Chapitre 7

Les sous-programmes

Notions avancées

1. La structure d'un programme

a) Un programme contient :

- Une fonction principale, appelée **`main()`.**
- Un ensemble de fonctions définies par le programmeur.
- Des instructions de déclaration de variables.

b) Les fonctions contiennent :

- Des instructions de déclaration de variables.
- Des instructions élémentaires (affectation, test, répétition, …).
- Des appels à des fonctions, prédéfinies ou non.

Chaque fonction est comparable à une boîte noire, dont le contenu n'est pas visible en dehors de la fonction.

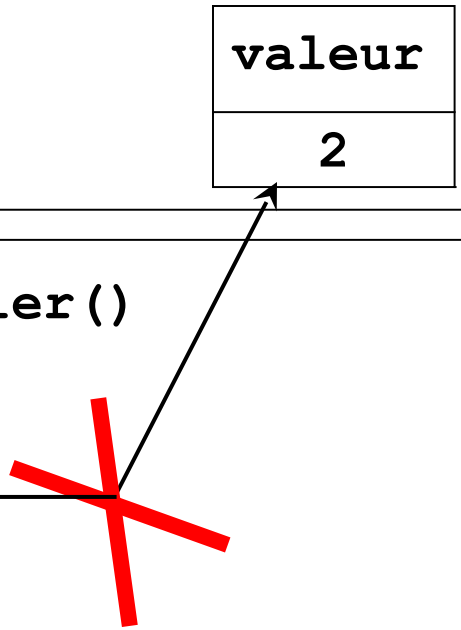
1.1 La visibilité des variables

Il existe une notion « d'intérieur » et « d'extérieur » aux fonctions.

La déclaration de variables est une opération réalisable à l'intérieur ou à l'extérieur d'une fonction.

Une fonction ne peut pas utiliser dans ses instructions une variable déclarée dans une autre fonction.

```
public class Visibilité
{
   public static void main( String[] arg )
   {
      // déclaration de variables
      int valeur = 2;
   }

   public static void modifier()
   {
      valeur = 3;
   }
}
```



Dans l'exemple :

- Impossible d'accéder à la variable **`valeur`** depuis la fonction **`modifier()`**.

- La variable **`valeur`** n'est définie que dans la fonction **`main()`**.

→ Erreur de compilation

1.2 Variable locale à une fonction

Une variable déclarée à l'intérieur d'une fonction est une **variable locale** à cette fonction.

Les variables locales n'existent que pendant le temps d'exécution de la fonction.

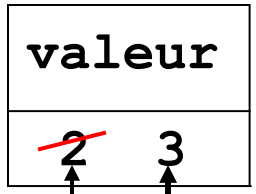
Attention : deux variables portant le même nom, locales à des fonctions différentes, n'utilisent pas la même case mémoire.

1.3 Variables de classe

Variables définies en **début** de la classe, hors de toute fonction.

Elles sont visibles depuis **toutes** les fonctions de la classe.

```
public class VariableDeClasse
{
   static int  valeur;

   public static void main( String[] arg )
   {
      valeur = 2;
      modifier();
   }

   public static void modifier()
   {
      valeur = 3;
   }

}
```



Résultats d'exécutions :

`valeur` = 2 avant **`modifier()`**
`valeur` = 3 dans **`modifier()`**
`valeur` = 3 après **`modifier()`**

L'ordinateur ne crée **qu'un seul emplacement mémoire**.

Remarques :

- Ne pas utiliser que des variables de classes (à la place des variables locales)

- Plusieurs variables peuvent porter le même nom

  - C'est la variable locale qui est utilisée.

  - Pour les différencier, utiliser le nom de la variable de classe préfixée par le nom de la classe.

    Ex: `NomClasse.NomVariable = 3;`

  - N'utiliser que les variables de classe que lorsque l'on ne peut pas faire autrement et en utilisant leur noms complets.

2. Les fonctions communiquent

L'utilisation des variables de classes peut être une source d'erreur.

Utiliser les paramètres des fonctions pour passer les valeurs entre sous-programmes.

Exemple : programme composé de deux fonctions **`main()`** et **`tripler()`**, et d'une variable, **`valeur`**, locale à la fonction **`main()`**. **`Tripler()`** permet de multiplier par trois le contenu de **`valeur`**.

2.1 Passage de paramètres par valeur

N'utiliser que des variables locales.

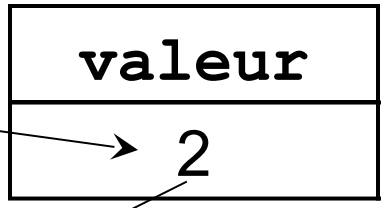
`Tripler()` doit connaître le contenu de la variable **`valeur`** pour pouvoir la multiplier par trois.

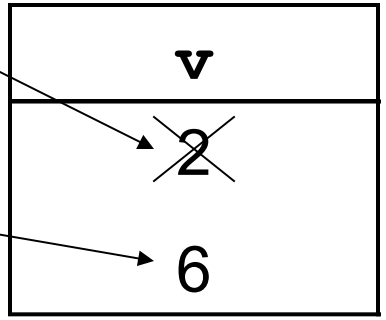
Utilisation d'un paramètre contenant **`valeur`**.

```
public class ParValeur
{
  public static void main (String [] arg)
  {
    int valeur = 2 ; // Déclaration des variables

    System.out.println("Valeur = " + valeur +
                       " avant tripler()");
    tripler(valeur); // Appel à la fonction

    System.out.println("Valeur = " + valeur +
                       " apres tripler()");
  } // fin de main()

...
```

```
...

  public static void tripler (int v)
  {
    System.out.println("Valeur = " + v +
                       " dans tripler()");

    v = 3 * v;

    System.out.println("Valeur = " + v +
                       " dans tripler() ");
  } // fin de tripler()

} //fin de class ParValeur
```

```
public class ParValeur
{
  public static void main (String [] arg)
  {
    int valeur = 2 ;
    tripler(valeur);
  }

  public static void tripler (int v)
  {
    v = 3 * v;
  }
}
```





tripler(valeur)

Résultat de l'exécution :

valeur=2 avant **tripler()**

valeur=2 dans **tripler()**
// valeur = valeur * 3
valeur=6 dans **tripler()**

valeur=2 après **tripler()**

2.2 Le résultat d'une fonction

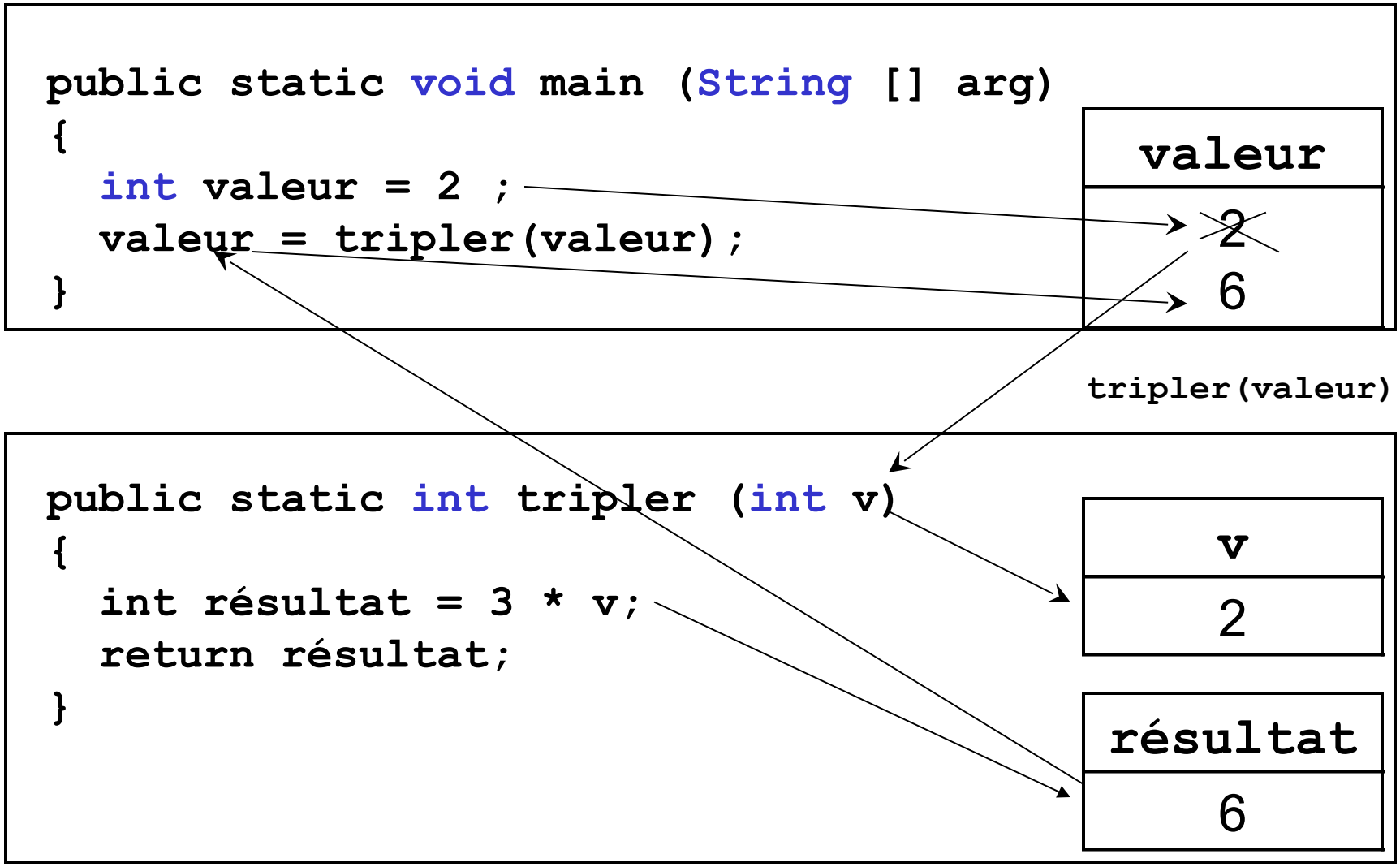
Retourner la valeur calculée par l'intermédiaire de l'instruction **`return`**

Exemple : le résultat de la fonction **`tripler()`** est maintenant renvoyé grâce à **`return`**.

```
public class RésultatFonction
{
  public static void main (String [] arg)
  {
    // Déclaration des variables
    int valeur = 2 ;

    System.out.println("Valeur = " + valeur +
                       " avant tripler()");

    valeur = tripler(valeur); // Appel à la fonction

    System.out.println("Valeur = " + valeur +
                       " apres tripler()");
  } // fin de main()

...
```

```
...


  public static int tripler (int v)
  {
    System.out.println("v =" + v + " dans tripler()");


    int résultat = 3 * v ;

    System.out.println("Resultat = " + résultat +
                       " dans tripler()");
    return résultat;  // Retour du résultat
  } // fin de tripler()

} //fin de class Resultat
```

```
public class RésultatFonction
{
  public static void main (String [] arg)
  {
    int valeur = 2 ;
    valeur = tripler(valeur);
  }

  public static int tripler (int v)
  {
    int résultat = 3 * v;
    return résultat;
  }
}
```



Résultat :

- valeur=2 avant tripler()
- v=2 dans tripler()
- résultat=6 dans tripler()
- valeur=6 après tripler()

Lorsqu'il y a plusieurs résultats à retourner :

- Il n'est pas possible avec **`return`** de renvoyer plusieurs valeurs.

➔ Utilisation des **objets**.