Programmation orientée objet

Allocation dynamique

Allocation automatique et allocation dynamique

- Automatique:

```
Point p(3,4);
Point p = Point(3,4);
```

- Dynamique:

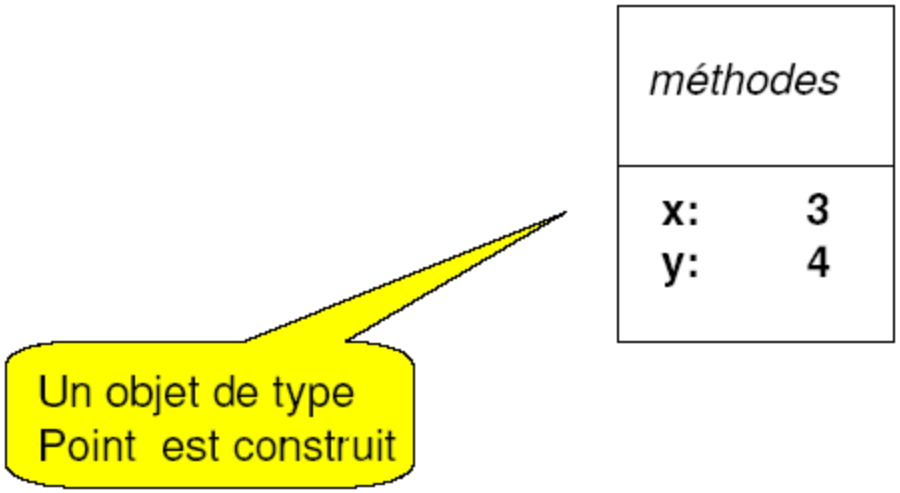
```
Point* p;
p = new Point(3,4);
```

Étapes de l'allocation d'un pointeur

```
Point* p = new Point(3,4);
```

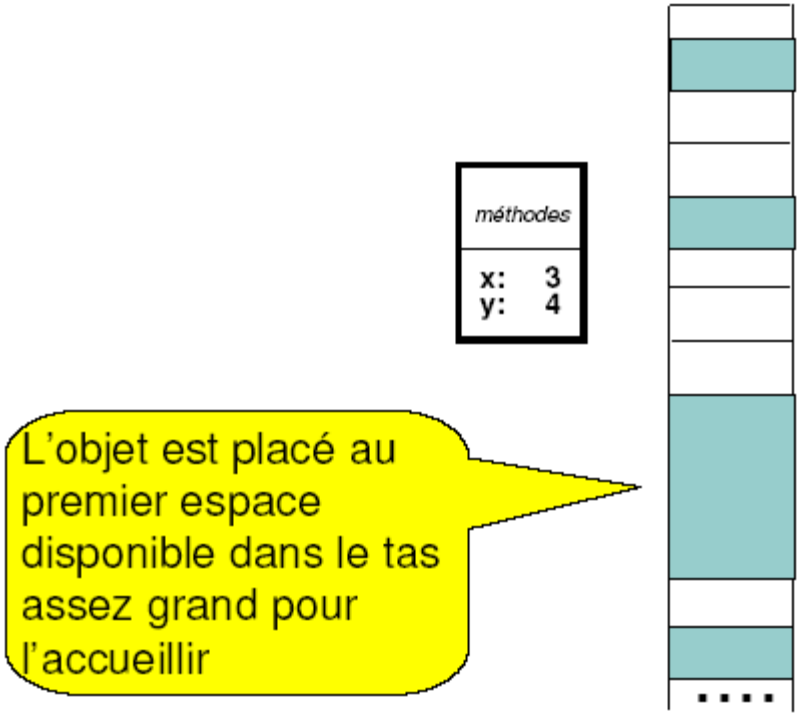
Étapes de l'allocation d'un pointeur

```
Point* p = new Point(3,4);
```



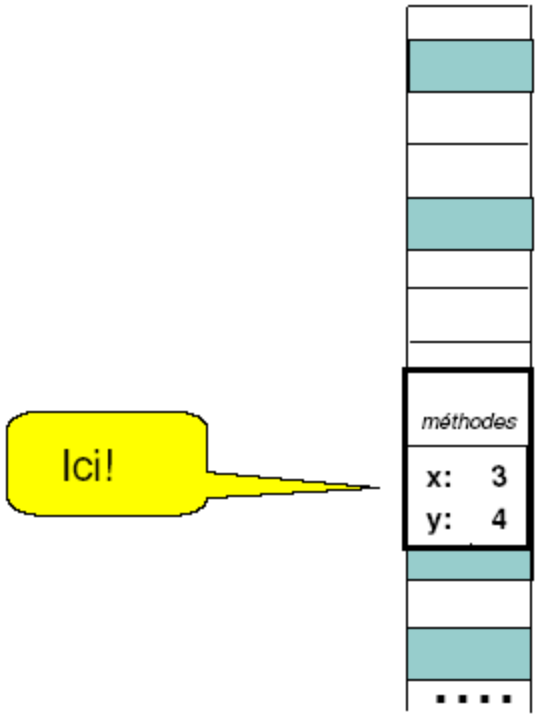
Étapes de l'allocation d'un pointeur

```
Point* p = new Point(3,4);
```



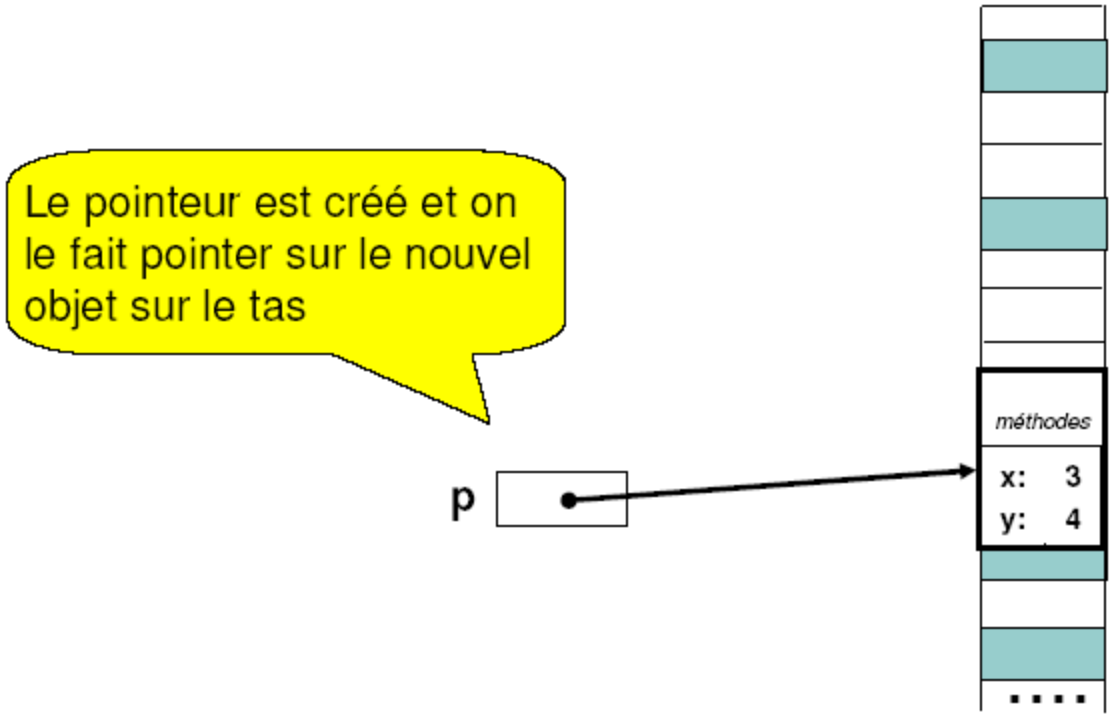
Étapes de l'allocation d'un pointeur

```
Point* p = new Point(3,4);
```



Étapes de l'allocation d'un pointeur

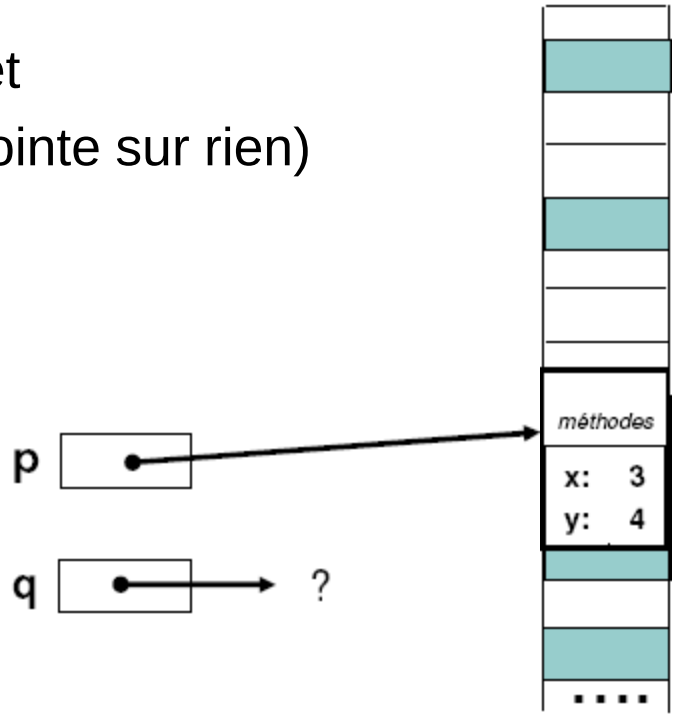
```
Point* p = new Point(3,4);
```



Pointeurs

```
Point* p = new Point(3,4);
Point* q;
```

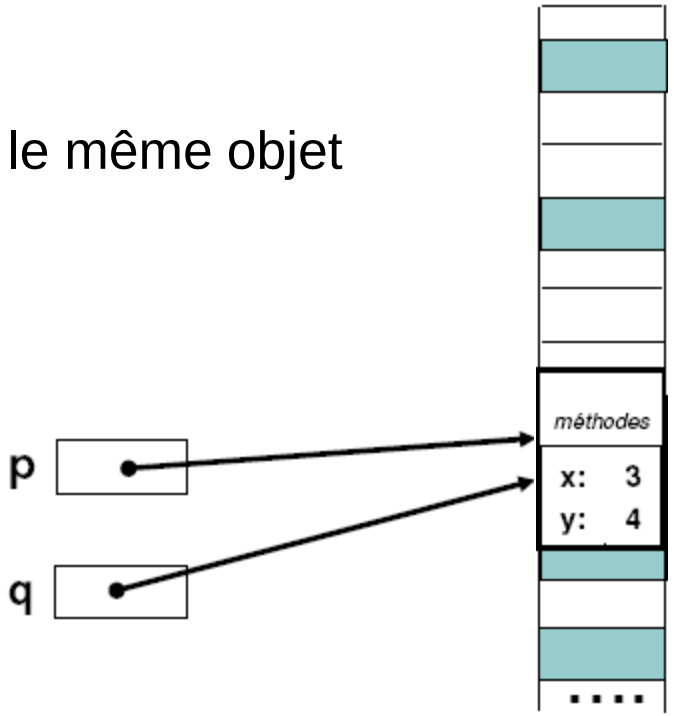
- Le pointeur **p** pointe sur un objet
- Le pointeur **q** est invalide (ne pointe sur rien)



Pointeurs

```
Point* p = new Point(3,4);
Point* q;
q = p;
```

- Les deux pointeurs pointent sur le même objet



Initialisation des pointeurs

- Prenez l'habitude de toujours initialiser vos pointeurs:

```
Point* p = 0;
```

Initialisation des pointeurs

- Dans un constructeur aussi, si la classe définit un attribut dynamique:

```
class UneClasse
{
public:
  UneClasse();
...
private:
  Point* monAttribut_;
  ...
};
```

```
UneClasse::UneClasse()
{
  monAttribut_ = 0;
}
```

Dé-référencement d'un pointeur

- Si **p** est un pointeur, l'expression ***p** retourne l'objet pointé par **p**
- Si on veut appliquer une méthode de l'objet en question:

```
(*p).getSalaire()
```

- Une autre forme synonyme et plus pratique:

```
p->getSalaire()
```

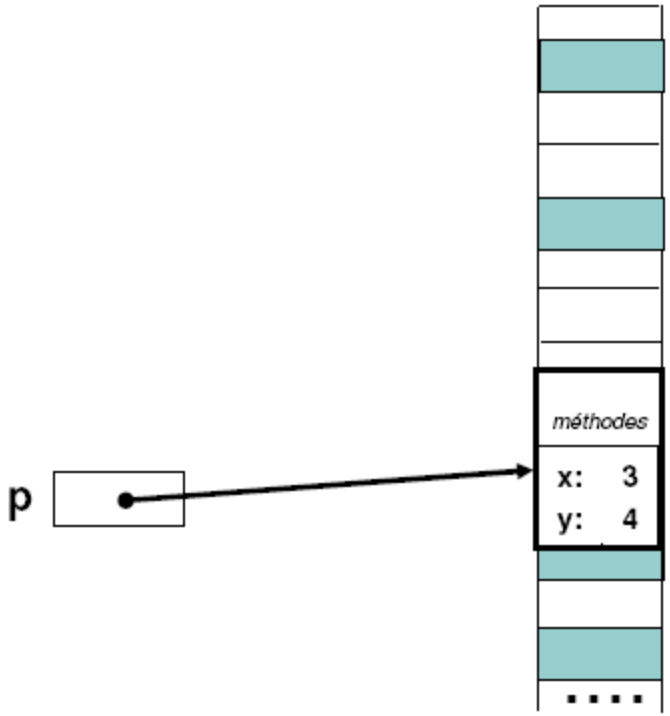
Désallocation de mémoire

- Pour tout appel à **new** il faut retrouver quelque part un appel à **delete** qui désalloue la mémoire:

```
int main()
{
  Point* p = new Point(3,4);
  ...
  delete p;
}
```

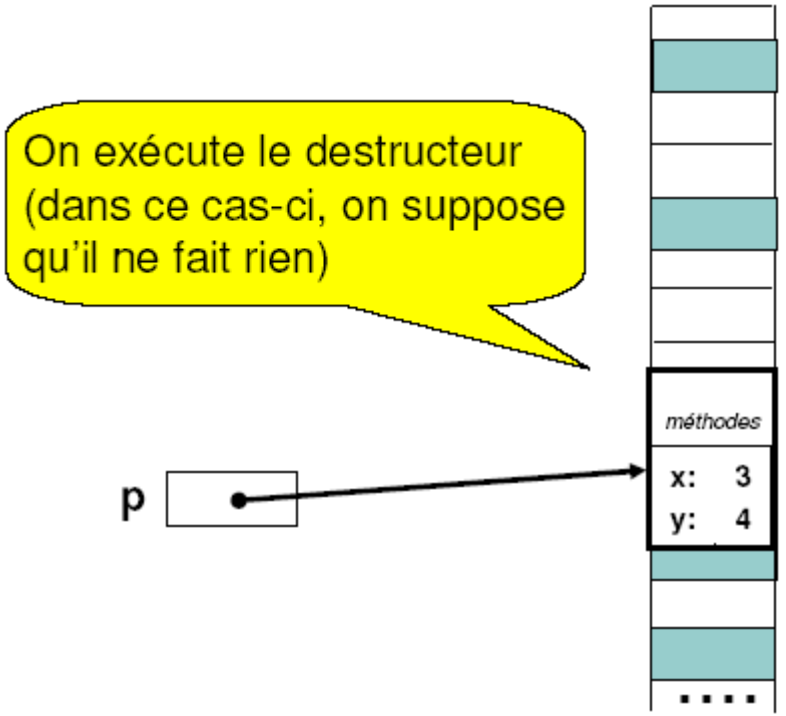
Étapes de la désallocation

```
delete p
```



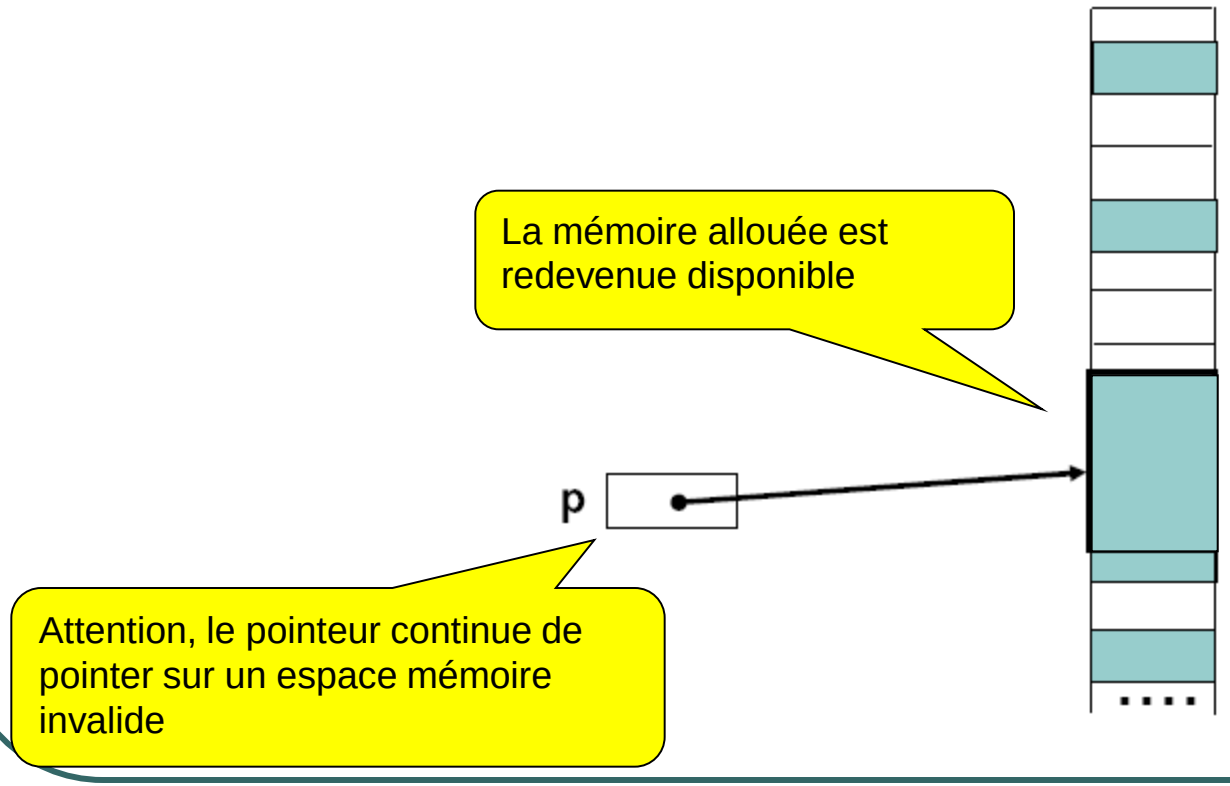
Étapes de la désallocation

```
delete p
```



Étapes de la désallocation

```
delete p
```



Désallocation

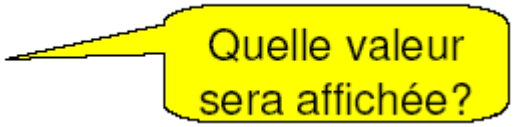
- Ainsi, après l'exécution de **delete**, le pointeur continue de pointer sur le même espace mémoire, devenu invalide
- Pour éviter toute tentative de déréférencer à nouveau ce pointeur, il est bon de le réinitialiser à 0:

```
delete p;
p = 0;
```

Tableaux et pointeurs

- En C++, l'adresse d'un tableau est en fait un pointeur qui pointe sur le premier élément du tableau:

```
int a[5] = {1,2,3,4,5};
int* p = a;
*p = 10;
cout << a[0];
```



Arithmétique des pointeurs

- Si **p** est un pointeur sur un entier, l'expression **p+3** pointe sur le troisième entier en mémoire situé après celui pointé par **p**
- Autre exemple:

```
int a[5] = {1,2,3,4,5};
int* p = a;
++p;
++p;
cout << *p;
```

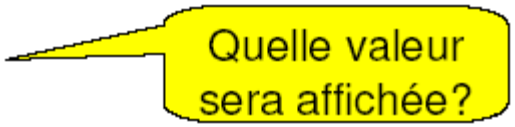


Tableau dynamique

- Considérons l'instruction suivante:

```
int n;
cin >> n;
Point* listeDePoints = new Point[n];
```

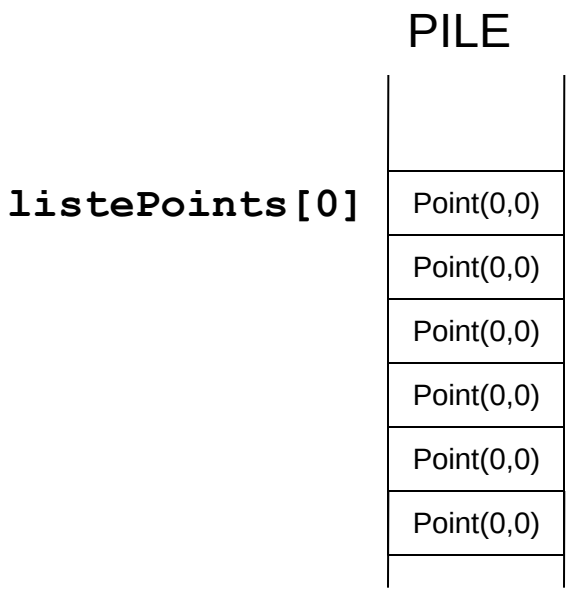
- L'effet de cette instruction est la création sur le tas d'un tableau dynamique dont la taille sera déterminée lors de l'exécution du programme
- Il ne faudra pas oublier de désallouer mémoire:

```
delete [] listeDePoints
```

Récapitulons

- Tableau de points alloué automatiquement:

```
Point
  listePoints[6];
...
```

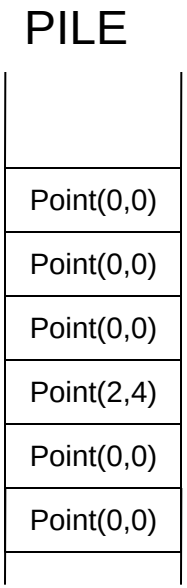


Récapitulons

- Tableau de points alloué automatiquement:

```
Point listePoints[6];
...
listePoints[3].modifierX(2);
listePoints[3].modifierY(4);
```

listePoints[3]

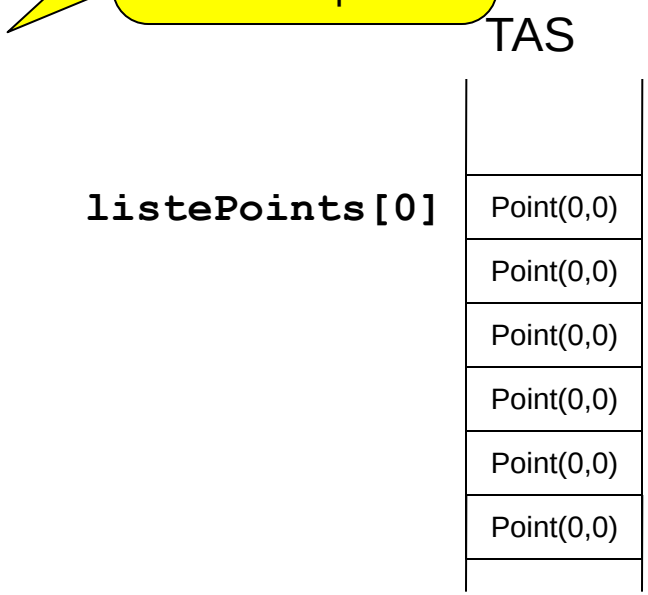


Récapitulons

- Tableau de points alloué dynamiquement:

```
Point* listePoints = 0;
...
listePoints = new Point[6];
```

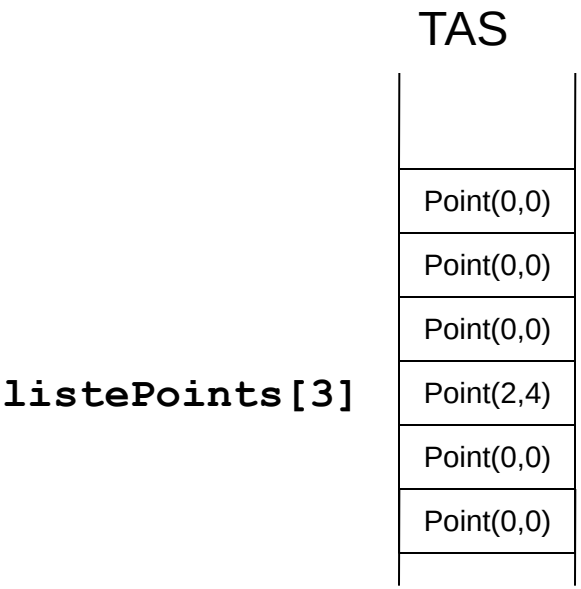
Une séquence de 6 objets de la classe Point est ajoutée dans le heap.



Récapitulons

Tableau de points alloué dynamiquement:

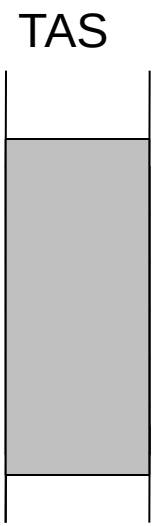
```
Point* listePoints = 0;
...
listePoints = new Point[6];
...
listePoints[3].modifierX(2);
listePoints[3].modifierY(4);
```



Récapitulons

Tableau de points alloué dynamiquement:

```
Point* listePoints = 0;
...
listePoints = new Point[6];
...
listePoints[3].modifierX(2);
listePoints[3].modifierY(4);
...
delete [] listePoints;
listePoints = 0;
```

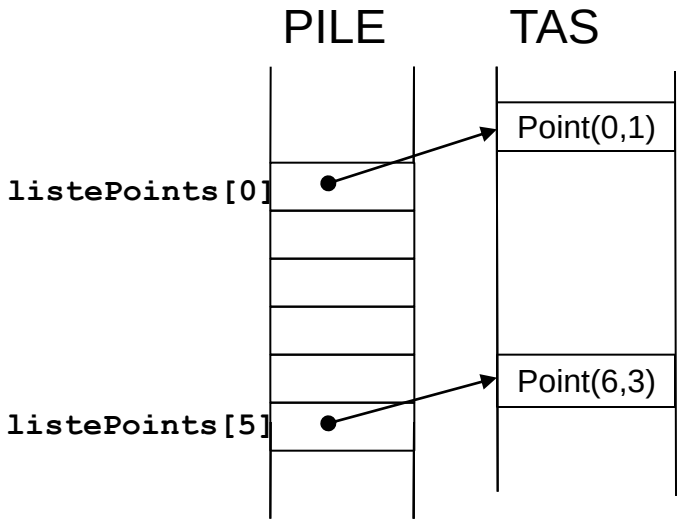


Récapitulons

Tableau de pointeurs de points alloué automatiquement:

```
Point* listePoints[6];
...
listePoints[0] = new Point(0,1);
...
listePoints[5] = new Point(6,3);
...
listePoints[3]->modifierX(2);
listePoints[3]->modifierY(4);
...
for (int i = 0; i < 6; ++i) {
    delete listePoints[i];
}
```

On crée un tableau automatique de 6 pointeurs (qui ne sont pas initialisés).



Récapitulons

Tableau de pointeurs de points alloué dynamiquement:

```
Point** listePoints;
...
listePoints = new Point*[6];
...
listePoints[0] = new Point(0,1);
...
listePoints[5] = new Point(6,3);
...
listePoints[3]->modifierX(2);
listePoints[3]->modifierY(4);
...
for (int i = 0; i < 6; ++i) {
    delete listePoints[i];
}
delete [] listePoints;
listePoints = 0;
```

Une séquence de 6 **pointeurs** est ajoutée dans le heap.