Université Mohammed V
FACULTE DES SCIENCES
RABAT / FSR
Département informatique

Programmation orientée objet en JAVA

Pr. REDA Oussama Mohammed

Programmation oriontée objet

Eléments du langage Java

Principes de programmation

Contenu

- **Langage Java I**
  - Identificateurs et littéraux
  - Types primitifs de java en détails
  - Variable (déclaration et affectation)
  - Opérateurs en détails
  - Conversion de type ou transtypage
  - Instructions de contrôle en détails
  - Entrées/sorties
  - Programme principal
  - Commentaires
- **Langage Java II**
  - Boucle for
  - Références
  - Définition et utilisation des tableaux
- **Langage Java III**
  - Bloc de code
  - Définition formelle (en-tête)
    - Procédure
    - Fonction
  - Portée des variables

Principes de programmation

- **Survol**
  - –Type de données (en mémoire)
  - –Opérateurs
  - –Instructions de contrôle
- **Langage Java**
  - –Identificateurs et littéraux
  - –Types primitifs de java en détails
  - –Variable (déclaration et affectation)
  - –Opérateurs en détails
  - –Conversion de type ou transtypage
  - –Instructions de contrôle en détails
  - –Entrées/sorties
  - –Programme principal
  - –Commentaires

Type de données

- **Type de données**
  - **Nécessite trois informations pour l'utilisation**
    - Catégorie de données
    - Opérations permises
    - Les limites (nombre maximum, minimum, taille, ...)

Exemple : Type entier

- **Catégorie de données** : nombres sans partie décimale
- **Opérations permises** : +, -, *, /, %, <, >, >=, ...
- **Limites** : -32768 à 32767 (16 bits)

Opérateurs

- **Deux sortes d'opérateurs** *(plus un ternaire)*
  - Unaire (un opérande)
  - Binaire (deux opérandes)
  - Ternaire (trois opérandes)
  - Le résultat est souvent du même type que les opérandes avec l'existence de quelques exceptions.
    - Exemple : 3/2 donnera 1 et non 1.5

      3==2 donnera *true* et non un entier

Opérateurs

- **Plusieurs opérateurs prédéfinis, utilisables selon les types de données**

  Unaires :

  – Entiers et réels : *incrémentation* ++
  décrémentation --
  valeur positive +
  valeur négative - *(opposé de +)*
  complément ~

  – Booléen : négation !

Opérateurs

- **Plusieurs opérateurs prédéfinis, utilisables selon les types de données**

  Binaires :

  – Entier et réel : *arithmétiques* (+, - , * , /, %)
  relationels (<, >, <=, >=)

  ➢ Entier : *opérateurs sur les bits* (&, |, ou exclusif(^))

  – Chaîne de caractères : + (concaténation)
  – Booléen : *conditionnels* (et(&&), ou logique(||), ou exclusif(^))

  – Pour tous ces types *(excépté les Chaînes de caractères avec sémantique différente pour l'opérateur d'affectation)*
    - ==(égalité), !=(différence), =(affectation)

Opérateurs

- **Plusieurs opérateurs prédéfinis, utilisables selon les types de données**

  Ternaires :

  – L'operateur ?:

  testCondition ? value1 : value2

  Abréviation de l'instruction *if-then-else*

Instructions de contrôle

- Deux catégories
  - **Sélectives**
    - Exécution unique d'instructions selon le résultat de l'évaluation d'une expression booléenne (exemple : if)
  - **Répétitives**
    - Exécution répétée d'instructions selon le résultat de l'évaluation d'une expression booléenne (exemple : while)

Identificateurs et littéraux

- **Identificateurs**
  - Le nom choisi pour une variable ou tout autre élément (ex: salaire)

- **Littéral (aux)**
  - Valeur fixe (exemple : 123, 12.45, "bonjour java" , ‘X’)
  - À éviter. Utilisez plutôt des constantes

Identificateurs et littéraux

- **Règles de création des identificateurs**
  - Doit commencer par une lettre, caractère de soulignement (_), ou un des symboles monétaires ($) Unicode (pas recommandé)
  - Ensuite, tout caractère alphanumériques, caractère de soulignement (_) et symboles Unicode mais aucun autre symbole n'est permis.
  - Un seul mot (sans éspace ni tiret)
  - Le langage Java est sensible à la casse des caractères.

  Exemple:

  Nom_Variable, NomVariable, nomVariable (correct)

  Salaire employé, Un+Deux, Hello!, 1er (illégal)

Types primitifs de java en détails

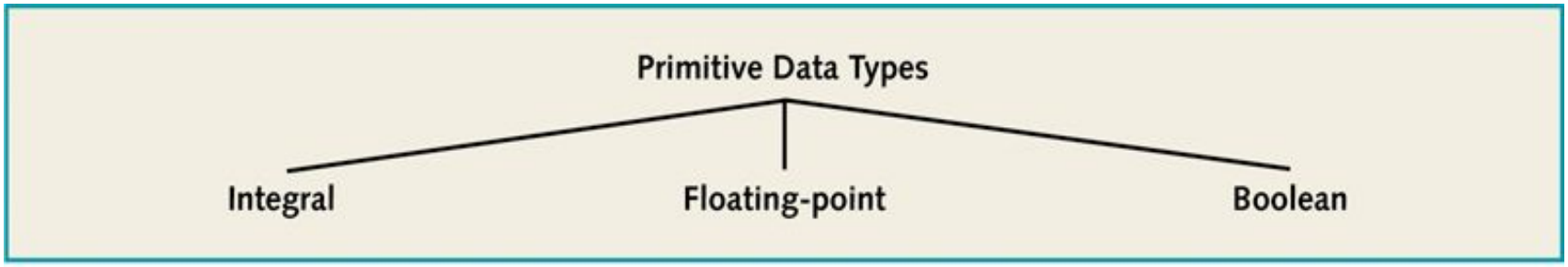


Figure 2-1 Primitive Data Types

- **Entiers**

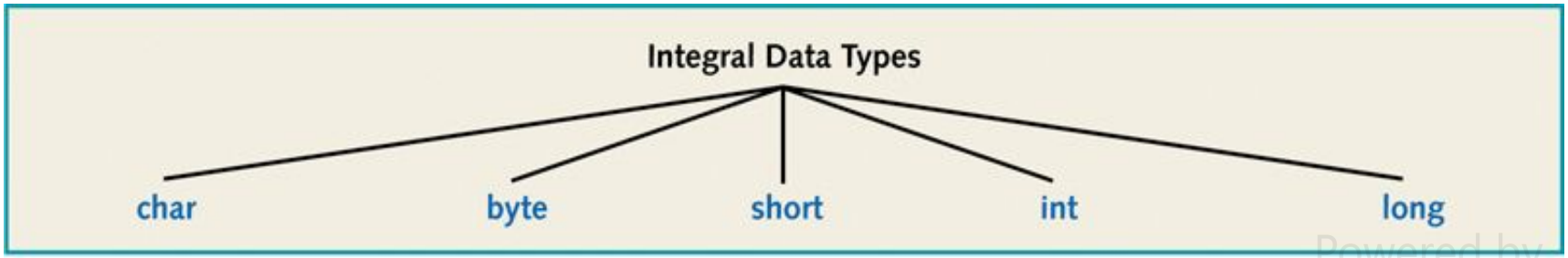


Figure 2-2 Integral Data Types

Types primitifs de java en détails

- **Entier**
  - byte (8 bits) -128 à +127
  - short(16 bits) -32768 à 32767
  - int (32 bits) -2147483648 à +2147483647 ***
  - long (64 bits) -9223372036854775808 à 9223372036854775807
- **Réel**
  - float (32 bits) $-3.4 * 10^{38}$ à $+3.4 * 10^{38}$
    (Précision = 7 en moyenne)
  - double (64 bits) $-1.7 * 10^{308}$ à $+1.7 * 10^{308}$ ***
    (Précision = 15 en moyenne)

***Les littéraux réels sont de type double

***utilisé la plupart du temps

Types primitifs de java en détails

- **Caractère**
  - char (16 bits) Unicode
    - Entre apostrophes (exemple : ‘a’, ‘A’, ‘1’,’%’ ou ‘\u0000’ à ‘\uFFFF’)
    - ‘\u0000’ : valeur par défaut
- **Booléen**
  - boolean
    - true, false

Variable (déclaration et affectation)

- **Variable**
  - Espace mémoire *modifiable*
  - Doit avoir un *identificateur*
  - Doit avoir un *type*
  - Accessible en tout temps via l'*identificateur* par la suite

Variable (déclaration et affectation)

- **Constante**
  - Espace mémoire *NON* modifiable
  - Doit avoir une valeur initiale
  - Doit avoir un identificateur
  - Doit avoir un type
  - Accessible en tout temps via l'identificateur par la suite
  - Mot réservé en Java : **final**

    Ex: final int MAX = 500;

Variable (déclaration et affectation)

- **Déclaration en Java**
  - **<type> identificateur;**
    - Exemple : byte age;
    - float salaire;
    - double precision;

Variable (déclaration et affectation)

- **Affectation**
  - Variable = littéral [op littéral ou variable] [op ...];
    - Exemple : age = 28;
      salaire = 125.72 + 50.1;
      precision = 0.00000001 * salaire;

- **Possible d'affecter lors de la déclaration(conseillé)**
    - Exemple : byte age = 28;
      float salaire = 125.72;
      double precision = 0.00000001 * salaire;
    - Utilisation d'une variable non-initialisée ne compile pas

Opérateurs en détails

- Six catégories d'opérateurs
  - Arithmétique, booléen, affectation, relationnel, bits, ternaire

Le tableau suivant montre l'ordre de priorité des opérateurs.

Opérateurs en détails

Operator Precedence

Operators	Precedence
postfix	*expr++ expr--*
unary	*++expr --expr +expr -expr ~ !*
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

- L'ordre de priorité de haut en bas.

- L'operateur le plus prioritaire se trouve en haut de la liste.

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html

Opérateurs en détails

- **Quelques points à considérer**
  - L'ordre de priorité des opérateurs est important dans l'évaluation d'une expression.
    - 3 + 8 * 9 – 4 + 8 / 2 = 75
    - (3 + 8) * 9 – (4 + 8) / 2 = 93
  - Le type des opérandes et du résultat sont importants
    - entier **op arithmétique** entier = entier 3/2 donne 1
    - entier **op relationnel** entier = booléen 3>2 donne vrai
    - entier **op arithmétique** réel= réel 3/2.0 donne 1.5
    - ...

Conversion de type ou transtypage

- Principe
  - Le résultat d'une opération est du type qui prend le plus de place en mémoire. C'est ce qu'on appelle la conversion de type ou le transtypage

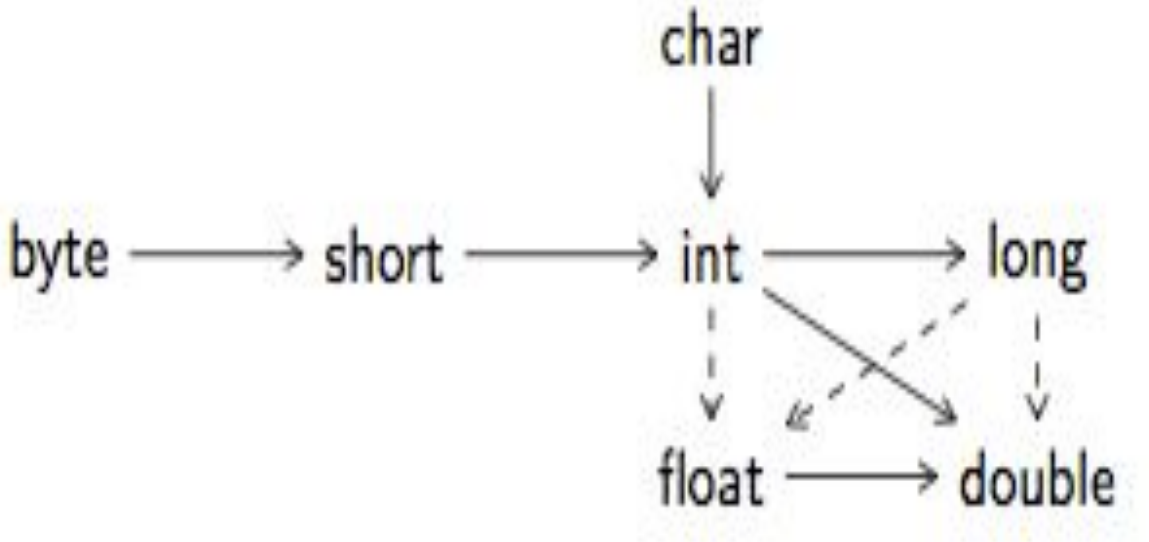
- Deux catégories
  - Implicite
    - Fait automatiquement par le compilateur
  - Explicite
    - Fait explicitement par le programmeur (Utilisé pour éviter la coercition implicite des Types)
    - (**<nouveau type>**) (valeur/variable)

Exemple : int x = 10;

```
x = 7,9 + 6,7;                  //Implicite   x = 14
x = (int) (7,9) + (int) (6,7) ;  //Explicite   y = 13
```

Conversion de type ou transtypage

Conversion implicite des types primitifs



Flèche pleine: Conversion sans perte de précision
Flèche ------>: Conversion avec possibilité de perte de précision

http://www.ukonline.be/cours/java/apprendre-java/chapitre2-4

Instructions de contrôle en détails

- Instructions sélectives

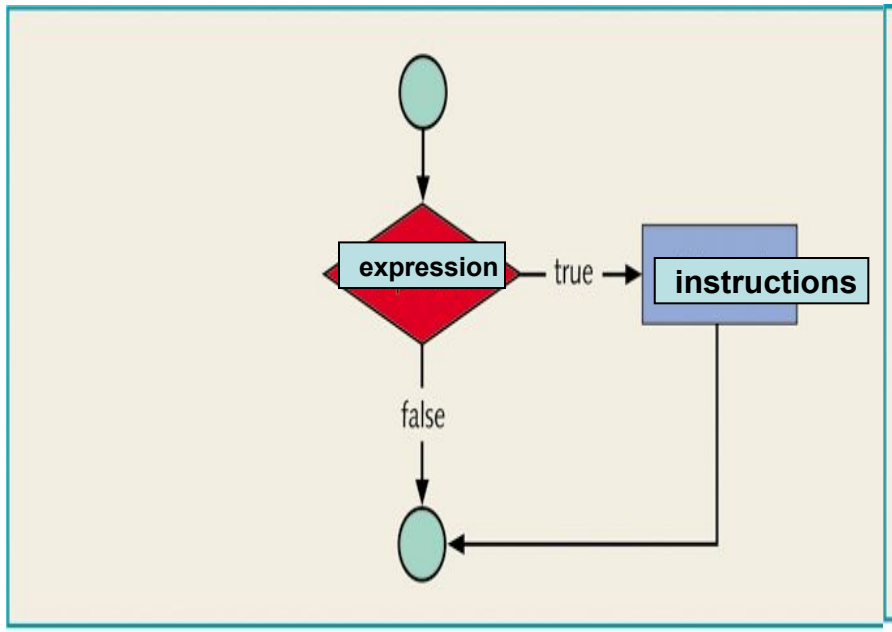


Figure 4-4 One-way selection

Si (if)

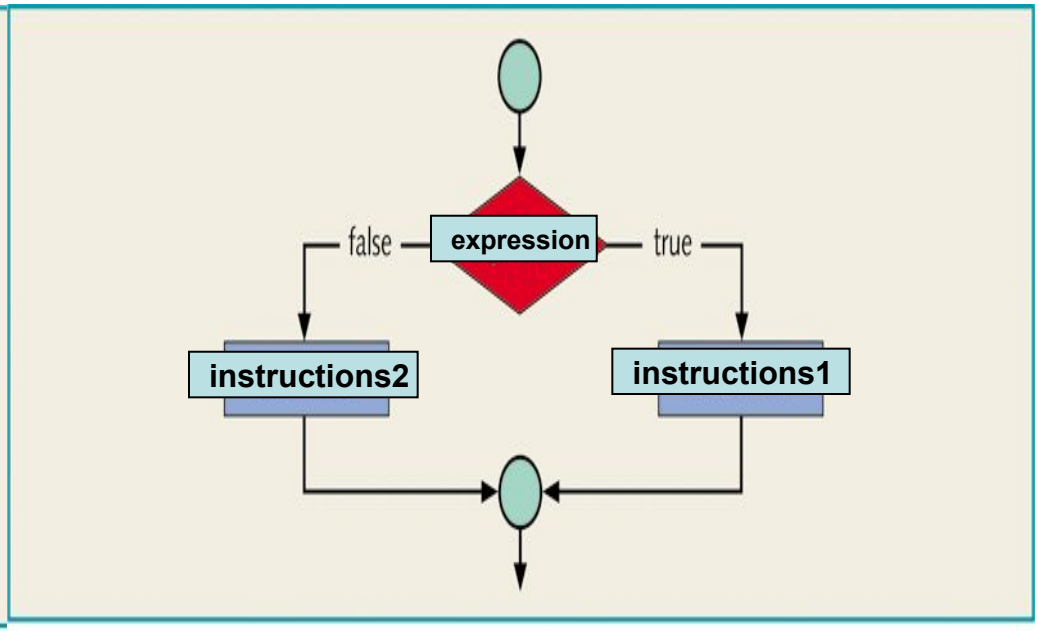


Figure 4-6 Two-way selection

si-sinon (if-else)

Instructions de contrôle en détails

- Instructions sélectives
  - Si (if); si-sinon (if-else)
    - **If** (expression booléenne) {
      Instructions
      }
    - **If** (expression booléenne) {
      Instructions
      }
      else{
      Instructions
      }
  - Opérateur conditionnel ou opérateur ternaire
    - **Expression booléenne ? Expression1 : Expression2**
      - Exemple : plusGrand = (valeur1 > valeur2) **?** valeur1 **:** valeur2

Instructions de contrôle en détails

- Instructions sélectives
  - Si (if); si-sinon (if-else)

  - ***Est ce qu'un étudiant a validé le module?***
  - ***Qui sont les étudiants qui ont validés le module?***
  - **if** (note>=10) {
    aValidé=true;
    return aValidé ;
    }

  - ***Afficher tous les étudiants avec la mention qu'ils ont eu au module?***

    - **if** (note>=10) {

      mention="réussi";
      }
      else{
      mention="échec";
      }
      System.out.println(mention);

Instructions de contrôle en détails

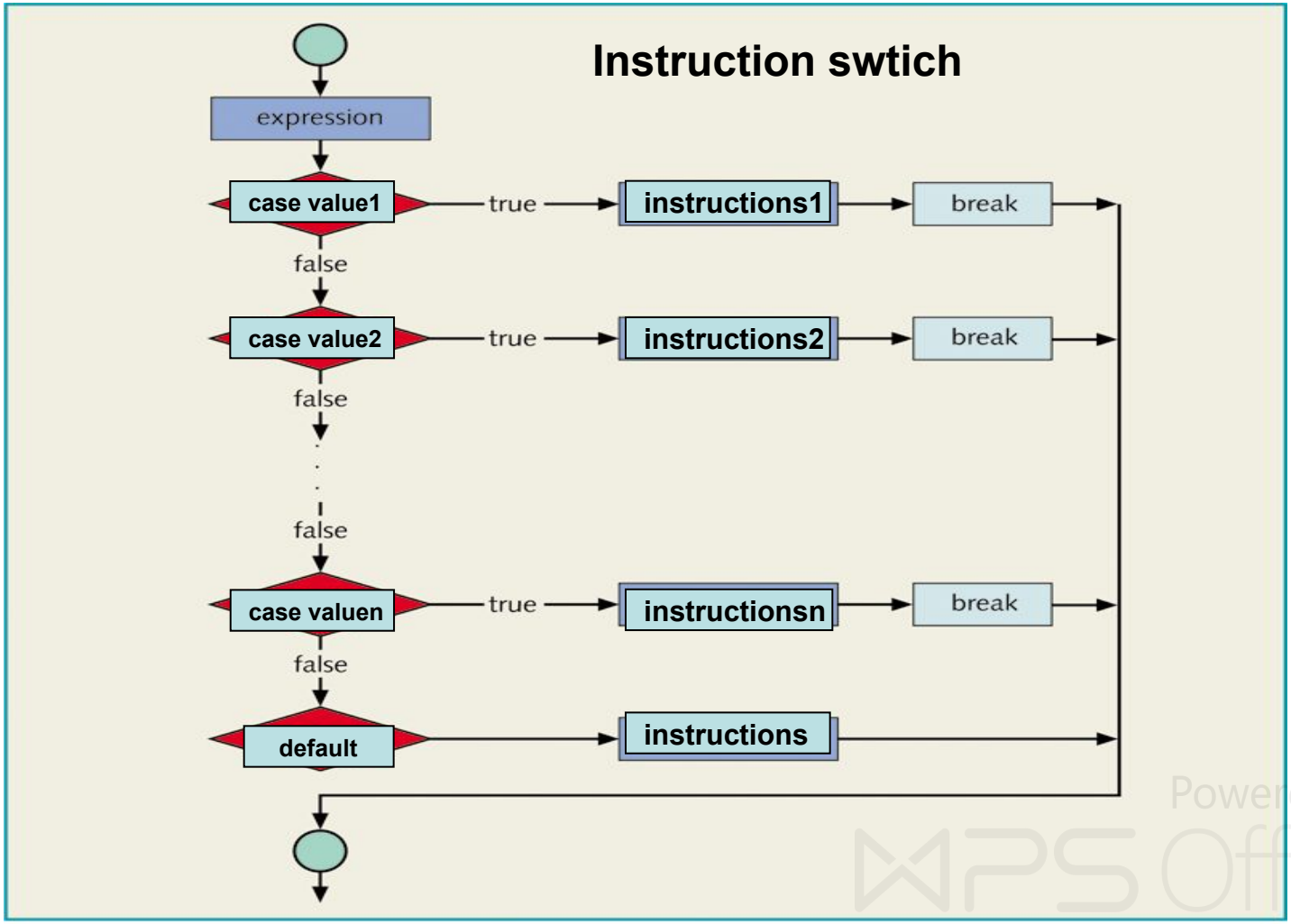


Figure 4-7 `switch` statement

Instructions de contrôle en détails

Example

```
switch (mention)
{
case 'A': System.out.println(" Mention Tres Bien.");
          break;
case 'B': System.out.println("Mention Bien.");
          break;
case 'C': System.out.println("Mention Assez Bien.");
          break;
case 'D': System.out.println("Mention Moyen.");
          break;
case 'E': System.out.println("Faible.");
          break;
default:  System.out.println("Pas de mention valide.");
}
```

Instructions de contrôle en détails

L'instruction swtich

```
switch (expression)
{
case valueur1: instructions1
               break;
case valeur2: instructions2
               break;
   ...
case valeurn: instructionsn
               break;
default: instructions
}
```

- `expression` est également connu sous le nom de sélecteur.
- `expression` peut être un identificateur.
- `valeur` est de type:

byte, short, char, et int

Instructions de contrôle en détails

- Instructions répétitives
  - Tant que (while, do-while)
    - **While** (expression booléenne) {
      Instructions
      }
    - **do**{
      Instructions
      }**while**(expression booléenne);

Instructions de contrôle en détails

- Instructions répétitives
  - Tant que (while)
    - **While** (expression booléenne) {
      Instructions
      }

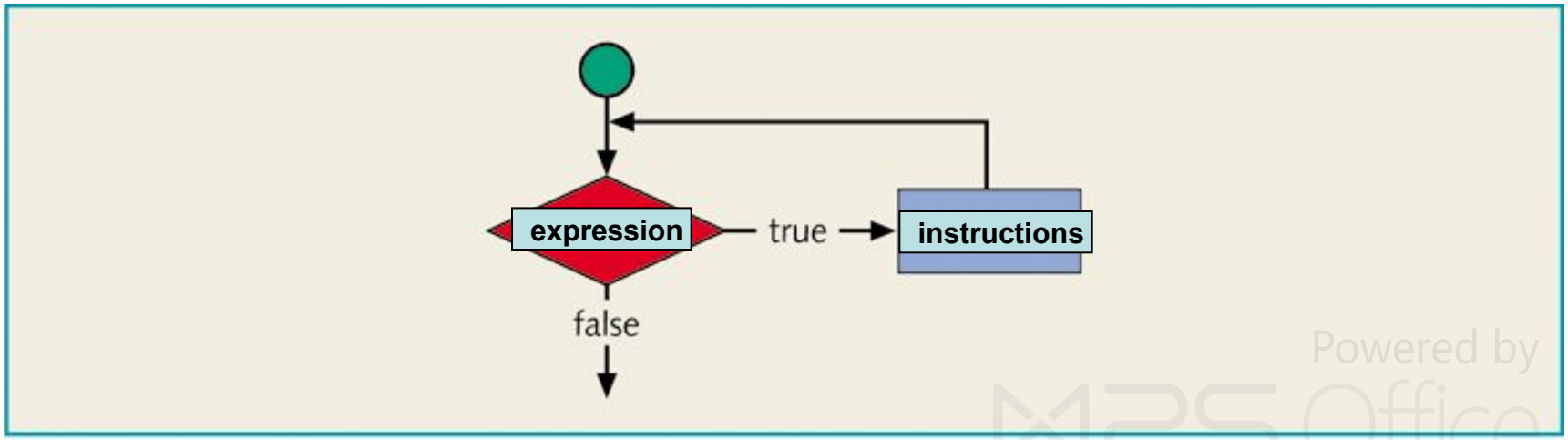


Figure 5-1 `while` loop

Instructions de contrôle en détails

- Instructions répétitives
  - Tant que (do-while)

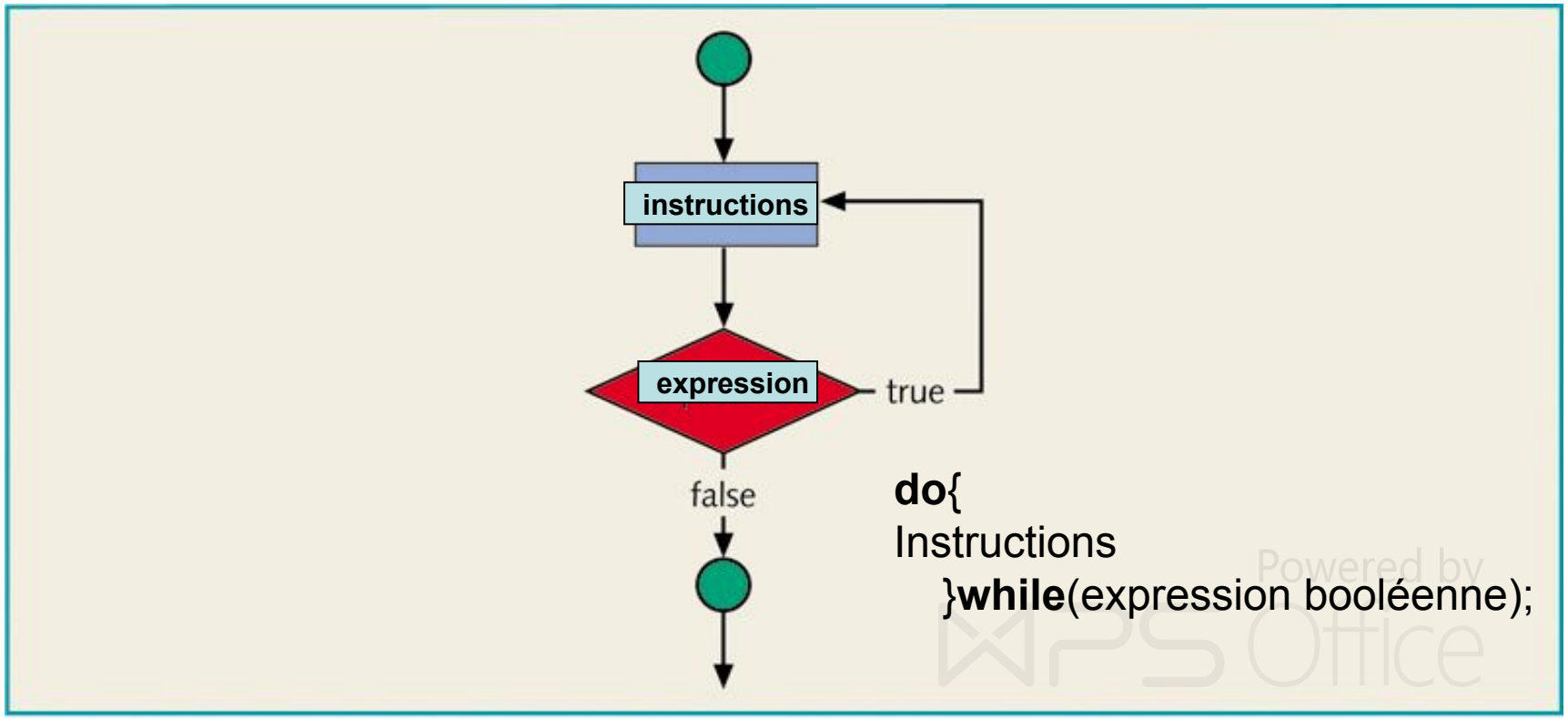


Figure 5-5 `do...while` loop

Instructions de contrôle en détails

Exemple while :

```
int i = 1;
while (i<=10) {
        //traitement
        i++;
}
```

Exemple do-while :

```
int i = 0;
do {
       i++;
       //traitement
}while(i<=10);
```

Instructions de contrôle en détails

- Imbrication
  - Toutes ces instructions de contrôle peuvent être imbriquées les unes à l'intérieur des autres.
    - Exemple :

```
if (condition booléenne){
    while(une condition booléenne){
        if(autre condition booléenne){
            traitement;
        else{
                autreTraitement;
        }
    }
}
```

Entrées/sorties

- **Affichage écran**
  - System.out.print()
  - System.out.println()
  - System.out.printf() //fonctionne comme en C
  - + //concatène les valeurs à afficher

  Exemple :

  System.out.print(" mon age est : "+21);

Entrées/sorties

- **Lecture clavier (interaction via la console)**

  static java.util.Scanner clavier = new java.util.Scanner(System.in)
  //à l'intérieur de la classe, avant le main()

  – **Dans le programme principal**
    - clavier.nextInt() //lit et retourne un entier en provenance du clavier
    - clavier.nextDouble() //lit et retourne un double
    - clavier.next() //lit et retourne le prochain mot (String)
    - clavier.nextLine() //lit et retourne toute la ligne (String)
    - etc.

***Pour l'instant, on ne se préoccupera pas des incompatibilités de type

Programme principal

- Il doit y avoir au minimum une classe dans un projet et une fonction principale qui s'appelle **main**
- La déclaration du clavier se fait avant la fonction main()
- L'utilisation se fait dans la fonction

Exemple :

```
public class exempleProgrammePrincipal {

        static java.util.Scanner clavier = new  java.util.Scanner(System.in);

        public static void  main(String[] args) {
                //code ici
                int x = clavier.nextInt();   //lit un entier au clavier
                                             //et le met dans x
        }
}
```

Commentaires

- Commentaire de ligne //
  - Tout ce qui se trouve après jusqu'à la fin de ligne
- Commentaire en bloc /* */
  - Tout ce qui se trouve entre /* et */
- Commentaire Javadoc /** */

Commentaires

- À quoi servent les commentaires
  - À éviter de lire le code pour savoir ce qu'il fait.
- À qui devraient servir le plus les commentaires
  - À vous
  - À n'importe quel autre lecteur
- Où mettre les commentaires ?
  - En-tête de programme
  - Variables et constantes
  - Bloc de code des instructions de contrôle
  - En-tête de sous-programme
  - Partout où cela clarifie le code

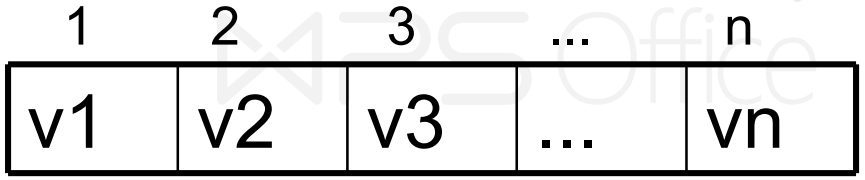
Principes de programmation (suite)

- **Rappel**
  - Tableaux

- **Langage Java**
  - Boucle for
  - Références
  - Définition et utilisation des tableaux

Tableaux

- C’est un contenant de données du même type
- On peut le considérer lui-même comme un type
  - Catégorie de données
    - Suite de variables du même type, consécutives en mémoire, accessibles par un indice (numéro)
  - Limite
    - Nombre de variables consécutives
  - Opérations
    - Accéder à une variable (lecture, écriture *(modification)*)
    - Toutes les autres opérations sont des fonctions du langage ou elles doivent être définies par le programmeur (comparer, copier, fouiller, ajouter, retirer, etc.)
  - Représentation graphique



LA BOUCLE FOR

- **For** (très utile pour les tableaux)
  - C'est comme un *while* optimisé pour les boucles dont on connaît le nombre de fois à itérer

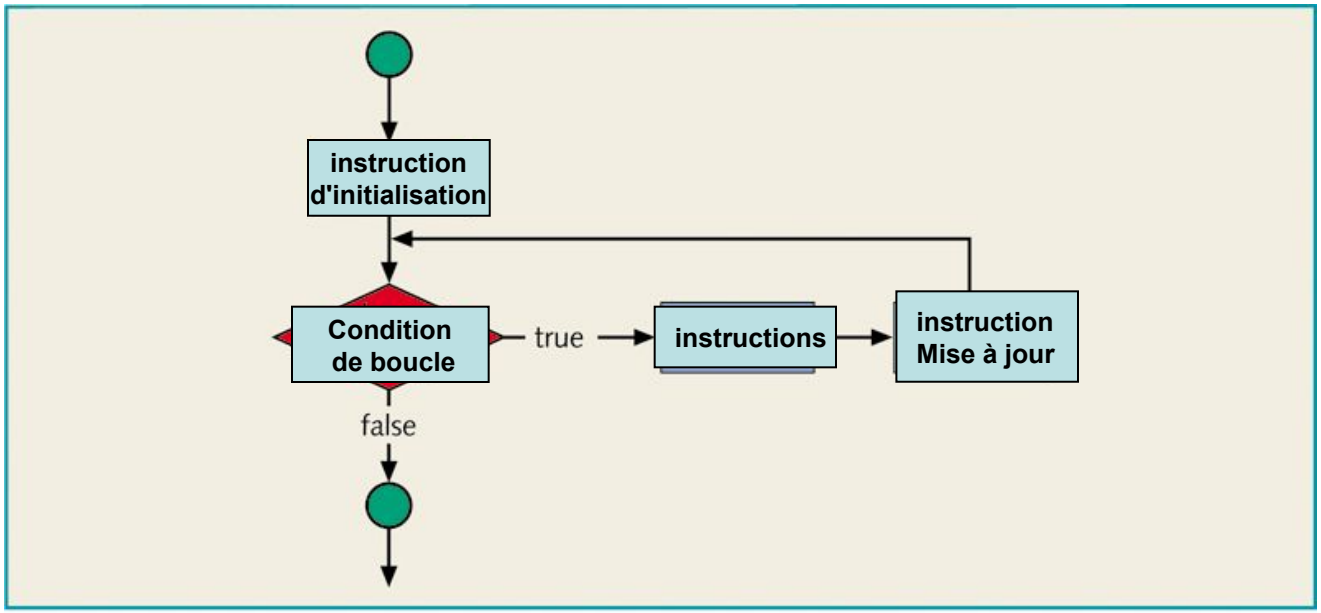
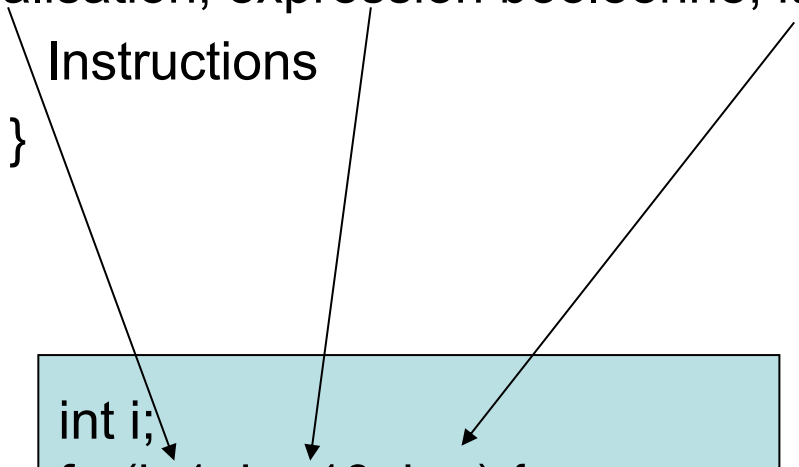


Figure 5-4 `for` loop

- **For** (très utile pour les tableaux)
  - C'est comme un *while* optimisé pour les boucles dont on connaît le nombre de fois à itérer
  - **for** (initialisation; expression booléenne; itération){
    Instructions
    }



Exemple for :

```
int i;
for(i=1; i<=10; i++) {
      //traitement
}
```

On peut aussi faire : for (**int i** = 1; i <= 10; i++)

Instructions `break`

- Utilisée pour la sortie prématurée (immédiate) d'une boucle.
- Utilisée pour sauter le reste des instructions dans une structure de Switch.
- Peut être placé à l'intérieur d'une instruction if d'une boucle.
  - Si la condition est remplie, on sort de la boucle immédiatement.

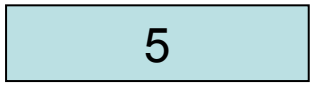
Instructions `continue`

- Utilisée dans les structures `while`, `for`, et `do`...`while`.
- Lorsqu'elle est exécuté dans une boucle, les instructions restantes dans la boucle sont ignorées; procède à la prochaine itération de la boucle.
- Lorsqu'elle est exécutée dans une structure `while`/`do`...`while` , l'éxpression conditionnelle de la boucle est évaluée immédiatement après l'instruction *continue*.
- Dans une structure `for`, l'instruction de mise à jour est exécutée après l'instruction continue; la condition de boucle est évaluée ensuite.

LES REFERENCES

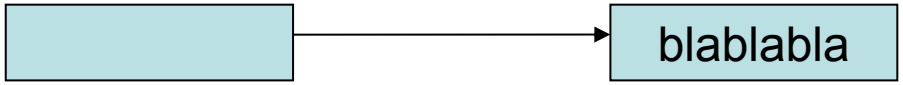
Références

- Les variables de type primitif ne sont pas des références.
  Exemple : int x = 5;



- Une référence est une variable dont le contenu fait référence à un emplacement mémoire différent, qui lui contient les données ***.

Exemple : Référence Donnée



- Toutes les variables d'un autre type que primitif sont des références

***Ressemble au pointeur du C sans les opérateurs '&' ou '*'

LES TABLEAUX EN JAVA

Tableaux

- **En Java**
  - Une variable-tableau est une référence
  - Le tableau doit être créé avant utilisation à l'aide de new ou des { }
  - Si on modifie le contenu d'un tableau dans une fonction, le paramètre effectif sera affecté.

Tableaux

- **En Java**
  - On définit les tableaux de la façon suivante :
    - Exemple : int [ ] tabInt = new int[20]; //définit un tableau de 20 entiers
    - Exemple : char [ ] tabCar = {‘a’,’l’,’l’,’o’}; //définit un tableau de 4 caractères
    - Forme générale :

      type[ ] ident_tableau = new ***type*** [nombre de cases]

      ou

      type[ ] ident_tableau ={ liste des valeurs séparées par une virgule};

Tableaux

- Syntaxe d'instantiation d'un tableeau (différentes façons de déclaration des tableaux):
  - `typeDonnée[ ] nomTableau;`
  - `nomTableau = new typeDonnée[intExp];`

  - `typeDonnée[ ] nomTableau =new typeDonnée[intExp];`
  - `typeDonnée[ ] nomTableau1, nomTableau2;`

- Syntaxe d'accés aux éléments d'un tableau:
  - `typeDonnée[indiceExp]`

    - `intExp` = nombre d'éléments dans un tableau `>= 0`
    - `0 <= indiceExp <= intExp`

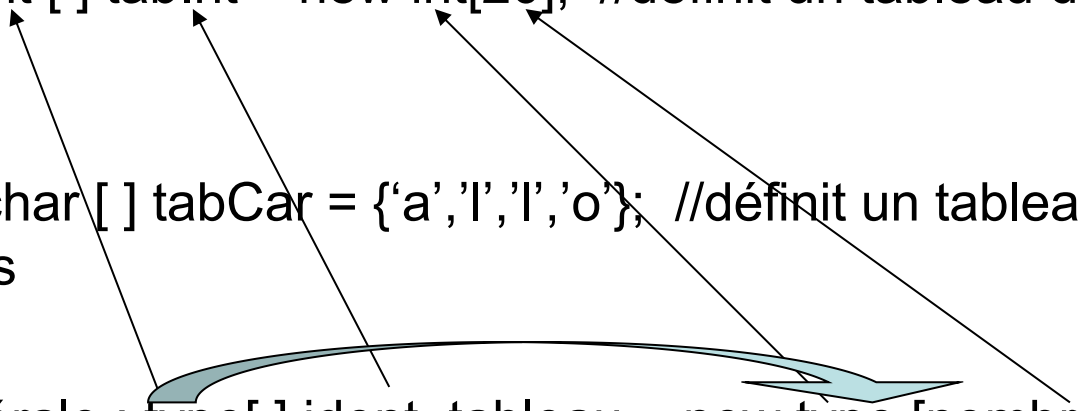
Initialisation des tableaux lors de la déclaration

```
double[] ventes = {12.25, 32.50, 16.90, 23, 45.68};
```

- Les valeurs, appelées valeurs initiales, sont placés entre accolades et séparés par des virgules.
- Ici, `ventes[0]= 12.25`, `ventes[1]= 32.50`, `ventes[2]= 16.90`, `ventes[3]= 23.00`, et `ventes[4]= 45.68`.
- Lors de la *déclaration-initialisation* des tableaux, la taille du tableau est déterminée par le nombre de valeurs initiales entre les accolades.
- Si un tableau est déclaré et initialisé simultanément, nous n'utilisons pas l'opérateur new pour instancier l'objet tableau.

Tableaux

- **En Java**
  - On définit les tableaux de la façon suivante :
    - Exemple : int [ ] tabInt = new int[20]; //définit un tableau de 20 entiers
    - Exemple : char [ ] tabCar = {‘a’,’l’,’l’,’o’}; //définit un tableau de 4 caractères
    - Forme générale : type[ ] ident_tableau = new type [nombre de cases] ou { liste des valeurs séparées par une virgule};



Tableaux

- **En Java**
  - Les indices commencent à 0
  - Toutes les cases sont initialisées avec une valeur nulle par défaut, selon le type (***0*** *pour les entiers*, ***0.0*** *pour les réels*, ***null*** *pour les réferences*, ***false*** *pour les booléens*,...)
  - On accède à une valeur à l'aide du nom de la variable tableau[indice]
    - Exemple : tabInt[3] fait référence à la 4ième case du tableau

Tableaux

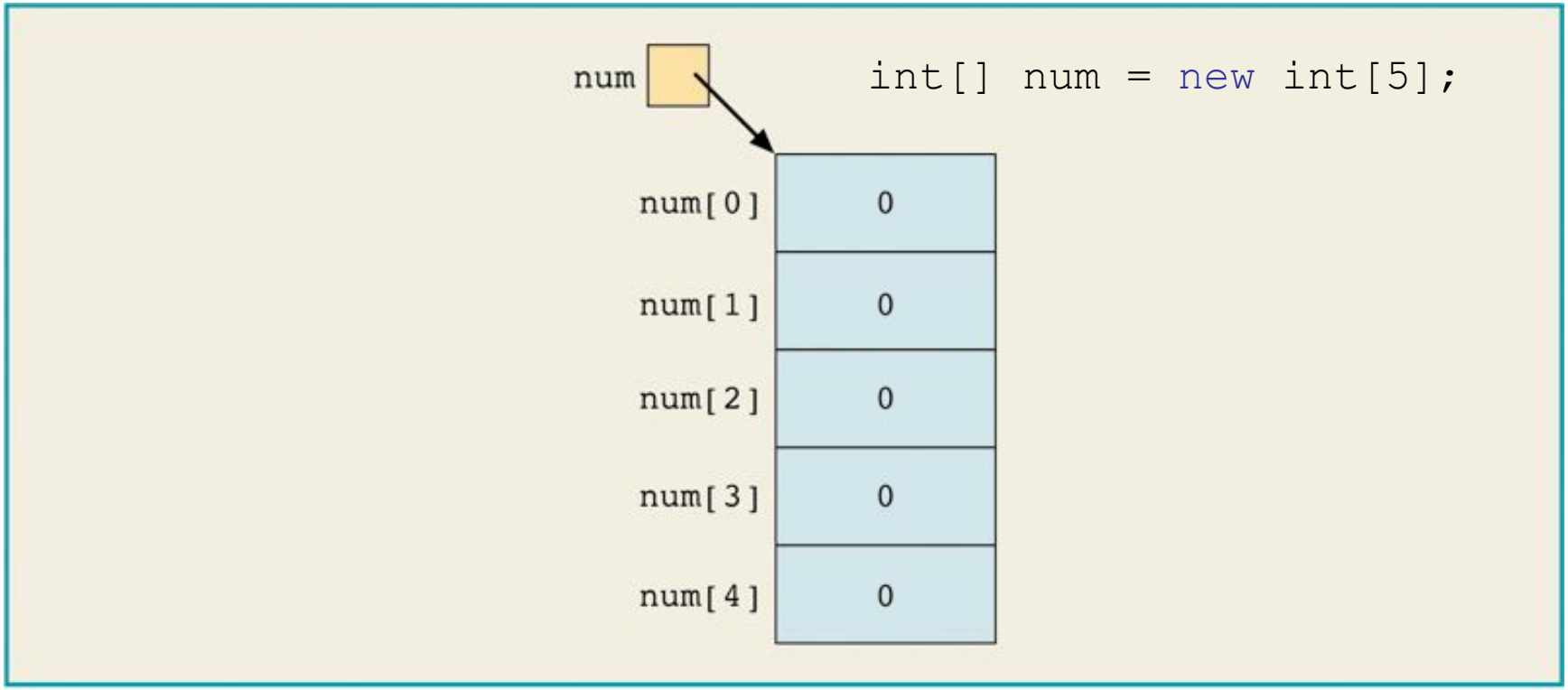


Figure 9-1 Array num

Tableaux

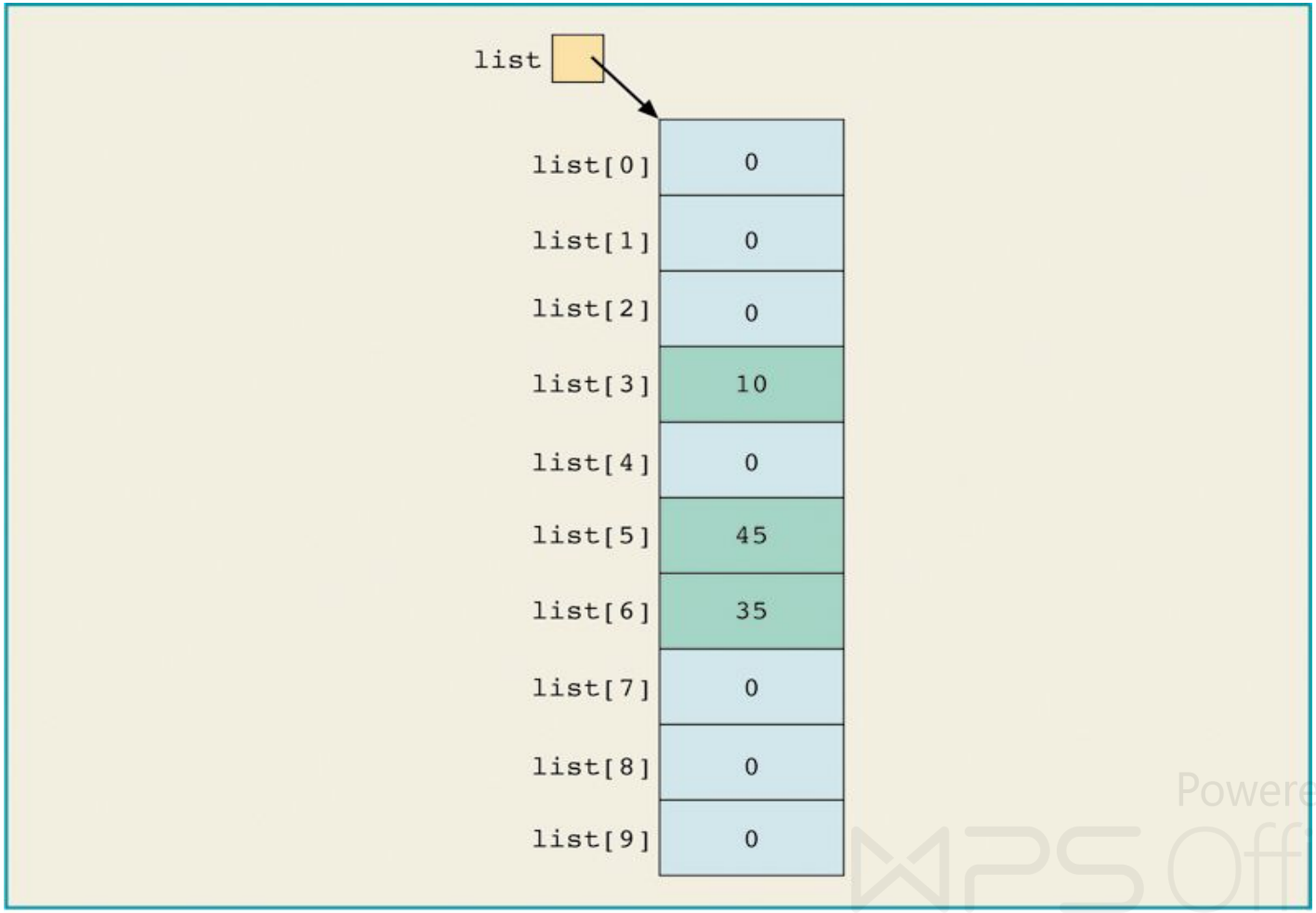


Figure 9-4 Array `list` after the statements `list[3]= 10;`, `list[6]= 35;`, and `list[5] = list[3] + list[6];` execute

Tableaux

- **En Java**
  - **On peut utiliser la variable (attribut) *length* pour obtenir le nombre de cases**

    - Une variable d'instance public *length* est associée à chaque tableau qui a été instancié.
    - La variable *length* contient la taille du tableau.
    - La variable *length* peut être directement accessible dans un programme utilisant le nom du tableau et l'opérateur point (.)

Tableaux

- **En Java**

  L'instruction suivante crée un tableau `list` de six éléménts et initialise les éléménts en utilisant les valeurs indiquées.

```
int[] list = {10, 20, 30, 40, 50, 60};
```

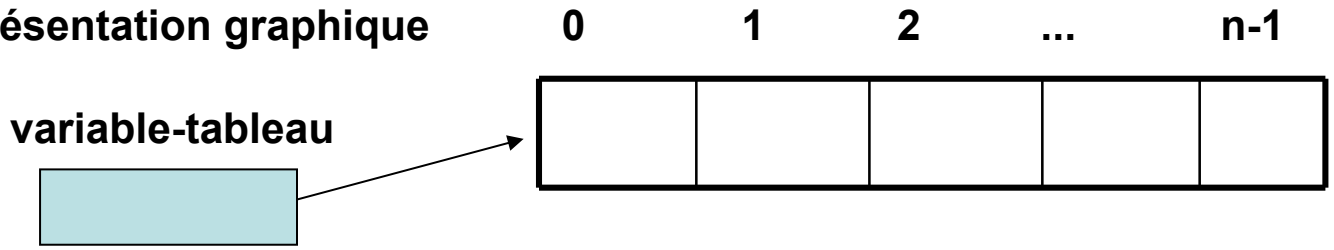
  - **Exemple: l'instruction *System.out.print(list.length);* affichera la valeur 6.**

  – **Les tableaux sont statiques (la taille est invariable)**

  – **L’accès à une case inexistante cause une erreur à l’exécution.**

Tableaux

- **Représentation graphique**



- **Boucle classique**

  for (var = 0; var < tableau.length; var++)

  Traitement de chaque case du tableau

  Exemple : char[] tabChar = new char[20];

```
for(int i = 0; i < tabChar.length; i++)
    System.out.println(tabChar[i]);
```

***Affichera 20 fois la valeur **null**

- **La boucle For EACH**

  (très utile pour les tableaux)

La boucle `foreach`

- La syntaxe pour utiliser cette boucle `for(foreach)` pour traiter les éléments d'un tableau est:

```
for (typeDonnée identificateur : nomTableau)
     instructions
```

- `identificateur` est une variable dont le type de données est le même que le type des éléments du tableau de données.

La boucle `foreach`

```
somme = 0;
for (double num : list)
    somme = somme + num;
```

- L'instruction `for` de la ligne 2 est lue comme suit:

pour chaque élment (identifié par `num`) dans le tableau `list`.

- L'identificateur `num` est initialisé à la valeur de `list[0]`. à la prochaine itération, la valeur de `num` est `list[1]`, et ainsi de suite.

```
for (double num : numList)
{
    if (max < num)
        max = num;
}
```

```
for (TYPE T : TableauTypeT)
    instructions Utilisant T;
```

Le type de `TableauTypeT est` TYPE[]
Le type de T `est` TYPE

Tableaux bi-dimensionels

- Les données se présentent parfois sous forme de tableau (difficile de représenter à l'aide d'un tableau à une dimension).
- Pour déclarer/instantier un tableau bidimensionnel:

```
typeDonnée[ ][ ] nomTableau = new typeDonnée[intExp1][intExp2];
```

- Pour accéder à un élément d'un tableau à deux dimensions:

- ***nomTableau*** [indexExp1] [indexExp2];
  - `intExp1, intExp2 >= 0`
  - ***`nomTableau.length`*** `= intExp1`
  - ***`nomTableau[i].length`*** `= intExp2` `// 0<i<=intExp1`
  - `indexExp1` = position de ligne
  - `indexExp2` = position de colonne

Tableaux bi-dimensionels

```
double[][]sales = new double[10][5];
```

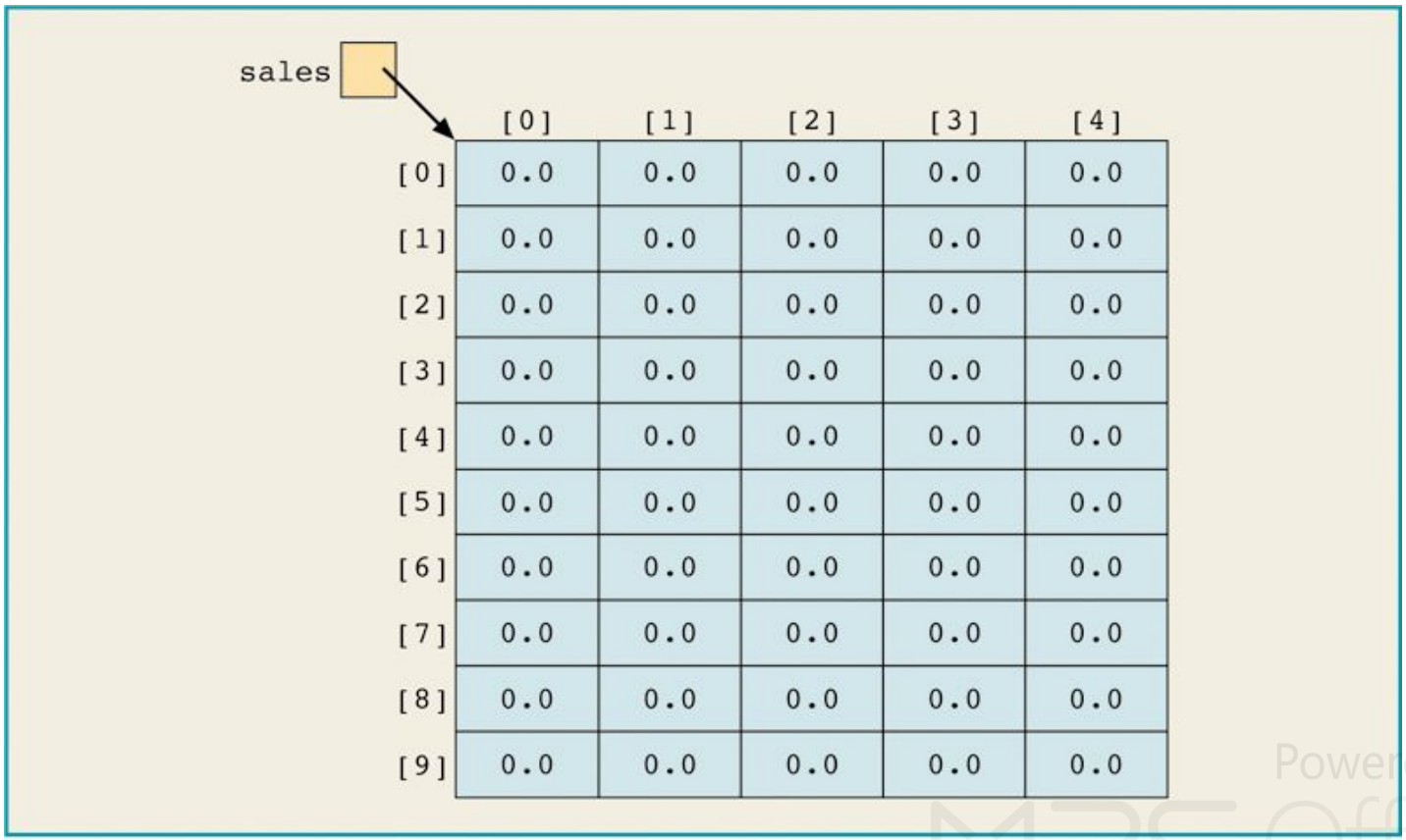


Figure 9-14 Two-dimensional array `sales`

Tableaux bi-dimensionels
Accés aux éléments

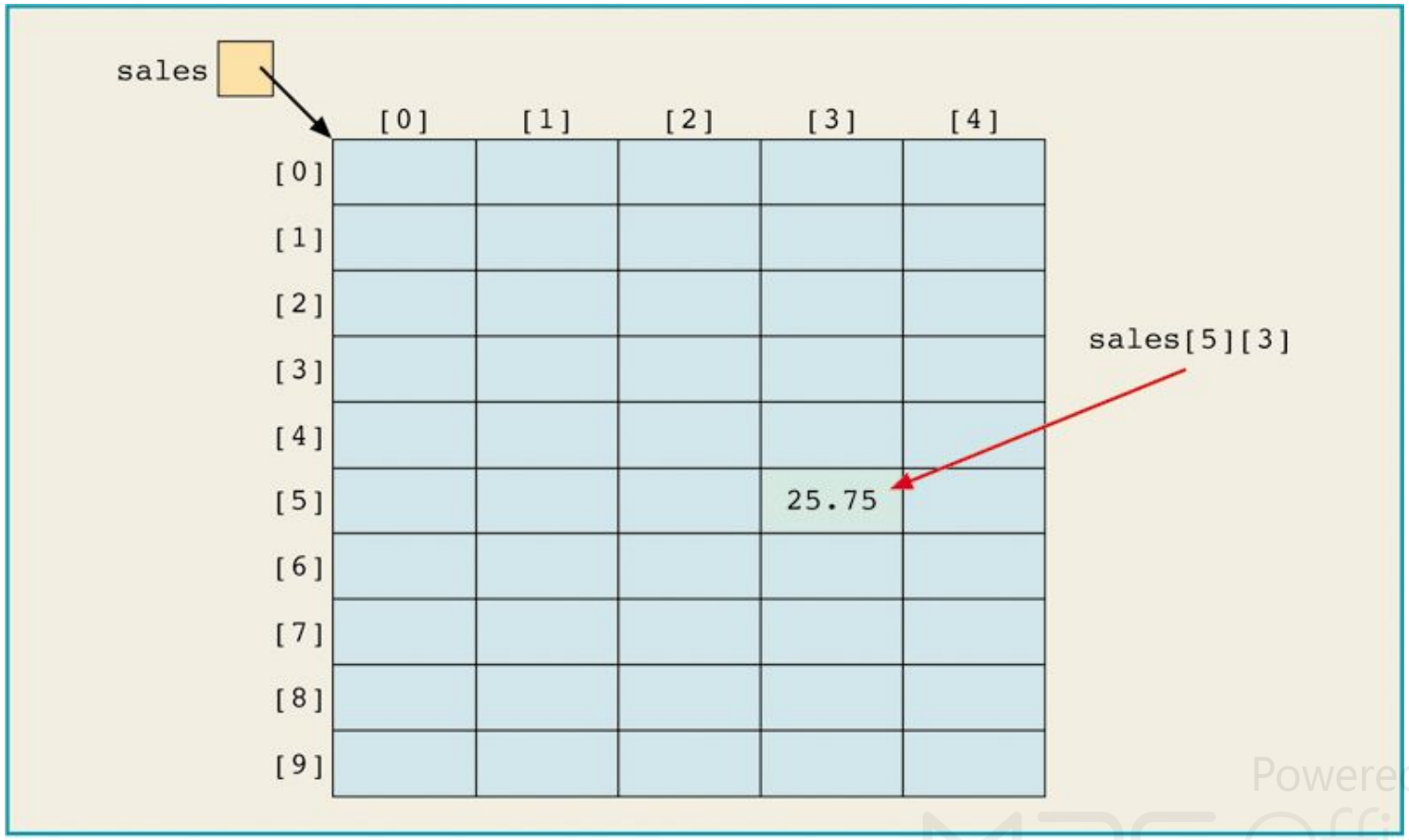


Figure 9-15 `sales[5][3]`

Tableaux bi-dimensionels: Traitements

Initialisation

```
for (ligne = 0; ligne < matrix.length; ligne++)
     for (col = 0; col < matrix[ligne].length;
                    col++)
          matrix[ligne][col] = 10;
```

Affichage

```
for (ligne = 0; ligne < matrix.length; ligne++)
{
     for (col = 0; col < matrix[ligne].length;
                    col++)
        System.out.printf(matrix[ligne][col]+"  ");
     System.out.println();
}
```

Tableaux bi-dimensionels: Traitements

Entrée

```
for (ligne = 0; ligne < matrix.length; ligne++)
     for (col = 0; col < matrix[ligne].length;
                    col++)
        matrix[ligne][col] = console.nextInt();
```

Somme par ligne

```
for (ligne = 0; ligne < matrix.length; ligne++)
{
     somme = 0;
     for (col = 0; col < matrix[ligne].length;
                    col++)
         somme  = somme  + matrix[ligne][col];
   System.out.println("Somme de ligne " + (ligne + 1)
                      + " = "+ somme);
}
```

Tableaux bi-dimensionels: Traitements

Somme par colonne

```
for (col = 0; col < matrix[0].length; col++)
{
      somme = 0;
    for (ligne = 0; ligne < matrix.length; ligne++)
            somme = somme + matrix[ligne][col];
      System.out.println("somme of colonne " + (col + 1)
                        + " = " + somme);
}
```

Tableaux bi-dimensionels: Traitements

Plus Grand Element de chaque ligne

```
for (ligne = 0; ligne < matrix.length; row++)
{
    plusgrand = matrix[ligne][0];
    for (col = 1; col < matrix[ligne].length;
                    col++)
        if (plusgrand < matrix[ligne][col])
            plusgrand = matrix[ligne][col];
 System.out.println("Le plus grand element de la ligne "
                   + (ligne + 1) + " = " + plusgrand);
}
```

Tableaux bi-dimensionels: Traitements

Plus Grand Element de chaque colonne

```
for (col = 0; col < matrix[0].length; col++)
{
    plusgrand = matrix[0][col];
    for (ligne = 1; row < matrix.length; row++)
        if (plusgrand < matrix[ligne][col])
            plusgrand = matrix[ligne][col];
System.out.println("Le plus grand element de la colonne "
                   + (col + 1) + " = " + plusgrand);
}
```

Tableaux Multidimensionels

- On peut définir des tableaux tri-dimensionels ou n-dimensionel (n peut etre n'importe quel nombre).
- Syntaxe pour déclarer et instantier un tableau:

```
typeDonnée[][]…[] nomTableau = new
        typeDonnée[intExp1][intExp2]…[intExpn];
```

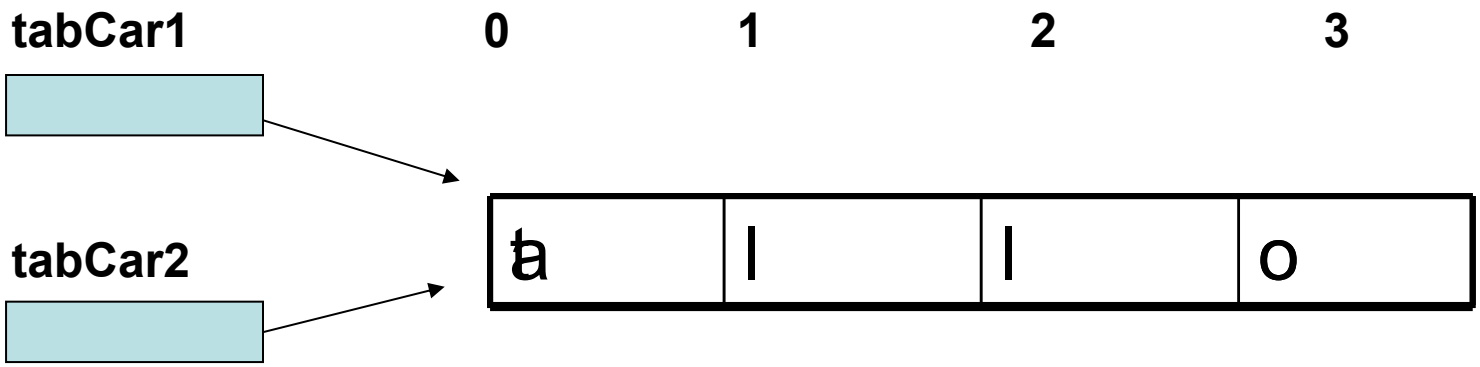
- Syntaxe pour accéder à un élémént:

```
typeDonnée[indexExp1][indexExp2]...[indexExpn]
```

  - `intExp1`, `intExp2`, ..., `intExpn` = entiers positifs
  - `indexExp1`, `indexExp2`, ..., `indexExpn` = entiers non-négatifs

Piège sur les références-tableau

- **Piège classique**
  char[ ] tabCar1 = {‘a’,’l’,’l’,’o’};



- **char[ ] tabCar2 = tabCar1;**
- **tabCar2[0] = ‘t’;**
- Si on veux travailler sur une copie il faut faire
  - **char [ ] tabCar2 = tabCar1.clone();**

Principes de programmation (suite)

- **Survol**
  - Sous-programmes
    - Aspects
    - Catégories
    - Paramètres formels et effectifs
    - Passage de paramètres
    - Mécanique d'appel
- **Langage Java**
  - Bloc de code
  - Définition formelle (en-tête)
    - Procédure
    - Fonction
  - Portée des variables

Sous-programmes

- **Trois aspects**
  - **Définition formelle** : décrit le nom, le type de la valeur de retour (s'il y a lieu) et la liste des informations (et leur type) nécessaires à son exécution.

    Exemple : double **sqrt**(double x)

  - **Appel effectif** : démarre l'exécution d'un sous-programme en invoquant son nom, en lui passant les valeurs demandées (du bon type) et en récupérant la valeur de retour (s'il y a lieu).

    Exemple : x = **sqrt**(y);

  - **Implémentation** : code qui sera exécuté par le sous-programme lors de l'appel.

Sous-programmes

- **Deux catégories**
  - **Fonction**
    - Un sous-programme qui retourne une valeur

      Exemple : sqrt(), cos(), sin(), power(), clavier.nextInt()

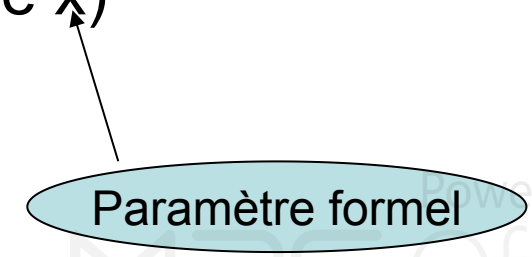
  - **Procédure**
    - Un sous-programme qui ne retourne rien (*void*)

      Exemple : System.out.println()

Sous-programmes

- **Paramètres formels**
  - Description des informations nécessaires et de leur type dans la définition formelle.
  - Ce sont des variables ou des constantes qui seront initialisées par les paramètres effectifs associés (par position) lors de l'appel.

    Exemple : double cos (double x)



Sous-programmes

- **Paramètres effectifs ou actuels (arguments)**
  - Valeur fournie à un sous-programme lors de l'appel.

    Exemple : x = cos(30);

  - Dans cet exemple, 30 est affecté à x de la fonction cos() lors de l'appel

Principes de programmation (suite)

EN-TÊTES FORMELLES

Définition formelle (en-tête)

- Procédure
  - void <nom> (liste des paramètres formels séparés par des ‘,’)
  - Le nom d’une procédure est habituellement un verbe à l’infinitif suivi d’un mot.

Exemple :

void afficherDate(int annee, int mois, int jour)

Définition formelle (en-tête)

- Fonction
  - <type de retour> <nom> (liste des paramètres formels séparés par des ‘,’)
  - Le nom d’une fonction désigne habituellement la valeur de retour.

Exemple :

double cos(double x)

int nbrJourMaxParMois(int annee, int mois)

Les références en paramètres

- Un paramètre formel reçoit une copie de la référence à une donnée et non la donnée.
- On peut modifier les données directement via le paramètre formel.
- On peut avoir plusieurs références sur une même donnée.

Variables de Types de données Primitives en Paramèters

- Un paramètre formel reçoit une copie de son paramètre effectif lui correspondant.

- Si un paramètre formel est une variable de type de données primitives:

  - Valeur du paramètre effectif est directement stocké.

  - Vous ne pouvez pas passer des informations en dehors de la méthode.

  - Fournit seulement un lien à sens unique entre les paramètres effectifs et les paramètres formels.

Les références en paramètres

Si un paramètre formel est une variable de référence:

- Copie la valeur du paramètre effectif lui correspondant.
- Valeur du paramètre effectif est l'adresse de l'objet où les données réelles sont stockées.
- Les deux paramètres effectifs et formels référencent un même objet.

Utilistaion des Variables références en paramètres

- Peut retourner plus d'une valeur à partir d'une méthode.
- Peut changer la valeur de l'objet réel.
- Lors du passage d'une adresse, économise de l'espace mémoire et du temps (par rapport à la copie de grande quantité de données).

Les réferences en paramètres
Exemple sur les tableaux

Exemple non fonctionnel :

```
//fonction  qui incrémente la valeur reçue
void incrementerValeur(int valeur) {
        valeur++;
}


//quelque part ailleurs dans le code
for( int i = 0; i<tab.length; i++)
        incrementerValeur(tab[i]);
```

- Aucune valeur ne sera modifiée dans le tableau puisque les types primitifs sont passés par copie.

Les réferences en paramètres
Exemple sur les tableaux

Exemple non fonctionnel :

```
//fonction  qui incrémente la valeur reçue
void incrementerValeur(int valeur) {
        valeur++;
}


//quelque part ailleurs dans le code
for( int i = 0; i<tab.length; i++)
        incrementerValeur(tab[i]);
```

- Aucune valeur ne sera modifiée dans le tableau puisque les types primitifs sont passés par copie.

Les réferences en paramètres
Exemple sur les tableaux

Exemple fonctionnel :

//fonction qui incrémente les valeurs d'un tableau

```
void incrementerValeur(int[] tab) {
        for( int i = 0; i<tab.length; i++)
              tab[i]++;
}
```

```
//quelque part ailleurs dans le code
incrementerValeur(tab);
```

- Le tableau est passé par référence donc son contenu sera modifié par la fonction.

Tableaux et liste des paramétres à longueur variable

- La syntaxe pour déclarer une liste de paramètres formels à longueur variable est:

```
typeDonnee ... identificateur
```

Tableaux et liste des paramétres à longueur variable

```
public static double plusGrand(double ... numList)
{
    double max;
    int index;
    if (numList.length != 0)
    {
        max = list[0];
        for (index = 1; index < numList.length;
                        index++)
        {
            if (max < numList [index])
                max = numList [index];
        }
        return max;
    }
    return 0.0;
}
```

Tableaux et liste des paramétres à longueur variable

```
double num1 = plusGrand(34, 56);
double num2 = plusGrand(12.56, 84, 92);
double num3 = plusGrand(98.32, 77, 64.67, 56);
System.out.println(plusGrand(22.50, 67.78,
                             92.58, 45, 34, 56));
double[] listNombre = {18. 50, 44, 56.23, 17.89
                       92.34, 112.0, 77, 11, 22,
                       86.62);

System.out.println(plusGrand(listNombre));
```

Surcharge de Méthodes

- La surcharge de méthodes: Plusieus méthodes peuvent avoir le même nom.

- Deux méthodes ont des listes de paramètres formels différentes :

  – Si les deux méthodes ont un nombre différent de paramètres formels.

  – Si le nombre de paramètres formels est le même dans les deux méthodes, le type de données des paramètres formels dans l'ordre que vous indiquez doivent différer dans, au moins, une position.

Surcharge de Méthodes

```
public void méthodeUn(int x)
public void méthodeDeux(int x, double y)
public void méthodeTrois(double y, int x)
public int méthodeQuatre(char ch, int x,
                         double y)
public int méthodeCinq(char ch, int x,
                       String nom)
```

Toutes ces méthodes ont des listes de paramétres formels différentes.

Surcharge de Méthodes

```
public void méthodeSix(int x, double y,
                       char ch)
public void méthodeSept(int one, double u,
                        char firstCh)
```

- Les méthodes *méthodeSix* et *méthodeSept* ont trois paramètres formels chacune, et le type de données des paramètres correspondants est le même.

- Ces méthodes ont toutes les mêmes listes de paramètres formels.

Surcharge de Méthodes

- La surcharge de méthode: Création dans une classe de plusieurs méthodes avec le même nom.

- La signature d'une méthode consiste en le nom de la méthode et sa liste de paramètres formels. Deux méthodes ont des signatures différentes si elles ont soit des noms différents ou des listes de paramètres formels différentes. (Notez que la signature d'une méthode ne comprend pas le type de retour de la méthode.)

Surcharge de Méthodes

- Les en-tetes des méthodes suivantes surchargent la méthode `methodXYZ` correctement:

```
public void methodXYZ()
public void methodXYZ(int x, double y)
public void methodXYZ(double one, int y)
public void methodXYZ(int x, double y,
                      char ch)
```

Surcharge de Méthodes

```
public void methodABC(int x, double y)
public int methodABC(int x, double y)
```

- Ces deux méthodes ont le même nom et la même liste de paramètres formels.
- Les declarations de ces deux méthodes pour surcharger la méthode methodABC sont incorrectes.
- Dans ce cas, le compilateur génère une erreur de syntaxe. (Notez que les types de retour dans les déclarations de ces deux méthodes sont différentes.)

Exercices sur les tableaux

Écrivez une fonction nbOccurence qui reçoit un tableau d'entiers et une valeur et qui retourne le nombre de fois où la valeur se trouve dans le tableau.

```
int nbOccurence (int[] tab, int valeur){

    int nb = 0;

    for(int i = 0; i < tab.length; i++){
            if(tab[i] == valeur)
               nb = nb + 1;            //on peut faire aussi nb++;

    }

   return nb;
}
```

Principes de programmation (suite)

BLOCS DE CODE
(portée et visibilité)

Sous-programmes

- Bloc de code
  - délimité par des accolades
  - contient des instructions
  - peut contenir des déclaration de variables
  - peut contenir d'autres blocs de code

Exemple :

```
{   int i;
    i = 5;
    while( i < 10) {
            System.out.println(i);
    }
}
```

Portée des variables

- La portée d'une variable est la partie du programme où une variable peut être utilisée après sa déclaration.
- Une variable définie dans un bloc est dite locale à ce bloc
- Une variable ne vit que dans le bloc où elle est définie et dans ses sous blocs

Exemple :

```
public class ExemplePortee {

        int i;
        void testPortee()
        {
                i=0;   //legal
        }
}
```

Portée des variables

- Deux variables peuvent avoir le même nom dans deux blocs différents

Exemple :

```
public class ExemplePortee {

    int i;  //local à la classe

    {   int i;  //legal
        i=0;  // le i local à ce bloc
        this.i = 0;   //le i de la classe (on y reviendra)
    }
    //le deuxième i n'existe plus ici
}
```

Portée des variables

- Une variable existe du début de sa déclaration jusqu'à la fin du bloc dans lequel elle a été définie
- Le compilateur prend toujours la variable dont la définition est la plus proche.
- À la fin d'un bloc les variables qui y ont été définies n'existent plus

Portée d'un identificateur à l'intérieur d'une Classe

- Identificateur local: Un identificateur qui est déclaré dans une méthode ou dans bloc et qui est visible uniquement dans cette méthode ou d'un bloc.

- Java ne permet pas l'imbrication des méthodes. Autrement dit, vous ne pouvez pas inclure la définition d'une méthode dans le corps d'une autre méthode.

- Dans une méthode ou un bloc, un identificateur doit être déclaré avant de pouvoir être utilisé. Notez qu'un bloc est un ensemble d'instructions entre accolades.

- La définition d'une méthode peut contenir plusieurs blocs. Le corps d'une boucle ou une instruction `if` constitue également un bloc.

- Dans une classe, en dehors de de la définition de toute méthode (et bloc), un identificateur peut être déclaré partout.

Portée d'un identificateur à l'intérieur d'une Classe

- Dans une méthode, un identificateur qui est utilisé pour nommer une variable dans le bloc externe de la méthode ne peut pas être utilisé pour nommer toute autre variable dans un bloc interne de la méthode. Par exemple, dans la définition de méthode suivante, la seconde déclaration de la variable x est illégalle:

```
public static void declarationIdentificateurIllegale()
{
    int x;

    //bloc
    {
        double x;     //déclaration illégale,
                      //x est déjà declaré
        ...
    }
}
```

Régles de portée

- Règles de portée d'un identificateur qui est déclaré dans une classe et accédé dans une méthode (bloc) de la classe.

- Un identificateur, disons X, qui est déclaré dans une méthode (bloc) est accessible:

  – Seulement dans le bloc à partir du point où il est déclaré jusqu'à la fin du bloc.

  – Par ces blocs qui sont imbriqués dans ce bloc.

- Supposons X est un identificateur qui est déclaré dans une classe et à l'extérieur de la définition de chaque méthode (bloc).

  – Si X est déclaré sans le mot réservé `static`, alors il ne peut pas être accessible dans une méthode `static`.

  – Si X est déclarée avec le mot réservé `static`, alors il peut être consulté dans une méthode (bloc) à condition que la méthode (bloc) n'a aucun autre identificateur nommé X.

Régles de portée

Example 7-12

```
public class régleDePortée
{
    static final double rate = 10.50;
    static int z;
    static double t;
    public static void main(String[] args)
    {
        int num;
        double x, z;
        char ch;
        //...
    }
    public static void one(int x, char y)
    {
        //...
    }
```

Régles de portée

```
    public static int w;
    public static void two(int one, int z)
    {
        char ch;
        int a;

        //block three
        {
            int x = 12;
            //...
        } //end block three here
      //...
    }
}
```

Régles de portée

Table 7-3 Scope (Visibility) of the Identifiers

Identifier	Visibility in one	Visibility in two	Visibility in block three	Visibility in main
rate (before main)	Y	Y	Y	Y
z (before main)	Y	N	N	N
t (before main)	Y	Y	Y	Y
main	Y	Y	Y	Y
local variables of main	N	N	N	Y
one (method name)	Y	Y	Y	Y
x (one's formal parameter)	Y	N	N	N
y (one's formal parameter)	Y	N	N	N
w (before method two)	Y	Y	Y	Y
two (method name)	Y	Y	Y	Y
one (two's formal parameter)	N	Y	Y	N
z (two's formal parameter)	N	Y	Y	N
local variables of two	N	Y	Y	N
x (block three's local variable)	N	N	Y	N

Programmation oriontée objet

Eléments du langage Java

support de présentation:

https://cours.etsmtl.ca/seg/FCardinal/

http://www.bhecker.com/itu/

Programmation oriontée objet

Eléments du langage Java

Université Mohammed V
FACULTE DES SCIENCES
RABAT / FSR
Département informatique

Programmation orientée objet en JAVA

Pr. REDA Oussama Mohammed