

« 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 2011

N. Japkowicz

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

© 2011, ÉITI, Université d'Ottawa

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

Table des matières

Section 1: Introduction	<u>3</u>
Section 2: Introduction à Java	<u>47</u>
Section 3: Algorithmes et leur traduction vers Java	<u>92</u>
Section 4: Traçage et débogage	<u>120</u>
Section 5: Branchements	<u>138</u>
Section 6: Tableaux et boucles	<u>163</u>
Section 7: Structure de programme	<u>214</u>
Section 8: Récursivité	<u>239</u>
Section 9: Matrices	<u>274</u>
Section 10: Introduction aux objets	<u>304</u>
Section 11 : Conception orientée-objet	<u>338</u>

« Si vous n'y réfléchissez pas réellement, vous pourriez croire que la programmation consiste simplement à coder des instructions dans un langage informatique. »

- *Ward Cunningham*, inventeur de WikiWiki

ITI 1520

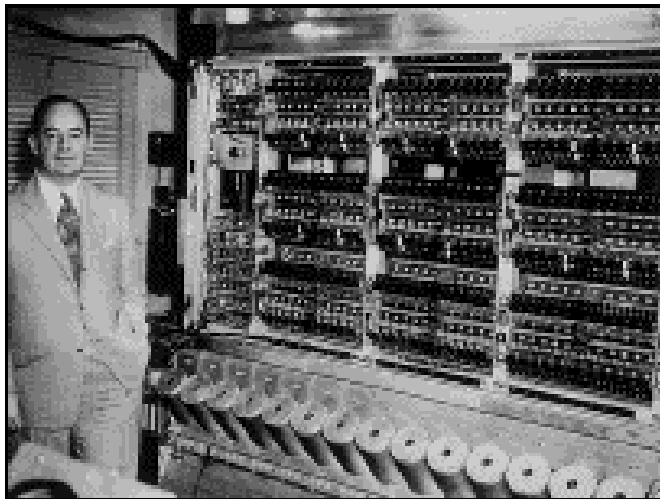
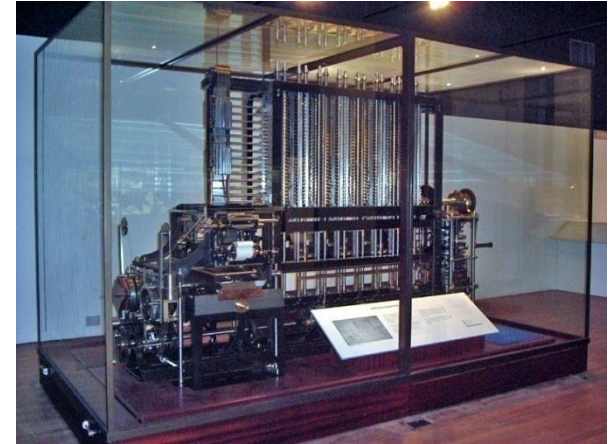
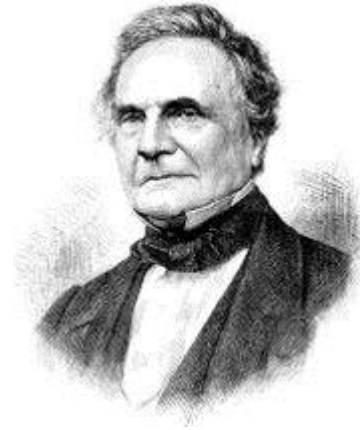
Section 1: Introduction

Objectifs:

- Introduction au génie logiciel
- Spécifications et algorithmes
- Variables et expressions

Note historique...

- Charles Babbage imagine et tente de réaliser, en 1833, une machine qui contient les concepts de ce que sera l'ordinateur moderne : unité de calcul, mémoire, registre et entrée des données par carte perforée.



- John von Neumann, mathématicien et informaticien, participe au développement du premier ordinateur: l'ENIAC (1945)
- Principes de l'architecture Von Neumann: les données et les programmes sont encodés en mémoire

Production de logiciels

- Ce cours porte sur la résolution de problèmes à l'aide de logiciels
- **Logiciels:**
 - Généralement de grande taille (milliers/millions de lignes de programme, plus la documentation, dans divers langages).
 - Impliquent de un à des milliers de développeurs/utilisateurs.
 - Développés et maintenus sur une longue période de temps (plusieurs mois, voire années; SABRE existe depuis 50+ ans).
 - Tout ceci nécessite des méthodes **systematiques** et **rigoureuses** de développement.
- **Génie Logiciel:**
 - Discipline pour le développement de logiciels corrects, en respectant le budget et le temps donnés.

Cycle de vie du logiciel

1. Analyse des exigences

- obtention des besoins des utilisateurs (qu'est-ce qu'on veut résoudre ?)
- estimation de la faisabilité
- estimation du coût, du temps, des ressources
- planification

2. Spécification de la solution

- définition de la structure du logiciel
- définition de la structure des données

3. Conception des algorithmes

- définir les étapes nécessaires pour résoudre un problème (ou sous-problèmes)

Cycle de vie du logiciel (suite)

5. Implémentation

- codage dans un langage de programmation
- on obtient un ensemble de programmes
- manuellement ou avec de l'aide d'outils

6. Tests et validation

- a-t-on les fonctionnalités et la qualité requises?
- débogage
- a-t-on vraiment résolu le problème ?

Cycle de vie du logiciel (suite)

6. Déploiement chez le client/utilisateur

7. Maintenance

- entretien, évolution, ajout/retrait de fonctionnalités
 - adaptation (au matériel, au système d'exploitation)
- La **documentation** est effectuée tout le long du cycle.
 - ITI 1520 couvre ce cycle mais se concentre surtout sur la conception des algorithmes (#3) et le codage (#4).
 - débute avec la spécification d'un problème
 - produit un programme pour le résoudre

Spécification du problème

- Notre but est de résoudre un problème, à l'aide d'un ordinateur.
- Le problème est généralement décrit en français (**description informelle**). En tant que concepteur de logiciel, il vous faut extraire les informations pertinentes (**spécification du problème**).

« Les ordinateurs excellent dans l'exécution d'instructions, mais ils ne peuvent pas lire vos pensées. » - D. Knuth

- Il vous faut ainsi **comprendre** le problème et clarifier:
 - Quelles sont les informations « **données** » fournies au programme?
 - Quels sont les **résultats** requis (produits par le programme)?
 - Quelles **hypothèses** pouvons-nous faire?
 - Quelles **contraintes** doivent être respectées par la solution?
- Cette étape est aussi appelée *analyse des exigences*.

Problème 1: Moyenne de trois nombres

- Description informelle:
 - Marc veut connaître **la moyenne du coût de** trois livres qu'il a achetés à la librairie.
- **DONNÉES**: description et noms des valeurs connues.
 - **Coût1**, **Coût2**, **Coût3** - nombres représentant le coût de chacun des livres de Marc.
- **RÉSULTATS**: description et noms des valeurs calculées à partir des données
 - **Moy** - moyenne de **Coût1**, **Coût2**, et **Coût3**.
- Aucune hypothèse ou contrainte (pour le moment!)

Qu'est-ce qu'un algorithme?

- Solution d'un problème à l'aide d'une **suite d'étapes** bien définies.
- Prend en entrée les données du problème et produit en sortie les résultats attendus.
- Le développement d'un algorithme est un processus créatif.
- Parfois, il est nécessaire de **décomposer** le problème en sous-problèmes moins complexes.

ATTENTION

Il n'y a pas d'algorithme pour tout problème.
Certains problèmes n'ont pas de solution algorithmique.
Il existe souvent plus d'un algorithme pour le même problème.

Représentation d'un algorithme (I)

- Les algorithmes sont décrits par un « modèle logiciel » à l'aide d'une notation précise compréhensible par les humains mais non-exécutable par les ordinateurs.
- Le modèle logiciel utilisé dans ce cours a le format suivant (**important!**):

DONNÉES

- Liste des noms de valeurs données (avec commentaires)

RÉSULTATS

- Liste des noms des résultats (avec commentaires)

EN-TÊTE: <RÉSULTATS> ← <nom_algorithme>(<DONNÉES>)

- Spécifie le nom de l'algorithme, l'ordre des données et l'ordre des résultats (qui peuvent être multiples)

(suite page suivante)

Représentation d'un algorithme (II)

HYPOTHÈSES

- Liste de conditions générales que l'on suppose vraies pour l'algorithme. On suppose que l'utilisateur de l'algorithme respecte ces conditions.

CONTRAINTES

- Liste de conditions générales que l'on suppose vraies au début de l'algorithme. Normalement, ces contraintes (souvent reliées aux DONNÉES) devraient être vérifiées.

MODULE

- Séquence d'instructions qui, lorsque exécutée, calcule les résultats désirés à partir des données.

Autre problème: triangle

- Description informelle:
 - Programme pour calculer la surface d'un triangle.
- **DONNÉES:**
 - Côté1, Côté2, et Côté3 - nombres représentant la longueur de chacun des côtés du triangle.
- **RÉSULTATS:**
 - Surface - surface du triangle.
- **HYPOTHÈSES:**
 - Côté1, Côté2, et Côté3 sont supérieurs à 0
- **CONTRAINTES:**
 - Côté1, Côté2, et Côté3 sont tels que les 3 côtés forment un triangle
 - Par exemple: 1 m, 1 m, 5 m formeraient une combinaison invalide.

Valeurs utilisées dans les algorithmes

- **Littéral**: valeurs constantes de données
 - Deux types: valeurs numériques, et texte
- Exemple de littéraux numériques:
`2, 3.14159, -14, -14.0`
- Un **caractère** est entouré de simple guillemets. Les guillemets ne font pas partie des données. Exemple de caractères:
`'a', 'A', '1', '!'`
- Une **chaîne** est une séquence de caractères entourée de double guillemets. Les guillemets ne font pas partie des données. Exemple de chaînes:
`"bonjour", " foo", "barre ", "2", " ", ""`

Stocker des valeurs dans un ordinateur

- La mémoire d'un ordinateur est un ensemble de « cellules » contenant des valeurs; chaque cellule est associée à une adresse.
- Une variable représente l'une ou plusieurs de ces cellules. Donc une variable est un **emplacement en mémoire** ayant **une adresse** et contenant une valeur.
- Une variable possède:
 - un nom (son adresse)
 - une valeur
 - (un type)

L'affectation de valeurs aux variables

- Donner une valeur à une variable s'appelle **affectation**.
- Dans notre modèle logiciel, nous utilisons
 $\text{uneVariable} \leftarrow \text{uneExpression}$
pour dénoter que la valeur de l'expression **uneExpression** est affecté à la variable **uneVariable**.
 - Exemple: $X \leftarrow 3$
- Du côté droit, **uneVariable** peut aussi être une variable, mais elle représente seulement sa valeur ou une expression (voir prochaine acétate).
 - Exemple: $X \leftarrow \text{AutreVariable}$
- Exemple: $X \leftarrow 4, Y \leftarrow 3, X \leftarrow Y$
 - Que valent X et Y après l'exécution de ces trois affectations?
- Les noms (variables) des DONNÉES d'un algorithme sont appelées **paramètres** (ou: *paramètres formels*)

Expressions simples

- Une expression est un énoncé dont l'exécution produit une valeur.
- Une expression peut utiliser des opérateurs, par exemple des opérateurs arithmétiques (+, -, * ou ×, / ou ÷)
 - Exemples: $40 + 10/5$, $\sqrt{49} * \log_2(32)$
- Une expression peut utiliser des variables
 - Exemple: $40 + |Y|/5$
- On peut affecter le résultat d'une expression à une variable
 - Exemple: $X \leftarrow 40 + Y/5$

Opérateurs

- Un opérateur représente un certain type de calcul
- Le nombre de valeurs qu'utilise un opérateur peut varier:
 - Binaire: utilise deux opérandes (valeurs)
 - Unaire: utilise une opérande
 - Exemples:
 - **Soustraction**: un opérateur binaire
 - **Négation**: un opérateur unaire
- Les opérateurs sont évalués (i.e. exécutés) dans un **ordre** spécifique, et certains opérateurs ont une **précédence** sur d'autres.
 - Pour nos modèles, nous utiliserons l'ordre et la précédence dictées par les règles mathématiques.

Division et restants

- Il est souvent utile de faire de la division de nombre entiers, au lieu de la division de nombres réels.
- Deux types de division:
 - Réel: $11.0 / 4.0$ résultat: 2.75
 - Entier: $11 \div 4$ résultat : 2
 - Le restant de la division entière est **tronqué**; la fraction est perdue.
- Il est aussi très utile d'obtenir le **restant** de la division de nombres entiers. Ceci se fait avec l'opérateur **modulo**:
 - $11 \bmod 4$: résultat: 3
- Trouver le restant donne de résultats importants:
 - La valeur de $X \bmod Y$ est toujours dans l'étendue 0 à $Y - 1$.
 - La valeur de $X \bmod 10$ donne toujours le dernier chiffre de X .

Exercice 1-1 - Algorithme pour moyenne



DONNÉES: Nbr1, Nbr2, Nbr3 (*trois nombres*)

RÉSULTATS:

Moy (*la moyenne de Nbr1, Nbr2 et Nbr3*)

EN-TÊTE:

MODULE:

Exercice 1-2 - Autre exemple



-
- Écrivez un algorithme (CalculP) qui prend en entrée trois valeurs (A, B, C) et retourne pour chacune d'elle la proportion de cette valeur sur le total.
 - Première solution:

Exercice 1-2 - Autre exemple (suite)



-
- Écrivez un algorithme (CalculP) qui prend en entrée trois valeurs (A, B, C) et retourne pour chacune d'elle la proportion de cette valeur sur le total.
 - Deuxième solution:

Variables intermédiaires

- Il est parfois utile d'utiliser, dans un algorithme, des variables autres que les données ou les résultats afin de conserver des valeurs temporaires.
- Ces variables **INTERMÉDIAIRES** devraient être listées et décrites au même titre que les données et résultats dans la définition d'un algorithme.
- Leurs valeurs ne sont pas retournées à l'algorithme invocateur, et elles ne sont pas conservées d'une invocation à l'autre.
- Les valeurs intermédiaires sont utilisées principalement pour améliorer la lisibilité d'un algorithme ou pour le rendre plus performant.

INTERMÉDIAIRES

- Les INTERMÉDIAIRES ont des valeurs qui peuvent varier:
 - Selon les valeurs DONNÉES à l'algorithme
 - Lors des calculs intermédiaires
- Les INTERMÉDIAIRES qui ne varient pas sont appelées **CONSTANTES**
 - Leurs valeurs sont fixes:
 - Peu importe la valeur des DONNÉES
 - Peu important les calculs
 - On leur donne un nom afin d'aider à documenter l'algorithme et de le rendre plus lisible (ex.: pi).
 - L'utilisation de constantes peut aussi amoindrir l'effort de maintenance (ex.: TPS).

Exercice 1-3: Moyenne sur 100



-
- Écrivez un algorithme qui considère trois notes (sur 25) et calcule leur moyenne (sur 100).
 - (Stratégie: faire la moyenne des notes, puis convertir le tout sur 100).

Exercice 1-4 - Dernier exemple...



- ... sans constante

- ...avec constantes TPS et TVQ

...

Module

...

$$T1 \leftarrow C1 * 0.07 \quad //TPS$$

$$T2 \leftarrow C2 * 0.07 \quad //TPS$$

$$T3 \leftarrow C3 * 0.07 \quad //TVQ$$

$$T4 \leftarrow C4 * 0.07 \quad //TPS$$

...

Qu'arrive-t-il lorsque la TPS passe à 6%?

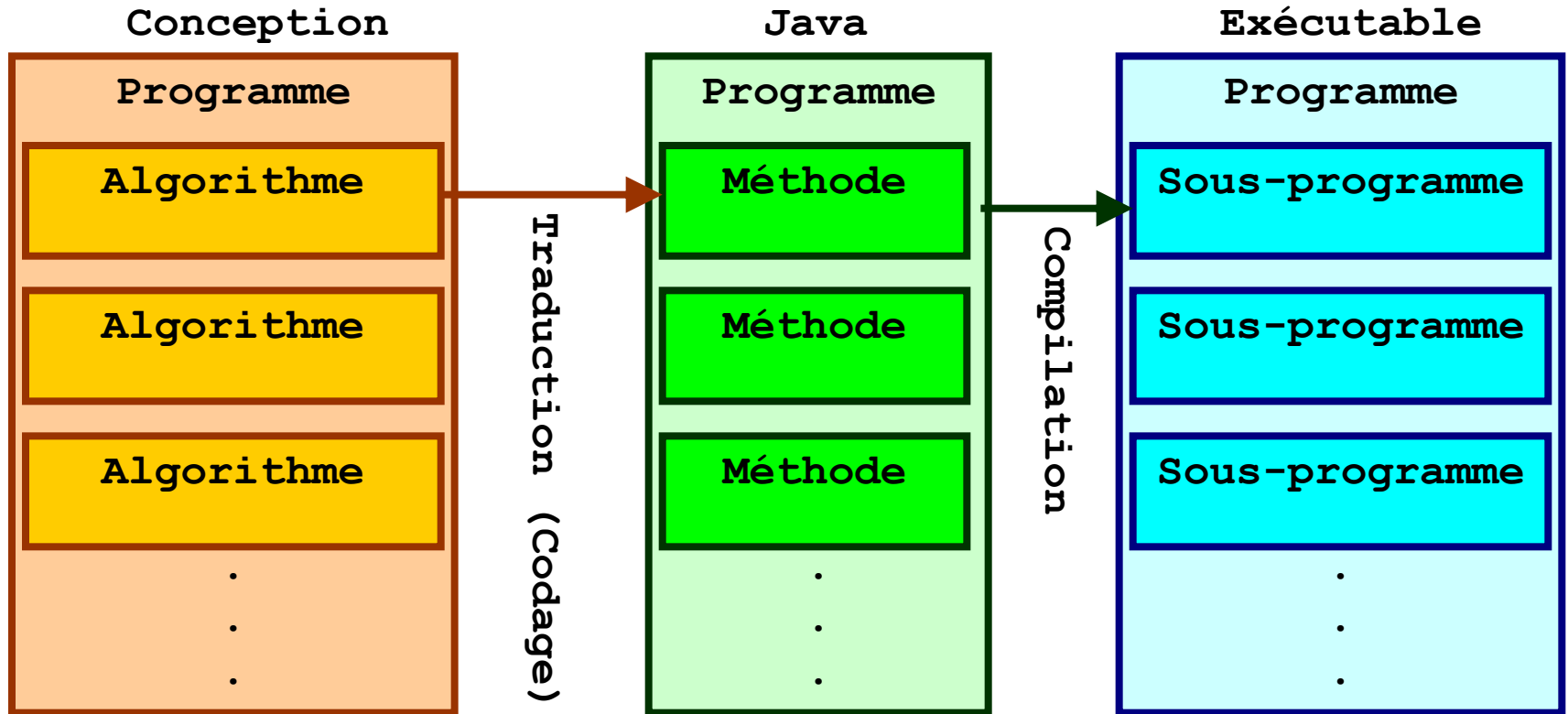
Sommaire de variables

- **DONNÉES** sont des variables contenant des valeurs données. Ces valeurs varient d'une invocation à une autre (voir prochaine diapositive), en d'autres mots, leurs valeurs changent pour chaque instance de problème.
- **RÉSULTATS** sont les variables qui contiennent les réponses trouvées par l'algorithme.
- **INTERMÉDIATES** sont les autres variables utilisées pour résoudre le problème.

Sous-programme

- Le sous-programme est une séquence d'instructions informatique pour accomplir une tâche
 - Un sous-programme normalement manipule un ensemble de variables (données et intermédiaires)
 - La source des données est un autre sous-programme
 - Un sous-programme normalement produit des résultats
 - Les résultats sont acheminés vers un autre sous-programme
 - L'algorithme est un modèle du sous-programme
 - Un sous-programme peut être « invoqué » par d'autres sous-programmes - à voir à la section 3.

Un programme est composé de sous-programmes



- Nous allons commencer avec un programme pour lequel on crée deux sous-programmes:
 - Un premier qui interagit avec l'utilisateur
 - Un deuxième qui résout un problème.

Le bloc d'instruction

- Le bloc d'instruction comprend une séquence d'instructions informatique
 - L'ordinateur exécute chaque instruction en séquence
 - Notez qu'une instruction:
 - Peut être une expression composée de plusieurs opérations (+, -, ← +)
 - L'invocation d'un autre algorithme
 - Le module de l'algorithme est un bloc d'instruction
 - Le bloc d'instruction jouera un rôle dans la définition d'instructions plus complexes (instructions de branchement et de boucle)

Représenter l'entrée par des algorithmes

- On veut pouvoir définir des algorithmes (sous-programmes) pour interagir avec l'utilisateur
- Doit pouvoir définir des algorithmes « standards » pour lire des valeurs du clavier et écrire des chaînes à la console
- Définissons les algorithmes suivants pour lire du clavier:
 - VarNbr ← LireRéal() pour lire un nombre réelle du clavier
 - VarEnt ← LireEntier() pour lire un nombre entier du clavier
 - VarChn ← LireLigne() pour lire une ligne d'entrée, i.e, une chaîne du clavier.
 - VarBool ← LireBooléen() pour lire "vrai" ou "faux".
 - VarCar ← LireCaractère() pour lire un seul caractère du clavier.
 - VarNbr ← Aléatoire() pour créer une valeur aléatoire plus grand ou égale à 0.0 et plus petit que 1.0.
- Exemples d'invocation des algorithmes.
 - X ← LireRéal()
 - A ← LireRéal()
 - (les instructions ci-dessus représentent des invocations d'algorithme)
 - (note que nous nous inquiétons pas comment ces algorithmes fonctionnent)

Représenter la sortie dans les algorithmes

- Définissons les algorithmes suivant pour écrire à la console
 - AfficheLigne(<liste d'arguments>)
 - Imprime une chaîne à la console (le curseur est ensuite placé à la prochaine ligne).
 - Affiche(<liste d'arguments >)
 - Imprime une chaîne à la console (le curseur est laissé à la fin de la chaîne imprimée).
 - La chaîne imprimée est une concaténation de tous les arguments dans la <liste d'arguments>. Un argument peut être un chaîne, littéral, un nom de variables, ou une expression.
 - Exemples:
 - Affiche("S.V.P. entrez une valeur: ")
 - AfficheLigne("Le résultat est ", x)
 - AfficheLigne("Additionner a et b donne ", a+b)
 - La liste d'arguments sera traduit à une expression de concaténation en Java (à voir dans la prochaine section)

Modèles vs. Calculs

- Dans nos algorithmes, nous utilisons des opérateurs et des valeurs mathématiques comme **modèle** de calculs.
- Un modèle n'est qu'une abstraction qui décrit l'essence du problème, mais pas ses détails. Lorsque vient le temps de faire les vrais calculs, des contraintes supplémentaires peuvent être introduites:
 - Pouvons-nous utiliser des valeurs illimitées?
 - Les valeurs doivent-elles avoir des types?
 - Y a-t-il des contraintes de temps pour les calculs?
 - Avons-nous suffisamment de mémoire pour toutes les variables?
 - Est-ce qu'un opérateur dans un langage de programmation fonctionne exactement comme dans représentation en mathématique?
 - ...

Codage

- **Codage** = traduction d'un algorithme vers un langage de programmation, produisant ainsi un **programme** exécutable par l'ordinateur.
- **Ne codez pas trop vite!** Il est plus important de passer du temps sur votre algorithme, en y réfléchissant, en le vérifiant mentalement, et en le « traçant » avec des valeurs de test afin de s'assurer qu'il soit correct.

« Le plus tôt vous commencez à coder, le plus de temps prendra votre programmation. » - R. Carlson

- Le codage est un processus surtout mécanique, et non créatif.
- Le développement d'algorithmes et le codage requièrent tous deux une attention particulière aux détails.

Types de langages de programmation

- Langage machine
 - Nombres (binaires: 0 et 1). Le seul langage que l'ordinateur comprend directement.
- Langage assembleur
 - Instructions faites de codes symboliques
 - Par exemple, MOV, LDA, BRA
 - Un Assembleur convertit le code source assembleur en langage machine
- **Langages de haut niveau** (3e génération)
 - Utilise des mots clés (anglais) comme instructions
- Langages de 4e génération
 - Syntaxe plus près de la langue naturelle (p.e., SQL)
 - Langages plus visuels (p.e., VB.Net), graphiques (p.e., SDL)

Paradigmes de programmation

- Pour les langages de haut niveau
 - Programmation impérative / procédurale
 - Basic, Pascal, Fortran, C, ...
 - Programmation orientée-objet
 - Java, C++, Smalltalk, ...
 - Programmation fonctionnelle
 - Lisp, Scheme, ML, ...
 - Programmation logique
 - Prolog, ...

Langages de programmation communs

Langage	Débuts	Utilisations typiques
FORTRAN	1956	Calculs scientifiques
LISP	1958	Systemes experts, intelligence artificielle
COBOL	1960	Affaires, gestion
APL	1962	Statistiques
Basic	1971	Enseignement, affaires
C	1972	Création de systèmes d'exploitation, de pilotes
C++	1984	Tout usage, communications
Java	1993	Tout usage, applications Internet

Autres langages

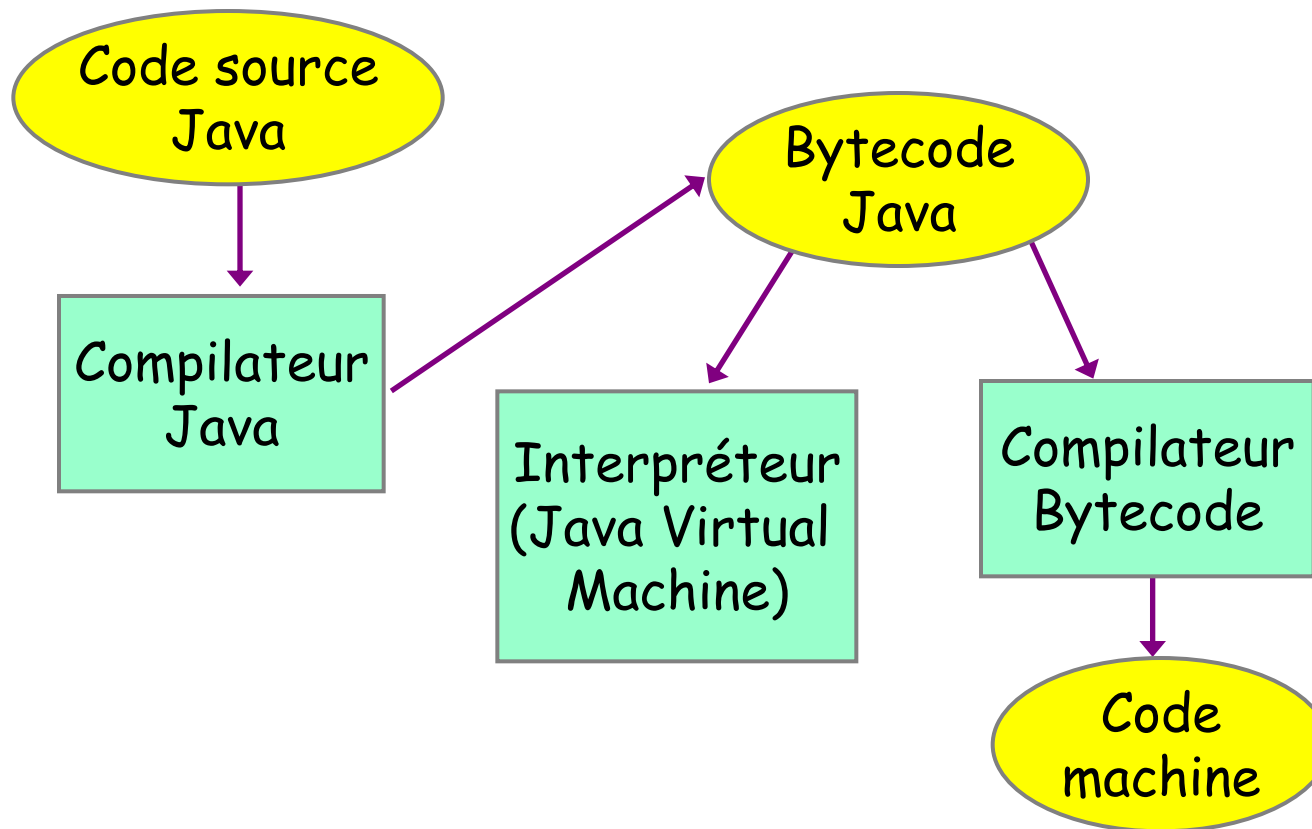
Langage	Débuts	Utilisations typiques
Pascal	1971	Enseignement
Smalltalk	1972	Orienté-objet, prototypage
Prolog	1972	Logique, systèmes experts
Ada	1979	Langage standard du Dépt. américain de la défense
*SQL = Structured Query Language	1979	Bases de données
*HTML = HyperText Markup Language	1992	Format pour décrire les pages Web
*XML = eXtensible Markup Language	1996	Structures et échanges de données (sur Internet)
C#	2002	Tout usage, réponse de Microsoft à Java (Sun)

* Langages qui ne sont pas pour la programmation générale

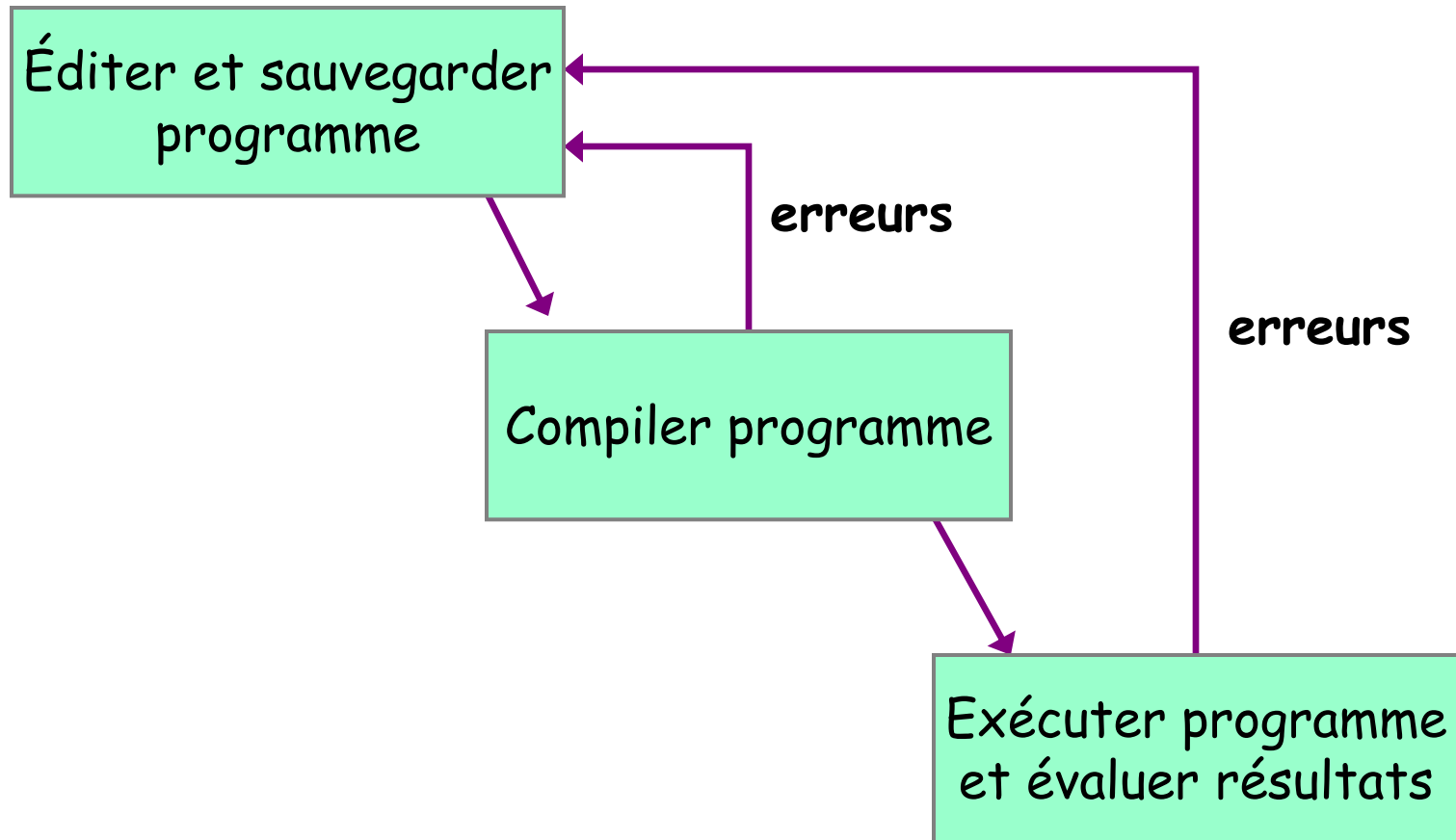
Langage de programmation et compilation

- 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 langage 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 »

De quel type d'erreur s'agit-il?

- Je m'appelle Daniel.
- m'Daniel. Je appelle
- La variable X vaut 0, et le résultat vaut $3/X$.
- Cette erreur est plus subtil.
- Tous les ballons sont ronds. Si cet objet n'est pas un ballon, alors il n'est pas rond.

Documentation

- La **documentation** regroupe tout le matériel qui rend facile la compréhension d'un programme pour l'utilisateur et le programmeur.

« Si vous ne pouvez l'expliquer simplement, alors c'est que vous ne le comprenez pas encore assez bien. » - *A. Einstein*

- La **documentation interne** (commentaires, noms de variables descriptifs, ...) se retrouve dans le programme.
- La **documentation externe** (manuel de l'utilisateur, manuel de référence, ...) se retrouve hors du programme.

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

ITI 1520

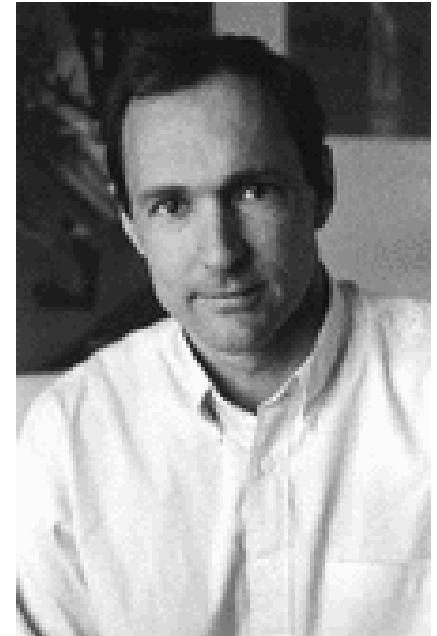
Section 2: Introduction à Java

Objectifs:

- Types de données
- Caractères et chaînes
- Opérateurs et précedence
- Lecture et affichage de valeurs
- Gabarit de classe

Note historique...

- Afin d'aider à la diffusion des informations, Tim Berners-Lee, chercheur du CERN de Genève, a mis au point, en 1990, le *World Wide Web*.
- Il est à l'origine de plusieurs standards parmi les plus utilisés comme HTTP, URL, et le langage HTML.
- Travaille maintenant sur le Web sémantique



Informaticien chez Sun Microsystems et Canadien d'origine, James Gosling et son équipe inventent, en 1992, le langage Oak qui sera renommé Java en 1994. Il produit le premier compilateur et la première machine virtuelle Java. Il a aussi écrit NeWS et Gosling Emacs.

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-programmes* 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 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 son initialisation (affectation d'une valeur), devez-vous considérer 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 choisit 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 ésoériques (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 priorité suivante (les opérateurs sur une même ligne ont la même priorité):

() (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:

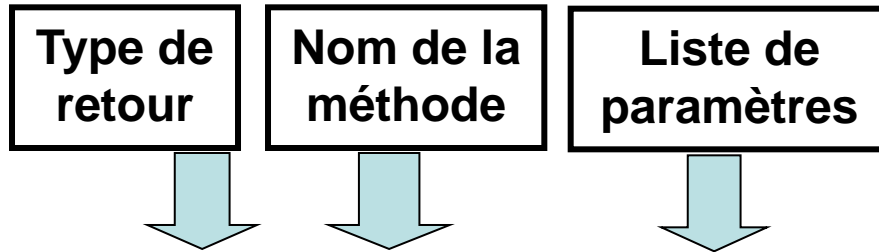
```
/**  
 * Ce texte sera inclus dans un document HTML.  
 *  
 * @author Daniel Amyot  
 * @version 2009  
 */
```
- Voir <http://java.sun.com/j2se/javadoc> et la documentation en ligne de la classe `iti1520`

Méthodes Java

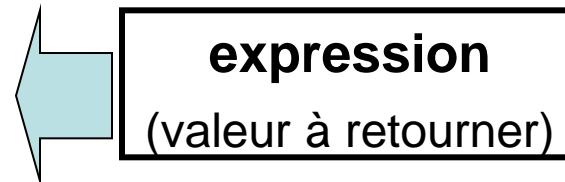
- Tel que déjà mentionné, la méthode Java est un sous-programme
- Chaque méthode Java a
 - Un **type de retour**: Spécifie le type du *résultat* de la méthode. S'il n'y a pas de résultat, ce type est mis à **void**.
 - Pour le moment, toujours ajouté les mots clefs « public static » avant le type de 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
```



```
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> ;  
}
```



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 [#55. Type double \(opérateurs\)](#).
 - 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:



Lecture à partir du clavier

- Lire une valeur du clavier en entrée n'est pas chose facile en Java.
- Pour simplifier le code, la classe Java **ITI1520** vous est fournie. Vous pouvez la télécharger du [site Web](#) du cours. Elle est compatible Java 1.4, 1.5/5.0, 1.6/6.0.
- Dans vos devoirs, vous devrez inclure le fichier **ITI1520.java**, dans le même répertoire que votre programme. Vous serez alors en mesure d'invoquer les méthodes de cette classe afin de lire une valeur (ou plusieurs valeurs) à partir du clavier.

Méthodes de la classe **ITI1520**

ITI1520.readInt() : Retourne un entier de type **int**
ITI1520.readDouble() : Retourne un réel de type **double**
ITI1520.readChar() : Retourne un caractère de type **char**
ITI1520.readBoolean() : Retourne une valeur de type **boolean**
ITI1520.readDoubleLine() : Retourne un tableau de **double**
ITI1520.readIntLine() : Retourne un tableau de **int**
ITI1520.readCharLine() : Retourne un tableau de **char**
ITI1520.readString() : Retourne une chaîne de type **String**

- 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 **ITI1520**

```
int x = ITI1520.readInt( );
```

- Si vous entrez **123** et appuyez ENTER, **x** contiendra alors la valeur **123**.
- La méthode **readDouble** fonctionne de la même façon.

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 );
    }
}
```

Alternative pour lire du clavier

Disponible sur Java 5.0 et Java 6.0

- Java permet l'utilisation de la classe **Scanner** comme alternative à la classe **ITI1520**:
 1. Créez un nouveau scanner (ceci est un objet) et affecter sa référence à une variable de référence **clavier**.
 2. Chaque fois que vous voulez une valeur du clavier, appeler une méthode via la variable de référence **clavier**.
- La méthode appelée dépend du type de valeur à être entrée (voir prochaine acétate).
 - Le « scanner » lira les caractères tapés et les convertira - si possible - au type de valeur désiré.

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

Note historique...

- Ada Byron, Comtesse de Lovelace, mathématicienne et collaboratrice de Babbage, définit vers 1840 le principe des itérations successives dans l'exécution d'une opération.
- Elle a créé le premier algorithme pour ordinateur (moteur analytique) et est souvent considérée comme étant la première programmeuse.
- Un langage porte son nom: Ada

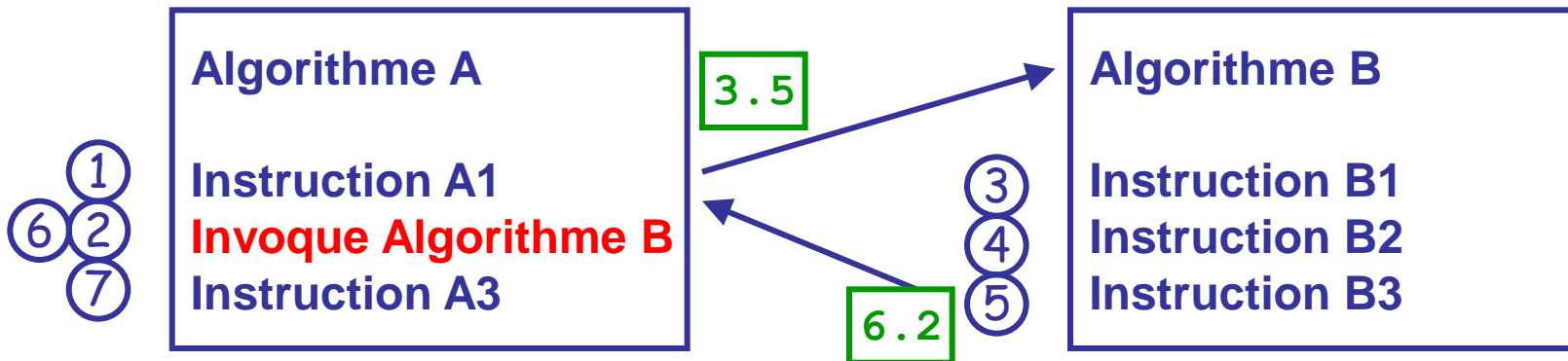


Utilisation d'algorithmes

- En développant un algorithme, il est habituellement utile de **réutiliser** des algorithmes existants (que vous avez écrits ou qui sont disponibles dans une librairie).
- Vous pouvez utiliser une instruction qui **INVOQUE** un algorithme existant où il est requis dans l'algorithme que vous développez.
- **Attention:** Assurez-vous que l'**ordre** des données, modifiées, et résultats soit correct!
- Pour invoquer un algorithme, vous n'avez besoin de connaître que son en-tête. Pas besoin de connaître son module (vous devez cependant avoir confiance en son bon fonctionnement!).

Invocation d'un algorithme

- Quand un algorithme en invoque un autre:
 - L'algorithme invocateur s'arrête au point d'invocation.
 - Des valeurs peuvent être "passées" à l'algorithme invoqué.
 - L'algorithme invoqué s'exécute.
 - Des valeurs résultantes peuvent être "retournées" à l'algorithme invocateur.
 - L'algorithme invocateur redémarre à l'instruction suivante.




Passage d'information

- En invoquant un algorithme, l'instruction d'invocation et l'en-tête doivent être identiques, sauf pour les noms des données, modifiées, et résultats. Ils correspondent un à un, dans le même ordre.

Invocation: MoySur25 ← Moyenne(N1, N2, N3)

EN-TÊTE: Moy ← Moyenne(Coût1, Coût2, Coût3)



- Les flèches indiquent de quelle façon l'information est passée. Une flèche à double tête serait utilisée pour un argument modifié.

Invocation d'un algorithme

- On appelle **invocation** l'exécution de l'algorithme avec des valeurs de données spécifiques.
- Représentée comme l'entête de l'algorithme, excepté pour les noms des données et des résultats qui sont remplacés par des valeurs. Exemple:

MaMoyenne ← Moyenne(10, 7, 4)

invoque notre algorithme avec les données Coût1=10, Coût2=7, Coût3=4 et retourne le résultat dans MaMoyenne (Moy)

- Notez: Si un nom de variable est utilisé comme argument, alors c'est sa valeur qui est passé à l'algorithme invoqué (passe par valeur).
- Notez: le résultat (valeur) est affecté à la variable spécifiée dans l'invocation.
- **ATTENTION:** L'échange de l'information entre l'invocation et l'algorithme est faite selon l'**ordre** des données et résultats (et non leurs noms).
- Les valeurs passées à un algorithme sont appelées des arguments (ou paramètres actuels). Les *arguments* (**paramètres actuels**) correspondent aux *paramètres* (**paramètres formels**) selon leur ordre.

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 (P_1, P_2, \dots, P_n), plus simples que P , dont les résultats seraient utiles pour résoudre P .
 2. Déterminez l'information d'en-tête (données, résultats, en-tête) pour $P_1 \dots P_n$, mais ne concevez pas leurs algorithmes tout de suite.
 3. Écrivez l'algorithme de P en supposant l'existence d'algorithmes pour $P_1 \dots P_n$.
 4. Développez les algorithmes de $P_1 \dots P_n$.

Programmes avec deux algorithmes

- Installment nous développerons des programmes ayant deux algorithmes (i.e deux sous-programmes)
 - Le premier, appelons-le `Principal()`, est utilisé pour interagir avec l'utilisateur
 - Le deuxième (ayant un nom approprié) est l'algorithme de résolution de problème.
- Considérez le développement d'un programme pour le problème suivant: calculer la moyenne sur 100 pour 3 notes sur 25.
- Nous avons déjà développer l'algorithme pour résoudre le problème.
- Dans l'algorithme `Principal()`, utilisons les algorithmes « standards » pour interagir avec l'utilisateur.

Algorithme RésNotes (résolution de problème)

Données: N1, N2, N3 (*notes sur 25*)
Résultats: MoyNotes (*moyenne de N1, N2, et N3, sur 100*)

Intermédiaires: Somme (*somme de N1, N2, N3*)
MoySur25 (*moyenne de N1, N2, N3 sur 25*)

Entête: MoyNotes \leftarrow RésNotes(N1, N2, N3)

Module: Somme \leftarrow N1 + N2 + N3
MoySur25 \leftarrow Somme / 3
MoyNotes \leftarrow MoySur25 * 4

Exercice 3-1 - L'algorithme Principal()

- L'exécution de notre programme commence toujours par cet algorithme (et donc se traduit à la méthode `main()` de Java)
- Aucune DONNÉE ni RÉSULTAT - i.e. aucune variable est utilisé pour recevoir des données ni de variable pour retourner un résultat.
- Utilise seulement des variables intermédiaires.
- Utilise les algorithmes `AfficheLigne(...)` et `Affiche(...)` pour afficher des messages à la console.
- Utilise les algorithmes `LireRéal()`, `LireEntier()`, `LireLigne()`, et `LireBooléen()` pour lire des valeurs du clavier.
- 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.

Exercice 3-1 - Algorithme Principal (interface à l'utilisateur)



DONNÉES:

RÉSULTATS:

INTERMÉDIAIRES:

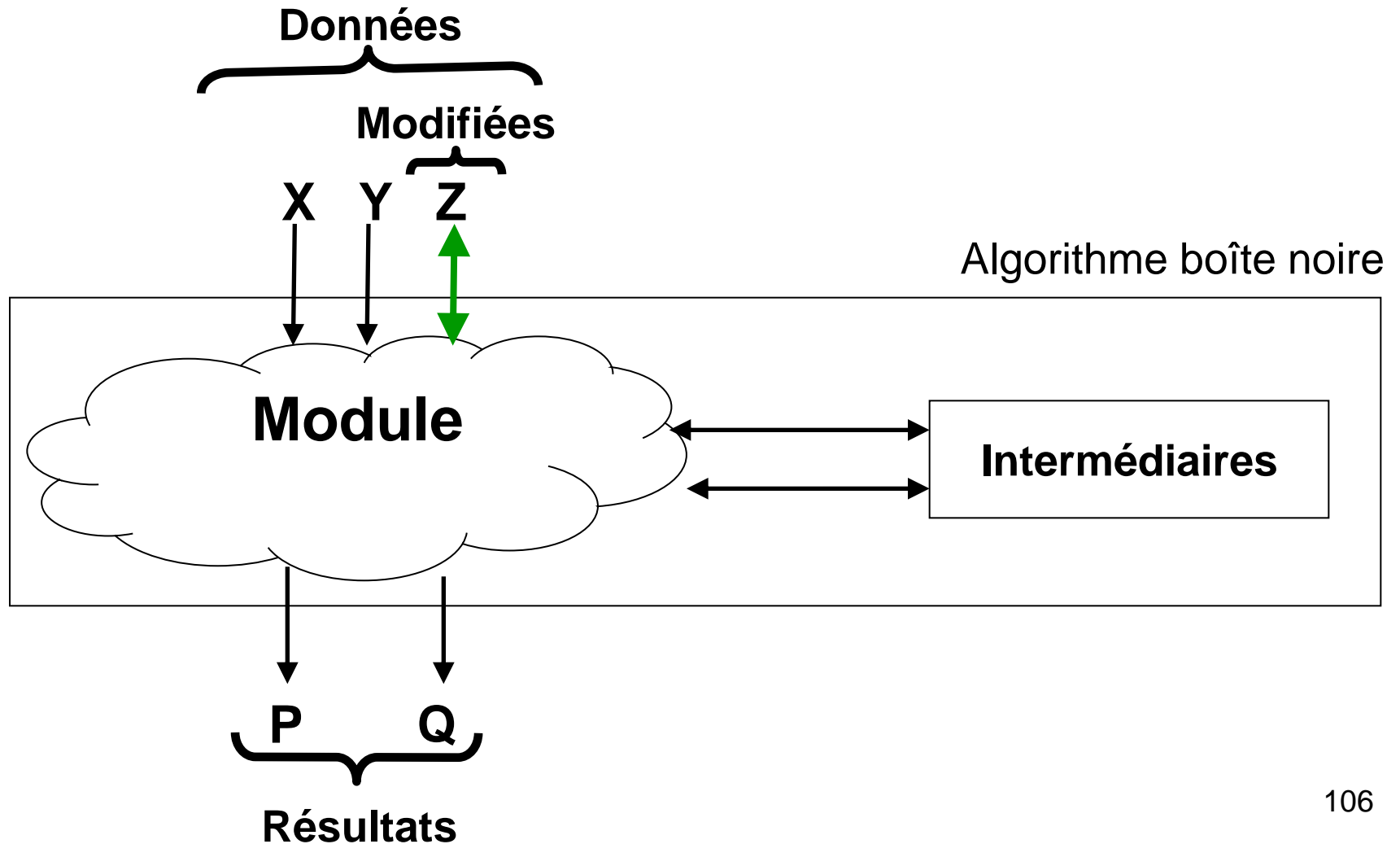
ENTÊTE:

MODULE:

La modification d'arguments

- Il est parfois nécessaire de changer la valeur d'une variable dans la liste d'arguments dans l'invocation d'un algorithme.
 - La DONNÉE correspondante de l'algorithme est appelée **paramètre modifié**. Elle est listée avec les **DONNÉES** et placée dans l'entête comme une DONNÉE, mais elle est aussi incluse dans une liste de **MODIFIÉES** séparée des **RÉSULTATS** normaux.
- Exemple: Écrivez un algorithme **Échange** qui échange les valeurs de deux variables X et Y.
 - Par exemple, si X=7 et Y=4, l'algorithme échange les valeurs pour produire X=4 et Y=7.
- Nous revenons à ce concept plus tard
 - La modification de paramètres de types primitifs n'est pas possible en Java
 - Mais peut se faire avec des Tableaux et Objets avec des variables de référence

$$(P, Q) \leftarrow \text{algorithme}(X, Y, Z)$$



Gabarit (format) complet pour les algorithmes

DONNÉES:
RÉSULTATS:
MODIFIÉES: *(optionnel)*
INTERMÉDIAIRES: *(optionnel)*
HYPOTHÈSES: *(optionnel)*
CONTRAINTES: *(optionnel)*
EN-TÊTE

(Liste des résultats) ← Nom_Algorithme (Liste des données)

MODULE

instruction
instruction
:
instruction

} **Bloc d'instructions**

(commentaires un peu partout!)

(les instructions peuvent être numérotées pour le traçage)

Exercice 3-2: Échanger deux valeurs



DONNÉES:

RÉSULTATS:

MODIFIÉES:

INTERMÉDIAIRES:

EN-TÊTE:

MODULE:

Variables, en résumé...

- Les **DONNÉES** sont les données en entrée. Elles varient d'une invocation à l'autre, c'est-à-dire que leurs valeurs peuvent varier pour chaque instance de problème.
- Les **RÉSULTATS** sont les réponses produites par l'algorithme.
- Les **INTERMÉDIAIRES** sont les autres variables utilisées.
- Les **MODIFIÉES** sont un sous-ensemble (peut-être vide) des variables **DONNÉES** dont les variables correspondantes de l'algorithme invoquant seront modifiées par l'algorithme invoqué.
- **NOTEZ:** Pour les **DONNÉES**, **RÉSULTATS**, et **INTERMÉDIAIRE**, les variables existent dans une mémoire de travail seulement durant l'exécution de l'algorithme, et n'existent pas entre les invocations de l'algorithme.

"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 } Bloc d'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
    }
}
```


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 LireRéal(), 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:

- $X \leftarrow \text{expression}$

- Java:

- `x = expression ;`

- Expressions

- $\text{Somme} \leftarrow N1 + N2 + N3$

- `somme = n1 + n2 + n3 ;`

- $\text{hypoténuse} \leftarrow \sqrt{x^2 + y^2}$

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

- Instruction d'invocation

- $\text{Moyenne} \leftarrow \text{RésMoy}(\text{Premier}, \text{Deuxième}, \text{Troisième})$

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

- $\text{Premier} \leftarrow \text{LireRéal}()$

- `premier = ITI1520.readDouble() ;`

Invocation en Java

Modèle logiciel: $X \leftarrow \text{Moy3}(10, J, K)$

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 traduits à des VARIABLES de Java.
 - Elles doivent être déclarées et données un type. Leur description sont traduits 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 placer cette instruction à toute autre endroit.
 - Malgré que les algorithmes permettent le retour de plusieurs résultats, la méthode Java ne peut que retourner 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 ici
```

```
    // 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
```

```
    }
```

```
}
```

« Pour comprendre un programme, vous devez
devenir à la fois le programme et la machine »
- A. Perlis

ITI 1520

Section 4: Traçage et débogage

Objectifs:

- Traçage manuel d'algorithmes
- Débogueur pour traçage de programmes

Note historique...

- Alan M. Turing, mathématicien anglais et père de l'informatique moderne, a conçu en 1936 une machine logique capable de résoudre **tous** les problèmes que l'on peut formuler en termes d'algorithmes et d'ordinateurs modernes: la machine de Turing
- Il est aussi à l'origine du test de Turing, très connu en intelligence artificielle
- La plus haute distinction en informatique au monde, décernée par l'ACM, porte son nom: Turing Award



Traçage d'un algorithme

- **Tracer** un algorithme, c'est l'exécuter manuellement, une instruction à la fois, en gardant trace de la valeur de chaque variable. Le traçage a pour but de détecter des erreurs potentielles **avant** le codage.
- Le traçage implique toujours une seule instance du problème (i.e. des valeurs précises pour les variables données). Vous pourriez avoir besoin de plusieurs traces (tests, avec des valeurs données **différentes**) d'un même algorithme pour trouver des erreurs.
 - Définissez des cas tests pour votre traçage, c'est-à-dire des résultats attendus pour un ensemble de valeurs de DONNÉES
 - Vous pouvez utiliser ces cas tests plusieurs fois pour vérifier votre logiciel.

Étapes du traçage

1. Numérotez chaque instruction de l'algorithme.
2. Préparez un modèle de programmation (utilisez autant de pages nécessaires) pour que vous puissiez « exécuter » votre algorithme.
3. Construisez un tableau de traçage avec
 - les variables dans les colonnes en commençant par les données, (les intermédiaires), puis les résultats selon leur ordre dans l'algorithme
 - les instructions dans les lignes en commençant par **valeurs initiales**
4. Dans le modèle de programmation, « créez » les variables et insérez les valeurs initiales des variables DONNÉES et ? Dans tous les autres variables. Remplissez le tableau en commençant par la première ligne (**valeurs initiales**). Indiquez pour chaque donnée la valeur lors de l'invocation de l'algorithme. Mettez '?' pour les autres variables.

Étapes du traçage (suite)

5. Remplissez ensuite la ligne correspondant à la première instruction et mettez à jour le modèle de programmation en changeant les valeurs des variables affectées par l'instruction. Pour chaque variable changée indiquant la nouvelle valeur dans le tableau de traçage (laissez les autres colonnes vides)
6. Procédez de la même façon pour les autres instructions jusqu'à la fin de l'algorithme.

Note: les valeurs d'une ligne représentent collectivement l'état du système à un moment donné.

Exercice 4-1 - Exemple de traçage



MoyNotes ← RésNotes(18, 23, 19)

Tracer une invocation

- Chaque fois qu'un autre algorithme est invoqué, il faut effectuer une trace séparée de l'algorithme appelé avec les données de l'invocation.
- Numérotez et référencez les pages de traces.
- Dans la colonne 1 du tableau de trace de l'algorithme où l'invocation est faite:
 - Indiquez le numéro de l'instruction d'invocation.
 - Le numéro de page où se trouve la trace de l'algorithme invoqué.
 - La correspondance des données et résultats entre l'instruction d'invocation et l'entête de l'algorithme invoqué (voir page précédente).
- Indiquez les valeurs retournées (résultats, modifiées) à la fin de l'algorithme invoqué, dans les colonnes correspondantes de la trace de l'algorithme invocateur.

Exercice 4-2 - Tracer un invocation

- Refaite le problème « moyenne sur 100 », mais cette fois-ci tracez l'algorithme Principal en utilisant le cas test suivant (montre l'interaction avec l'utilisateur)

S.V.P. tapez trois notes sur 25

23 16 21

La moyenne est 80 pourcent.

Exercice 4-2 - Tracer Principal (page 1) ?

- Trace: Principal

Exercice 4-2 - Tracer MoyNotes (page 2)

Exercice 4-3 - Problème de moyenne à nouveau!



- Refaite l'algorithmes « notes sur 100 », mais cette fois-çi en utilisant l'algorithme « moyenne ».

MoyNotes \leftarrow RésNotes(23, 16, 21)

DONNÉES:

RÉSULTATS:

INTERMÉDIAIRES:

EN-TÊTE:

MODULE:

Exercice 4-4 - Exemple de traçage (page 1)



-
- Trace: MoyNotes \leftarrow RésNotes(23, 16, 21)

Exercice 4-4 - Exemple de traçage (page 2) ?

Exercice 4-5 - Chiffres inversés



- Écrivez un algorithme qui, étant donné un nombre positif de 2 chiffres N , inverse les chiffres de N pour constituer un nouveau nombre $N_{\text{inversé}}$.
- Supposez qu'il existe un algorithme avec l'entête suivante:
 $(\text{Premier}, \text{Second}) \leftarrow \text{Chiffres}(X)$
qui retourne dans *Premier* le chiffre de gauche et dans *Second* le chiffre de droite du nombre à 2 chiffres X .

DONNÉES:

RÉSULTATS:

INTERMÉDIAIRES:

EN-TÊTE:

MODULE:

Exercice 4-6 - Trace chiffres inversés ?

Exercice 4-7 - Joindre 4 nombres



- Écrivez un algorithme qui prend 4 entiers positifs et les combine (joint) en un seul nombre.
 - Ex: avec 11, 35, 200, et 7 l'algorithme devrait produire 11352007.
- Tracez votre algorithme avec les données ci-dessus.
- Vous pouvez supposer que vous disposez de l'algorithme:

$C \leftarrow \text{Joindre}(A, B)$

Données: A, B (*deux entiers positifs*)

Résultat: C (*le nombre constitué des chiffres de A suivis des chiffres de B*)

- Exemple: $\text{Joindre}(120, 43)$ produit 12043

Tracer des programmes

- Lors du traçage d'un programme, l'un des outils les plus utiles se nomme « **débogueur** ». Cet outil peut fournir la trace d'un programme en exécution.
 - Souvent, ce processus demande de dire au compilateur d'ajouter de l'information supplémentaire pour le débogueur.
- Deux modes d'opération:
 - Exécuter jusqu'à un « **breakpoint** »: le programme s'arrête à un endroit prédéfini dans le code source.
 - Mode « **étape par étape** » (*single step*): le programme s'exécute une instruction à la fois.
- Lorsque le programme est arrêté, vous pouvez inspecter les valeurs courantes des variables.
 - Vous pouvez donc vérifier que les valeurs des variables correspondent à celles que vous avez dans votre trace d'algorithme, à la rangée attendue.

Utilisation d'un débogueur

Plusieurs débogueurs offrent quatre façons de faire progresser le programme une fois arrêté:

1. **Resume**: Le programme continue jusqu'au prochain point d'arrêt ou jusqu'à la fin de l'exécution.
2. **Step into**: passe à la prochaine instruction, même si celle-ci se trouve dans une autre méthode Java. Pour un débogage détaillé.
3. **Step over**: Passe à la prochaine instruction dans la méthode courante. Utilisée très fréquemment.
4. **Step out**: Va jusqu'à la fin de la méthode courante.

"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

Note historique...

- 1945 : Un insecte coincé dans les circuits bloque le fonctionnement du calculateur Mark I. L'informaticienne Grace Murray Hopper décide alors que tout ce qui arrête le bon fonctionnement d'un programme s'appellera « bug »!
- 1951 : Invention du premier compilateur (AO) pour générer un programme binaire à partir du code source d'un programme
- Elle sera l'une des principales créatrices de l'un des premiers langages de programmation: COBOL.



92 Ph 5

9/9

0800 Anttan

1000

1300 (032) MP-MC 1.98210000 1.051846995 check

(033) PRO 2 2.130476415

check 2.130676415

Relays 6-2 in 033 failed special speed test in relay

Relays changed

1100 Started Cosine Taps (Sine check)

1525 Started Multi-Adder Test.

1545

Relay #70 Panel F (moth) in relay.

1630/1630 Anttan started.

1700 closed down.

First actual case of bug being found.

Relay 3145

Relay 3370

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.

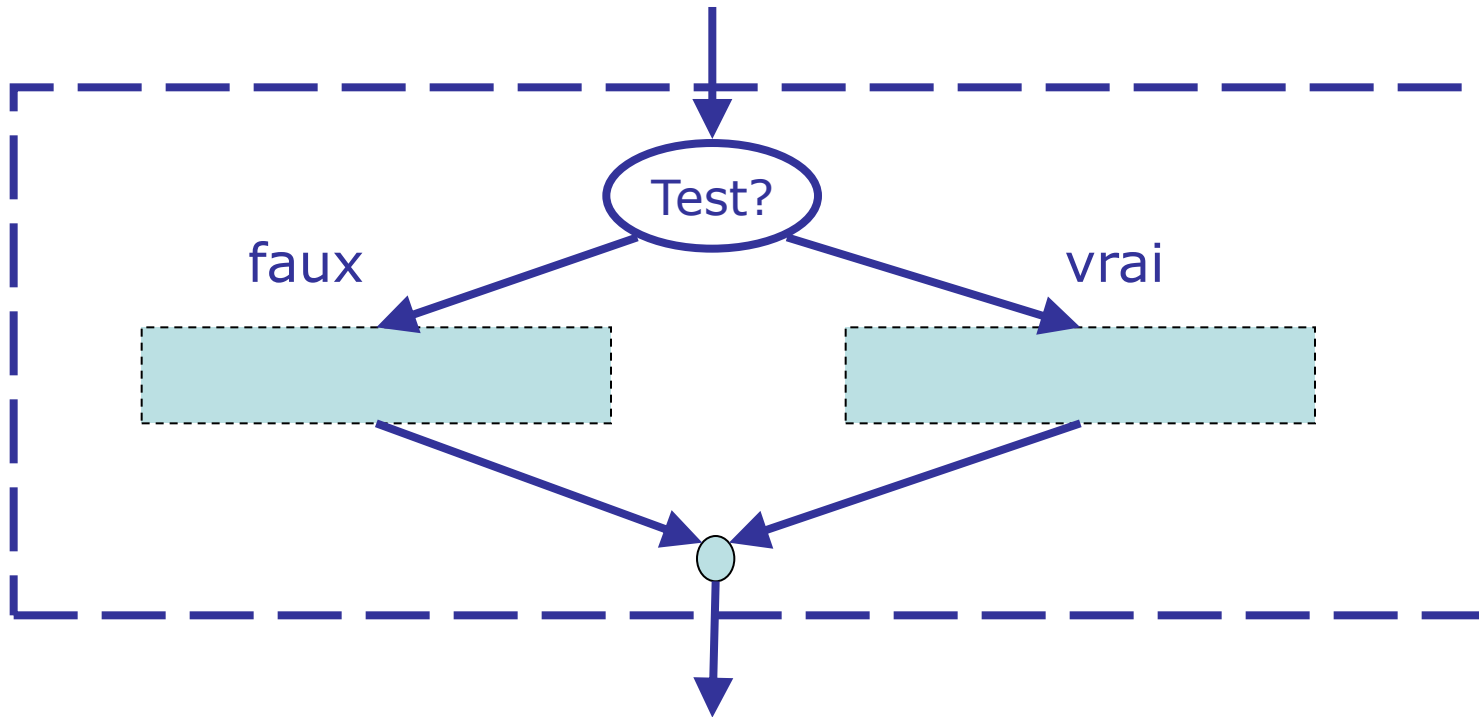
DONNÉES: X, Y (*deux nombres*)

RÉSULTAT: M (*maximum entre X et Y*)

EN-TÊTE: $M \leftarrow \text{Max2}(X, Y)$

- 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).



- 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)



Contenu d'un bloc d'instruction

- Instruction simple (invocation, affectation)
- Instruction vide (\emptyset = "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

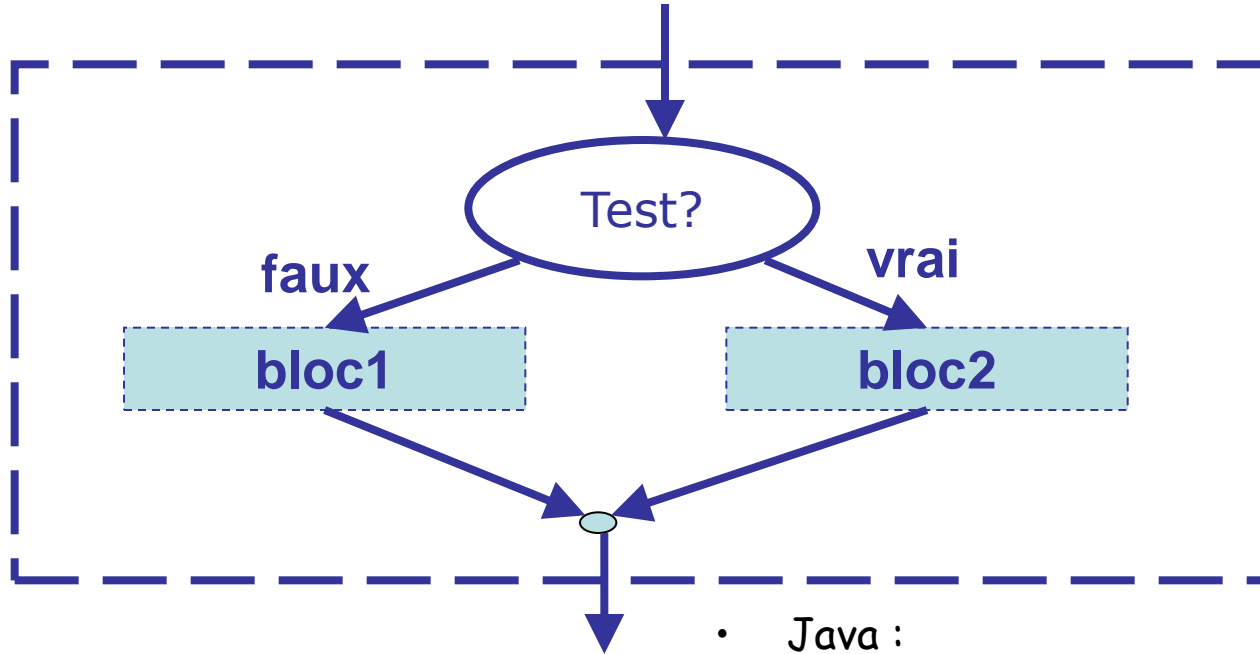


-
- Comment écrire l'algorithme trouvant le maximum (M) entre deux valeur (X et Y) dans notre modèle logiciel?

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



• Java :

```
if (Test)
{
    // Instructions
}
else
{
    // Instructions
}
```

Bloc d'instruction 2 {

Bloc d'instruction 1 {

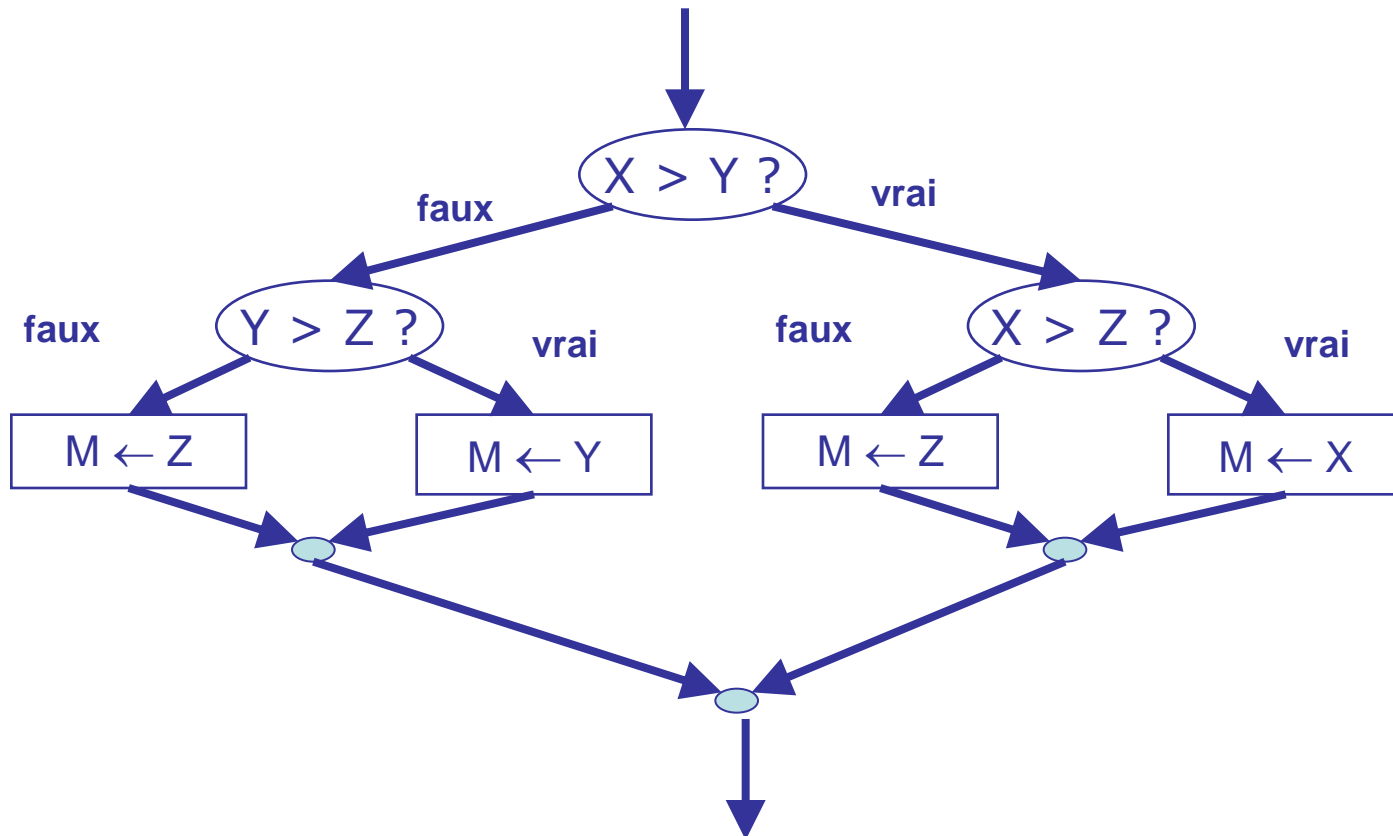
Exercice 5-3 - Traduction de branchements 1



Données: X, Y, Z (*trois nombres*)
Résultat: M (*la plus grande des données*)
En-tête: $M \leftarrow \text{Max3}(X, Y, Z)$

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

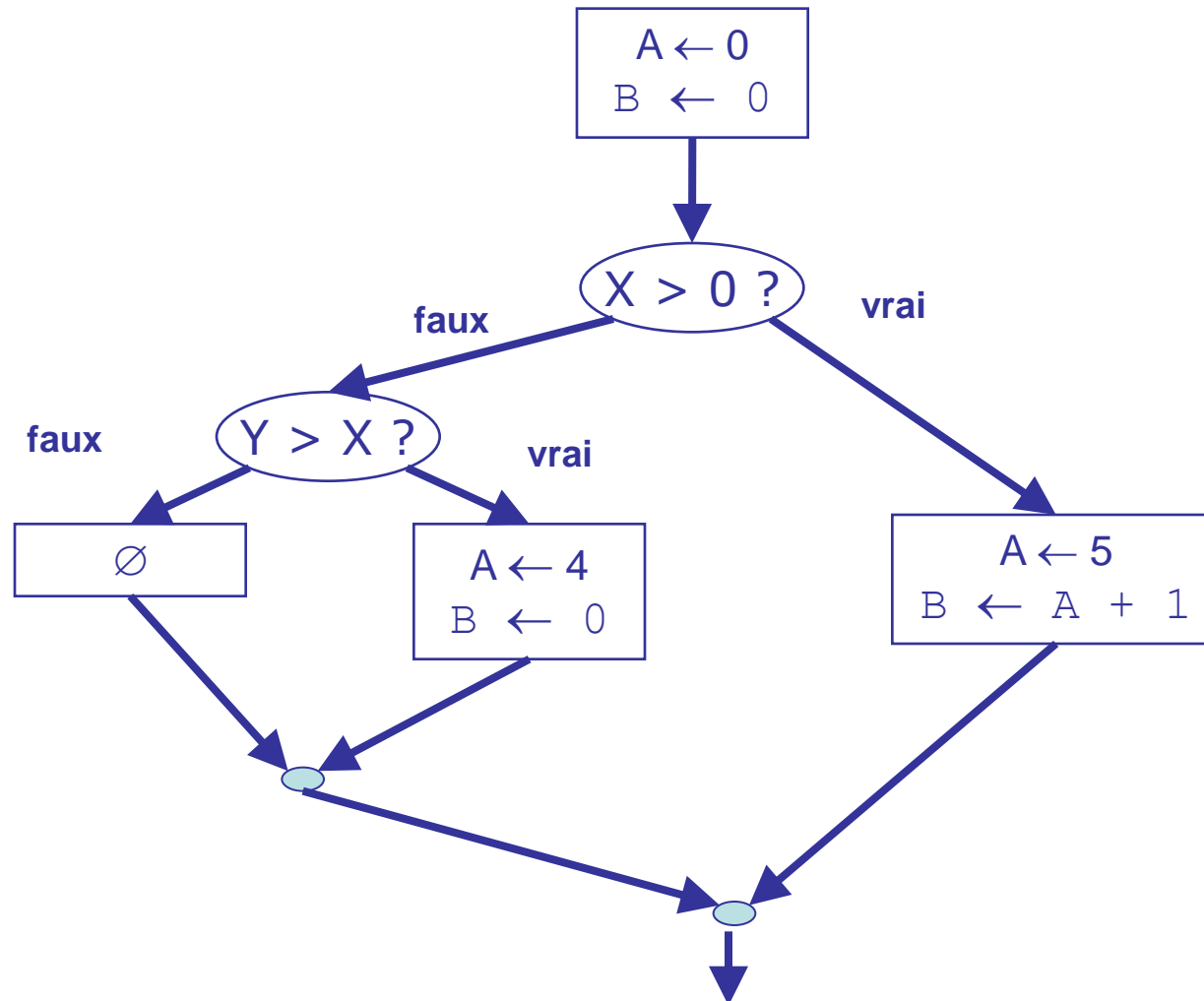
Solution avec branchements imbriqués



Exercice 5-4 - Traduction branchements imbriqués



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



Exercice 5-5 - Traduction d'un bloc d'instruction



Traçage d'algorithmes avec branchements

- Lors du traçage d'algorithmes avec tests (branchements ou boucles):
 - Numérotez les tests et les instructions.
 - Ajoutez une ligne pour les conditions évaluées ainsi que le résultat de l'évaluation (*vrai* ou *faux*).
 - Indiquez uniquement les instructions exécutées dans la trace.

Exercice 5-6 - Trace du problème maximum de trois nombres, version 2



-
- Trace: $\text{MAX3}(5, 11, 8)$

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
 - $X \leftarrow \text{VRAI}$
 - $Y \leftarrow \text{FAUX}$
- Le résultat d'un test (expression Booléenne) peut aussi être affecté à une variable Booléenne:
 - $X \leftarrow (A < 0)$

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 $\hat{\text{Age}} = 15$, alors:
 - L'expression $\hat{\text{Age}} > 16$ sera évaluée à FAUX, et NON ($\hat{\text{Age}} > 16$) aura comme valeur VRAI.
 - L'expression $\hat{\text{Age}} < 65$ sera évaluée à VRAI, et NON ($\hat{\text{Age}} < 65$) aura comme valeur FAUX.

Exercice 5-10 - Encore des expressions Booléennes complexes



Supposons que $X = 5$ et que $Y = 10$.

Expression	Valeur
$(X > 0) \text{ ET } (\text{NON } Y = 0)$	
$(X > 0) \text{ ET } ((X < Y) \text{ OU } (Y = 0))$	
$(\text{NON } X > 0) \text{ OU } ((X < Y) \text{ ET } (Y = 0))$	
$\text{NON } ((X > 0) \text{ OU } ((X < Y) \text{ ET } (Y = 0)))$	

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!

Note historique...

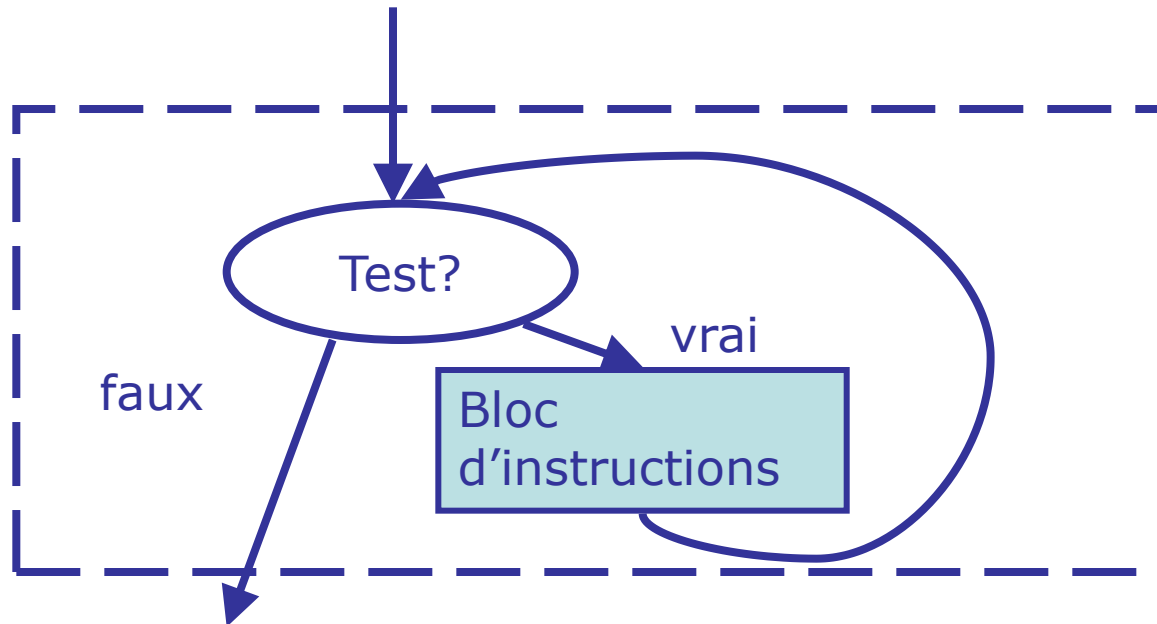
- Donald Knuth, informaticien américain, est l'un des pionniers du domaine de l'analyse d'algorithmes.
- Il est l'auteur de l'un des livres les plus respectés en informatique (The Art of Computer Programming) et du système de composition pour textes (scientifiques) TeX.



- Edsger Dijkstra, informaticien hollandais, a développé l'algorithme du chemin le plus court (qui porte son nom) et du concept de sémaphore pour les programmes et processeurs parallèles.
- Son article de 1968, "Go To Statement Considered Harmful" a révolutionné l'utilisation de l'instruction GOTO au profit des structures de contrôle telles que la boucle while.

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):

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

Exercice 6-2: Boucle "déterminée"

DONNÉE:

RÉSULTAT:

INTERMÉDIAIRE:

HYPOTHÈSE:

EN-TÊTE:

MODULE:

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.

Exercice 6-3: Boucle "indéterminée"

DONNÉES:

RÉSULTATS:

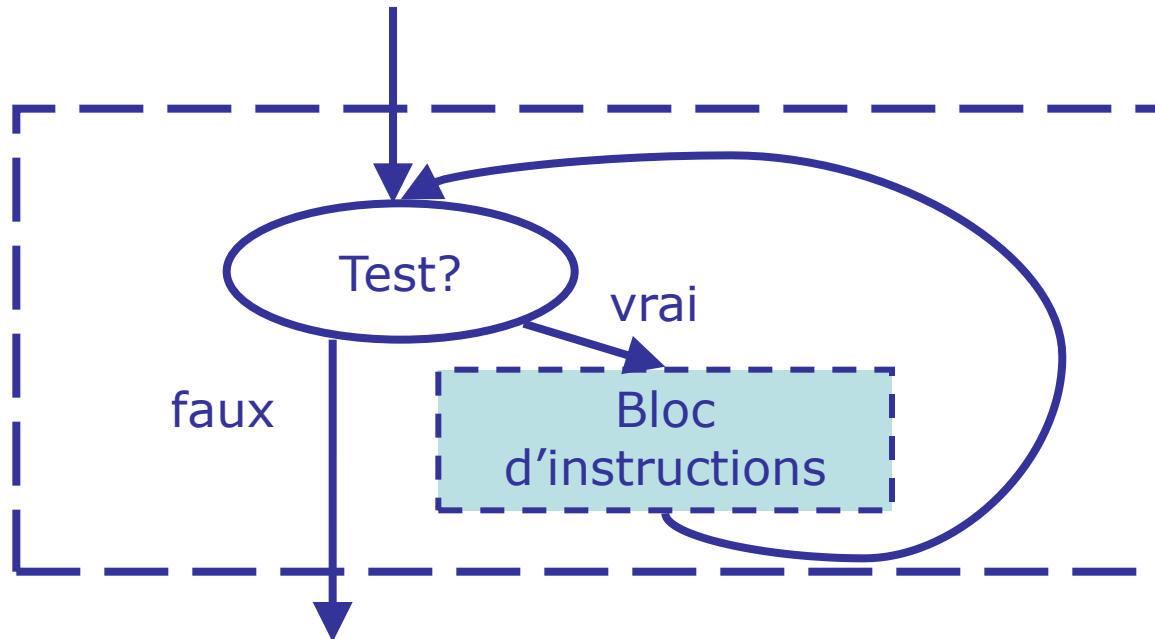
INTERMÉDIARES:

HYPOTHÈSES:

EN-TÊTE:

MODULE:

Traduction de boucles au Java



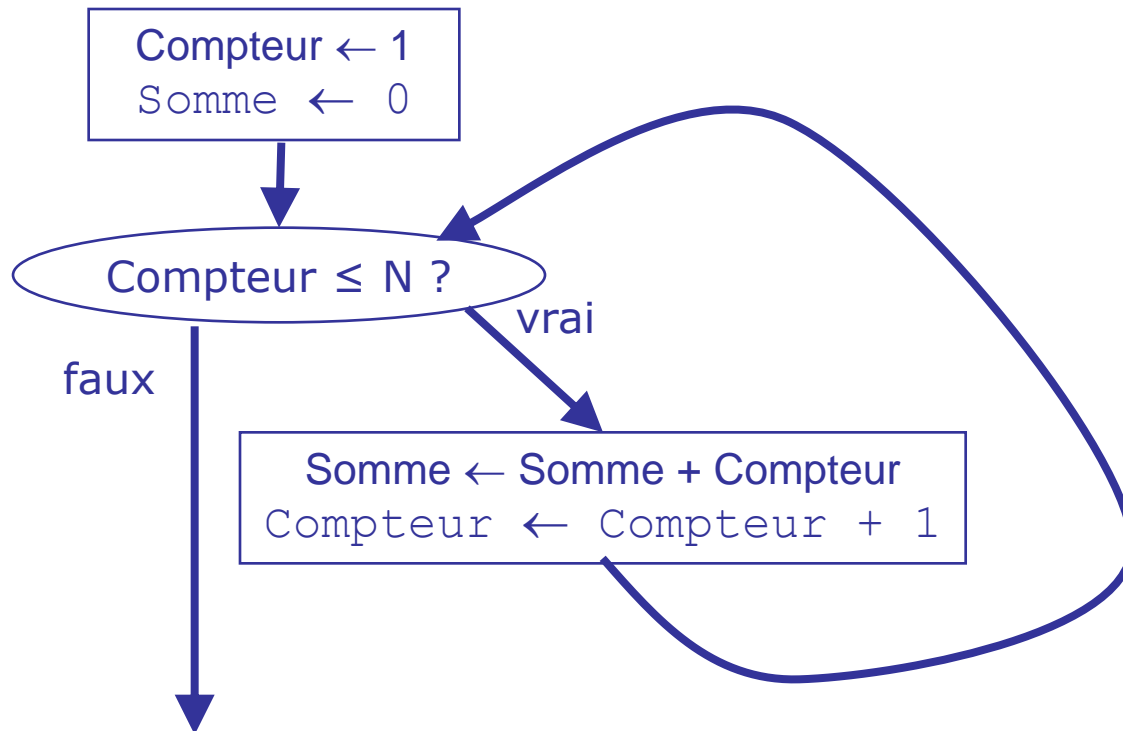
- Java

```
while (Test)
{
    // Instructions
}
```

Bloc d'instructions

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 \leftarrow Somme1àN(N)
MODULE:



Exercice 6-4: Traduire la boucle au Java ?

```
import java.io.* ;
```

```
class 
```

```
{
```

```
    public static void main (String args[ ])
```

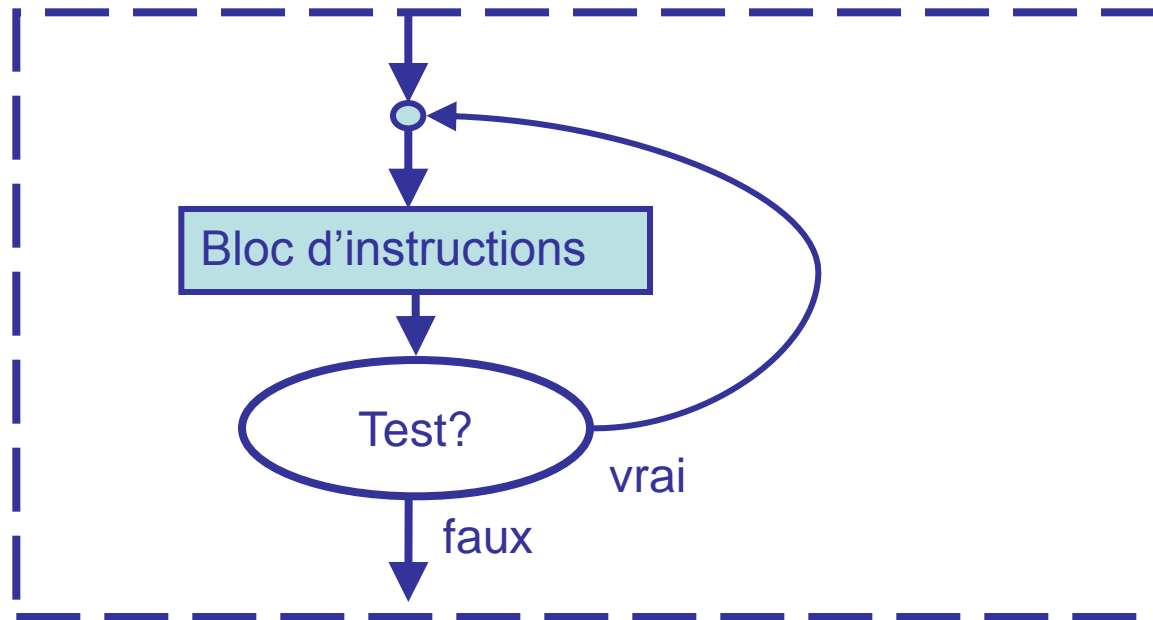
```
{
```

```
}
```

```
}
```

Instruction de boucle post-test

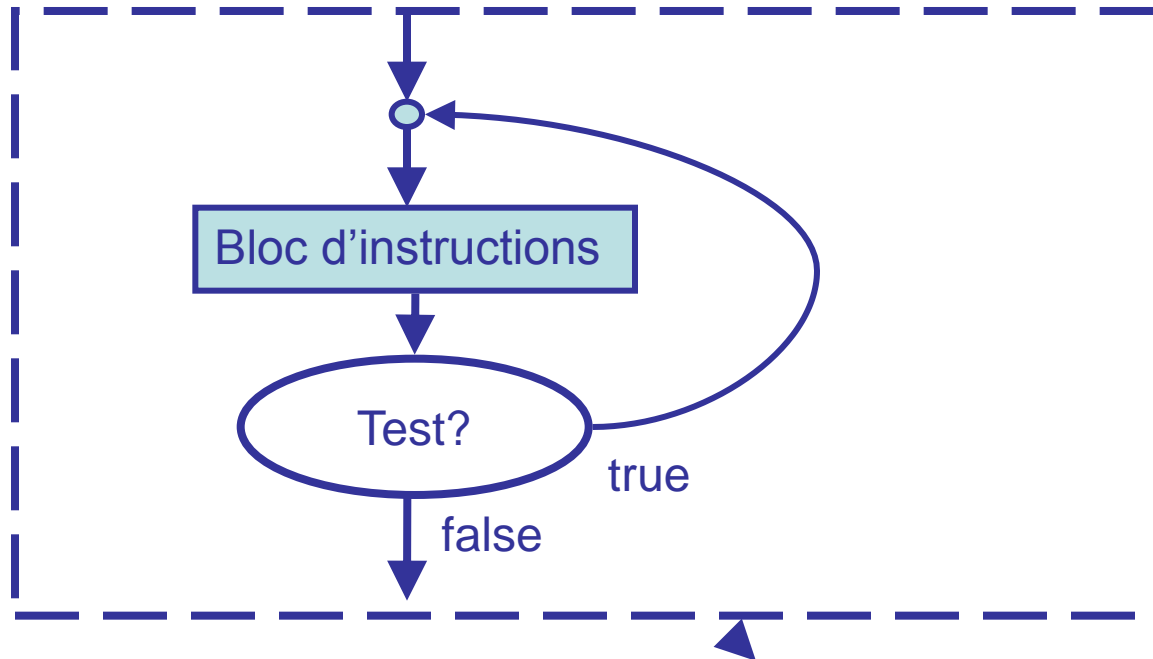
- Dans l'instruction de boucle post-test, le test se fait après l'exécution du bloc d'instructions.



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

L'instruction de boucle post-test

Traduction au Java



• Java:

```
Block d'instructions {  
    do  
    {  
        // Instructions  
    }  
    while(test);  
}
```

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

}
```

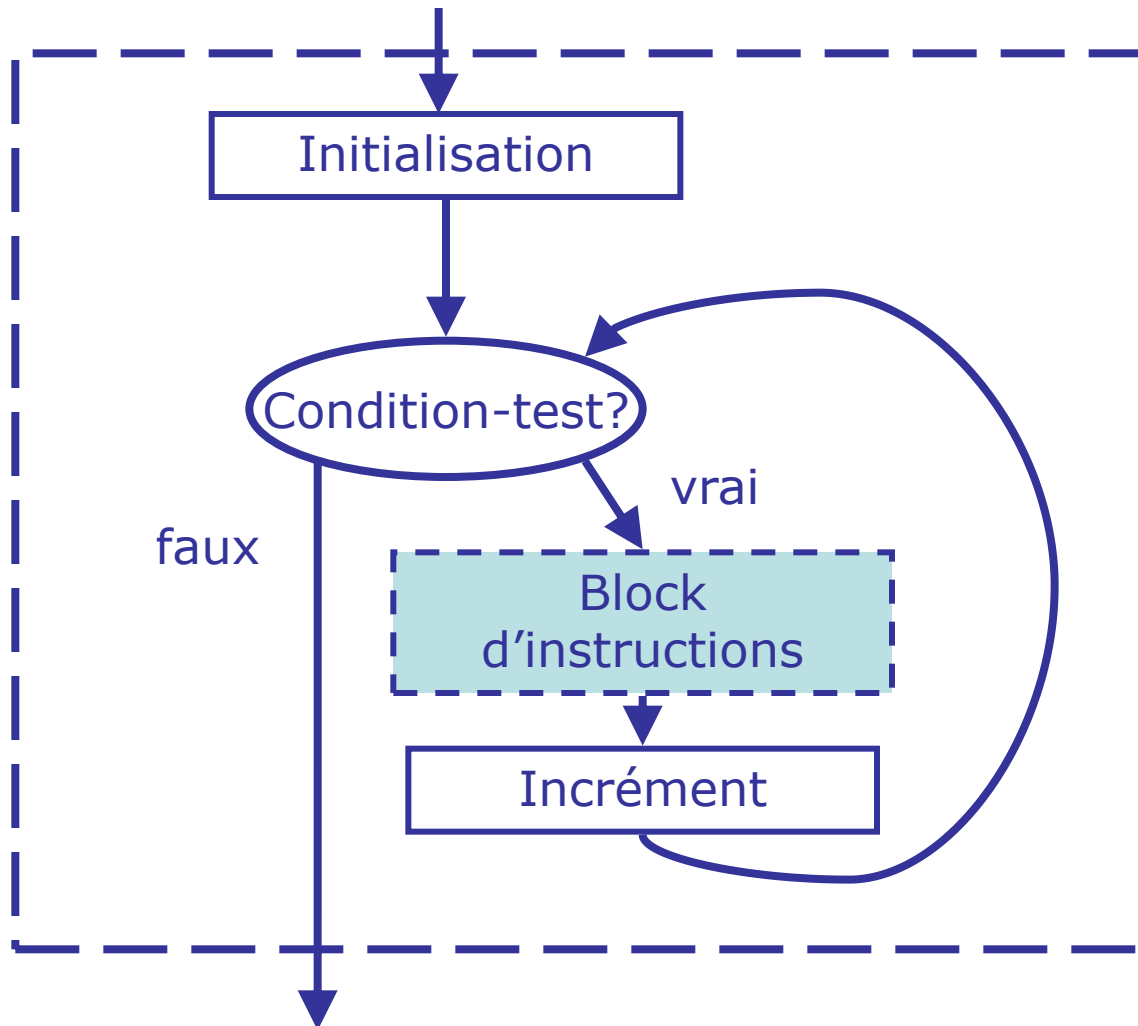
Boucle FOR

- 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:

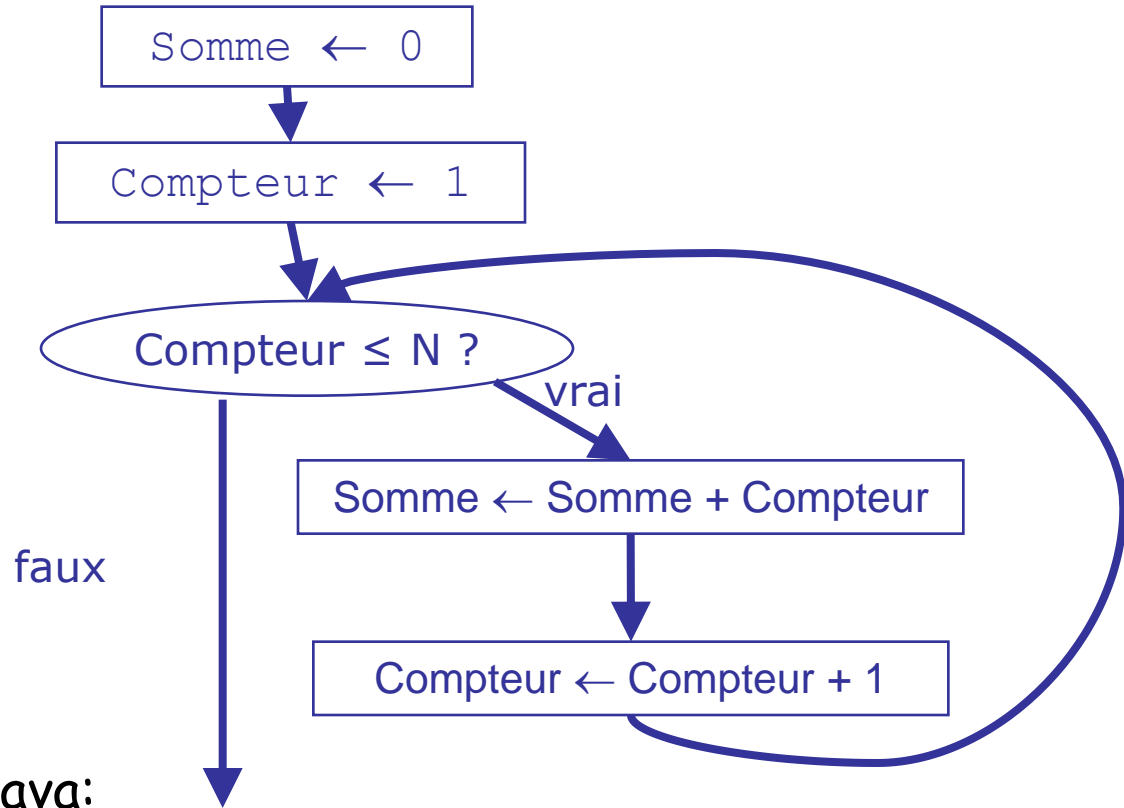
```
for (<initialisation>; <condition-test>; <incrément>)  
{  
    // instructions  
}
```

- 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



- Traduction à Java:

Problème avec variables simples...

- Supposons le module d'un algorithme lisant 5 entiers et qui les affiche dans l'ordre inverse:

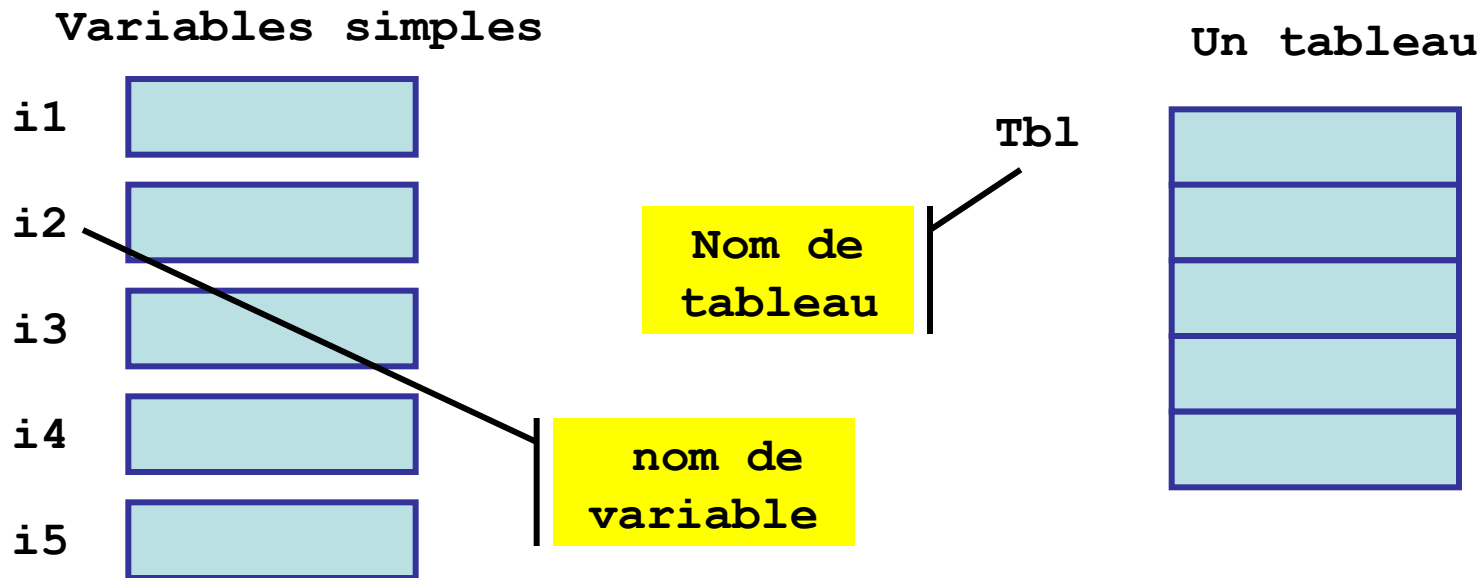
Module:

```
i1 ← LireEntier()  
i2 ← LireEntier()  
i3 ← LireEntier()  
i4 ← LireEntier()  
i5 ← LireEntier()  
AfficheLigne(i5)  
AfficheLigne(i4)  
AfficheLigne(i3)  
AfficheLigne(i2)  
AfficheLigne(i1)
```

- Qu'arrive-t-il lorsqu'on a 1000 entiers? X entiers?

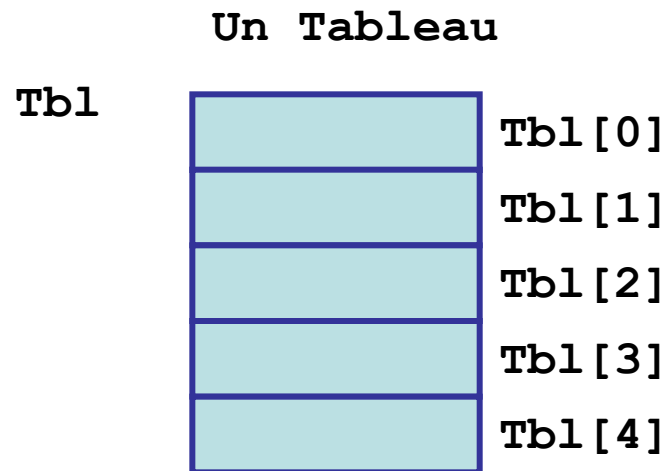
Tableaux informatiques

- "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.

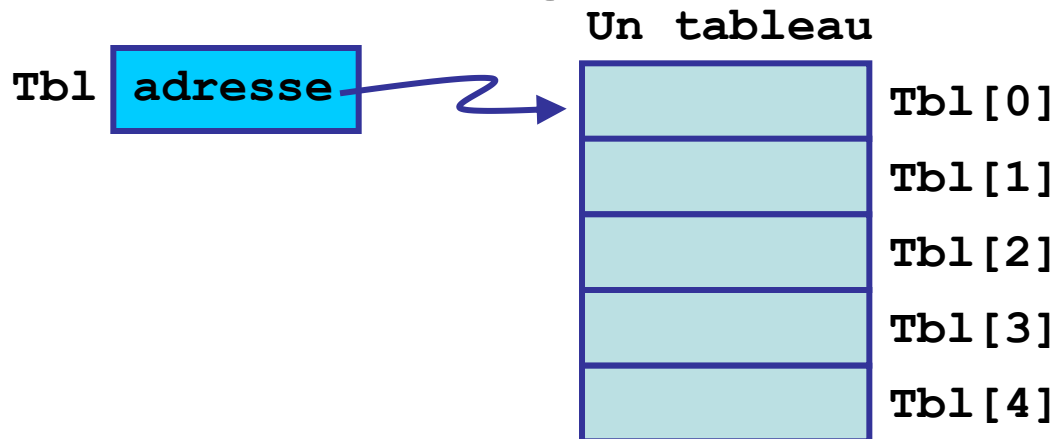


Tableaux informatiques (*suite*)

- 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. les 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

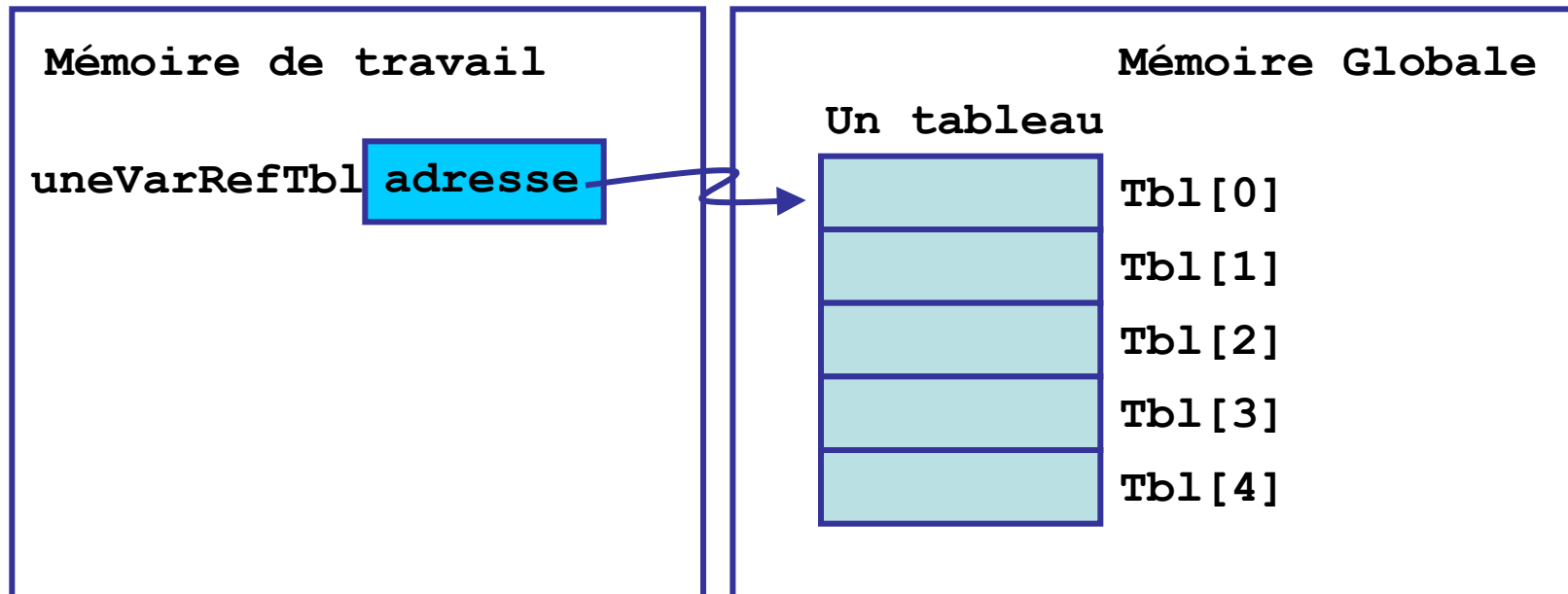
- Que représente le nom d'un tableau?
- Il peut représenter l'adresse en mémoire où se trouve le tableaux.
 - Ceci est semblable à un nom de variable, mais se fait traité différemment d'un nom de variable
 - Cette approche est utilisée dans des langages tel que le C
 - L'image dans les acétates précédentes reflète cette représentation.
- Le nom de tableau peut 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` \leftarrow 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` \leftarrow 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)

$T[2] =$

$T[K] =$

$T[2 * K - 1] =$

$T[T[0] + 1] =$

- $T[\text{expression}]$ 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 ($1+2+\dots+N$).

? 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 créés dans la mémoire de travail.

Mémoire pour tableaux

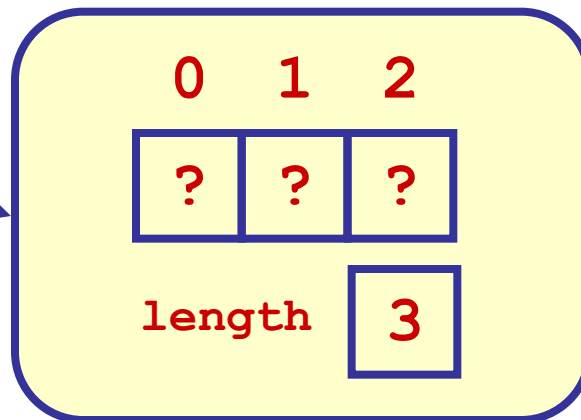
```
double[] unTableau ;
```

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



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;
```

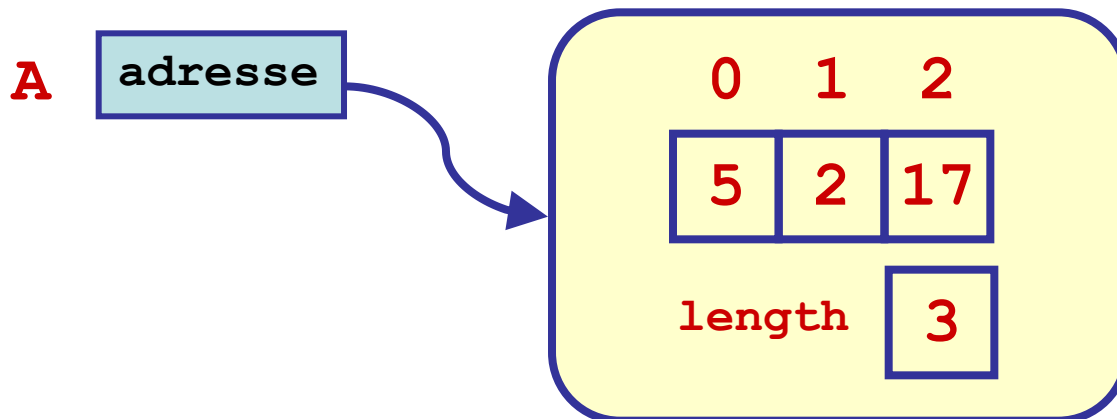
```
intTableau[2] = 4;
```

- 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

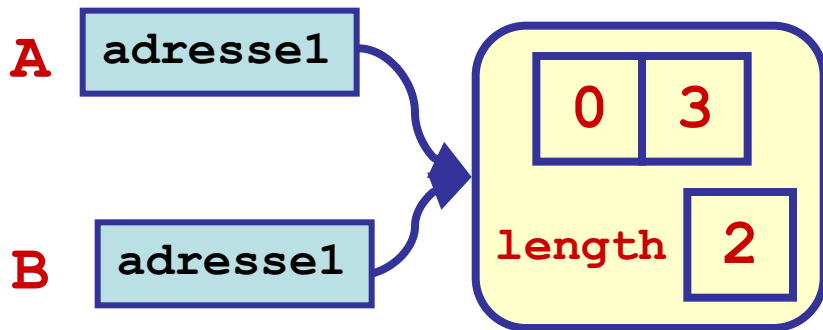
- Un type tableau est un **type de référence**, à cause du « *pointeur* » vers le tableau.
- Il est important de distinguer la référence (pointeur) de l'item référencé.
 - Dans le diagramme suivant, **A** est la référence, et le tableau est ce qui est référencé.
 - Java ne nous permet pas de regarder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



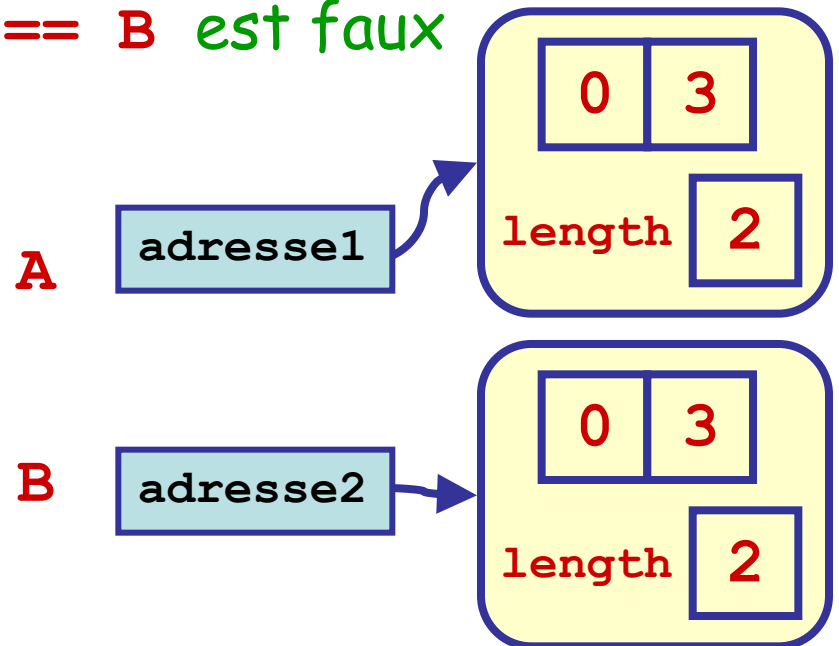
Types de référence

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

A == B est vrai

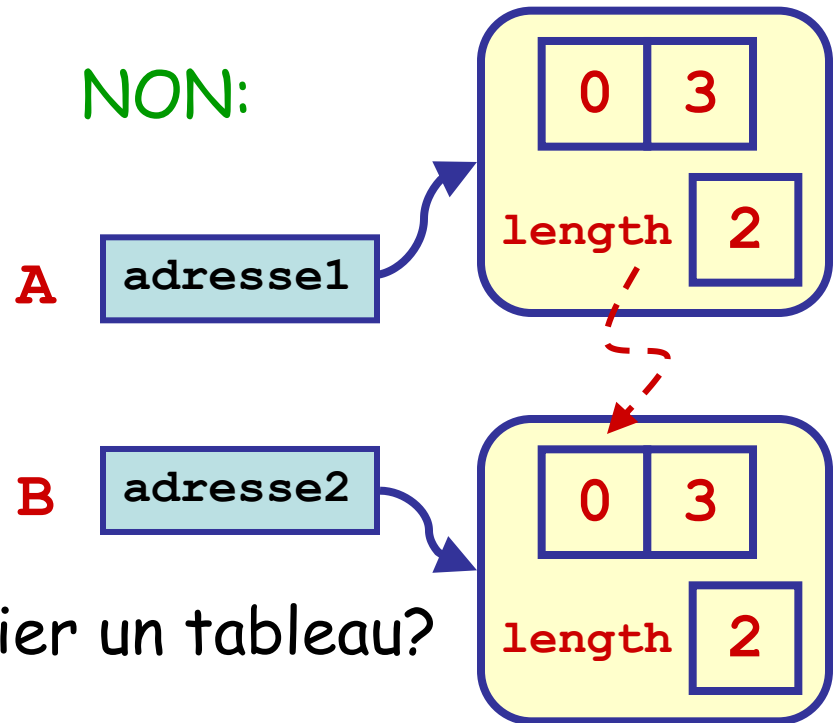
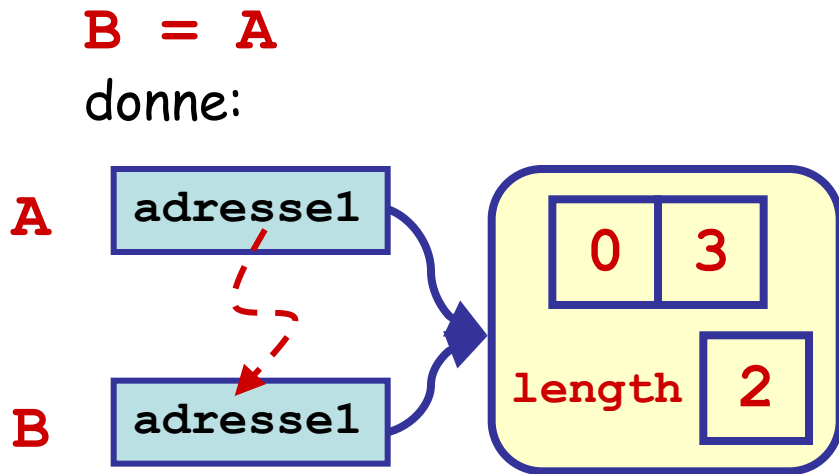


A == B est faux



Types de référence

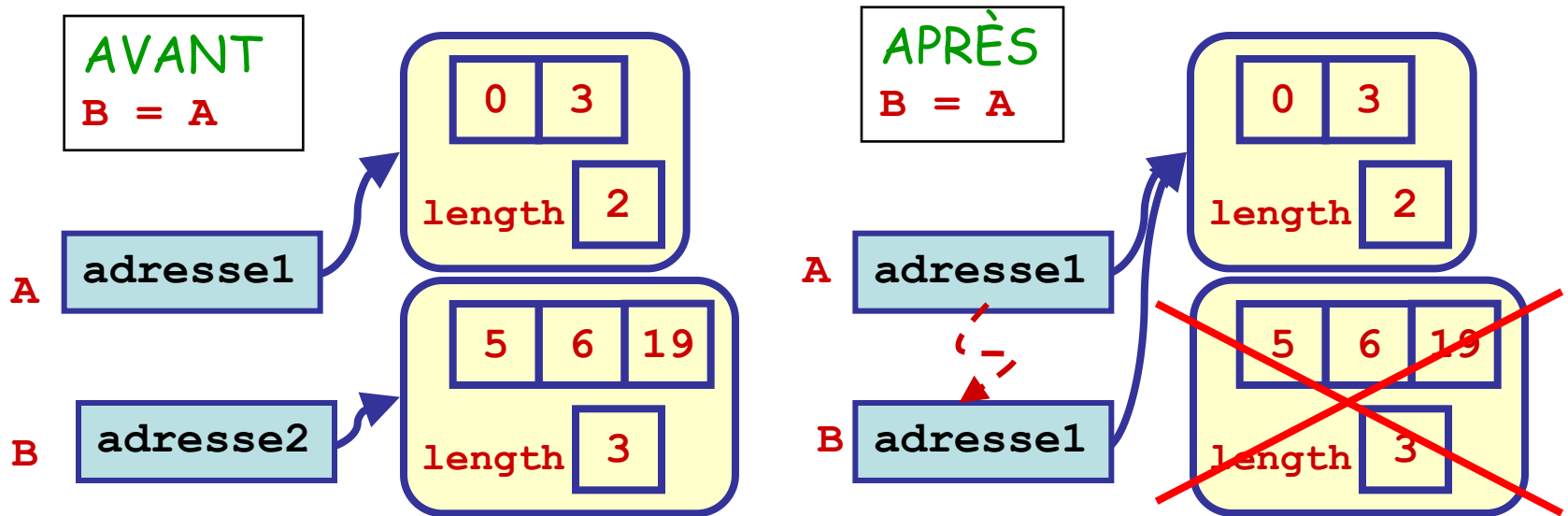
- L'affectation ne copie que la référence, pas l'objet référencé.



- 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: [AssiginationTableaux.ppt](#)

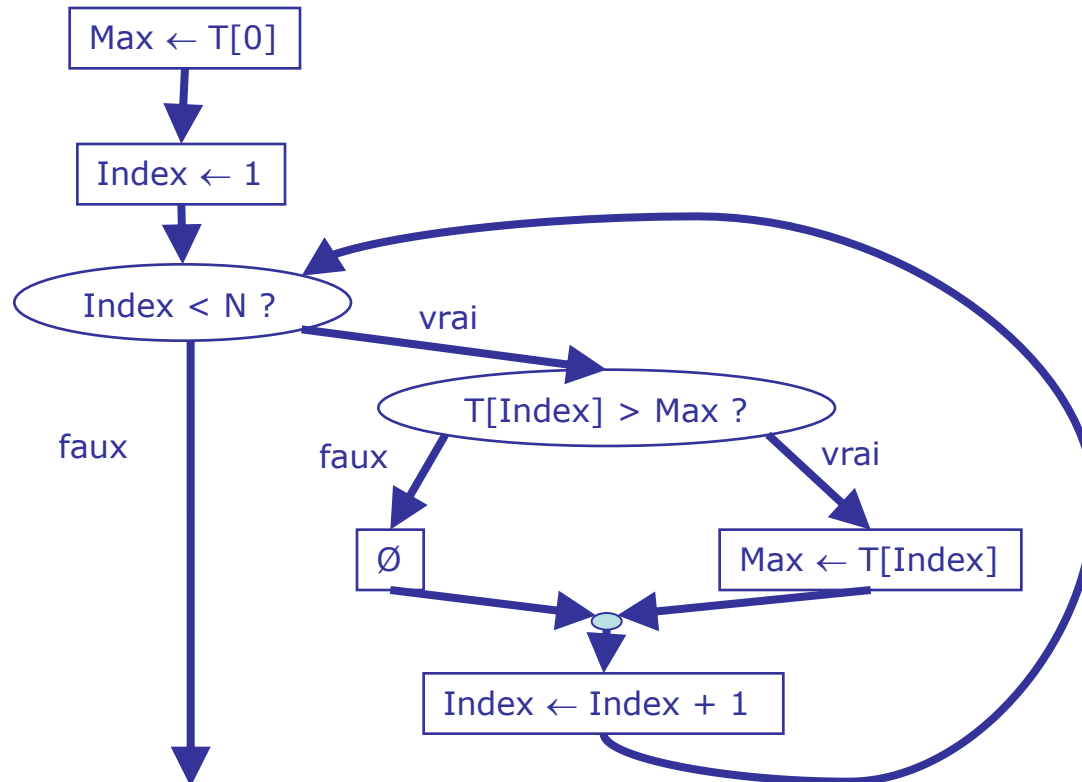
“Les objets peuvent être classifiés scientifiquement en trois catégories: ceux qui fonctionnent, ceux qui se brisent, et ceux qui se perdent.” - R. Baker

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
MODULE Max \leftarrow MaxDansTableau(T, N)



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 ont 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

Note historique...

- 1976 : Steve Jobs et Steve Wozniak créent leur premier ordinateur personnel qu'ils baptisent Apple I.
- L'ordinateur sera vendu pour 666.66 \$ avec 256 octets de ROM, 4 K octets de RAM et une sortie vidéo sur téléviseur.



- En juin 1975, Bill Gates et Paul Allen renomment leur compagnie Traf-O-Data en **Microsoft**.
- Créateurs de MS-DOS, de **Windows**, du Basic-Microsoft puis de **Visual Basic**.

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.

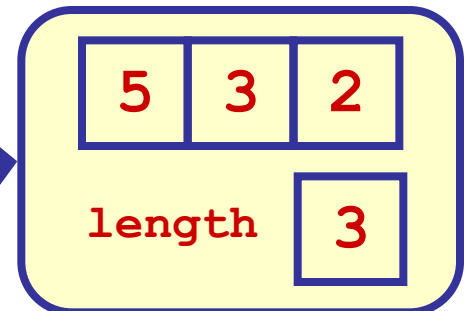
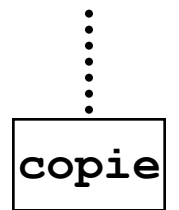
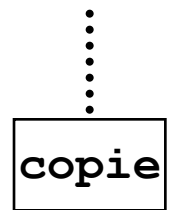
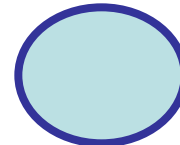
Passage de types primitifs et de référence

Invocateur:

`m(unInt, unTableau)`

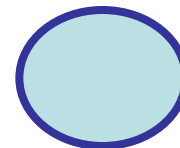
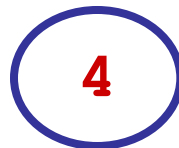
`unInt`

`unTableau`



Méthode invoquée:

`m(int x, int[] y)`



`x`

`y`

Qu'arrive-t-il
si on a:

`y[2] = 5;`

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 7-1: Trace (Table 2)



échangeTbl(a, i, 5-i)
 ↑ ↓ ↓
échangeTbl(x, i, j)

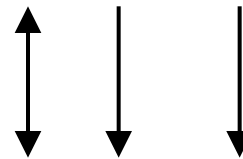
Instruction	x	i	j	temp	Tableau de Table 1
Valeurs initiales					
1. temp = x[i]					
2. x[i] = x[j]					
3. x[j] = temp					



Exercice 7-1: Trace (Table 3)



échangeTbl (a , i , 5-i)



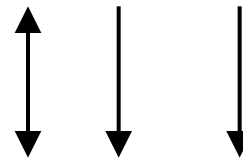
échangeTbl (x , i , j)

Instruction	x	i	j	temp	Tableau de Table 1
Valeurs initiales					
1. temp = x[i]					
2. x[i] = x[j]					
3. x[j] = temp					

Exercice 7-1: Trace (Table 4)



échangeTbl (a , i , 5-i)



échangeTbl (x , i , j)

Instruction	x	i	j	temp	Tableau de Table 1
Valeurs initiales					
1. temp = x[i]					
2. x[i] = x[j]					
3. x[j] = temp					

Programmes avec plus d'une classe

- Un programme peut avoir plus d'une classe. Si vous conservez toutes les classes d'un programme dans un même répertoire, n'importe quelle de ces classes peut invoquer une méthode **public** de n'importe quelle autre classe dans le répertoire.
- Quand une méthode dénotée **static** est invoquée, il faut utiliser le nom de la classe avec l'**opérateur point** (.).
 - Par exemple, en ayant la classe **ITI1520** dans le même répertoire que vos programmes, vous pouvez invoquer la méthode **readInt()** avec:
ITI1520.readInt();

Classes « librairie »

- Au lieu de placer toutes vos méthodes dans la même classe que **main** (la classe qui contient votre programme principal), il vaut mieux les séparer en groupes cohérents et placer chaque groupe dans une classe séparée.
- Ces classes ne seront pas des programmes car elle n'auront pas de méthode **main**. Chacune sera une petite librairie de méthodes pouvant être utilisées par d'autres méthodes. **ITI1520.java** est un exemple de classe « librairie ».
- De telles classes peuvent être compilées séparément, mais elles ne peuvent pas être exécutées comme programmes autonomes. Elles doivent être compilées avant de compiler toute autre classe les utilisant.

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.

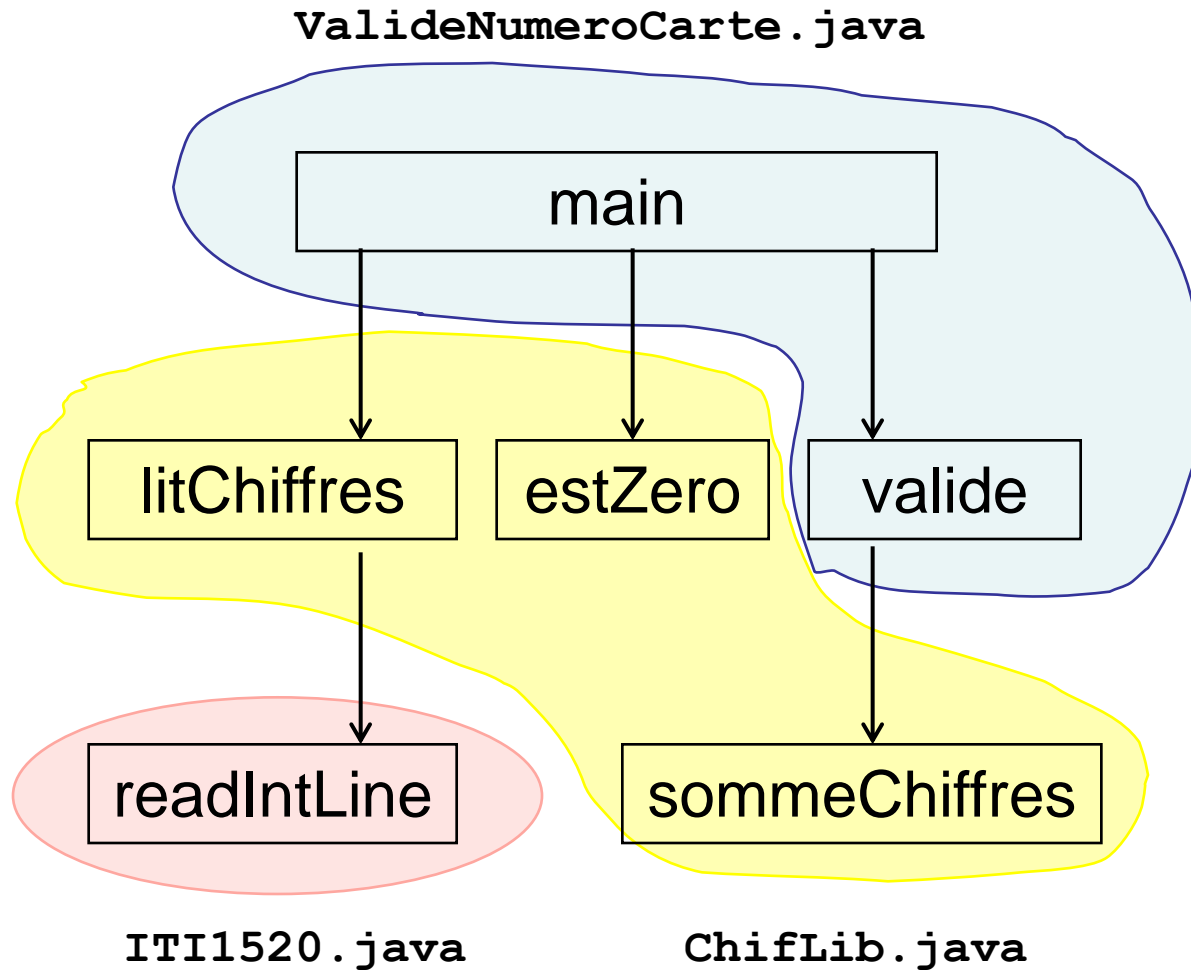
Exercice 7-2: Structure de données pour numéros

- 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 validité
    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 😊

Récurtivité

- Définition: voir « Récurtivité ».

ITI 1520

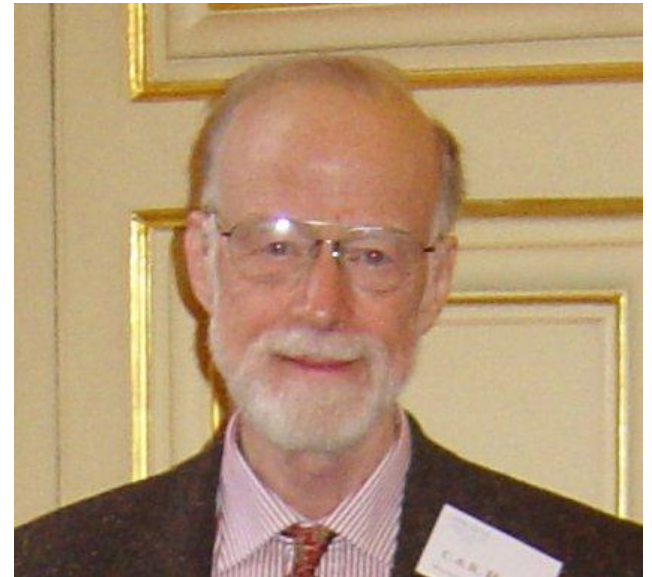
Section 8: Récurtivité

Objectifs:

- Définition et illustration de la récurtivité
- Utilisation systématique
- Exemples

Note historique...

- Charles Antony Richard (Tony) Hoare, informaticien anglais, a développé en 1960 l'algorithme de tri (récuratif) le plus utilisé: Quicksort
- Il a aussi développé la logique de Hoare utilisée en génie logiciel pour la vérification de programmes et la programmation par contrats
- Il est aussi à l'origine du langage concurrent Communicating Sequential Processes



*"Inside every large problem is
a small problem struggling to get out."*

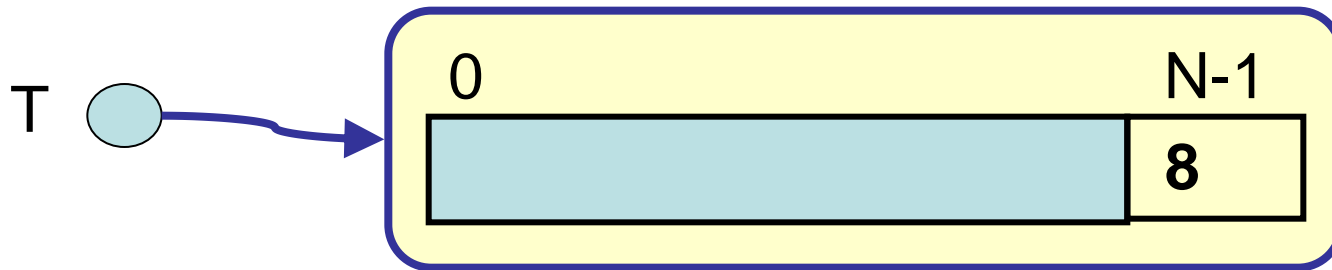
-- C.A.R. Hoare

Récurtivité

- La **récurtivité** est une technique de résolution de problème à l'aide d'un sous-problème plus petit; un paramètre (habituellement un entier) du sous-problème est plus petit.
- Avec la récurtivité, le sous-problème est semblable au problème initial, mais en plus simple.
- Quand le paramètre est assez petit (le **cas de base**), le problème peut être résolu directement.
- Si le paramètre est grand, le **problème** est reformulé en terme d'une sous-problème avec un paramètre plus petit.
 - Il faut trouver la solution problème à l'aide de la solution du sous-problème.
 - Si le paramètre du sous-problème est encore trop grand, il faut le réduire jusqu'à nous rejoignons le cas de base.
- Le problème et les sous-problèmes sont résolu avec plusieurs exécutions d'un seul sous-programme (algorithme/méthode) qui s'invoque lui-même.
 - On peut ainsi parlé d'instance d'exécution du sous-programme.

Récurtivité: Exemple 1 (idée de base)

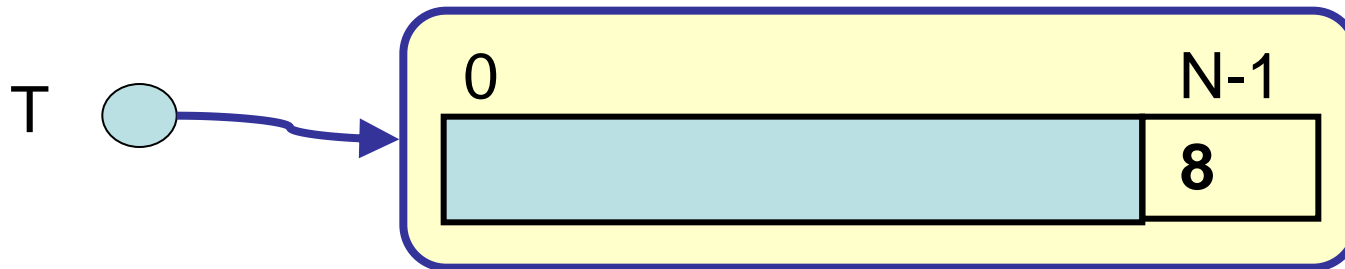
- Quelle est la valeur maximale parmi les éléments aux positions $0 \dots (N-1)$ d'un tableau T de taille N ?



- (la valeur maximale dans la partie ombrée est M)
- Réponse:
 - le maximum entre M et la valeur à l'indice $N-1$

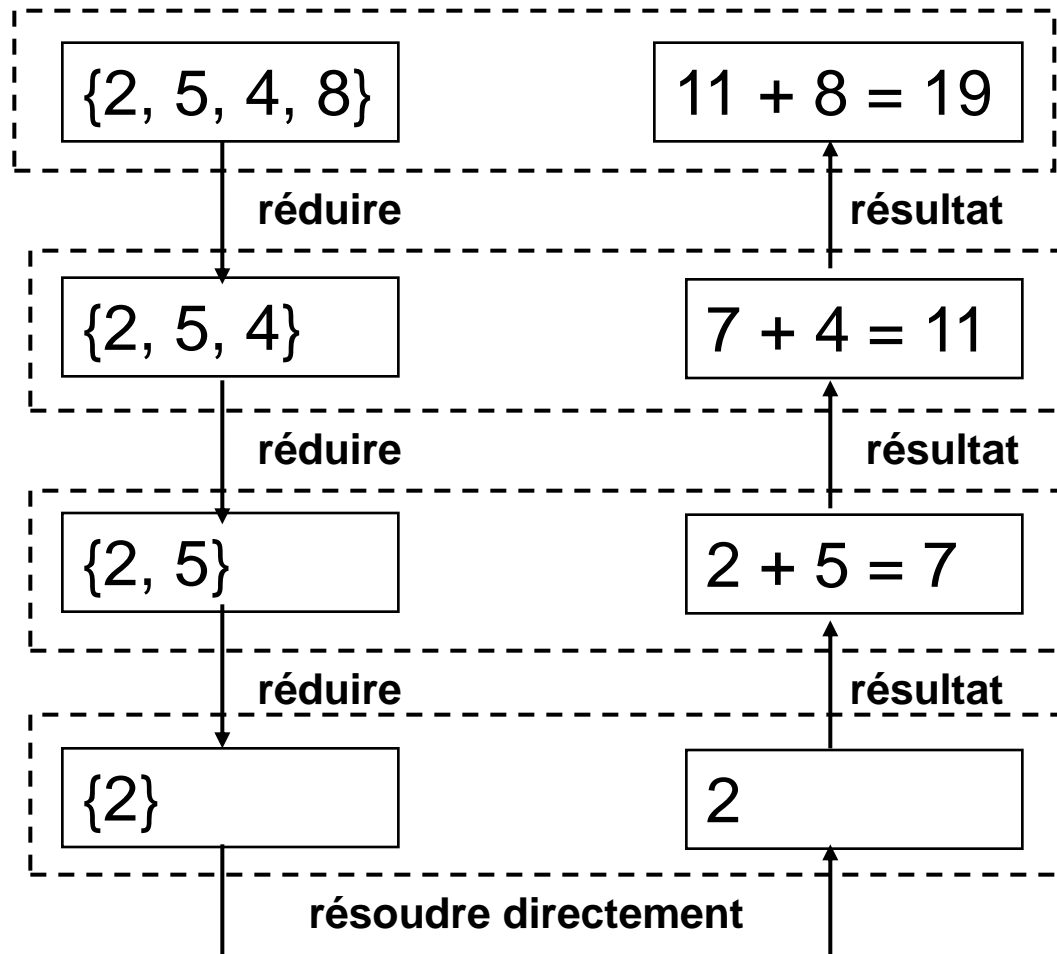
Récurtivité: Exemple 2 (idée de base)

- Quelle est la somme des éléments (nombres) aux positions $0 \dots (N-1)$ d'un tableau T de taille N ?

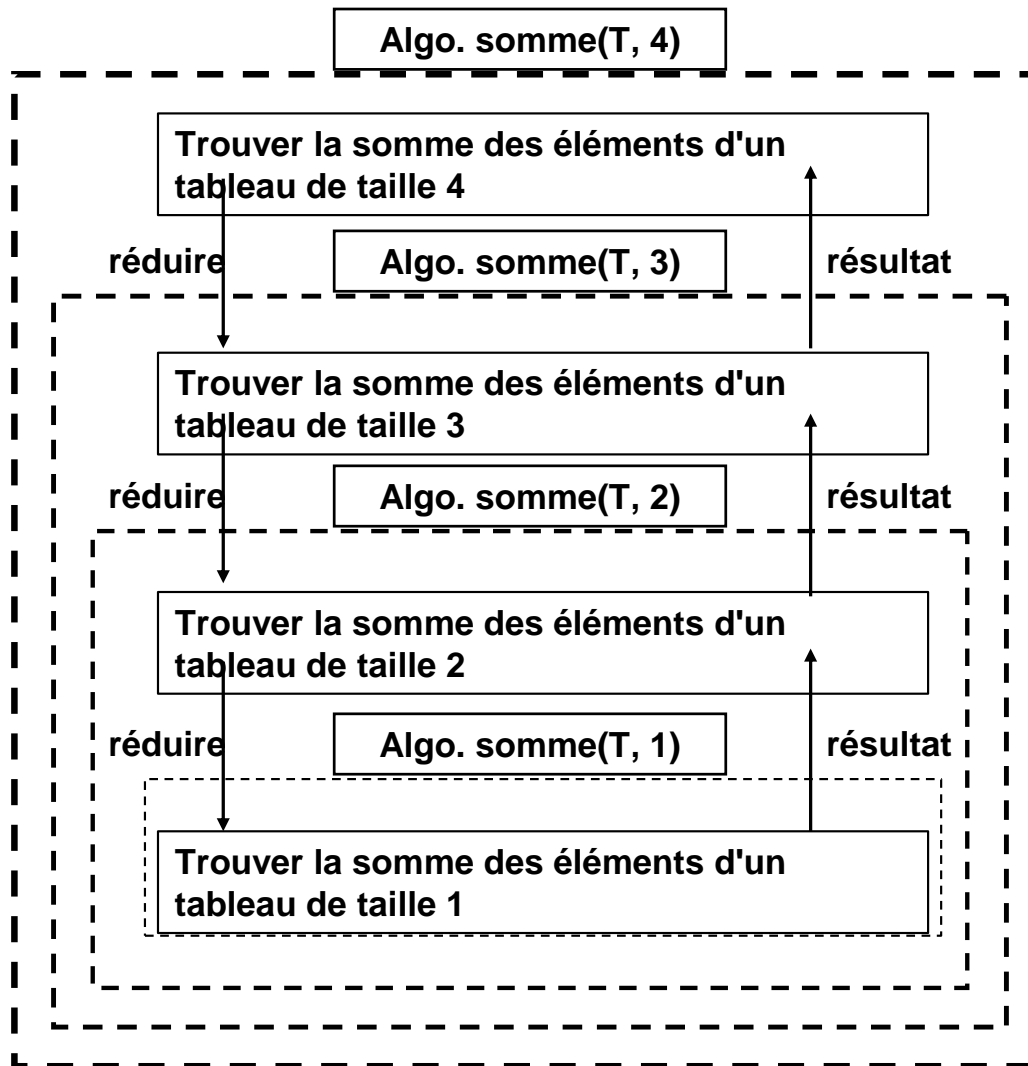


- Voir [SommeRecursive.ppt](#)

Exemple avec $T = \{2, 5, 4, 8\}$



Invocations récursives

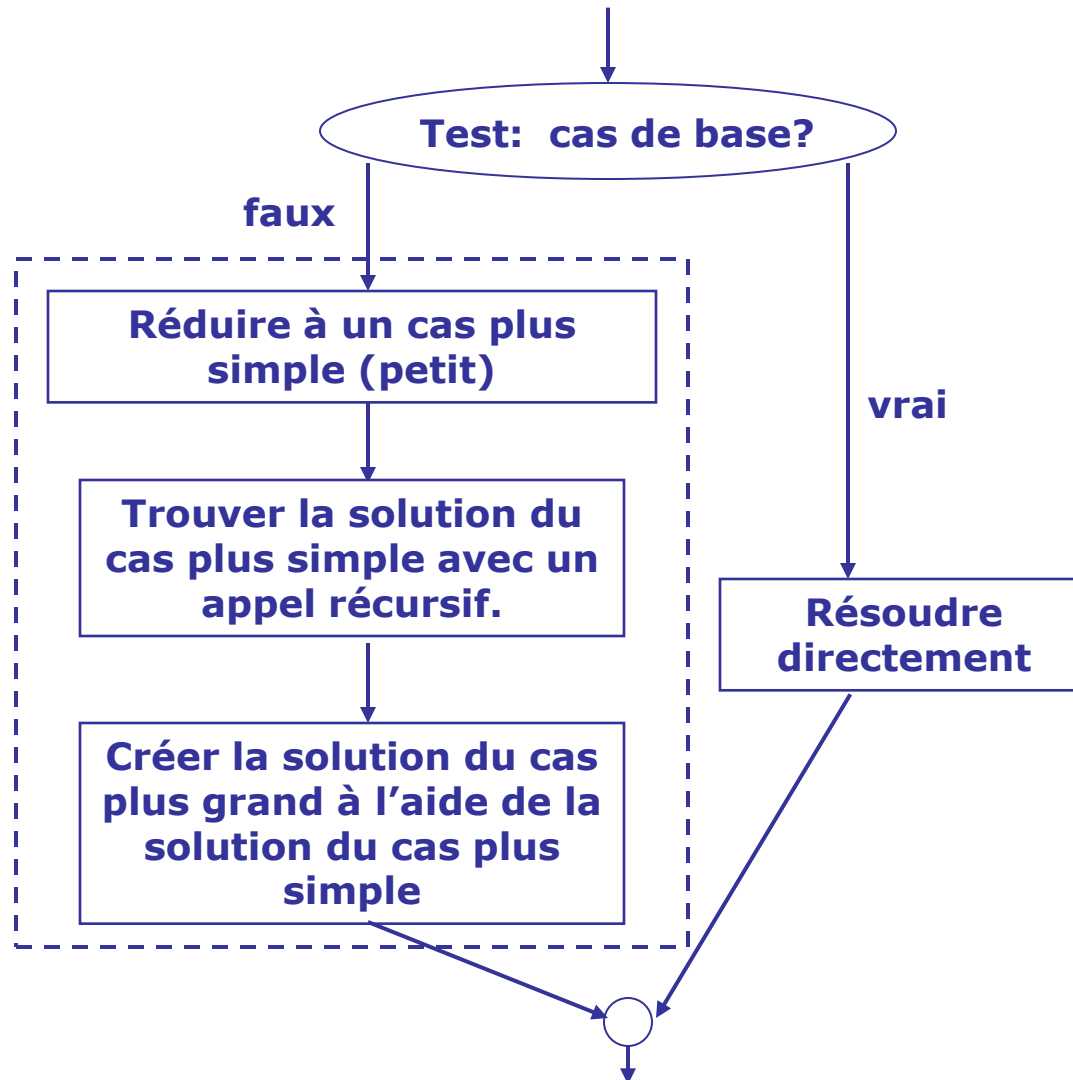


Composantes de la récursivité

Il y a 3 composantes à la récursivité:

1. Un test pour voir si le problème est assez simple pour résoudre directement (sans récursivité): le **cas de base**.
2. La solution pour le cas de base.
3. Une solution au problème qui implique la solution de un (ou plusieurs) versions plus petites du même problème.

Gabarit pour algorithmes récursifs



Exercice 8-1: Algorithme pour somme récursive (I)

- Développez un algorithme récursif pour trouver la somme des valeurs dans les positions du tableau 0...(N-1)

Données: T *(réfère à un tableau d'entiers)*
 N *(nombre d'éléments à additionner
 dans T)*

Résultat: S *(somme des éléments du tableau)*

Intermédiaires:

 M *(vaut N - 1 ; plusPetit!)*
 SPartielle *(somme partielle des M
 premiers éléments du tableau)*

En-tête:

S ← SommeRéc(T, N)

Exercice 8-1: Algorithme pour somme récursive (I)



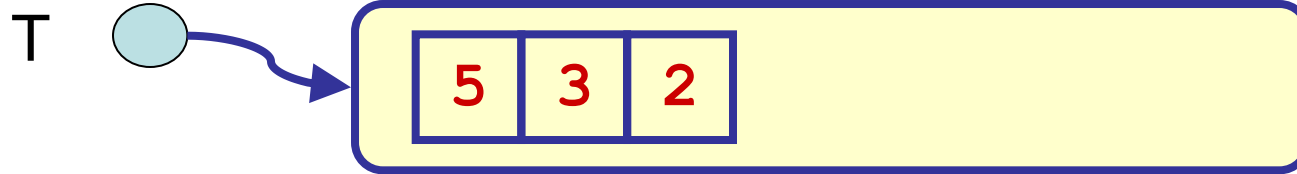
MODULE:

Exercice 8-1: Version simplifiée



-
- M peut être substitué directement dans l'invocation

Exercice 8-1: Tracez pour cette valeur de T: ?



instruction	Réf. Par T	N	SPartielle	S
valeurs initiales				

Exercice 8-1: Trace, Table 2



instruction	Réf. Par T	N	SPartielle	S
valeurs initiales				

Exercice 8-1: Trace, Table 3



instruction	Réf. Par T	N	SPartielle	S
valeurs initiales				

Algorithmes récursifs et méthodes récursives

- Un algorithme qui s'invoque lui-même, avec des données différentes, est un **algorithme récursif**.
- Une méthode Java qui s'invoque elle-même, avec des arguments (données) différents, est une **méthode récursive**.
- On peut aussi avoir de la récursivité circulaire, plus difficile à détecter et à gérer
 - Ex: A invoque B, B invoque C, et C invoque A.
 - Pas vu dans le cours.

Gabarit pour méthode Java récursive

```
public static typeDeRetour methodeRécursive
    (int taille, <autresParamètres>)
{
    typeDeRetour résultat;
    typeDeRetour résPartiel;

    if (CasDeBase(taille))
        { <Trouver le résultat directement > }
    else
    {
        { < Réduire à une instance avec plusPetit >; }
        < résPartiel > =
            methodeRécursive(plusPetit, < autresParamètres >);
        { < Calculer le résultat, basé sur résPartiel > }
    }
    return résultat;
}
```

Remarque: Il n'y a PAS de boucle!

Exercice 8-2: Traduction de SommeRec en Java



Variation

- Le fait d'avoir un cas de base qui ne fait rien est une variation assez commune.
- Par exemple, dans l'algorithme SommeRec, nous pouvons supposer que le cas de base est 0 et que son résultat est 0.
- Dans ce cas, la méthode sommeRec vue précédemment peut être reformulée ainsi:

```
public static int sommeRec(int [] t, int n)
{
    int s = 0; // RÉSULTAT
    if (n >= 1)
    {
        s = sommeRec(t, n - 1);
        s = s + t[n - 1];
    }
    return s;
} // Est-ce facile à lire ou à comprendre...?
```

Exemples de récursivité

Exemple 1: Quelle est la valeur maximale parmi les éléments aux positions 0...(N-1) d'un tableau A de taille N?

8-1 Quelle est la somme des éléments (nombres) aux positions 0...(N-1) d'un tableau T de taille N? (Exercice 8-2: Traduction en Java)

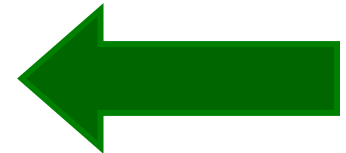
8-3 Trouver X^N où X et N sont des entiers et $N \geq 0$, $X \geq 1$.

(a) Algorithme direct

(b) Alternative alternatif sachant que:

$$X^N = X^I * X^{N-I}$$

- Trouvez I pour obtenir la version la plus efficace.



8-4 Dans un tableau T de plus de N nombres, retourner VRAI si tous les nombres aux positions 0...N-1 de T sont égaux, sinon FAUX.

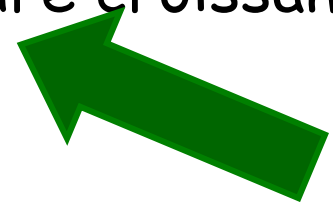
Exemples de récursivité (encore!)

8-5 Calculer $N!$ (factoriel).

8-6 Trouver la somme de $1+2+\dots+N$.

8-7 Inverser l'ordre des caractères dans un tableau A de N caractères.

8-8 Trier un tableau A de N nombres en ordre croissant.



Exercice 8-3: Algorithme pour trouver X^N (version 1)



Données: X (*base*)
 N (*entier, puissance*)
Résultat: X^{à la}N (*X à la puissance N*)
Intermédiaires:
 M (*vaut N - 1 ; plusPetit!*)
 X^{à la}M (*résultat partiel*)
En-tête:
 X^{à la}N ← Puissance(X, N)

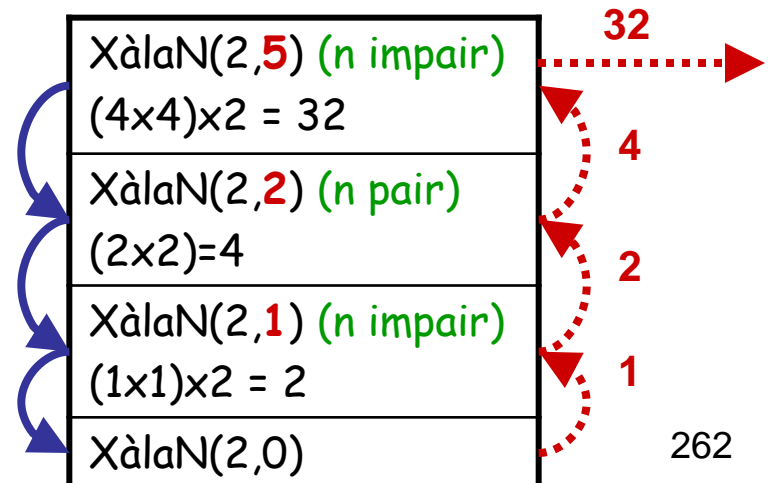
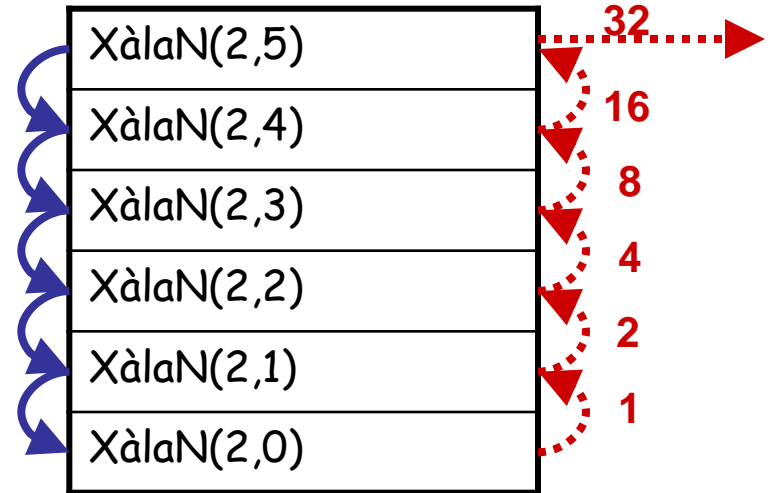
Exercice 8-3: Version plus efficace pour trouver X^N

Exercice 8-3: Réduction de la taille du problème

- La 2e version de X^N est plus efficace parce que nous coupons la taille du problème par un facteur 2, au lieu de le réduire par 1.
- Il y a donc moins d'invocations récursives:

→ Invocation récursive

⋯ Valeur retournée

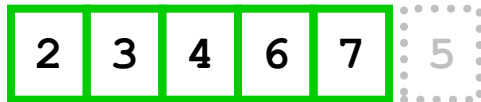


Exercice 8-8: Trier un tableau de nombres, version 1

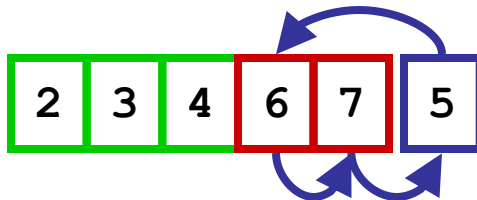
- Idée générale: "tri par insertion": insérer la dernière valeur dans un tableau trié précédemment.



- Premièrement, trier le tableau de taille $n-1$ par récursion



- Deuxièmement, déterminer où placer la valeur située à la dernière position du tableau



- Troisièmement, déplacer les autres valeurs, et insérer.



Exercice 8-8: Trier un tableau de nombres, version 2

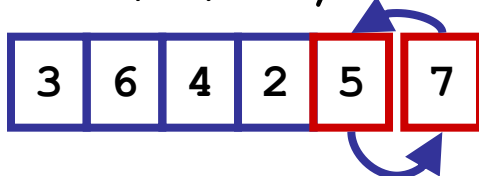
- Idée générale: "tri par sélection": sélectionner l'élément le plus grand et le placer à la bonne position.



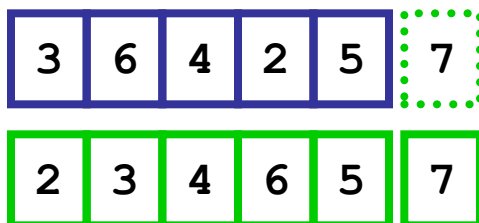
- Premièrement, trouver la position ★ contenant le maximum



- Deuxièmement, interchanger cette position avec la dernière



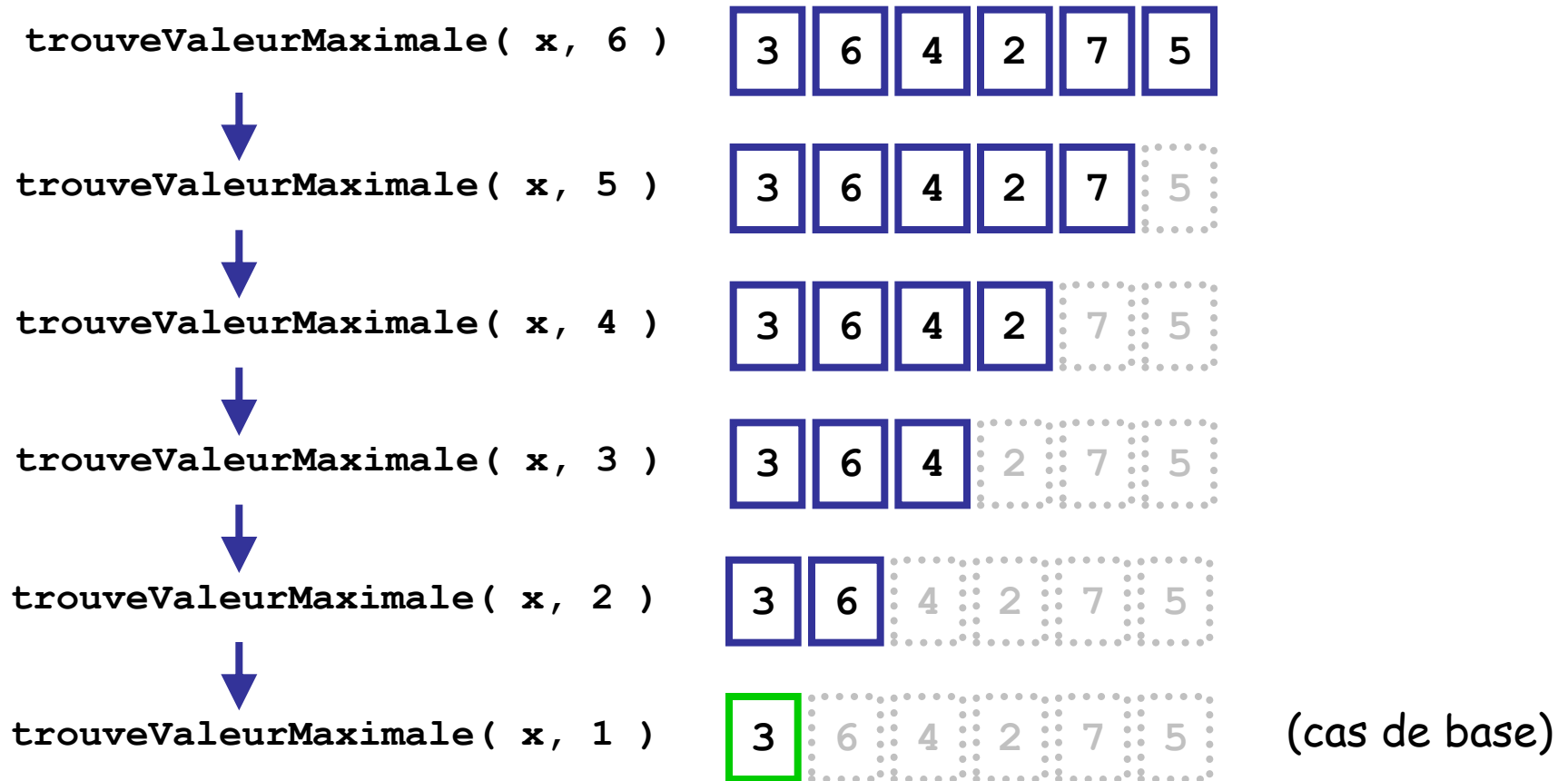
- Troisièmement, appel récursif pour trier le tableau plus petit



Exercice 8-8: Tri par sélection récursif ?

Exercice 8-8: Trouver récursivement la position de la plus grande valeur

Exercice 8-8: Trouver récursivement la position de la plus grande valeur



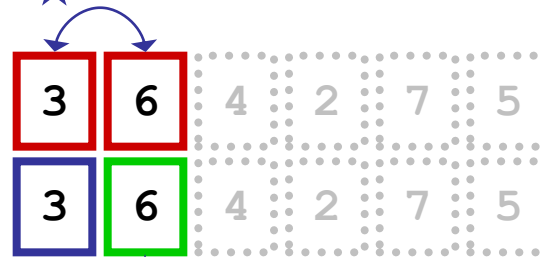
Exercice 8-8: Trouver récursivement la position de la plus grande valeur

(répété de la page précédente)
`trouveValeurMaximale(x, 1)`



(cas de base)

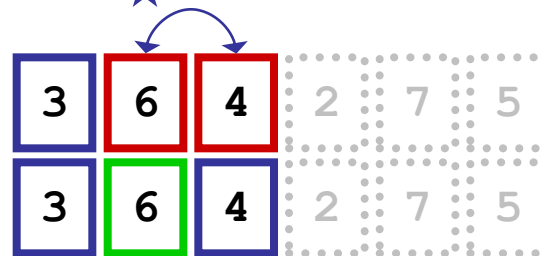
retour à
`trouveValeurMaximale(x, 2)`



(compare)

(choisit)

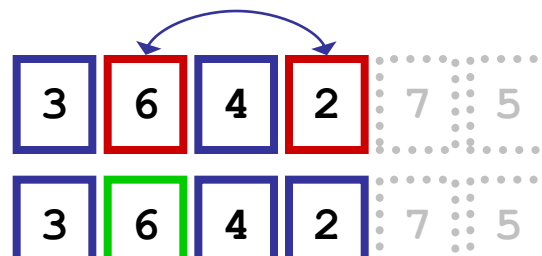
retour à
`trouveValeurMaximale(x, 3)`



(compare)

(choisit)

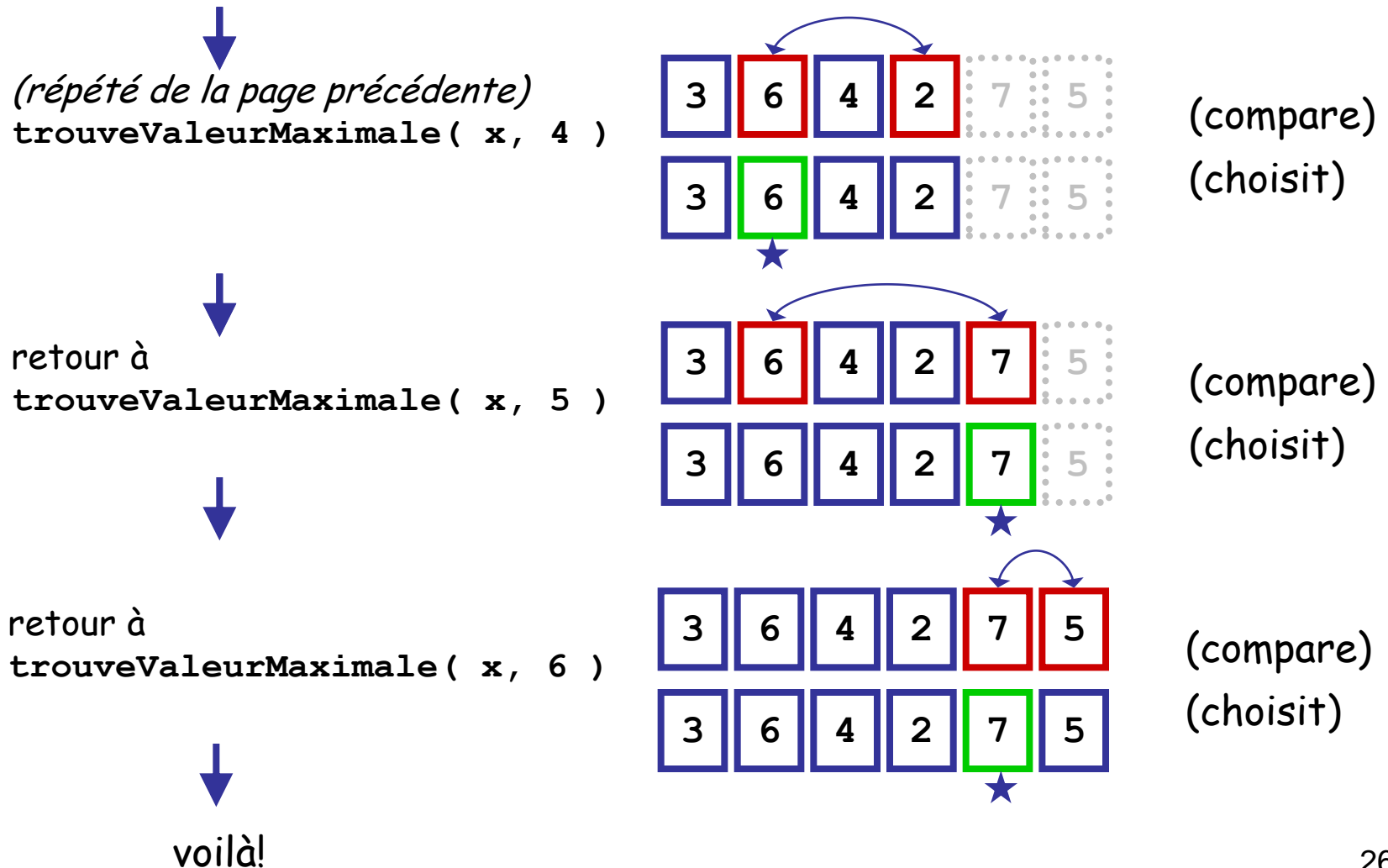
retour à
`trouveValeurMaximale(x, 4)`



(compare)

(choisit)

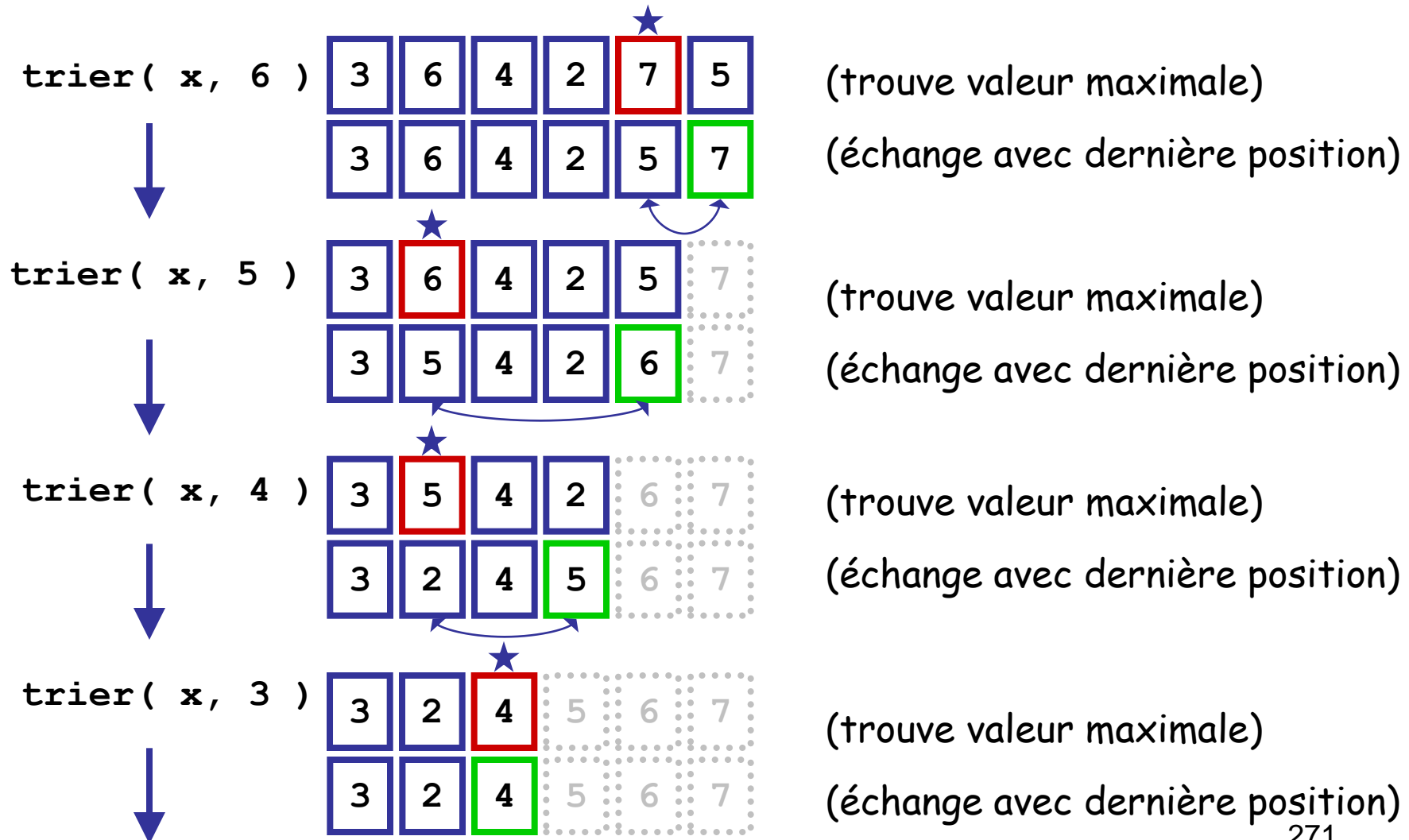
Exercice 8-8: Trouver récursivement la position de la plus grande valeur



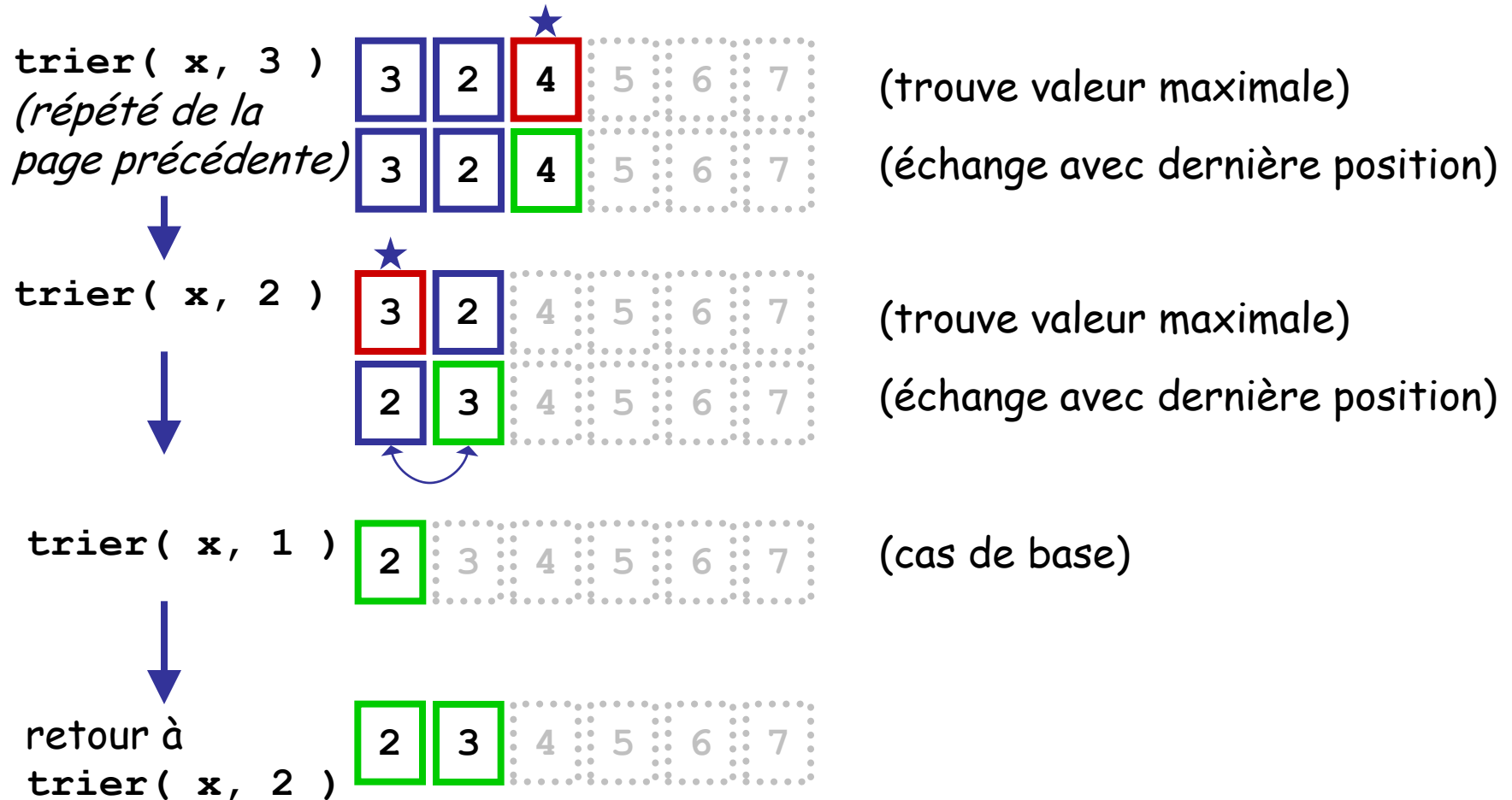
À quel moment le problème est-il résolu?

- Dans l'exemple précédent, nous avons d'abord besoin du résultat d'un appel récursif:
 - Donc, des appels récursifs ont été faits sans même avoir fait de "travail", jusqu'à ce que le cas de base ait été atteint.
 - C'est **après** que les appels récursifs aient commencé à retourner des résultats que les comparaisons ont été faites et que le problème a été résolu.
- Ce n'est pas toujours le cas. Par exemple, dans le tri par sélection, le "travail" est fait **avant** l'appel récursif, et le problème est résolu quand le cas de base est atteint.
 - Cependant, il faut quand même retourner pour toutes les invocations récursives! Voir l'exemple des pages suivantes.
- Parfois, l'appel récursif est fait **au milieu** du travail.

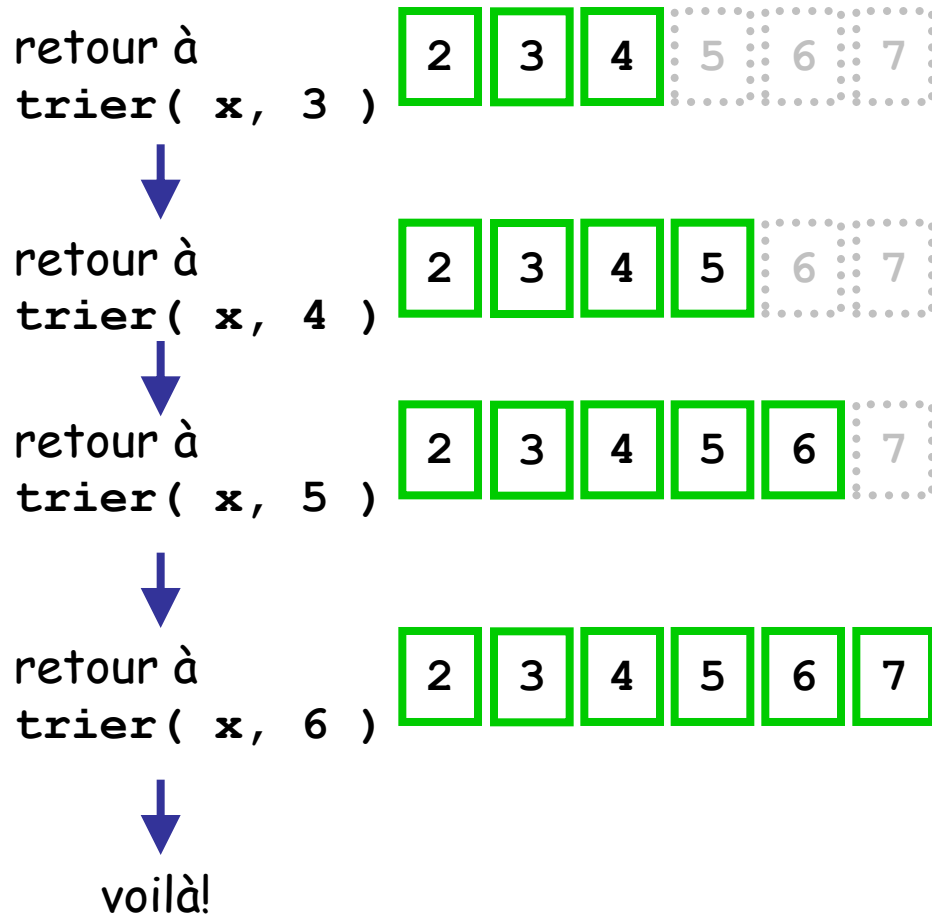
Exercice 8-8: Tri par sélection récursif



Exercice 8-8: Tri par sélection récursif



Exercice 8-8: Tri par sélection récursif



"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

Note historique...

- 1998: Larry Page et Sergey Brin, fondent une compagnie qui révolutionne le monde des moteurs de recherche et de l'informatique répartie: **Google!**
- Plusieurs applications...
- 450 000 serveurs
- Plus d'un milliard de requêtes par jour!



Matrices

- Une matrice $L \times C$ a R rangées et C colonnes.
- Exemple. Une matrice 4×6 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.

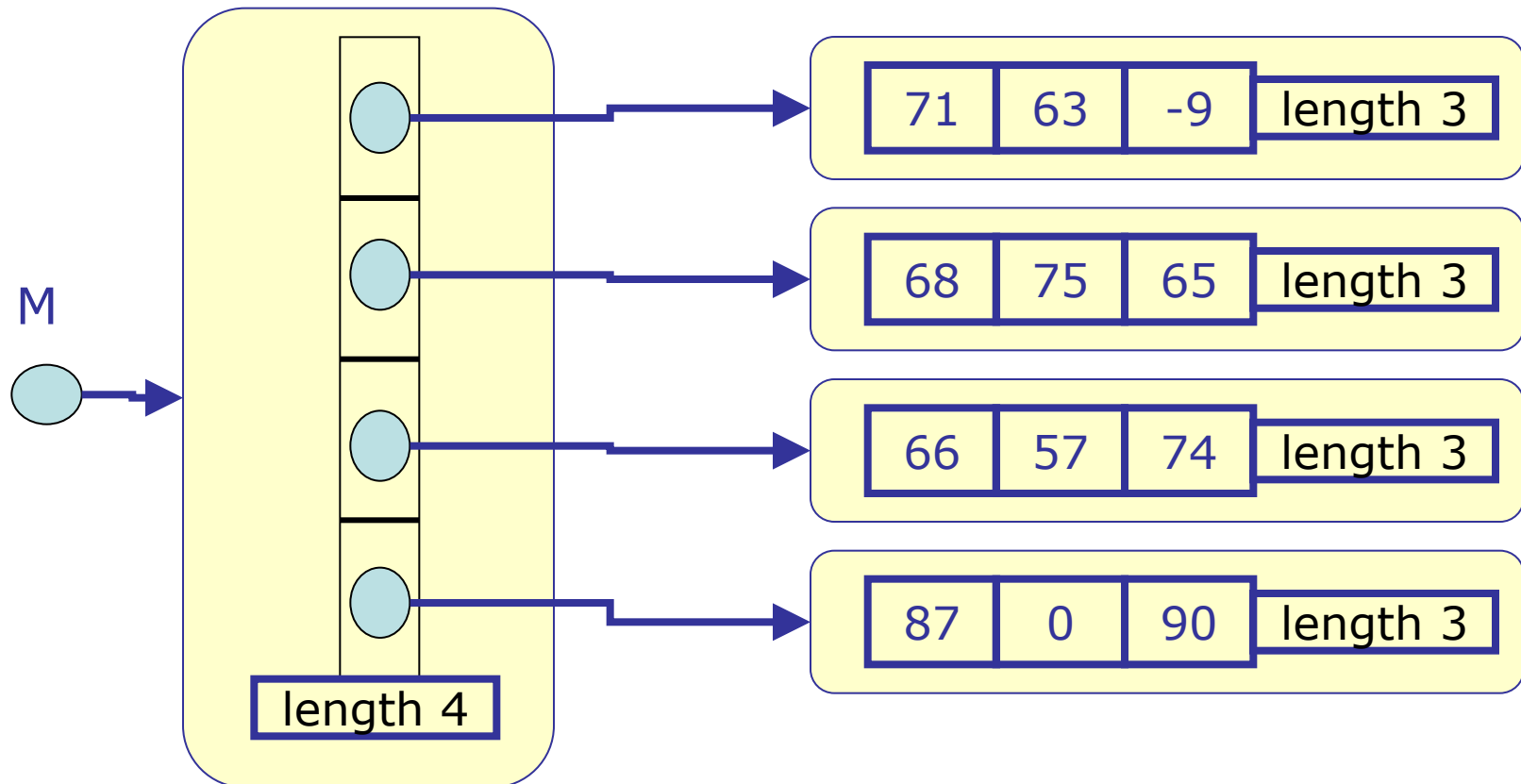
Exercice 9-3: Algorithme VérifDiag

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



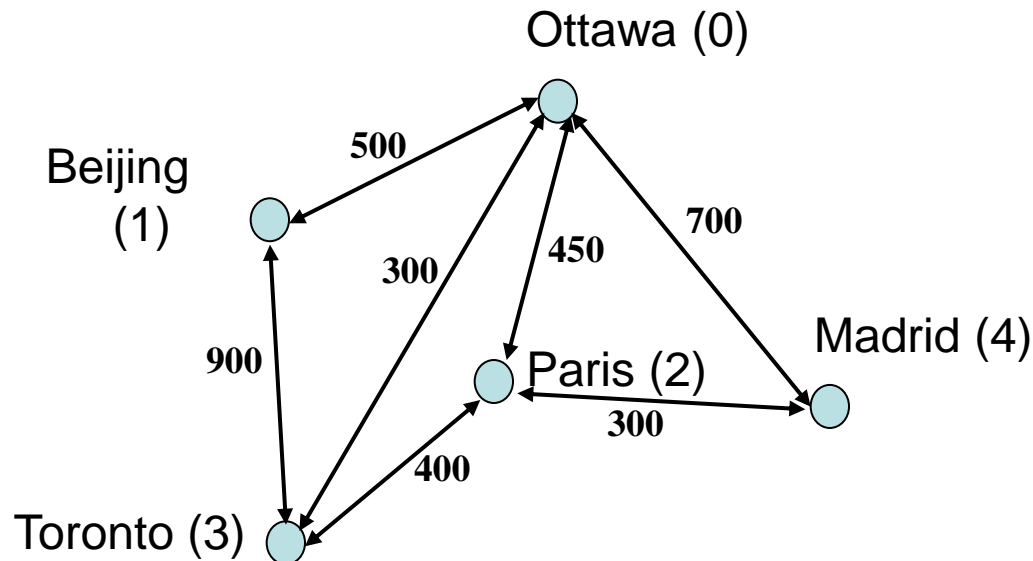
-
- Traduire l'algorithme trouvant la valeur maximale dans une matrice:
 - Note: `Integer.MIN_VALUE` est la valeur entière la plus négative permise pour un `int` Java, et peut être utilisé pour $-\infty$.

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()`.

Matrice d'adjacence

- *Air Escape* offre des vols entre plusieurs villes. Ces vols et leurs coûts peuvent être représentés par un graphe où un arc entre les villes X et Y avec un poids (étiquette) de W indique que *Air Escape* a un vol entre X et Y au coût de W dollars.



Représentation de matrices d'adjacence

- Ce graphe peut être représenté par une **matrice d'adjacence**. Il y a une ligne et une colonne pour chaque ville, et $\text{COÛT}[X][Y]$ est le coût d'un vol entre X et Y si un tel vol existe. La valeur *infini* (∞) est utilisée lorsqu'il n'y a pas de vol.

$$\text{COÛT} = \begin{bmatrix} 0 & 500 & 450 & 300 & 700 \\ 500 & 0 & \infty & 900 & \infty \\ 450 & \infty & 0 & 400 & 300 \\ 300 & 900 & 400 & 0 & \infty \\ 700 & \infty & 300 & \infty & 0 \end{bmatrix}$$

- Ici, ∞ est en fait un nombre très grand, plus grand que tout autre nombre
 - En Java, utiliser la constante **`Integer.MAX_VALUE`**

Trouver le vol direct le moins cher

- Vous habitez, l'une des villes visitées par *Escape*, et vous avez D\$ à dépenser. Écrivez un algorithme qui retourne un tableau de villes auxquelles vous pouvez vous permettre de vous rendre **directement**.
- Ce que vous savez (Données)
 1. La ville où vous vivez.
 2. Le coût d'un vol entre deux villes.
 3. Le nombre total de villes.
 4. Le montant que vous pouvez dépenser.
- Ce que vous désirez (Résultat)
 - Un **tableau** de villes visitables.
- Idée:
 - Premièrement, trouver le nombre de villes visitables.
 - Ensuite, créer un tableau de la bonne taille.
 - Enfin, placer les villes visitables dans ce tableau.

Exercice 9-6: Trouver le vol direct le moins cher (p. 1)



Exercice 9-6: Trouver le vol direct le moins cher (p. 2)



Exercice 9-7: Traduction vers Java (1) ?

Exercice 9-7: Traduction vers Java (2) ?

Solution alternative

- Afin de simplifier le problème, nous pourrions utiliser des algorithmes séparés pour trouver le nombre de villes visitables et pour remplir le tableau.
 - Algorithme `TrouveNombre`
 - Trouve le nombre de villes visitables
 - Algorithme `TrouveVilles`
 - Trouve les villes visitables
- En entrée, ces algorithmes n'ont besoin que d'une ligne du tableau `COÛT`: celle qui correspond à la ville où vous êtes.
- Nouveau module principal:
`NbVilles ← TrouveNombre (Coût[Départ], D, N)`
`Villes ← CréerTableau(NbVilles)`
`TrouveVilles(Coût[Départ], D, N, Villes)`

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 $i^{\text{ème}}$ 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.

Exercice 9-7: Algorithme EffaceRang (1) ?

Exercice 9-7: Algorithme EffaceRang (2) ?

Exercice 9-7: Algorithme DéplaceHaut ?

Exercice 9-7: Algorithme MetRangÀZéro ?

Exercice 9-7: Traduction vers Java



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.

"Politics is the skilled use of blunt objects."
-- L.B. Pearson

ITI 1520

Section 10: Introduction aux objets

Objectifs:

- Enregistrements
- Classes et objets
- Dissimulation de l'information
- Accesseurs et modificateurs
- **this**

Note historique...

- Barbara Liskov, informaticienne, est la première femme à avoir obtenu son doctorat en informatique aux États-Unis (en 1968, de l'Université Stanford)
- Elle est à l'origine de CLU, le premier langage à supporter l'abstraction de données (1975), ce qui a influencé de nombreux langages OO, incluant Java
- En 1993, elle et Janette Wing ont développé une définition particulière du sous-typage Principe de substitution de Liskov, utilisé dans la conception et la programmation orientée-objet.



Information - Étudiant

- Comment emmagasiner toute l'information sur un étudiant pour un cours?
 - Numéro d'étudiant (entier)
 - Note examen partiel (réel)
 - Note examen final (réel)
 - Inscrit pour crédits (Booléen)
- **Exercice 10-1:** Qu'est-ce qui ne va pas avec les solutions suivantes:
 - Chaque valeur est une variable séparée:
 - Mettre toutes les valeurs dans un tableau:



« Enregistrements »

- Tel un tableau, un « **enregistrement** » (*record*) permet à une variable d'emmagasiner plusieurs valeurs **de différents types**.
 - Une autre façon de voir l'enregistrement est comme un groupe de variables de différents types.
- Les enregistrements et les tableaux ont deux différences importantes:
 - Les valeurs/variables (appelées *champs*) d'un enregistrement peuvent être de types différents.
 - Chaque champ d'un enregistrement a un **nom**. Une valeur est accédée en indiquant le nom du champ.
- Exemple (un enregistrement simple avec 4 champs):

Nom du champ

Valeur du champ

NumÉtud	1234567
ExamPartiel	60.0
ExamFinal	80.0
Créditable	VRAI

Utiliser les enregistrements

- Supposons que l'enregistrement précédent soit emmagasiné dans une variable appelée **E**.
- Pour accéder au résultat de l'examen partiel:

E.ExamPartiel

- Ceci fait référence à un champ dans l'enregistrement **E**. Un champ peut être utilisé de la même façon qu'une variable du même type. Par exemple:

T ← E.ExamPartiel + E.ExamFinal

E.Créditable ← Faux

- L'enregistrement entier peut être utilisé dans une affectation ou encore être passé en paramètre.

X ← E

(Pas en Java - ceci se produit dans d'autres langage de programmation tel que le C).

Définir un **type** d'enregistrement

- Pour les types primitifs, nous regardions:
 - Quelles valeurs le type permet-il?
 - Quelles opérations peut-on faire avec des valeurs de ce type?
- Un enregistrement est un type défini par le concepteur et construit à partir des types déjà rencontrés:
 - Types primitifs
 - Autres types définis par le concepteur
- Créer un enregistrement est semblable à un type primitif:
 - Quelles sont les composantes d'une valeur d'enregistrement?
 - Quelles opérations peut-on faire avec des valeurs de cet enregistrement?

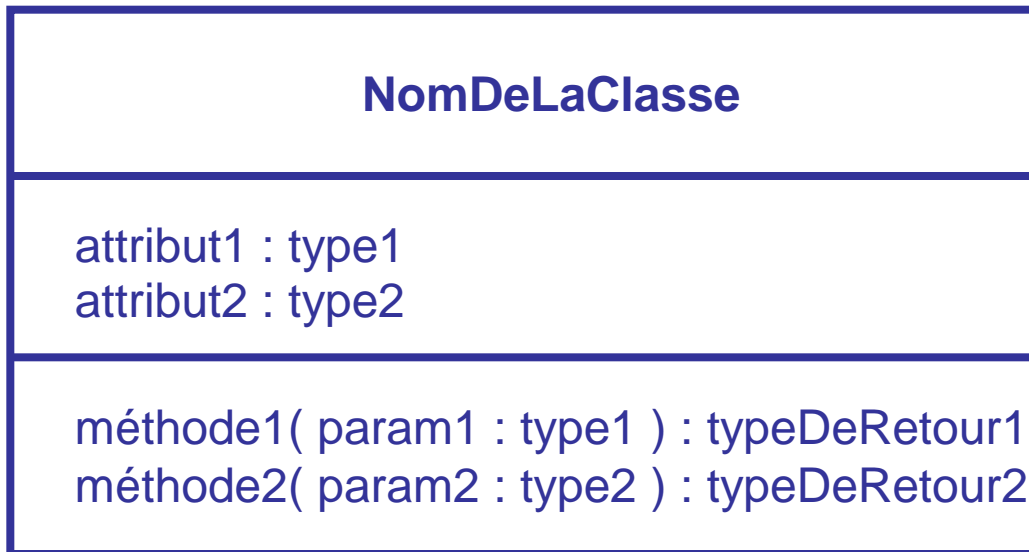
Enregistrements et classes/objects

- Quelques langages permettent la création d'enregistrements, sans la possibilité de définir des opérations sur ces enregistrements. Par exemple:
 - Le langage Pascal a des « record »
 - Le langage C a des « struct » (structures)
 - (Il est possible de définir des références à des fonctions dans les structures C).
- D'autres langages vous permettent aussi de définir des opérations sur ces types.
 - Les types d'enregistrement sont habituellement appelés « classes », et les enregistrements spécifiques (*instances*) des « objets ».
 - Exemples:
 - Classes dans les langages C++ et Java

Classes et objets

- Un objet peut être considéré comme un enregistrement car il y a un ensemble d'**attributs** (valeurs avec noms (**variables**) emmagasinées dans l'objet).
- Chaque objet est créé à partir d'une classe. **Un objet est référé à partir d'une variable de référence (comme le tableau).**
- Une "classe" peut être vue comme un « gabarit » permettant de créer des objets avec des ensembles identiques d'attributs.
 - La classe peut aussi contenir des méthodes (algorithmes) pour faire des calculs/transformations sur les attributs de l'objet (et/ou sur des données externes).
- Une méthode est invoquée sur un objet en utilisant l'opérateur point (.), comme pour les champs d'un enregistrement
résultat ← unObjet.uneMéthode(unParamètre)

Diagrammes de classes



← Les "attributs" sont comme les variables de champs d'un enregistrement

- Cette forme de diagramme provient d'une notation normalisée appelée "Unified Modelling Language" (ou UML).

Traduction vers Java

```
public class <Nom de classe>
{
    // Déclarations de variables
    // public <type> <nom>;

    // Méthodes
}
```

- Ceci n'est que la définition de la structure d'un enregistrement pour construire des objets.
 - Les objets doivent être créés en utilisant l'opérateur **new**.
 - Un objet est référé avec par une **variable de référence**.

```
// Déclaration de la variable de référence
<Nom de classe> varRef;
// Création de l'objet
varRef = new <Nom de classe>();
```

Exercice 10-2: Première version de la classe Étudiant



- Pour chaque étudiant, nous voulons emmagasiner son # d'étudiant, ses résultats d'examens partiel et final, et la confirmation que le cours est suivi pour des crédits (ou non). *Nous parlerons de sa note finale plus tard.*

Étudiant
<i>(aucune méthode encore!)</i>

Exercice 10-2: Traduction vers Java

```
public class Étudiant  
{
```

```
    // méthodes
```

```
}
```

```
    // Déclare variable de référence unÉtudiant
```

```
    // Crée un objet Étudiant référé par unÉtudiant
```

Exercice 10-3: Utilisation des objets en Java

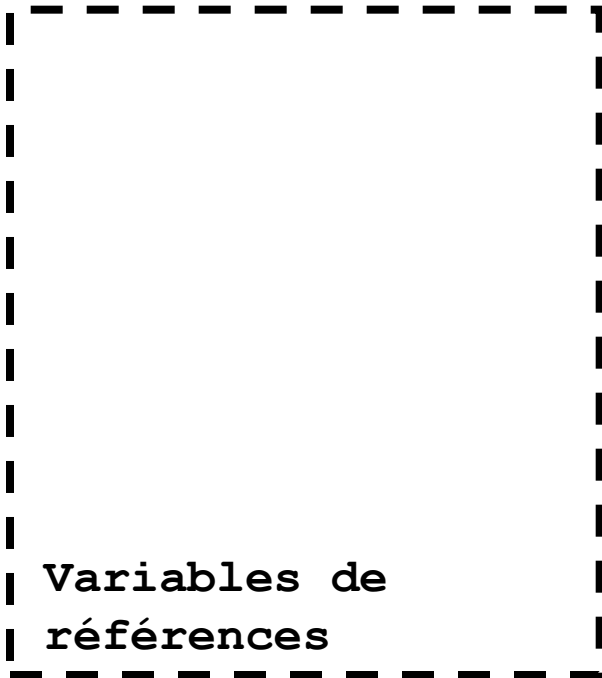
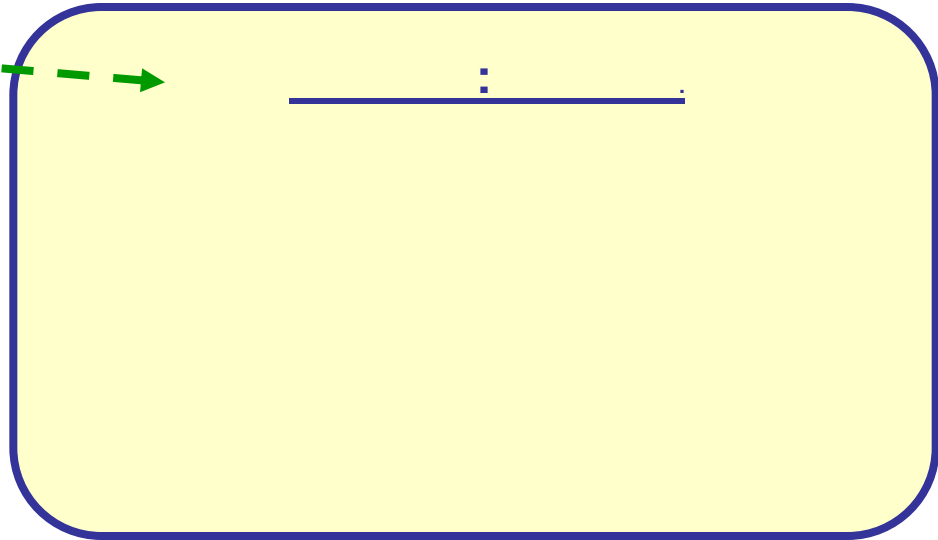
```
Étudiant unÉtudiant;           // déclare la variable de réf.  
unÉtudiant = new Étudiant();   // création de l'objet  
unÉtudiant.numÉtud = 1234567;  
unÉtudiant.examPartiel = 60.0;  
unÉtudiant.examFinal = 80.0;  
unÉtudiant.créritable = true;
```

```
Étudiant moiAussi = new Étudiant();  
moiAussi.numÉtud = 1069665;  
moiAussi.examPartiel = 73.0;  
moiAussi.examFinal = 79.0;  
moiAussi.créritable = false;
```

Exercice 10-3: Utilisation des objets en Java ?

format:
<nomObjet> : <nomClasse>

(le soulignement indique
qu'il s'agit d'un diagramme
d'instances)

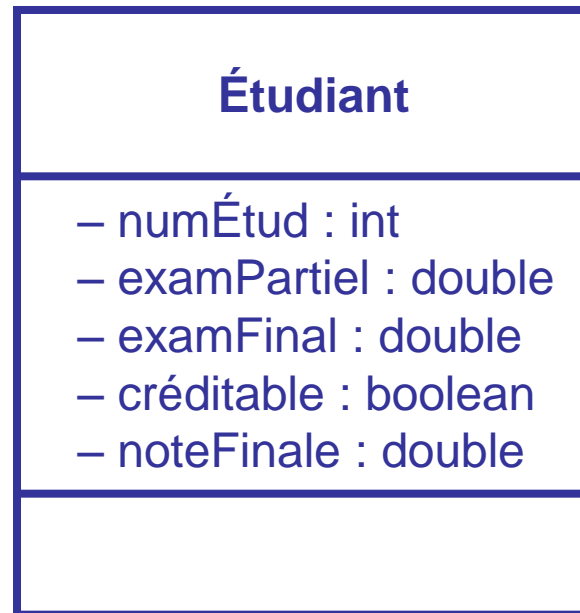


Dissimulation de l'information

(Information Hiding)

- Si nous voulions modifier nos objets Étudiant pour conserver la note finale, qui vaut 20% de l'examen partiel plus 80% de l'examen final:
 - Nous pourrions ajouter un champ **noteFinale** à notre classe.
- Nous voulons nous assurer que:
noteFinale == (0.2*examPartiel + 0.8*examFinal)
est toujours vrai, pour fins de cohérence.
- Il serait utile de pouvoir interdire à quiconque de mettre à jour **noteFinale** de façon arbitraire.
 - En fait, la note finale devrait changer uniquement en modifiant la valeur de **examPartiel** ou de **examFinal**.
- Restreindre l'accès s'appelle la **dissimulation de l'information**.

Attributs privés dans une classe



- Le signe négatif (-) avant une variable indique que l'attribut est privé.
- En déclarant un attribut privé, seules les méthodes **déclarées à l'intérieur de la classe** peuvent y accéder (en lecture et écriture).

Dissimulation de l'information

- Les noms et types des champs représentent l'*implémentation* d'une classe.
 - Afin de rendre votre classe relativement indépendante par rapport aux autres parties de votre programme (ce qui réduit l'effort de maintenance), les champs sont (presque) toujours **private**.
 - Cette **dissimulation de l'information** s'appelle aussi **abstraction de données** ou encore **encapsulation**).
- Les champs et méthodes privées ne peuvent être accédés directement mais seulement à partir des méthodes de la classe.
- Si (et seulement si) nécessaire, nous pouvons définir quelques méthodes publiques permettant aux autres parties du programme d'avoir accès aux champs.
- Les méthodes publiques représentent l'**interface** de la classe par rapport aux autres parties du programme.

Deuxième version de la classe Étudiant

- Cette fois, nous utiliserons l'encapsulation.

```
public class Étudiant
{
    // étaient publiques
    private int numÉtud;
    private double examPartiel;
    private double examFinal;
    private double noteFinale; // nouveau champ
    private boolean créditable;
    // méthodes
}
```

Comment utiliser cette deuxième version?

- Si nous essayons ceci:

```
Étudiant unÉtudiant = new Étudiant();  
unÉtudiant.numÉtud = 1234567; // erreur!
```

le compilateur retournera une erreur parce que nous n'aurons plus accès à **numÉtud** hors de la classe.

- Cependant, nous pouvons créer des méthodes d'accès supplémentaires dans la classe **Étudiant**:
 - "accesseur": demande de voir la valeur d'un champ privé.
 - "modificateur": demande de modifier la valeur d'un champ privé.

Accesseurs et modificateurs

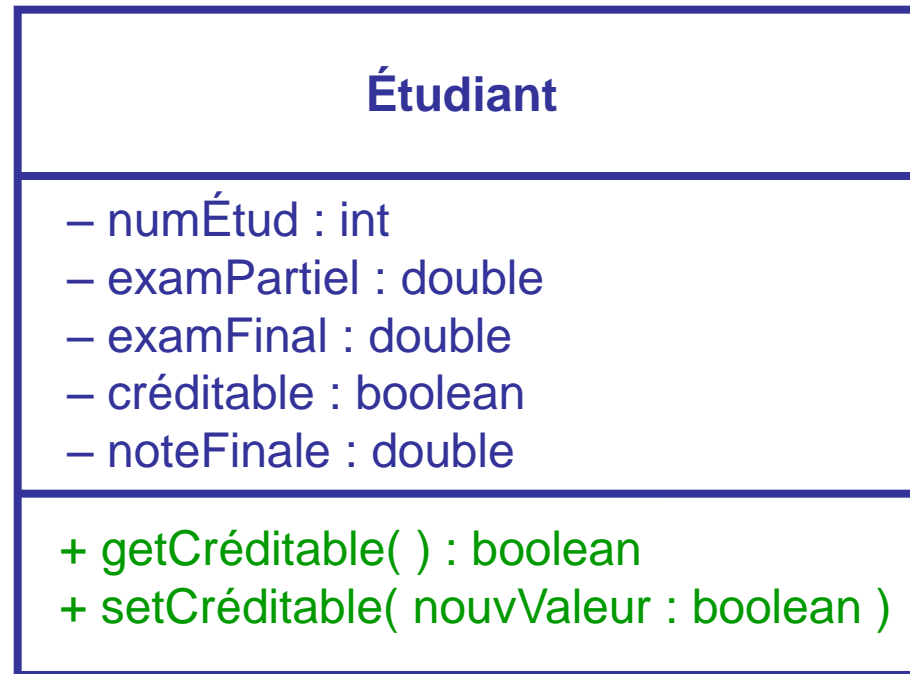
- **Accesseur**
 - une méthode d'instance publique (invoquée via la référence à un objet);
 - retourne la valeur d'un champ de l'objet;
 - ne possède aucun paramètre explicite;
 - est souvent nommée **getNomDuChamp** (on l'appelle parfois *getter* en anglais).

- **Modificateur**
 - une méthode d'instance publique (invoquée via la référence à un objet);
 - affecte une valeur à un champ;
 - accepte en paramètre une valeur du type champ;
 - est souvent nommée **setNomDuChamp** (on l'appelle parfois *setter* en anglais).

Accesseurs et modificateurs

- Exemples pour le champ `créditable` de la classe:
 - + `getCréditable() : boolean`
 - méthode pour retourner la valeur de `créditable`
 - le `+` indique que la méthode a une visibilité `publique`
 - le type de retour est `boolean` et, avec la notation UML, est placé à la fin de la méthode (inverse de Java).
 - + `setCréditable(nouvValeur : boolean)`
 - méthode pour modifier la valeur de `créditable`
 - un paramètre, `nouvValeur`, de type `boolean`
 - `aucune` valeur de retour

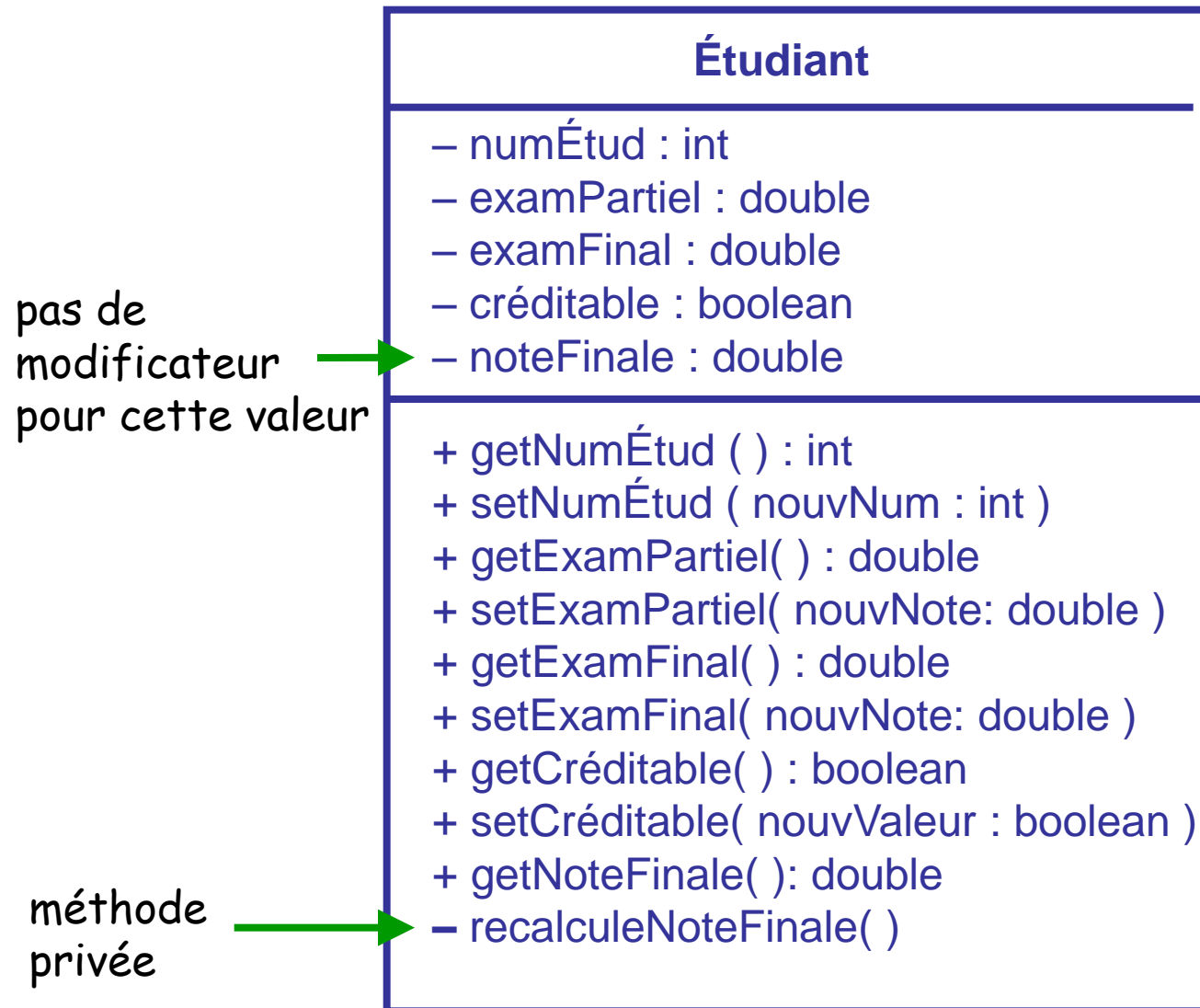
Diagramme de classe avec accesseurs et modificateurs



De retour à l'encapsulation

- Afin de suivre notre stratégie pour dissimuler le champ `noteFinale`:
 - Nous créerons une méthode accesseur pour `noteFinale`, mais **PAS** une méthode modificateur.
 - Nous pouvons ajouter une méthode `recalculeNoteFinale()` pour recalculer la note finale si les notes de l'examen partiel ou de l'examen final ont changé.
 - Les méthodes modificateur `setExamPartiel()` et `setExamFinal()` vont invoquer `recalculeNoteFinale()` afin d'automatiser la mise à jour de la note finale.
- Nous devrions aussi restreindre l'accès à `recalculeNoteFinale()` car cette méthode n'est pas conçue pour être utilisée à l'extérieur de la classe.

Classe Étudiant avec dissimulation de l'information



Traduction vers Java

```
public class Étudiant
{
    // Attributs
    private int numÉtud;
    private double examPartiel;
    private double examFinal;
    private boolean créditable;
    private double noteFinale;

    // Méthodes
    public int getNumÉtud()
    {
        // insérez votre code ici
    }
    public void setNumÉtud( int nouvNum )
    {
        // insérez votre code ici
    }
    public double getExamPartiel()
    {
        // insérez votre code ici
    }
    public void setExamPartiel(double nouvNote)
    {
        // insérez votre code ici
    }
    // continue à droite...
```

```
// ...suite du côté gauche

    public double getExamFinal()
    {
        // insérez votre code ici
    }
    public void setExamFinal( double nouvNote )
    {
        // insérez votre code ici
    }
    public boolean getCréditable()
    {
        // insérez votre code ici
    }
    public void setCréditable(boolean nouvValeur)
    {
        // insérez votre code ici
    }
    public double getNoteFinale()
    {
        // insérez votre code ici
    }

    private void recalculeNoteFinale()
    {
        // insérez votre code ici
    }
} // fin de la classe Étudiant
```


Invocation d'accesseurs et modificateurs

- Encore l'utilisation de l'opérateur point (.)
- Le code suivant causera des erreurs, pourquoi?

```
Étudiant unÉtudiant = new Étudiant();  
unÉtudiant.numÉtud = 1234567;           // erreur ici!  
int monNuméro = unÉtudiant.numÉtud;    // erreur ici!  
System.out.println( monNuméro );
```
- Le compilateur s'assure que l'accès à **numÉtud** est privé
- Solution: utilisation d'accesseurs/modificateurs

```
Étudiant unÉtudiant = new Étudiant();  
unÉtudiant.setNumÉtud ( 1234567 );     // OK!  
int monNuméro = unÉtudiant.getNumÉtud(); // OK!  
System.out.println( monNuméro );
```

Accesseurs et modificateurs en Java

- Exemples de méthodes pour le champ **créditable** :

```
public class Étudiant
{
    boolean créditable;
    // ...autres attributs préalablement déclarés

    // accesseur:  retourne la valeur requise
    public boolean getCréditable()
    {
        return this.creditable ;
    }

    // modificateur:  place la valeur fournie en paramètre dans
    //                  l'attribut
    public void setCréditable( boolean nouvValeur)
    {
        this.creditable = nouvValeur ;
    }

    // ...les autres méthodes sont semblables
}
```

Ce fameux `this`

- Quand les champs de notre classe `Étudiant` étaient publiques, nous distinguons un même champ dans deux enregistrements différents avec le nom de la variable et l'opérateur point:
 - `unÉtudiant.créritable` versus `moiAussi.créritable`
- De même, quand une méthode `dans la classe` veut utiliser « la valeur du champ de l'objet sur lequel la méthode a été invoquée », alors `this` fait référence à l'objet invoqué.
- Lors de l'appel `unÉtudiant.getCréritable()`, `this` fait référence à `unÉtudiant`
 - ... et donc `this.créritable` est `unÉtudiant.créritable`, qui vaut `true`.
- Lors de l'appel `moiAussi.getCréritable()`, `this` fait référence à `moiAussi`
 - ... et donc `this.créritable` est `moiAussi.créritable`, qui vaut `false`.

Opération opère sur les données d'objet

- Ajouter des méthodes à une classe telle que **Étudiant** revient à créer des opérations qui opère sur les données des objets créés avec la classe (nos propres types de données).
- Notez que nous avons invoqué la méthode **getCréditable** sans paramètres à deux différents endroits, mais comme chaque invocation était associée à un objet différent, nous avons obtenu des résultats différents.
- Une analogie:
 - Avec des entiers: $3 + 5$ et $4 + 3$; la même opération (+) est invoquée, mais sur des valeurs différentes, ce qui mène à des résultats différents.
 - Avec des étudiants: même opération (**getCréditable**), mais sur des objets différents \Rightarrow résultats différents

Implémenter la dissimulation en Java

- Le code suivant implémente notre stratégie où la note finale ne peut être changée qu'en modifiant les valeurs de l'examen final ou de l'examen partiel.

```
public class Étudiant
{
    // attributs et autres méthodes...

    public void setExamPartiel( double nouvNote )
    {
        this.examPartiel = nouvNote ;
        this.recalculeNoteFinale( ) ;
    }

    public void setExamFinal( double nouvNote )
    {
        this.examFinal = nouvNote ;
        this.recalculeNoteFinale( ) ;
    }

    private void recalculeNoteFinale ( )
    {
        this.noteFinale = 0.2 * this.examPartiel + 0.8 * this.examFinal ;
    }
}
```

Bénéfices de l'encapsulation (1)

- L'une des causes les plus communes de problèmes informatiques dans le passé était que toutes les parties du programme avaient accès à toutes les variables (de façon globale).
 - Par exemple, quelqu'un pouvait modifier une variable dans un long programme, alors qu'une autre partie du programme présupposait que cette variable ne serait pas modifiée.

"Successful software always gets changed." - *F. Brooks*

- Avec l'encapsulation (dissimulation de l'information), nous pouvons plus facilement compartimenter le code afin de réduire ce genre d'effet indésirable.

Bénéfices de l'encapsulation (2)

- Nous pouvons aussi apporter des changements à l'intérieur de la classe qui n'affecteront pas les utilisateurs de cette classe.
- Exemple: Si nous décidons que le champs `noteFinale` n'a pas vraiment sa place à l'intérieur de la classe `Étudiant`
 - Nous pourrions calculer la note finale à chaque requête:

```
public double getNoteFinale()
{
    return 0.2 * this.examPartiel + 0.8 * this.examFinal;
}
```
 - Nous devrions aussi enlever ses accesseurs/modificateurs, ainsi que leurs invocations dans `setExamPartiel()` et `setExamFinal()`.
- Apporter ces changements n'affectera pas les utilisateurs de la classe:
 - Ainsi, `moiAussi.getNoteFinale()` se comporte de la même façon.
 - Comme `recalculeNoteFinale()` était privée, le code hors de la classe ne pouvait pas invoquer cette méthode, et donc elle peut être enlevée en toute quiétude..
- Donc, nous n'avons pas à changer le code hors de la classe!

Deux versions possibles

Étudiant

- numÉtud : int
- examPartiel : double
- examFinal : double
- créditable : boolean
- **noteFinale : double**

- + getNumÉtud () : int
- + setNumÉtud (nouvNum : int)
- + getExamPartiel() : double
- + setExamPartiel(nouvNote: double)
- + getExamFinal() : double
- + setExamFinal(nouvNote: double)
- + getCréditable() : boolean
- + setCréditable(nouvValeur : boolean)
- + getNoteFinale() : double
- **recalculeNoteFinale()**

Étudiant

- numÉtud : int
- examPartiel : double
- examFinal : double
- créditable : boolean

- + getNumÉtud () : int
- + setNumÉtud (nouvNum : int)
- + getExamPartiel() : double
- + setExamPartiel(nouvNote: double)
- + getExamFinal() : double
- + setExamFinal(nouvNote: double)
- + getCréditable() : boolean
- + setCréditable(nouvValeur : boolean)
- + getNoteFinale() : double

Encore **this**

- Dans la plupart des cas, nous n'avons pas à utiliser **this** pour faire référence à l'objet où la méthode est invoquée.
 - À l'intérieur de la classe **Étudiant**:
 - **examFinal** peut être utilisé au lieu de **this.examFinal**.
 - **recalculeNoteFinale()** peut être utilisé au lieu de **this.recalculeNoteFinale()**.
- Il y a deux situation où nous avons vraiment besoin de **this**:
 1. Un objet veut se passer en paramètre à une méthode d'une autre classe
 2. Un objet veut retourner une référence à lui-même comme résultat d'une méthode.

"Design and programming are human activities;
forget that and all is lost."

-- Bjarne Stroustrup, inventeur de C++

ITI 1520

Section 11: Conception orientée-objet

Objectifs:

- Constructeurs
- Classes avec champs tableau
- Méthodes/variables de classe et d'instance
- Conception de classes

Note historique...



- Le Xerox Palo Alto Research Center (PARC), fondé en 1970, est à l'origine de plusieurs contributions importantes en informatique:
 - Première station de travail (Alto) avec interface utilisateur graphique (GUI, avec fenêtres et icônes) et souris
 - Premier éditeur de texte WYSIWYG
 - Langage InterPress (précurseur de PostScript) pour la description de pages imprimées
 - Protocole Ethernet pour réseaux locaux
 - Langage orienté-objet Smalltalk, avec environnement de développement graphique (le concepteur était Alan Kay)
 - L'imprimante laser
 - Les Remote Procedure Calls, le protocole two-phase commit
 - ...

Orienté-objet

- L'approche utilisée pour notre classe Étudiant est « orientée objet »:
 - Nous avons une classe comme gabarit pour la création d'objets.
 - Les objets Étudiant sont des **INSTANCES** de la CLASSE Étudiant.
 - Les instances ont des **méthodes d'instance** qui utilisent les valeurs des champs pour un objet spécifique
 - ex: **getNoteFinale()** aura des résultats différents pour des objets différents parce que **this.examFinal** peut être différent pour des objets différents
 - Si vous voulez que l'objet fasse quelque chose pour vous, il faut lui demander en invoquant sa méthode correspondante.
 - Vous ne pouvez pas *fouiner* à l'intérieur d'un objet (quand vous êtes hors de sa classe) et modifier la valeur de ses champs privés.
 - Vous ne pouvez pas non plus invoquer une méthode d'instance sans la référence à un objet:
 - $X \leftarrow 3.0 + \text{getNoteFinale}()$ ne veut rien dire. À quelle note finale faisons-nous référence?

Initialisation d'objets

- Lorsque nous créons un nouvel **Étudiant**, nous voulons habituellement donner des valeurs à tous les champs dans l'objet.

```
Étudiant unÉtudiant ← new Étudiant();  
unÉtudiant.numÉtud ← 1234567;  
unÉtudiant.examPartiel ← 60.0;  
unÉtudiant.examFinal ← 80.0;  
unÉtudiant.créritable ← True
```

- Java permet la création d'une sorte spéciale de méthodes, appelées **CONSTRUCTEURS**, qui peuvent être utilisés pour initialiser les champs d'un objet lors de sa création.

```
Étudiant unÉtudiant = new Étudiant(1234567, 60.0, 80.0, true);
```

Constructeurs

- Un **constructeur** est une méthode spéciale dans une classe, utilisée pour créer un objet.
 - son nom est le même que celui de la classe;
 - aucun type de retour, mais retourne quand même un objet de cette classe (**opérateur new**);
 - habituellement publique;
 - peut avoir des paramètres.
- Les paramètres d'un constructeur, si présents, sont utilisés pour initialiser les valeurs d'un objet.
- Comme il peut y avoir plusieurs façons d'initialiser un objet, une classe peut avoir plusieurs constructeurs (de mêmes noms), que l'on peut **distinguer par des listes de paramètres différentes**.

Implémentation en Java

- Constructeur qui spécifie des valeurs pour tous les champs d'un objet **Étudiant**:

```
class Étudiant
{
    // ... Les champs seraient définis ici ...
    public Étudiant(int n, double ePartiel, double eFinal, boolean crédit)
    {
        numÉtud = n;
        examPartiel = ePartiel;
        examFinal = eFinal;
        créditable = crédit;
    }
    // ... Autres méthodes ...
}
```

- Ce constructeur peut être utilisé ainsi:

```
Étudiant unÉtudiant = new Étudiant(1234567, 60.0, 80.0, true);
```

Autre constructeur de la classe Étudiant

- Lors de l'inscription, nous ne pouvons pas encore déterminer les notes d'examen
- Nous pourrions donc fournir **aussi** ce constructeur:

+ Étudiant(n : int, crédit : boolean)

```
public Étudiant(int n, boolean crédit )
{
    numÉtud = n;
    examPartiel = 0.0;    // une valeur « sûre »
    examFinal = 0.0;    // une valeur « sûre »
    créditable = crédit;
}
```

- Usage:

```
Étudiant moiAussi = new Étudiant( 1069665, false );
```

- Quand il y a plus d'un constructeurs, ils doivent avoir des listes de paramètres qui peuvent être distinguées par le **nombre**, **l'ordre** et le **type** de paramètres.

Diagramme de classe avec constructeurs

Étudiant

– numÉtud : int
– examPartiel : double
– examFinal : double
– créditable : boolean

+ Étudiant (n : int, ePartiel : double, eFinal : double, crédit : boolean)
+ Étudiant (n : int, crédit : boolean)

+ getNumÉtud () : int
+ setNumÉtud (nouvNum : int)
+ getExamPartiel() : double
+ setExamPartiel(nouvNote: double)
+ getExamFinal() : double
+ setExamFinal(nouvNote: double)
+ getCréditable() : boolean
+ setCréditable(nouvValeur : boolean)
+ getNoteFinale(): double

Constructeur par défaut

- Voici un constructeur sans paramètre:

+ Étudiant()

```
public Étudiant( )  
{  
    numÉtud = 0;  
    examPartiel = 0.0;  
    examFinal = 0.0 ;  
    créditable = false;  
}
```

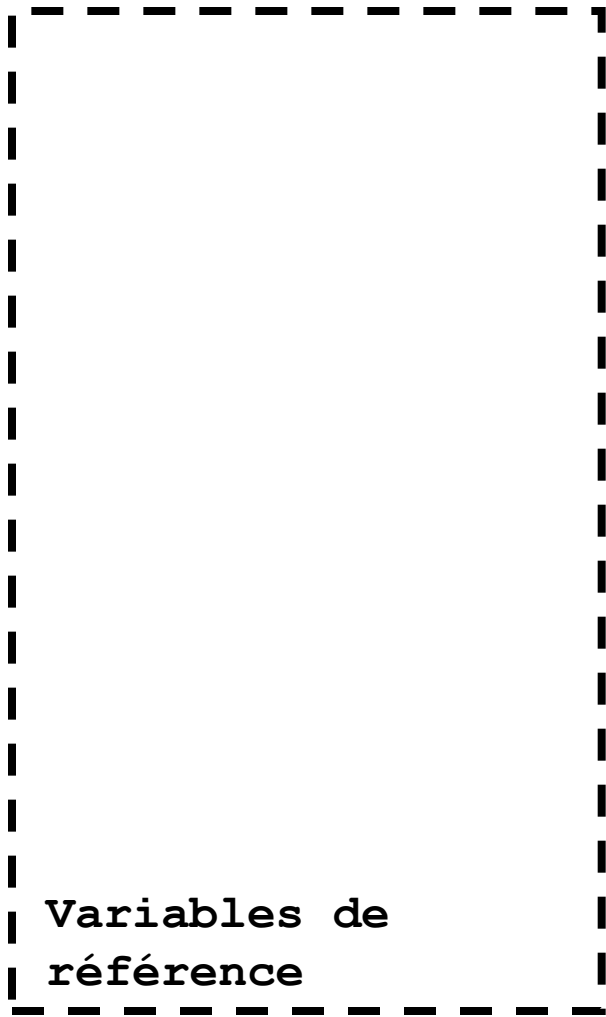
- Un constructeur sans paramètre est appelé **constructeur par défaut**.
 - Il est recommandé de toujours définir un constructeur par défaut qui affecte des valeurs « sûres » aux champs de l'objet.
- Si une classe ne définit aucun constructeur, alors le compilateur Java crée un constructeur par défaut.

Exercice 11-1: Utilisation des constructeurs

- Montrez comment les objets sont créés et référés par la méthode `main` suivante:

```
public class Section11
{
    public static void main(String [] args)
    {
        Étudiant étudiantA; // variable de référence
        Étudiant moiAussi; // une autre variable de référence
        Étudiant étudiantB; // une troisième variable de référence
        •
        •
        étudiantA = new Étudiant(1234567,60.0,80.0,true);
        moiAussi = new Étudiant(7654321,true);
        étudiantB = étudiantA;
        •
        •
        •
    }
}
```

Exercice 11-1: Utilisation des constructeurs



Classes avec champs tableau

Étudiant

```
- numÉtud : int
- devoirs : double[]
- examPartiel : double
- examFinal : double
- créditable : boolean

+ Étudiant ( n : int, ePartiel : double,
             eFinal : double, crédit : boolean )
+ Étudiant ( n : int, crédit : boolean )
+ getNumÉtud ( ) : int
+ setNumÉtud ( nouvNum : int )
+ getExamPartiel( ) : double
+ setExamPartiel( nouvNote: double )
+ getExamFinal( ) : double
+ setExamFinal( nouvNote: double )
+ getCréditable( ) : boolean
+ setCréditable( nouvValeur : boolean )
+ getNoteFinale( ) : double
- calcMoyDevoirs ( ) : double
```

- Le champ d'une classe peut être de n'importe quel type, incluant un type tableau.
- Ajoutons un tableau de nombres réels à la classe **Étudiant** afin de conserver les notes de devoirs.
- **ATTENTION:** encore ici, les **tableaux ne sont pas créés automatiquement!** Le tableau **devoirs** devra être créé après la création de l'objet **Étudiant**.

Classes avec champs tableau

```
public class Étudiant
{
    private int numÉtud ;
    private double[] devoirs;
    private double examPartiel ;
    private double examFinal ;
    private boolean créditable;
    // méthodes
}
```

- Lors de la création de l'objet, la variable de référence **devoirs** est encore à **null**...

Initialisation de champs tableau

- Voici un constructeur qui crée et initialise un tableau dans un objet. Ce constructeur a comme paramètre le nombre de devoirs.

```
public Étudiant( int nbDevoirs )
{
    numÉtud = 0;
    examPartiel = 0.0;
    examFinal = 0.0 ;
    créditable = false;
    devoirs = new double[nbDevoirs];

    // boucle pour initialiser chaque élément du tableau
    int index;
    for ( index = 0; index < nbDevoirs; index = index + 1 )
    {
        devoirs[index] = 0.0;
    }
}
```

Accesseurs pour champs tableau

- Un accesseur pour un champ tableau peut:
 - Retourner une référence au tableau tout entier:
+ `getDevoirs() : double[]`

```
public double [] getDevoirs()  
{  
    return devoirs;  
}
```

- Retourner l'une des valeurs du tableau, ce qui requiert un paramètre supplémentaire (index):
+ `getDevoirs(numéro : int) : double`

```
public double getDevoirs( int numéro )  
{  
    return devoirs[numéro];  
}
```

- Quelle est la meilleure approche?

Exercice 11-2: Calcul de la note finale ?

- Écrivez des méthode `getNoteFinale()` et `calcMoyDevoirs()` pour notre classe `Étudiant` qui retournent des `double` :
 - La note finale égale 55% de l'examen final, plus 20% de l'examen partiel, plus 25% de la moyenne de 5 devoirs.

```
private double calcMoyDevoirs()  
{
```

```
}
```

Exercice 11-2: Calcul de la note finale

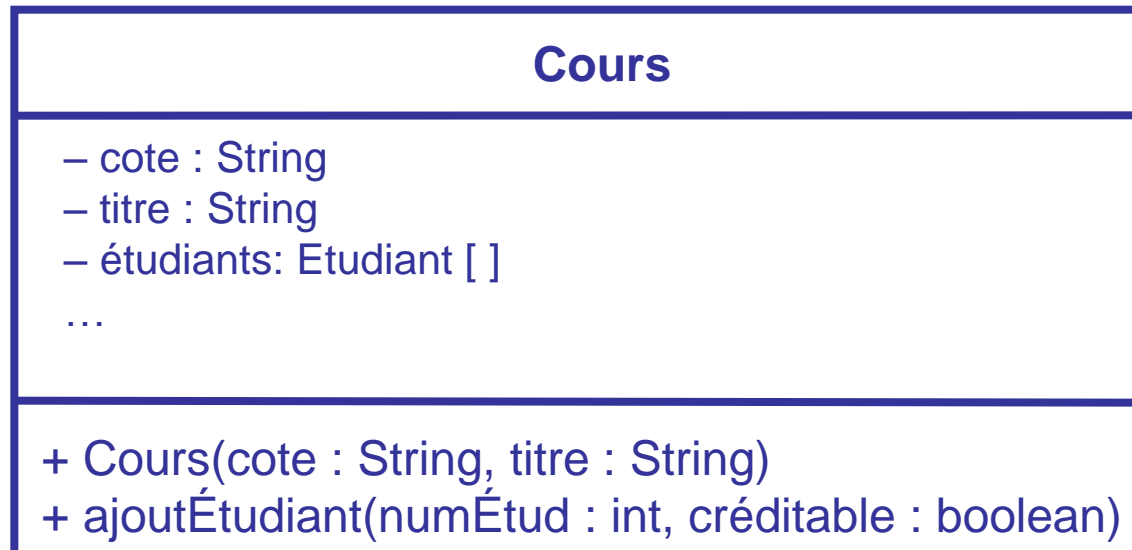
- Écrivez des méthode `getNoteFinale()` et `calcMoyDevoirs()` pour notre classe `Étudiant` qui retournent des `double` :
 - La note finale égale 55% de l'examen final, plus 20% de l'examen partiel, plus 25% de la moyenne de 5 devoirs.

```
public double getNoteFinale()  
{
```

```
}
```

Information sur un cours

- Comme nous avons maintenant une classe qui emmagasine l'information sur un **Étudiant**, comment pouvons-nous créer une classe **Cours** pour l'information sur tous les étudiants d'un cours?



Exercice 11-3: Tableau d'objets

- Montrez comment les objets sont créés et référés par la méthode **main** suivante:

```
public class Section11
{
    public static void main(String [] args)
    {
        Cours unCours;
        unCours = new Cours("ITI1520","Intro. à l'info.");
        unCours.ajoutÉtudiant(123456,true);
        unCours.ajoutÉtudiant(654321,false);
    }
}
```

Exercice 11-3: Tableau d'objets



Valeurs communes à une classe

- Dans nos objets **Étudiants** et **Cours**, nous avons utilisé des **variables d'instances** et créé des **méthodes d'instances**.
 - Un ensemble de variables d'instances est créé pour chaque nouvel objet
 - Les méthodes d'instance utilisent les variables d'instance de l'objet où la méthode est invoquée.
- Si nous avons une valeur que nous aimerions voir partagée par tous les étudiants.
 - Exemples: Les poids de l'examen final, de l'examen partiel et des devoirs, pour calculer la note finale.
- Le code suivant ne nous permet pas de modifier ces poids sans avoir besoin de recompiler:

```
public double getNoteFinale()  
{  
    double moyDevoirs = calcMoyDevoirs( );  
    return 0.2 * examPartiel + 0.55 * examFinal + 0.25 * moyDevoirs;  
}
```

Poids communs pour la classe Étudiant

Étudiant

- poidsDevoirs: double = 0.25
- poidsPartiel: double = 0.20
- poidsFinal: double = 0.55
- numÉtud : int
- devoirs : double[]
- examPartiel : double
- examFinal : double
- créditable : boolean

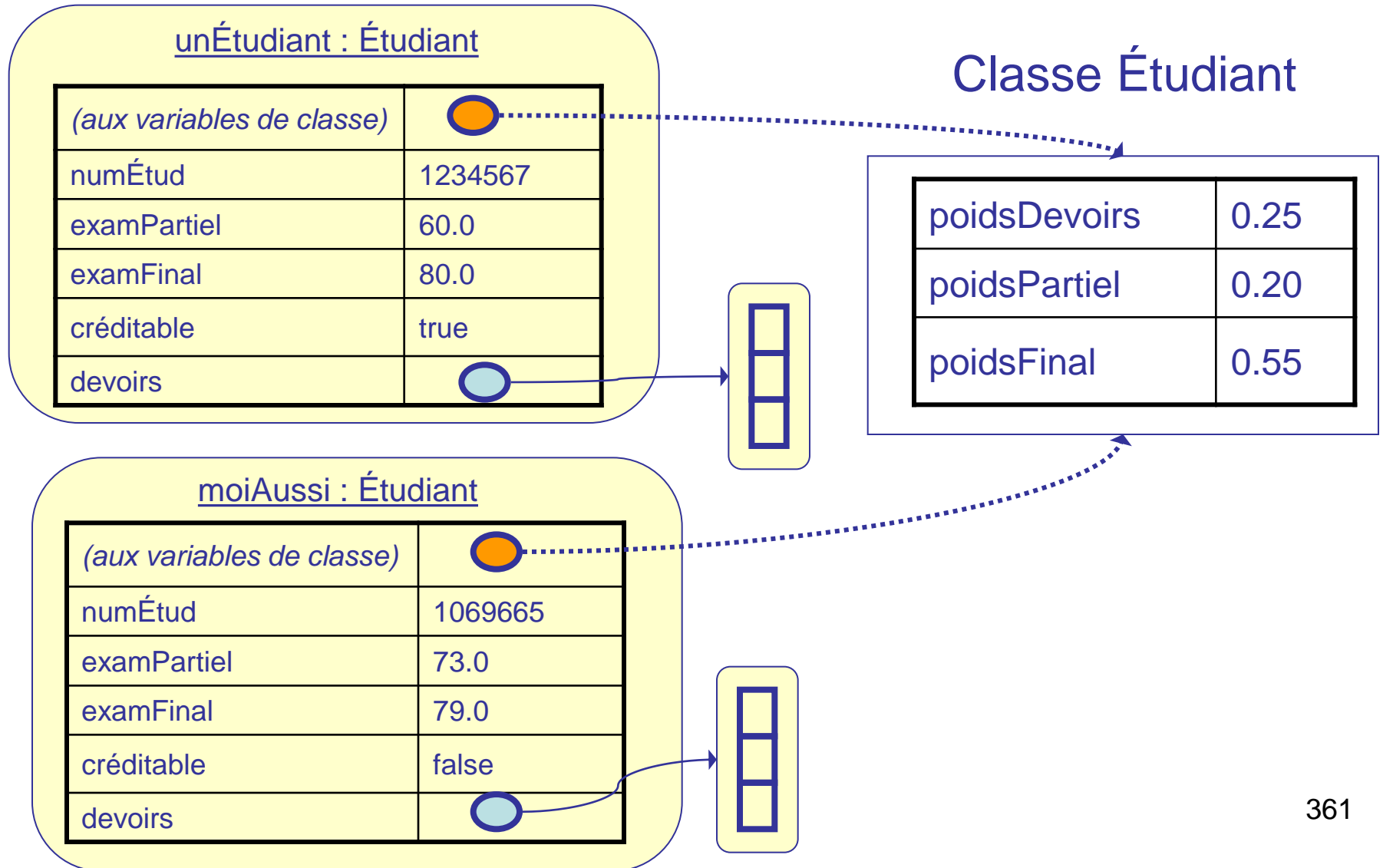
- + getNumÉtud () : int
- + setNumÉtud (nouvNum : int)
- + getDevoirs(numDevoir : int) : double
- + setDevoirs(numDevoir : int,
 nouvNote : double)
- + getExamPartiel() : double
- + setExamPartiel(nouvNote: double)
- + getExamFinal() : double
- + setExamFinal(nouvNote: double)
- + getCréditable() : boolean
- + setCréditable(nouvValeur : boolean)
- + getNoteFinale() : double
- calcMoyDevoirs() : double

- Le problème avec cette approche est que TOUS les objets étudiant auront leur propre copie de **poidsDevoirs**, **poidsPartiel**, et **poidsFinal**, ce qui prendrait de l'espace mémoire supplémentaire (inutilement).
- Si ces poids changent, tous les objets devraient être mis à jour!
- De plus, il pourrait y avoir des incohérences entre les poids de différents objets.

Variables de classe

- Un autre type de variable utile en orienté-objet se nomme "variable de classe" (aussi appelé "variable statique").
- Les variables de classe ne sont PAS emmagasinées dans les objets individuels; elles appartiennent à la classe, où elles sont emmagasinées.
- CHAQUE objet créé de la classe a accès aux variables de classe, même si elles sont privées.
 - Si les variables de classe sont publiques, alors des méthodes à l'extérieure de la classe ont aussi accès à eux.

Variables de classe et d'instance



Variables de classe en diagrammes UML

Étudiant

– poidsDevoirs: double = 0.25
– poidsPartiel: double = 0.20
– poidsFinal: double = 0.55
– numÉtud : int
– devoirs : double[]
– examPartiel : double
– examFinal : double
– créditable : boolean

+ getNumÉtud () : int
+ setNumÉtud (nouvNum : int)
+ getDevoirs(numDevoir : int) : double
+ setDevoirs(numDevoir : int, nouvNote : double)
+ getExamPartiel() : double
+ setExamPartiel(nouvNote: double)
+ getExamFinal() : double
+ setExamFinal(nouvNote: double)
+ getCréditable() : boolean
+ setCréditable(nouvValeur : boolean)
+ getNoteFinale() : double
– calcMoyDevoirs() : double

- Les variables de classe sont soulignées, les variables d'instance ne le sont pas.
- Notez que les valeurs initiales des variables de classe ont été fournies.

Traduction vers Java

```
public class Étudiant
{
    // Variables de classe                (mêmes pour tous les étudiants)
    static private double poidsDevoirs = 0.25;
    static private double poidsPartiel = 0.20;
    static private double poidsFinal = 0.55;

    // Variables d'instances              (une copie par objet étudiant)
    private int numÉtud ;
    private double[] devoirs;
    private double examPartiel ;
    private double examFinal ;
    private boolean créditable;

    public double getNoteFinale()
    {
        double moyDevoirs = calcMoyDevoirs( );
        double noteFinale = Étudiant.poidsPartiel * this.examPartiel
            + Étudiant.poidsFinal * this.examFinal
            + Étudiant.poidsDevoirs * this.moyDevoirs;
        return noteFinale;
    }
}
```

Méthodes de classe

- Nous pouvons maintenant changer le calcul de la note finale pour tous les étudiants en changeant la valeur des poids.
- Nous devons écrire une méthode modificateur pour chaque poids.
 - Ce modificateur devrait être une méthode de **CLASSE**, parce que les valeurs sont associées à la classe et non aux objets.
 - Nous pouvons aussi initialiser les poids avant de commencer à créer des objets.
- Une méthode de classe est invoquée comme suit:
`NomDeLaClasse.unNomDeMéthode()`
 - Tout comme les classes **ITI1520** et **Math!**
- Une méthode de classe **NE PEUT PAS** utiliser de variables d'instance parce qu'une méthode n'est associée à aucun objet en particulier.

Méthodes de classe en diagrammes UML

Étudiant

– poidsDevoirs: double = 0.25
– poidsPartiel: double = 0.20
– poidsFinal: double = 0.55
– numÉtud : int
– devoirs : double[]
– examPartiel : double
– examFinal : double
– créditable : boolean

+ getPoidsDevoirs (): double
+ setPoidsDevoirs (double poids)
+ getPoidsPartiel (): double
+ setPoidsPartiel (double poids)
+ getPoidsFinal (): double
+ setPoidsFinal (double poids)
+ getNumÉtud () : int
+ setNumÉtud (nouvNum : int)
+ getDevoirs(numDevoir : int) : double
+ setDevoirs(numDevoir : int, nouvNote : double)
+ getExamPartiel() : double
+ setExamPartiel(nouvNote: double)
+ getExamFinal() : double
+ setExamFinal(nouvNote: double)
+ getCréditable() : boolean
+ setCréditable(nouvValeur : boolean)
+ getNoteFinale() : double
– calcMoyDevoirs() : double

- Les méthodes de classe sont soulignées, les méthodes d'instance ne le sont pas.

Méthodes **static** en Java

- Comme avec les variables de classe, les méthodes de classe sont spécifiées à l'aide du mot-clé **static**.
 - Notez: **main** est toujours une méthode de classe.
- Écrivez des méthodes de classe pour **Étudiant** afin de modifier les valeurs des poids pour l'examen final, l'examen partiel, et les devoirs:

```
public static void setPoidsFinal ( double poids )
{
    Étudiant.poidsFinal = poids;
}
public static void setPoidsPartiel( double poids )
{
    Étudiant.poidsPartiel = poids;
}
public static void setPoidsDevoirs( double poids )
{
    poidsDevoirs = poids; // Étudiant est optionnel
}
```

Exercice 11-4: Utilisation des variables et méthodes de class

- Quelle sortie sera produite par la méthode `main` suivante?

```
public class Section11
{
    public static void main(String [] args)
    {
        int numDev;
        Étudiant étudiantA; // variable de référence
        Étudiant moiAussi; // une autre variable de référence
        étudiantA = new Étudiant(1234567, 60.0, 79.0, true);
        moiAussi = new Étudiant(7654321, 54.5, 83.4, true);
        for(numDev=0 ; numDev<5 , numDev=i+1)
        {
            étudiantA.setDevoirs(numDev, 60.0);
            moiAussi.setDevoirs(numDev, 65.0);
        }
        System.out.println("La note pour étudiant "+étudiantA.getNumÉtud()+
            " est "+ étudiantA.getNoteFinale());
        Étudiant.setPoidsPartiel(0.30);
        Étudiant.setPoidsDevoirs(0.15);
        System.out.println("La note pour étudiant "+moiAussi.getNumÉtud()+
            " est "+ moiAussi.getNoteFinale());
        System.out.println("La note pour étudiant "+étudiantA.getNumÉtud()+
            " est "+ étudiantA.getNoteFinale());
    }
}
```

Exercise 11-4: Utilisation des variables et méthodes de class



Fenêtre de terminal



Résumé de la conception de classes (1)

- Dans un langage orienté objet comme Java, concevoir une classe représente une grande partie de l'effort de création de logiciels.

"Classes struggle, some classes triumph, others are eliminated." --[Mao Zedong](#)

- Il y a plusieurs décisions à prendre:
 - Quelle information devrait être dans la classe?
 - Quelles sont les champs que chaque objet devrait avoir?
 - Quelles champs qui devraient être associées à la classe?
 - De quels types sont ces champs?
 - Comment initialiser, accéder, et modifier ces champs?
 - Quelles sont les opérations que nous pourrions demander à la classe d'exécuter?
 - Quelles autres méthodes d'instance sont requises?
 - Quelles autres méthodes de classe sont requises?
 - Quels sont les algorithmes de toutes ces méthodes?

Résumé de la conception de classes (2)

- À considérer lors des décisions liées à la conception d'une classe:
 - Comment cette classe pourrait-elle être modifiée dans le futur?
 - Y a-t-il quelque chose de codé de façon permanente qui devrait être fait avec des variables?
 - Problèmes potentiels:
 - Y a-t-il une possibilité que la variable soit utilisée avant de se voir affecter une valeur?
 - Quand une méthode est invoquée, qu'est-ce que la méthode présuppose par rapport aux valeurs des paramètres? Ces hypothèses sont-elles vérifiées?

Exemple de classe: Fraction

- Nous allons définir une classe dont les objets représentent des fractions telles que $2 / 3$, $(-1) / 5$, ou $7 / 4$.
- Comme $2 / 3 = 4 / 6 = 6 / 9$, et $(-1) / 2 = 1 / (-2)$, etc., des fractions avec des numérateurs et dénominateurs différents peuvent quand même représenter une même fraction.
- Nous allons emmagasiner une fraction dans sa *forme réduite*:
 1. Le numérateur et le dénominateur n'ont pas de facteur commun autre que 1 ou -1.
 2. Le dénominateur doit être positif.
- En mathématiques, le dénominateur d'une fraction ne peut jamais être 0. Mais nous n'avons pas de façon d'interdire aux utilisateurs de créer une fraction avec 0 comme dénominateur. Java a un mécanisme appelé **gestion des exceptions** (*exception handling*) qui permet de gérer de telles situations inhabituelles. Cependant, nous ne couvrirons pas ce mécanisme dans ce cours.

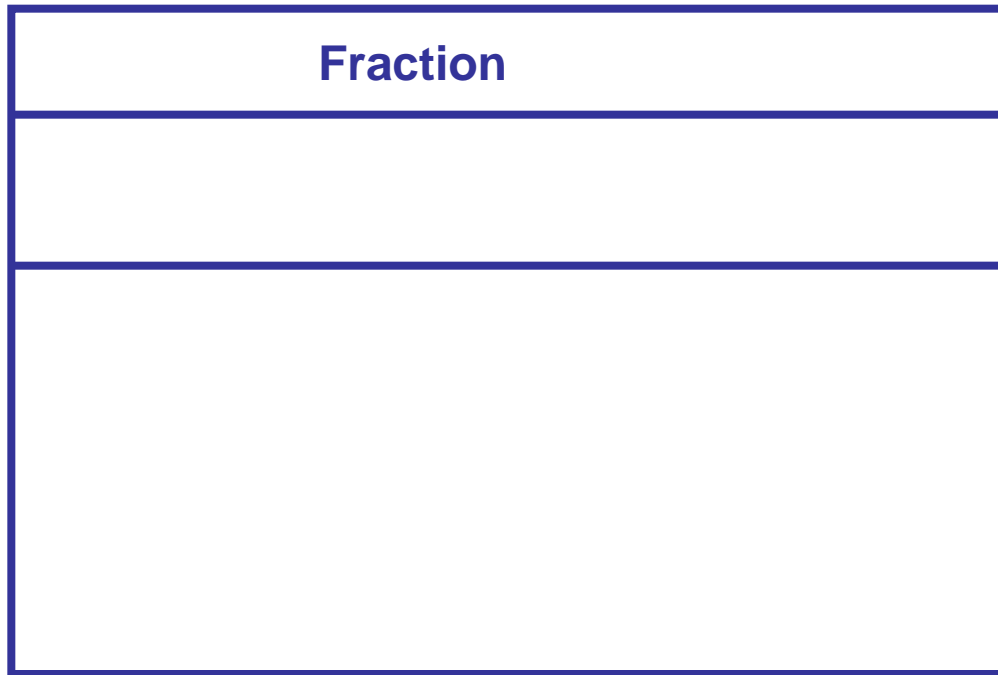
Exigences de la classe Fraction

- Une fraction a deux entiers: un numérateur et un dénominateur.
 - Interdiction d'accéder aux champs individuels...
- Le numérateur d'une fraction est un entier.
- Le dénominateur d'une fraction est un entier différent de 0.
 - Si le dénominateur n'est pas spécifié lors de la création, nous supposons qu'il vaut 1.
- Une fraction est toujours en forme réduite:
 - Le plus grand commun diviseur (PGCD) du numérateur et du dénominateur est toujours 1.
 - Le dénominateur est toujours positif.
 - Exemple: $6/-9$ doit être représenté par $-2/3$
 - Cas spécial: si le numérateur est 0, la fraction est représentée par $0/1$
- Une fraction avec dénominateur 1 devrait être affichée comme un entier, et selon la forme numérateur/dénominateur dans les autres cas.

Exercice 11-5: Conception de la classe Fraction ?

- Quelle information avons-nous besoin d'emmagasiner dans une Fraction?
- De quelles opérations avons-nous besoin?
 - Nous nous limiterons à l'addition comme opération mathématique

Exercice 11-5: Conception de la classe Fraction



Exercice 11-6: Réduction d'une Fraction ?

- Afin d'obtenir la forme réduite d'une **Fraction**, nous utiliserons une méthode de support appelée **réduit**, qui pourra faire appel à **pgcd(a,b)**, une méthode qui retourne le PGCD de deux entiers (à faire, page suivante). Écrivez ces deux méthodes en Java:

Exercice 11-7: Méthode pour PGCD



- Algorithme récursif pour calculer le PGCD(a , b):
 - Si $(a \text{ modulo } b)$ est 0, PGCD(a , b) vaut b
 - $a \text{ modulo } b$ est le reste de a divisé par b
 - Sinon, PGCD(a , b) vaut PGCD(b , $a \text{ modulo } b$)
- Question: Cet algorithme converge-t-il vers le cas de base?
 - Notez que $(a \text{ modulo } b)$ vaut au plus $b-1$.
- Attention: et si b vaut 0?

Exercice 11-8: Constructeurs pour Fraction

- Écrivez des constructeurs pour une Fraction qui:
 - accepte 2 entiers: le numérateur et le dénominateur
 - accepte 1 entier, à être converti en une Fraction

Exercice 11-9: Affichage de fractions

- Écrivez une méthode Java pour afficher une **Fraction**.
- Usage:
 - **Fraction f1 = new Fraction (6, -9);**
 - **f1.affiche();**
- Résultat: **-2/3**

Exercice 11-10: Somme de fractions



- Écrivez une méthode Java pour additionner deux objets Fraction

- Usage:

```
Fraction f1 = new Fraction( 1, 2 );
```

```
Fraction f2 = new Fraction( 1, 3 );
```

```
Fraction somme = f1.plus( f2 );
```

```
somme.affiche( );
```

- Résultat: 5/6

Exercice 11-11: Ajouter un entier à une fraction



- Écrivez une méthode Java dont l'usage est:

```
Fraction f1 = new Fraction( 5, 2 );  
Fraction somme = f1.plus( 3 );  
somme.affiche( );
```

- Résultat: 11/2

Autres opérations arithmétiques

- Nous vous encourageons à écrire des méthodes similaires pour les autres opérations (soustraction, multiplication, division) sur deux fractions et sur une fraction et un entier.
- Qu'y a-t-il de spécial pour la division?

Somme en méthode de classe (1)



- Afin de rendre les opérandes symétriques, nous pourrions transformer ces méthodes en méthodes de classe.
- `this` ne peut pas être utilisé ici dans ces méthodes.

```
public
```

```
{
```

```
    int n = f.numérateur * g.dénominateur
```

```
        + f.dénominateur * g.numérateur;
```

```
    int d = f.dénominateur * g.dénominateur;
```

```
    return
```

```
}
```

Somme en méthode de classe (2)



- Il faut cependant ajouter une troisième méthode pour l'autre combinaisons de paramètres:



- Usage: Ces méthodes de classe sont utilisées comme dans les exemples suivants:

```
Fraction    a = new Fraction(2, 3),  
             b = new Fraction(1, 5);  
  
Fraction    somme1 = Fraction.somme(a, b),  
             somme2 = Fraction.somme(a, 3),  
             somme3 = Fraction.somme(2, b);
```

Classe Fraction

```
public class Fraction
{
    private int numérateur;
    private int dénominateur;

    // MÉTHODES DE SUPPORT
    private static int pgcd (int a, int b) ...
    private void reduit( ) ...
    // ACCESSEURS
    public int getNumérateur() ...
    public int getDénominateur() ...
    // CONSTRUCTEURS
    public Fraction(int n, int d) ...
    public Fraction(int entier) ...
    // SOMME, EN MÉTHODE D'INSTANCE
    public Fraction plus(Fraction g) ...
    public Fraction plus(int entier) ...
    // SOMME, EN MÉTHODE DE CLASSE
    public static Fraction somme(Fraction f, Fraction g) ...
    public static Fraction somme(Fraction f, int entier) ...
    public static Fraction somme(int entier, Fraction f) ...
    // MÉTHODE AFFICHE
    public void affiche( ) ...
} // Classe Fraction
```


Programme-Test pour Fraction

```
// Cette classe teste notre réalisation de la classe Fraction
public class TestFraction
{
    public static void main(String [ ] args)
    {
        // teste les constructeurs et l'affichage
        Fraction a = new Fraction (1, -4);
        Fraction b = new Fraction (-14, -24);
        Fraction c = new Fraction (0, -3);
        Fraction d = new Fraction (2);
        Fraction e = new Fraction (0);
        int i = 2;
        a.affiche( ); System.out.println( ); // devrait afficher -1/4
        b.affiche( ); System.out.println( ); // devrait afficher 7/12
        c.affiche( ); System.out.println( ); // devrait afficher 0
        d.affiche( ); System.out.println( ); // devrait afficher 2
        e.affiche( ); System.out.println( ); // devrait afficher 0
        // teste somme
        a.plus(b).affiche( ); System.out.println( ); // devrait afficher 1/3
        a.plus(d).affiche( ); System.out.println( ); // devrait afficher 7/4
        c.plus(d).affiche( ); System.out.println( ); // devrait afficher 2
        e=Fraction.somme(-1, a.plus(i));
        e.affiche( ); System.out.println( ); // devrait afficher 3/4
    }
}
```

Dernier retour sur la dissimulation

- La classe Fraction représente un bon exemple de **dissimulation de données**:
 - Le code du programme TestFraction n'a pas d'accès direct à la réalisation de la classe Fraction (et ses champs numérateur et dénominateur).
 - Le code du programme TestFraction ne connaît pas et n'a pas à connaître comment la classe Fraction est réalisée.
 - Par exemple, la classe Fraction pourrait tout aussi bien représenter une fraction par un tableau de deux entiers:

```
public class Fraction
{
    private int [ ] éléments = new int[2];
    // élément[0] est le numérateur,
    // élément[1] est le dénominateur
    ...
}
```

- La réalisation des méthodes peut changer, mais si l'**interface** demeure la même, alors le code du programme invocateur n'a pas à changer!

Quelques pensées finales...

- « À la source de toute erreur blâmée sur l'ordinateur, on retrouve deux erreurs humaines, l'une d'entre elles étant de blâmer l'autre sur l'ordinateur. »
 - Anonyme
- "We shall do a much better programming job, provided we approach the task with a full appreciation of its tremendous difficulty, provided that we respect the intrinsic limitations of the human mind and approach the task as very humble programmers. "
 - Alan Turing

C'est tout!

- Merci de votre attention.
- Révision et préparation de l'examen final...