

---

**UE Programmation Orientée Objet**


---

## Bases non objet de Java

**Q 1 .** Déclarer et initialiser deux variables entières, puis écrire une séquence d'instructions qui échange leurs valeurs.

```
{
  int x = 1; // 1ere variable
  int y = 2; // seconde variable
  int tmp; // variable temporaire
  tmp = x; x = y; y = tmp;
}
```

**Q 2 .** Ecrire une séquence d'instructions qui calcule le maximum de deux variables entières x et y dans une troisième variable res.

```
if (x >= y)
  res = x;
else
  res = y;
```

**Q 3 .** Idem avec le max de 3 nombres x, y, z, en utilisant un opérateur booléen.

```
if ((x >= y) && (x >= z))
  res = x;
else
  if (y >= z)
    res = y;
  else
    res = z;

ou

if ((x >= y) && (x >= z)) {
  res = x;
} else {
  if (y >= z) {
    res = y;
  } else {
    res = z;
  }
}
```

**Q 4 .** Calculer dans res le PGCD de 2 entiers x et y par l'algorithme d'Euclide.

Algorithme d'Euclide :

- si un des nombres est nul, l'autre est le PGCD;
- sinon il faut soustraire le plus petit du plus grand et laisser le plus petit inchangé; puis, recommencer ainsi avec la nouvelle paire jusqu'à ce que un des deux nombres soit nul. Dans ce cas, l'autre nombre est le PGCD.

```
while ( (x != 0) && (y != 0) ) { // ou x*y != 0
  if (x < y)
    y = y - x;
  else
    x = x - y;
}
if (x == 0)
  res = y;
else
  res = x;
```

**Q 5 .** Mettre un booléen à vrai ou faux selon qu'un entier x est premier ou non ?

On teste s'il est divisible par div, pour div de 2 à x (ou  $\sqrt{x}$  pour être plus efficace) par pas de 1 ou 2 (plus efficace).

```
int div = 2; // diviseur potentiel de x
boolean premier = true; // x est a priori premier
// invariant : \forall d, 1 < d < div : x%d != 0
while ((div < x) && premier) { // ou div < Math.sqrt(x)
  premier = (x%div != 0); // si div divise x, x pas premier
  div = div + 1; // ou div+2
}
```

**Q 6 .** Initialiser un tableau tabn avec les entiers de 1 à n.

---

```

    for (int i=0; i<tabn.length; i++)
        tabn[i] = i+1;

```

---

**Q 7 .** Somme des éléments sur la diagonale d'une matrice carrée.

```

int [][] mat;
mat = new int [5] [5];
... // remplissage de mat int
somme = 0;
for (int i=0; i<mat.length; i++) {
    somme = somme + mat[i][i];
}

```

---

**Q 8 .** Ranger dans max la plus grande valeur d'un tableau tab.

```

int max = tab[0];
// invariant \forall j < i, tab[j] <= max
for (int i = 1; i < tab.length; i++)
    if (max < tab[i])
        max = tab[i];

```

---

**Q 9 .** Ranger dans index le plus petit indice de l'élément qui vaut valeur dans un tableau, sinon mettre length.

```

int valeur = 4; // valeur cherchée
int[] t = {3,4,4,9}; // dans t
int index = 0;
while (index < t.length && t[index] != valeur) {
    index++;
}

```

---

**Q 10 . Triangle de Pascal.** Initialiser, pour un  $n$  donné, un tableau avec les coefficients  $C_n^p$ ,  $p$ ième coefficient binomial d'ordre  $n$ . Rappel :

$$C_n^p = \frac{n!}{p!(n-p)!} \quad \text{soit } C_n^0 = 1$$

$$C_n^n = 1$$

$$C_n^p = C_{n-1}^{p-1} + C_{n-1}^p$$

À l'ordre 4 :

$n = 0$	1				
$n = 1$	1	1			
$n = 2$	1	2	1		
$n = 3$	1	3	3	1	
$n = 4$	1	4	6	4	1

Pour l'ordre  $n$  on utilise un tableau `tp` de dimension 2, avec  $n$  sur la première dimension et  $p$  sur la seconde. On a donc `tp[n][p] = C_n^p`.

```

tp[n][0] = 1
tp[n][n] = 1
tp[n][p] = tp[n-1][p-1] + tp[n-1][p] sinon

```

La taille du tableau en première dimension est l'ordre+1. D'où :

```

// triangle de Pascal de ordre lignes
int ordre = 4;
int [][] tp;
tp = new int [ordre+1] [];
for (int n = 0; n <= ordre; n++) {
    tp[n] = new int [n+1];
    for (int p=0; p<=n; p++)
        if ((p==0) || (p==n))
            tp[n][p] = 1;
        else
            tp[n][p] = tp[n-1][p-1] + tp[n-1][p];
}

```

---

bien réfléchir aux indices aux bornes!

**Q 11 .** Calculer le nombre d'entiers positifs en tête d'un tableau.

```

int[] tpos = {1,2,3,4,-2,4,5};
int nb = 0; // nombre d'entiers positifs en tête de tpos
int i = 0; // variable de boucle
while (i < tpos.length && tpos[i] > 0) { // ?? paresseux !!
    nb = nb+1;
    i = i+1;
}

```

---

**Q 12 .** Calculer la taille de la plus longue séquence d'entiers positifs dans un tableau.

```

int[] t1 = {1,2,3,4,-2,4,5,6,6,6,-2,1};
int taille_max = 0; // le résultat
int taille = 0; // taille de la séquence courante
for (i = 0; i < t1.length; i++){
    // raz taille et maj taille_max si élément négatif
    if (t1[i] < 0) {
        if (taille > taille_max)
            taille_max = taille;
        taille = 0;
    }
    else // incrémenter la taille courante
        taille = taille + 1;
}

```

---

**Q 13 . Le tri bulle.** Idée de l'algorithme : parcourir les  $n$  premières cases du tableau en échangeant deux éléments successifs si le premier est plus grand que le second (soit échanger  $t[i]$  et  $t[i+1]$  si  $t[i] > t[i+1]$ ), ce qui fait remonter comme une bulle le plus grand élément de ces  $n$  cases dans la case d'indice  $n - 1$ , où il est bien placé. Puis on recommence en excluant du parcours les éléments bien placés. Reste à faire varier  $n$  correctement.

ex : Les étapes successives sont représentées verticalement. Après chaque étape un cadre montre le parcours de tableau restant à faire.

0	1	2	3	4	5	
4	6	5	2	1	3	tableau de départ

0	1	2	3	4	5	
4	5	2	1	3	6	après ce premier parcours l'élément d'indice 5 est maintenant bien placé.

0	1	2	3	4	5	
4	2	1	3	5	6	l'élément d'indice 4 est maintenant bien placé aussi.

0	1	2	3	4	5	
2	1	3	4	5	6	

0	1	2	3	4	5	
1	2	3	4	5	6	

0	1	2	3	4	5	
1	2	3	4	5	6	fini!

```

int[] aTrier = ...;

for (i = aTrier.length - 1; i > 0; i--) {
    for (int j = 0; j < i; j++) {
        if (aTrier[j] > aTrier[j+1]) { // mal triés ? on les échange
            tmp = aTrier[j];
            aTrier[j] = aTrier[j+1];
            aTrier[j+1] = tmp;
        }
    } // for j
} // for i

```

---