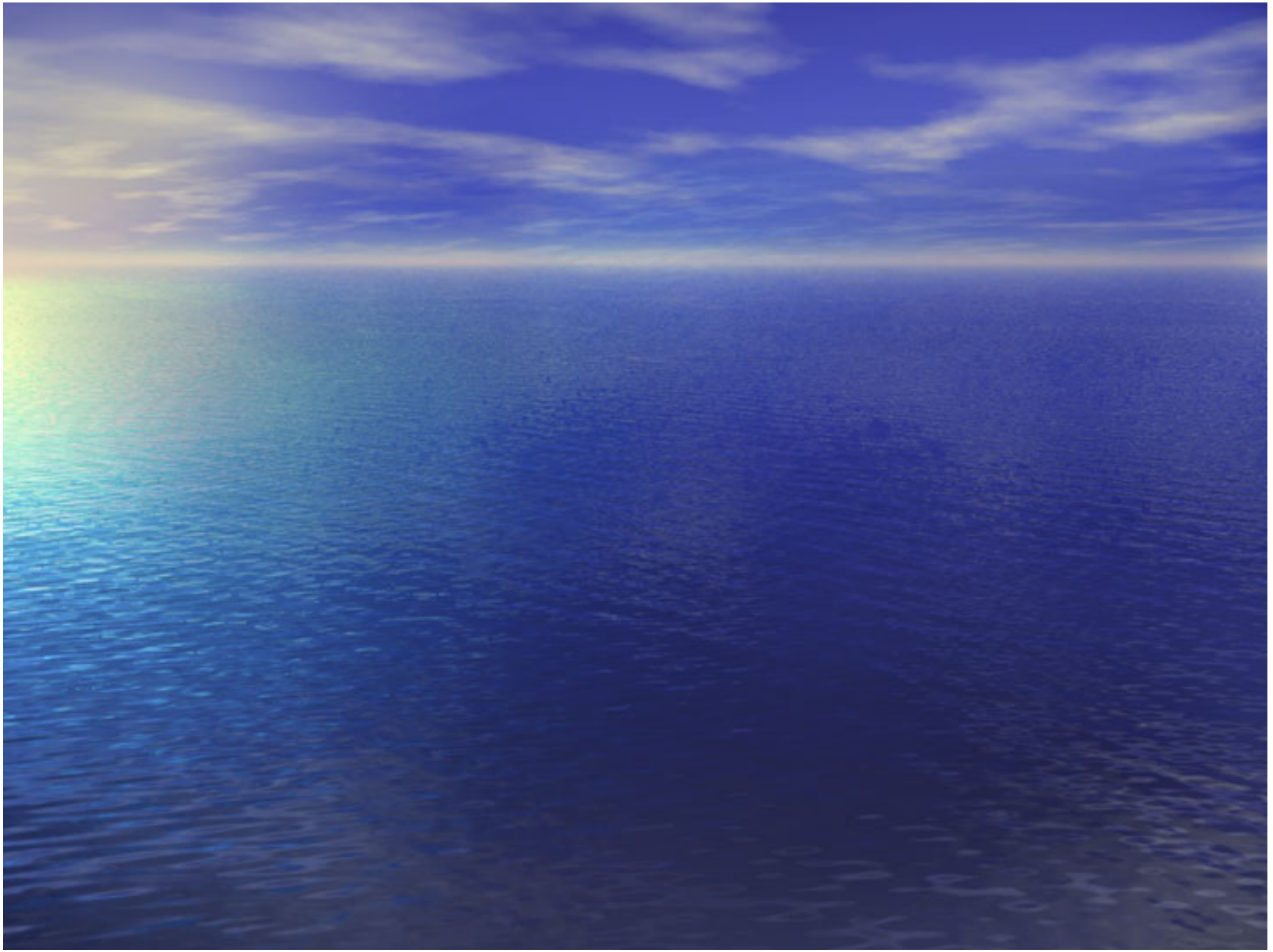


I- Les chaines de caractères (String)

Types Fondamentaux

String

- Chaîne de caractères entourée de **""**

```
String nom;  // déclaration
nom = "Jacques";  // affectation
String unAutre = "Nicolas";  //décl. & affect.
unAutre = "";  // String vide
```

  - Méthode **length()** donne le nb. de caractères

```
System.out.println(nom.length());
-> 7
System.out.println(unAutre.length());
-> 0
```

Types Fondamentaux

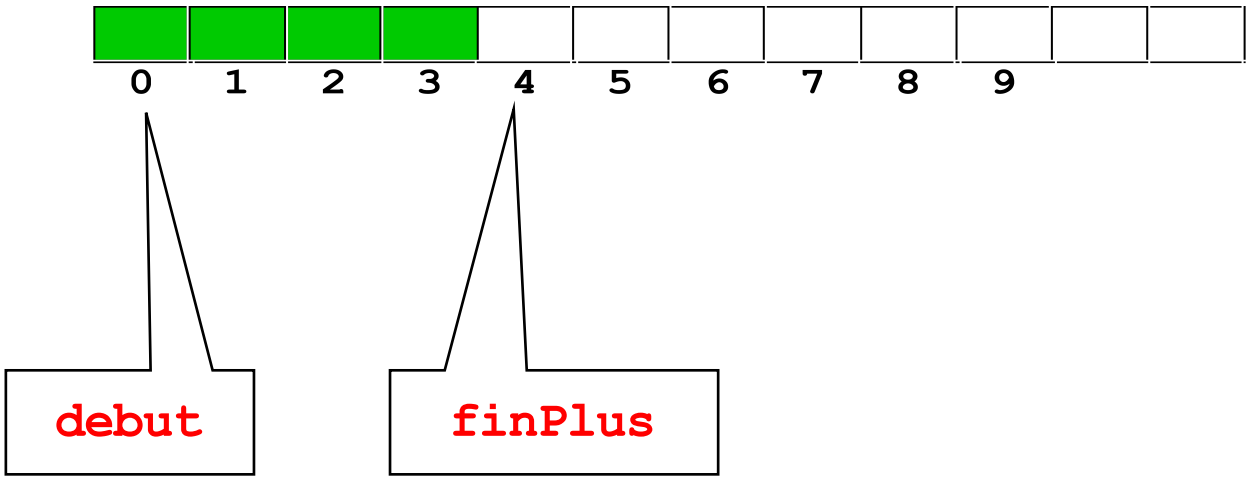
substring()

- **substring(debut, finPlus)** extrait un morceau

```
String salut = "Hello, life!";
String antiSalut = salut.substring(0, 4);
System.out.println("Life is " + antiSalut);
-> Life is Hell
```

- **debut** est le premier caractère à prendre
- **finPlus** est le premier caractère à ne pas prendre en compte

Types Fondamentaux
substring()



- Premier élément du **String** est l'élément 0
- **length()** est dernier élément - 1
  - **substring.length() == finPlus - debut**

Types Fondamentaux
String (2)

- Opérateurs

`s.length()`	Longueur de **s**
`s.substring(i, j)`	Sous-chaÓne de**s** de pos. **i ß j-1**
`s.toUpperCase()`	**s** toute en majuscules
`s.toLowerCase()`	**s** toute en minuscules
`"" + x`	Le nombre **x** en caractËres
`Numeric.parseDouble(s)`	**double** reprÈsentÈ par**s**
`Integer.parseInt(s)`	**int** reprÈsentÈ par**s**

Types Fondamentaux

Mise en Forme

- Caractères spéciaux

`\n`	ß la ligne
`\t`	tab

- Sont équivalents

```
System.out.println("Ah que coucou");
System.out.print("Ah que coucou\n");
```

Types Fondamentaux
String (3)

- L'opérateur «**+**» enchaîne deux **String**

```
String s1 = "Ah que ";
System.out.println(s1 + "coucou");
-> Ah que coucou
```

Comparaison des `String`

```
String nom = "Fred";
if (nom.equals("Frod")) {
   System.out.println("Alors, Frod…");
}
```

- C'est la méthode **`equals()`** appliquée à une variable de type **`String`**
- Caractère par caractère :

F r e d

F r o d

La méthode compareTo

La méthode *compareTo s'utilise ainsi :*

chaîne1.compareTo(chaîne2)

Elle fournit :

- un entier négatif si *chaîne1 arrive avant chaîne2,*
- un entier nul si *chaîne1 et chaîne2 sont égales (on a alors chaîne1.equals(chaîne2)),*
- • un entier positif si *chaîne1 arrive après chaîne2.*

La Classe StringBuffer

Un StringBuffer peut être *utilisé partout ou un String utilisé : Il est simplement plus flexible et on peut modifier son contenu.*

Cette classe contient 3 constructeurs et plus de 30 méthodes:

append

charAt

replace

delete

…


II- Tableaux

Structures de Données

- Une variable de type primitif
  - Stockage d'un élément d'un type donné
    - **`int`**, **`double`**, **`char`**, **`boolean`**, **`String`**…
- Un tableau
  - Stockage d'éléments tous du même type
    - **`int[]`**, **`double[]`**, **`char[]`**, **`boolean[]`**, **`String[]`**…
- Une collection d'objets
  - Stockage divers et varié
    - **`Vector`**, **`ArrayList`**, **`HashMap`**, **`LinkedList`**…

Stockage d'Éléments Similaires

- Peu pratique en variables…

```
String dept01 = "Ain";
String dept02 = "Aisne";
…
String dept95 = "Val d'Oise";
```

- Mieux en tableau…

```
String[] dept = {"Québec","Ain","Aisne",
                 …,"Val d'Oise"};
```

Tableaux

- *Array*
  - Stockage d'éléments tous du même type
  - Structure à part entière
  - Un tableau est un objet référencé
  - Assimilable à une classe

  - Création en trois étapes:
    1. déclaration
    2. allocation de mémoire
    3. initialisation des éléments

Tableaux
Déclaration

- Indiqué par `[]`
  - Deux possibilités

```
type[] nom;
type nom[];
```

```
int[] tableau1;
int tableau2[];
int[][] matrice;  // tableau bidimensionnel
int[] x, y[]; //équivalent à int x[],y[][];
int tab[10];  // ne compile pas
```

Tableaux
Allocation

- Alloué dynamiquement
  - à l'aide du mot clé **new**

```
int[] tableau1;  // déclaration
tableau1 = new int[10];  // allocation
int tableau2[]; // déclaration
tableau2 = new int[35]; // allocation
int[][] matrice; // déclaration
matrice = new int[2][4]; // allocation
int[] x, y[]; // déclaration
x = new int[5]; // allocation
y = new int[3][2]; // allocation
```

Tableaux

Déclaration et Allocation

- Peut combiner déclaration et allocation

```
int[] tableau1 = new int[10];
int tableau2[] = new int[35];
int[][] matrice = new int[2][4];
int[] x = new int[5];
int[] y[] = new int[3][2];
```

Tableaux
Initialisation (1)

- Chaque élément initialisé séparément

```
int[] tablo = new int[10];
for (int i = 0; i < 10; i++) {
    tablo[i] = i;
}
```

Tableaux
Initialisation (2)

- Valeurs initiales peuvent être énumérées

```
int[] joursParMois = {31, 28, 31,
  30, 31, 30, 31, 31, 30, 31, 30, 31};
```

  - Expédie déclaration, allocation, intialisation

Tableaux
De `String`

- Pareil que pour tableaux de primitifs

```
String[] jours = {"lundi", "mardi",
  "mercredi", "jeudi", "vendredi",
  "samedi", "dimanche"};
```

Les tableaux en Java

- Les tableaux sont considérés comme des **objets**
- Fournissent des collections ordonnées d'éléments
- Les éléments d'un tableau peuvent être :
  - Des variables d'un type primitif (int, boolean, double, char, …)
  - Des références sur des objets (à voir dans la partie Classes et Objets)
- Création d'un tableau
  1. Déclaration = déterminer le type du tableau
  2. Dimensionnement = déterminer la taille du tableau
  3. Initialisation = initialiser chaque case du tableau

Tableaux : Déclaration

① Déclaration

- La déclaration précise simplement le type des éléments du tableau

```
int[] monTableau;
```

monTableau | *null*

- Peut s'écrire également

```
int monTableau[];
```

Attention : une déclaration de tableau ne doit pas préciser de dimensions

```
int monTableau[5]; // Erreur
```

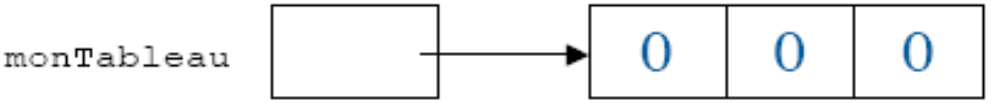
Tableaux : Dimensionnement

② Dimensionnement

- Le nombre d'éléments du tableau sera déterminé quand l'objet tableau sera effectivement créé en utilisant le mot clé **new**
- La taille déterminée à la création du tableau est fixe, elle ne pourra plus être modifiée par la suite
- Longueur d'un tableau : « monTableau.**length** »

```
int[] monTableau; // Déclaration
monTableau = new int[3]; // Dimensionnement
```

- La création d'un tableau par **new**
  - Alloue la mémoire en fonction du type de tableau et de la taille
  - Initialise le contenu du tableau à 0 pour les types simples



Tableaux : Initialisation

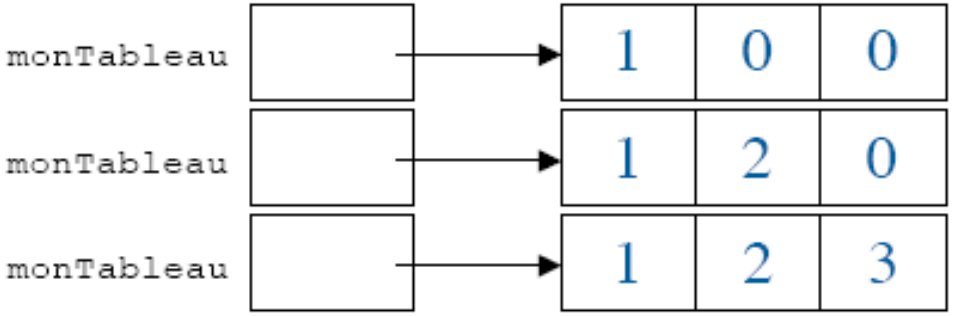
③ Initialisation

- comme en C/C++ les indices commencent à zéro
- l'accès à un élément d'un tableau s'effectue suivant cette forme

```
monTab[varInt]; // varInt >= 0 et <length
```

- Java vérifie automatiquement l'indice lors de l'accès (lève une exception)

```
monTab[0] = 1;
monTab[1] = 2;
monTab[2] = 3;
```



- Autre méthode : en donnant explicitement la liste de ses éléments entre {…}

```
int[] monTab = {1, 2, 3}
```

- est équivalent à

```
monTab = new int[3];
monTab[0] = 1; monTab[1] = 2; monTab[2] = 3;
```

Tableaux en Java : Synthèse

① Déclaration

```
int[] monTableau;
```

② Dimensionnement

```
monTableau = new int[3];
```

③ Initialisation

```
monTableau[0] = 1;
monTableau[1] = 2;
monTableau[2] = 3;
```

Ou ①② et ③

```
int[] monTab = {1, 2, 3};
```

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