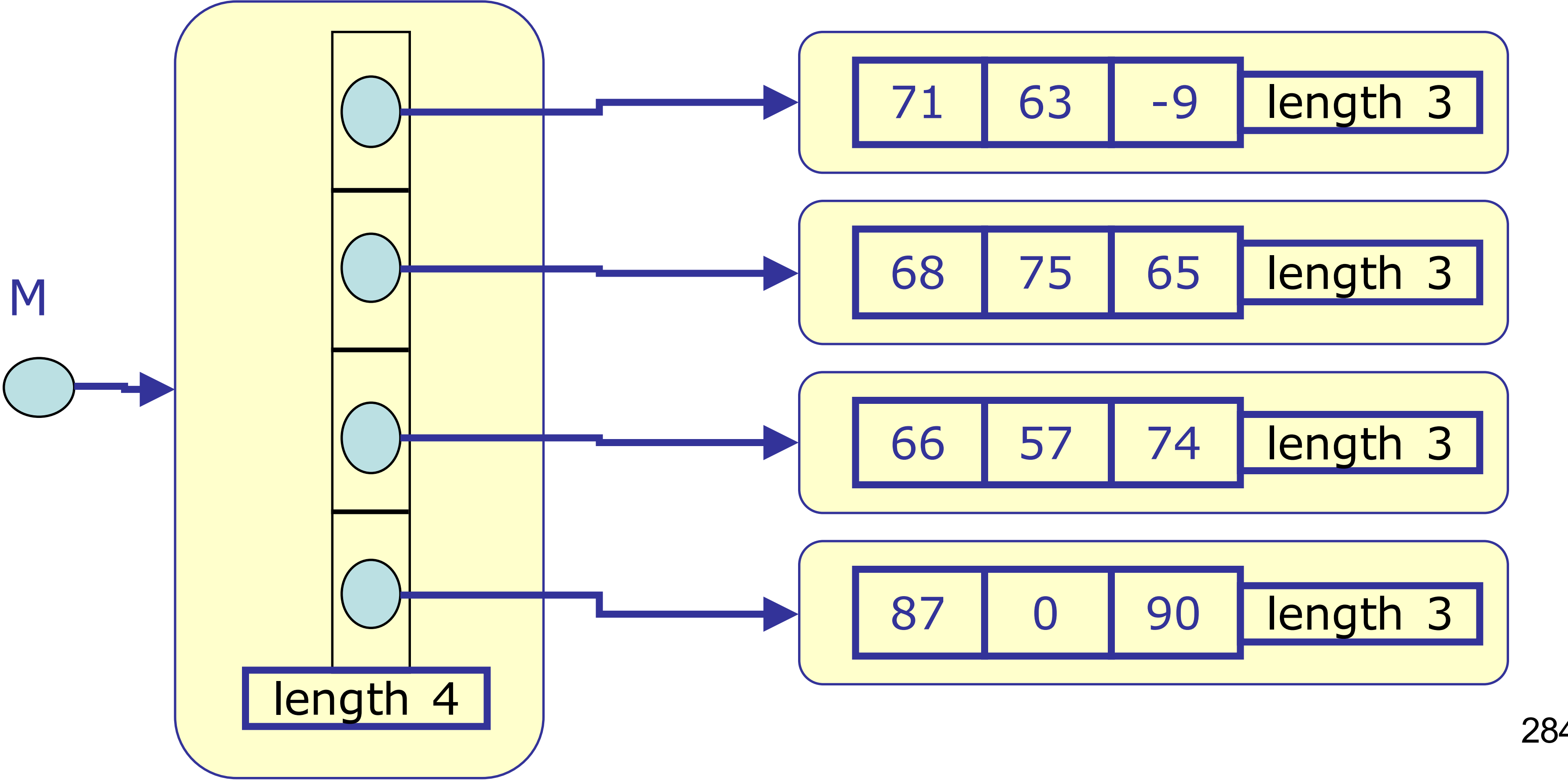
- *M1* est une matrice diagonale alors que *M2* ne l'est pas.
- Écrivez un algorithme qui vérifie si une matrice carrée donnée est une matrice diagonale.

Exercice 9-3: Algorithme VérifDiag ?

Exercice 9-3: Version efficace

Matrices en Java

- Un tableau 2-D (matrice) en Java est littéralement un tableau de tableaux; chaque élément du premier tableau est une référence à un tableau.



Déclarer un tableau 2-D

- Pour déclarer que M est référence à un tableau 2-D d'entiers:

```
int [][] m;
```

- Pour créer un tableau 2-D de 2x3 (en réservant l'espace-mémoire nécessaire) et en l'affectant sa référence à **M** :

```
m = new int[2][3];
```

- Pour créer et initialiser un tableau 2-D de 2x3:

```
int [][] m;
m = new int[][] { {1, 2, 3}, {4, 5, 6} };
```

- L'attribut Java `length` retourne la taille unique d'un tableau:

  `m.length` vaut 2

  `m[0].length` vaut 3

  Quiz: Que dire de `m[0][0].length`?

Exercice 9-4: Valeur maximale dans une matrice en Java

- Traduire l'algorithme trouvant la valeur maximale dans une matrice:
  - Note: **`Integer.MIN_VALUE`** est la valeur entière la plus négative permise pour un **`int`** Java, et peut être utilisé pour $-\infty$.

Exercice 9-5: Lecture d'une matrice

- Écrivez une méthode Java qui permettra de lire une matrice d'entiers. Les nombres de rangées et de colonnes sont fournis. La méthode demande les valeurs de la rangée 0, puis de la rangée 1, etc. Ces valeurs sont saisies une à une à l'aide de **`ITI1520.readInt()`**.

Effacer des lignes et des colonnes

- *Air Escape* a décidé d'arrêter ses vols à partir de la ville X (disons Paris, X=2). Les numéros des villes > X ont été réduits de 1 (Madrid est donc devenue la ville #3).
  - Ce problème peut être résolu en effaçant la rangée et la colonne correspondant à la ville retirée.

- Écrivez deux algorithmes, l'un pour effacer une rangée d'une matrice, l'autre pour effacer une colonne.
- Nous ne changerons pas la taille de la matrice, mais les éléments de la dernière rangée ou de la dernière colonne seront tous mis à 0.

Effacer une rangée

- **Idée:**
  - Pour effacer la $i^{ième}$ rangée d'une matrice, nous pouvons déplacer les rangées du bas vers le haut et placer des zéros à la dernière rangée.
  - Nous développerons un algorithme séparé pour copier une rangée vers la rangée du dessus, qui écrasera les valeurs déjà présentes.
    - *Ainsi*, notre algorithme principal n'aura qu'à faire une boucle pour copier la rangée i + 1 vers la rangée i, la rangée i + 2 vers la rangée i + 1, …, la rangée n-1 vers la rangée n - 2.
  - Nous utiliserons un autre algorithme pour mettre à zéro les éléments de la dernière rangée.

Effacer une colonne

- Effacer une colonne d'une matrice se fait de la même façon. À faire comme exercice!
- Un autre exercice possible consiste à généraliser le problème: Effacer une rangée ou une colonne d'une matrice qui n'est pas nécessairement carrée.