Le langage Java
- les bases -

Types de données

Java	Français	Limites ou example
int	Nombres entiers	-2^{31} … $2^{31} - 1$
long	Nombres entiers	-2^{63} … $2^{63} - 1$
double	Nombres réels	3.14 -2.5 6.022e23
boolean	Valeurs logiques	true false
char	Caractères	‘a’, ‘z’, ‘#’
String	Chaînes de caractères	"Bonjour"

Déclaration

```
type nom;
```

Déclare une *variable* appelée nom, de type type

Ex:

```
int nombreEtudiants;
double poids;
```

Affectation

```
variable = expression;
```

Stocke la valeur de expression dans variable.
Le contenu précédent de variable est écrasé !

Ex:

```
nombreEtudiants = 120;
poids = 54.5;
nom = "Gildas";
reponse = true;
```

Type tableau

- **Tableau**
  - Structure de données qui stocke exactement N éléments du même type (N ≥ 1)
  - Les éléments sont stockés de manière *contiguë* en mémoire

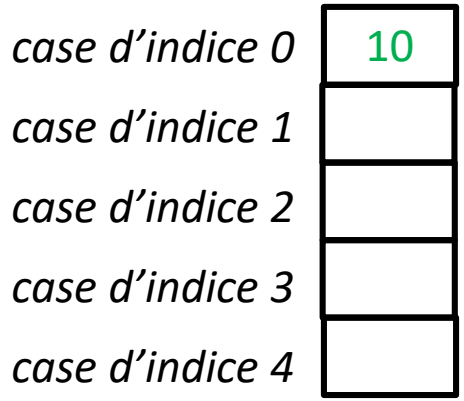
- Déclaration

  type [] nom = new type[N]; *// N nombre entier ≥ 1*

  Ex: int[] tableau = new int[5];

- Accès
  - Chaque case du tableau a un indice entre 0 et N-1
  - Une case du tableau s'utilise comme une variable

  - Ex: tableau[0] = 10;



Input/Output

- Pour afficher des messages texte à l'écran:
  - System.out.println(*valeur ou variable*);

  - Ex: System.out.println("Bonjour");
    System.out.println(nombreEtudiants);

- Pour lire ce que l'utilisateur tape au clavier:
  - En haut du code : import java.util.Scanner;
  - Dans votre programme :

```
Scanner scan = new Scanner(System.in);      // commande « magique » pour l'instant
int x = scan.nextInt();
String reponse = scan.nextLine();
double d = scan.nextDouble();
```

…suite au tableau…

Opérateurs

Type	Opérations
int	+ - * / % ++ ---
long	+ - * / % ++ --
double	+ - * /
boolean	&& || !
char	++ --
String	+

Ex:

```
facile = 1 + 1;
poids = 54.5 * 1.1;
nom = "Gildas" + " Kermarrec";
reponse = !true;
lettre = ‘a’; lettre++;
```

Comparaisons

Opérateur en Java	Signification
==	Egalité
!=	Différence
<	Strictement plus petit
<=	Plus petit ou égal
>	Strictement plus grand
>=	Plus grand ou égal

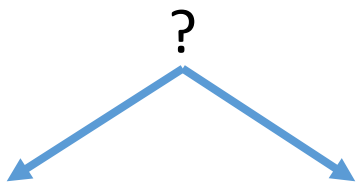
Ex:

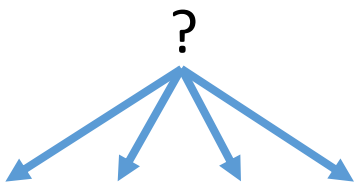
```
x = 42;
y = 100;
reponse = (x > y); // false
```

…autres exemples au tableau. ***Attention au piège des String !****…*

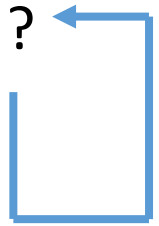
Contrôle de flux

- On ne veut pas toujours que le programme s'exécute « linéairement »
- Trois outils :
  - Conditions

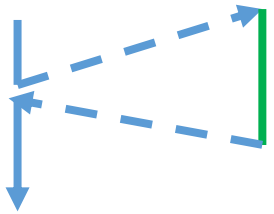




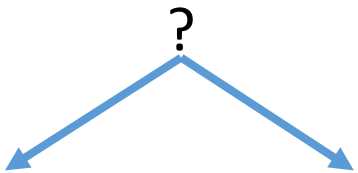
  - Boucles



  - Fonctions



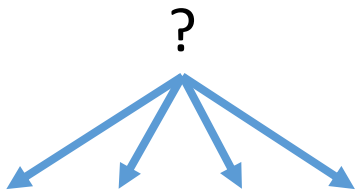
Conditions : le **if**




```
if (condition) {
   ….code exécuté si la condition est vraie…
} else {
   …code exécuté si la condition est fausse…
}
```

Ex:
```
if (age < 18) {
        System.out.println("Mineur");
    } else {
        System.out.println("Majeur");
    }
```

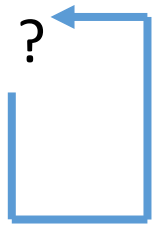
Conditions : le **switch**




```
switch (variable) {
        case valeur1: …code exécuté si variable == valeur1…
                      break;
        case valeur2: …code exécuté si variable == valeur2…
                      break;
        …
        default: …code exécuté si variable n'a aucune des valeurs ci-dessus…
}
```

Ex: *…fait au tableau…*

Itération : le **for**



- Itération : répéter les mêmes instructions
- On utilise **for** quand on sait combien de fois on veut répéter

```
for (initialisation; condition; incrémentation) {
        …instructions à répéter…
}
```

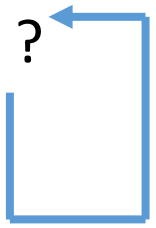
Ex:

```
for (int i = 0; i<5 ; i++) {
        System.out.println(i);
}
```

…autres exemples au tableau…

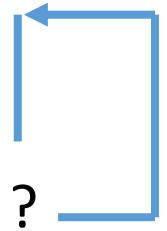
Itération : le **while**

Parfois on ne sait pas à l'avance combien d'itérations on va faire :
boucles while ou do…while




```
while (condition pour continuer) {
    … instructions à répéter…
}
```

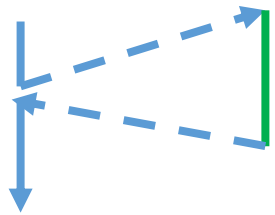
```
do {
    … instructions à répéter…
} while (condition pour continuer)
```



Ex: *…au tableau :*

- *le code PIN*
- *ne pas se tirer une balle dans le pied avec un while !*

Les fonctions



- Idée : transférer le contrôle de l'exécution à une autre partie du code

- Permet de bien séparer les tâches : *une fonction est chargée d'une tâche*
  - Permet de n'écrire qu'une fois le code pour cette tâche (réutilisation de code)

Un code avec deux fonctions

```
class Truc {
        public static void tache1() {
                ….
        }
        public static void tache2() {
                …
        }

        public static void main(String[] args) {
                tache1();
                tache2();
        }
}
```

 : flux d'exécution

…complété au tableau…
…exemple simple d'exécution…

Anatomie d'une fonction + return

modifieurs type_de_retour nom(*….paramètres…*) { *….instructions….* }

Appel: nom(valeurs des paramètres);

- Une fonction renvoie **une** valeur de type type_de_retour
  - Sauf si type_de_retour est void, dans ce cas pas de valeur retournée
  - Dans la fonction, la valeur retournée est précédée du mot clé **return**
  - **return** termine immédiatement la fonction !

Ex: …au tableau…

Les paramètres de la fonction

- Paramètre = moyen d'envoyer des informations à la fonction
- Un paramètre s'utilise comme une variable dans la fonction
- Il prend la valeur donnée au moment de l'appel
- Déclaration de paramètres :

  …(type_parametre_1 nom_parametre_1, …, type_parametre_N nom_parametre_N)

Ex:

```
public static int soustraction(int a, int b) {
        return a – b;
}
….
```

dans le main: int x = soustraction(10,5);

Transmission par valeur ou par référence

Qu'est ce qu'il y a dans les paramètres ?

- Types de base (int, float, double, char, boolean) sont « transmis » par **valeur**
  - i.e. une copie de la valeur donnée lors de l'appel est mise dans le paramètre
  - Toute modification de cette valeur dans la fonction n'est pas répercutée dans l'appelant

  Ex: *…au tableau…*

- Autres types (tableaux,….) sont « transmis » par **référence**
  - Ils peuvent être très gros (tableau de millions d'éléments) donc la copie est trop risquée
  - Java donne juste leur « adresse » dans le paramètre (transparent pour le programmeur)
    - *Pour être précis: Java donne une copie de cette adresse*
  - Toute modification faite dans la fonction est répercutée dans l'appelant !
  - Cas spécial : String n'est jamais modifiable (on dit **immutable**)

  Ex: *…au tableau…*

Récursivité

- Utiliser l'appel de fonction pour une autre forme de boucle
- Fonction récursive = fonction qui s'appelle elle-même
- Il faut :
  - Un **cas de base** pour arrêter la récursion
  - Un **cas récursif** pour passer aux itérations suivantes

Ex:

```
public static int fibonacci(int n) {
        if ((n == 0) || (n == 1)) {
            return 1;
        } else {
            return fibonacci(n-1) + fibonacci(n-2);
        }
}
```

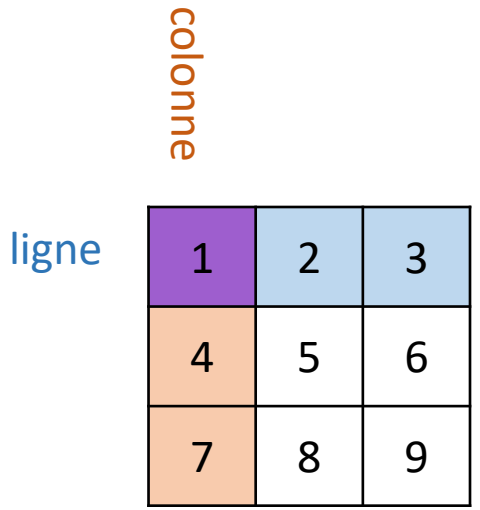
…autre exemple au tableau…

Durée de vie des variables

- Une variable « n'existe » pas partout dans le code
- En général : elle n'existe que dans le bloc où elle est définie
  - Bloc : commence par **{** et finit par **}**
  - Ex: *…au tableau…*

- Une variable d'un bloc intérieur peut masquer une variable d'un bloc extérieur
  - Ex: *…au tableau…*

Tableaux multidimensionnels (1/2)



- Déclaration

```
type[][] nom = new type[nb lignes][nb colonnes];
```

- Accès

```
nom[ligne][colonne];
```

- Ex:

```
int[][] tab = new int[3][3];
for (int i=0 ; i<3 ; i++)
    for (int j=0 ; j<3 ; j++)
        tab[i][j] = i+j ;
```

```
int[][] tab2 = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};
```

Tableaux multidimensionnels (2/2)

- Représentation en mémoire : tableau de tableaux

  …schéma au tableau…

- « Ragged arrays » : lignes de taille différente
  - Ex :

```
int[][] ragArray = new int[3][];
for (int i=0 ; i<3 ; i++)
    ragArray[i] = new int[i];
```