Rappels et compléments

Déclaration de méthode d'instance

[Access] [Modif] Type IdMéth([Args]) [ClauseExc] [Corps]

Access →	*Modif* →	*Args* →	*ClauseExc* →
public	**abstract**	*Type*$_1$ arg$_1$, ..., *Type*$_n$ arg$_n$	**throws** *Exc*$_1$, ..., *Exc*$_n$
protected	**final**		
private			

```
public        void m(int a, int b)      { ... }
```

signature

~~**abstract final**~~
~~**private abstract**~~
private final ← inutile

déclaration imbriquée

```
void m() {
    void n() {
        ...
    }
    ...
}
```

récursivité

```
long fact(int k) {
    if (k <= 1) {
        return 1;
    }
    return k * this.fact(k - 1);
}
```

variable locale `final`

```
class A {
    void m() {
        final int j = 1;
        j = 2;
    }
}
```

paramètre `final`

```
class A {
    void m(final int j) {
        j = 2;
    }
}
```

En Java, les paramètres des méthodes sont passés *par valeur*.

paramètre formel = variable locale contenant la valeur du paramètre effectif

```
class Test {
    void t(int n) {
        n = 2;
        System.out.println(n);
    }
}
```

```
Test obj = new Test();
int x = 5;
obj.t(x); // 2
System.out.println(x); // 5
```

```
class A {
    int n = 5;
}

class Test {
    void t(A u) {
        u.n = 2;
        System.out.println(u.n);
    }
}
```

```
Test obj = new Test();
A x = new A();
System.out.println(x.n); // 5
obj.t(x); // 2
System.out.println(x.n); // 2
```

la variable `x` n'a pas changé de valeur...

... mais l'état de l'objet `x` a été modifié durablement !

Rappels et compléments

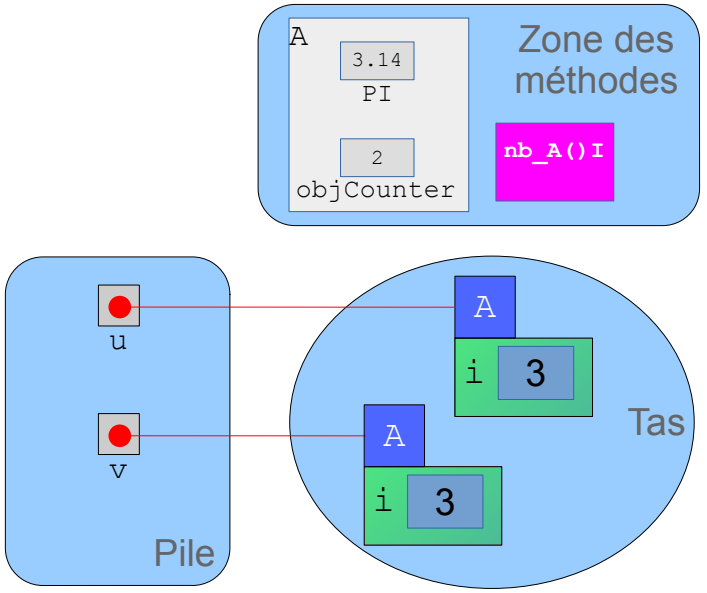
Caractéristiques de classe : attributs ou méthodes déclarés avec le mot clé **`static`**.

```java
public class A {
    // ATTRIBUTS STATIQUES
    static final double PI = 3.14;
    private static int objCounter = 0;

    // ATTRIBUTS D'INSTANCES
    private int i;

    // CONSTRUCTEURS
    public A() {
        i = 3;
        objCounter += 1;
    }

    // OUTILS
    public static int nb() {
        return objCounter;
    }
}
```



bloc statique d'initialisation

```java
public class A {
    // ATTRIBUTS STATIQUES
    private static final String[] DIR_NAMES = new String[] {
        "N", "E", "S", "W"
    };

    private static final Map N_TO_D;
    static {
        N_TO_D = new HashMap();
        N_TO_D.put("N", new Direction("North"));
        N_TO_D.put("E", new Direction("East"));
        N_TO_D.put("S", new Direction("South"));
        N_TO_D.put("W", new Direction("West"));
    }

    ...
}
```

Rappels et compléments

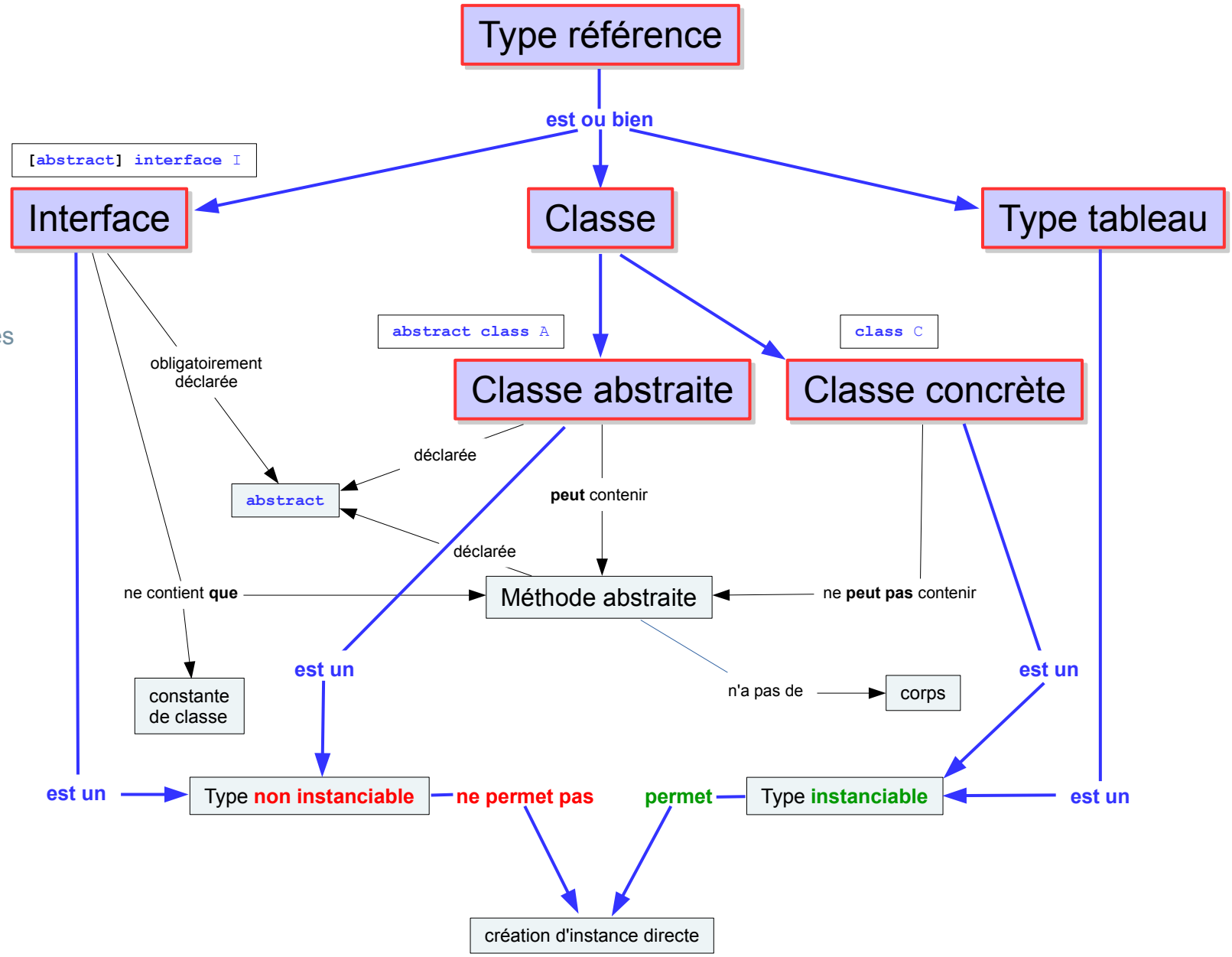
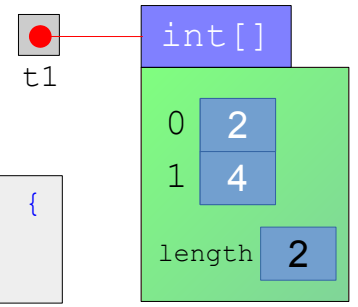


Tableau simple : tableau à une seule dimension.

```
1  int[] t1;
2  t1 = new int[2];
3  for (int i = 0; i < t1.length; i++) {
       t1[i] = 2 * (i + 1);
   }
```



instance directe de tableau d'`int`

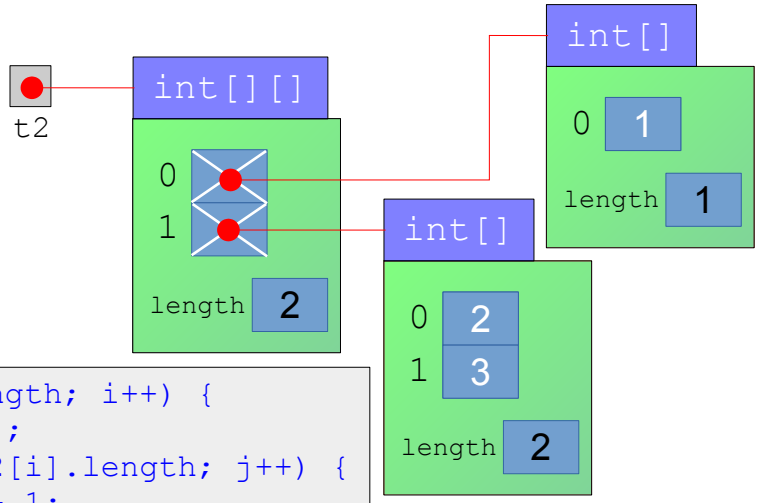
ou aussi 1-2-3

```
int[] t1 = new int[] { 2, 4 };
```

Tableau à n dimensions : tableau simple de tableaux à n-1 dimensions.

n > 1

```
1  int[][] t2;
2  t2 = new int[2][];
3  for (int i = 0; i < t2.length; i++) {
       t2[i] = new int[i + 1];
       for (int j = 0; j < t2[i].length; j++) {
           t2[i][j] = i + j + 1;
       }
   }
```



ou aussi 1-2-3

```
int[][] t2 = new int[][] {
    { 1 }, { 2, 3 }
};
```

Boucle `for` étendue : boucle **`for`** utilisable
- sur un tableau*
- lors d'un parcours en lecture de ce tableau
- de la forme **`for`** `(`**`[final]`** *`Type IdVar : Exp`*`)` `I`

  `Exp` est une expression de type `T[]`

  `Type` est un type avec lequel `T` est compatible

```
long s = 0;
for (long n : getLongTab()) {
    s += n;
}
```

équivalent

```
long s = 0;
long[] t = getLongTab();
for (int i = 0; i < t.length; i++) {
    long n = t[i];
    s += n;
}
```

Rappel : boucle **`for`** classique de la forme

`for` `(`**`[`**`e1`**`];` **`[`**`e2`**`];` **`[`**`e3`**`]`**`)` `I`

`e1` : *`Type IdVar = Init`*
initialisation des variables de contrôle

`e2` : *`BoolExpr`*
critère de poursuite

`e3` : *`StepExpr`*
variation des variables de contrôle

* : et aussi sur un objet *itérable*, mais on verra ça plus tard...

Type enum : classe (synthétisée) qui est le type générateur d'un nombre fixe d'objets accessibles par des constantes statiques de cette classe.

```
public static final Civ
public enum Civ { UKN, MR, MRS, MS }
```

```
for (Civ c : Civ.values()) {
    System.out.print(c + " ");
}
```

boucle **for** classique correspondante

UKN MR MRS MS

```
Civ[] v = Civ.values();
for (int i = 0; i < v.length; i++) {
    Civ c = v[i];
    System.out.print(c + " ");
}
```

```
UKN.compareTo(MR) < 0
UKN.toString() → "UKN"
Civ.valueOf("UKN") → UKN
Civ.valueOf("nawak") → null
```

~~UKN.clone()~~

~~new Civ()~~

méthodes **final**

```
UKN.name() → "UKN"
UKN.ordinal() → 0
MR.ordinal()  → 1
MRS.ordinal() → 2
MS.ordinal()  → 3
```

```
Civ e = ...;
switch (e) {
    case UKN:
        ...
    case MR:
        ...
    case MRS:
        ...
    case MS:
        ...
    default:
        ...
}
```