# 2I002
# COMPOSITION D'OBJETS

Vincent Guigue & Christophe Marsala

SCIENCES SORBONNE UNIVERSITÉ

LIP6

MLIA Machine Learning & Deep Learning for Information Access

## Un objet complexe = un objet qui utilise des objets

- Chaque classe reste petite, lisible et facile à débugguer
- Par agrégation, on construit des concepts complexes

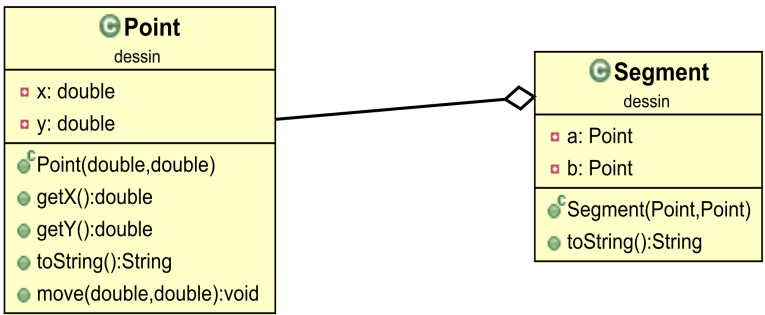
**Syntaxe**: simple et intuitive

```
1  public class Segment{
2    private Point a, b; // simple declaration
3
4    public Segment(Point a, Point b) {
5      this.a = a;
6      this.b = b;
7    }
8    public String toString() {
9      return "Segment [a=" + a + ", b=" + b + "]";
10     // note " "+a <=> " "+a.toString() => implicite en JAVA
11   }
12   public void move(double dx, double dy) {
13      a.move(dx, dy); // vision public du Point
14      b.move(dx, dy);
15   }
16 }
```

```java
public class Segment{
  private Point a,b;
  ...
```

Deux représentations usuelles:

1) Lien d'agrégation: Un segment **est composé de** Point(s)

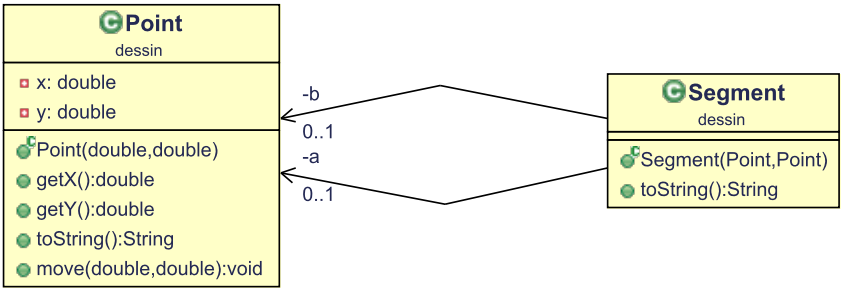


```java
public class Segment{
  private Point a,b;
  ...
```

Deux représentations usuelles:

2) Lien d'utilisation:

- Le segment **utilise** un point en attribut privé nommé a
- Le segment **utilise** un point en attribut privé nommé b

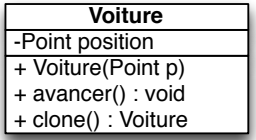


Cas classique : besoin de dupliquer une `Voiture` dont la position est définie par un attribut `Point`

| **Voiture** |
| --- |
| -Point position |
| + Voiture(Point p)<br>+ avancer() : void<br>+ clone() : Voiture |

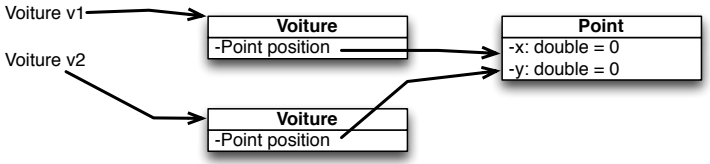
```
1 // Dans voiture
2 Voiture clone(){
3   return new Voiture(position);
4 }
5 // dans le main
6 Voiture v1 = new Voiture(new Point(0,0));
7 Voiture v2 = v1.clone();
```

Cas classique : besoin de dupliquer une `Voiture` dont la position est définie par un attribut `Point`




```java
// Dans voiture
Voiture clone(){
  return new Voiture(position);
}
// dans le main
Voiture v1 = new Voiture(new Point(0,0));
Voiture v2 = v1.clone();
```

⚠ GROS PROBLEME !!




Il y a deux instances de Voiture, mais une seule position... Si l'une bouge, l'autre aussi (on va avoir l'impression qu'elle s'est téléporté)

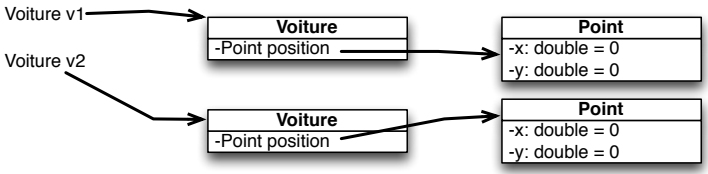
Cas classique : besoin de dupliquer une `Voiture` dont la position est définie par un attribut `Point`

| Voiture |
| --- |
| -Point position |
| + Voiture(Point p)<br>+ avancer() : void<br>+ clone() : Voiture |

```java
// Dans voiture
Voiture clone(){
  return new Voiture(position);
}
// dans le main
Voiture v1 = new Voiture(new Point(0,0));
Voiture v2 = v1.clone();
```

## Solution:

```java
// Dans voiture
Voiture clone(){ // clonage recursif
  return new Voiture(position.clone());
}
```



# Egalité structurelle: attention aux attributs

- Structure standard classique...
- jusqu'au moment du test sur les attributs:

penser à l'égalité structurelle (au lieu de ==)

```
1  public boolean egalite(Voiture v) {
2    if (! position==v.position)
3      return false;
4    return true;
5  }
```

# Egalité structurelle: attention aux attributs

- Structure standard classique...
- jusqu'au moment du test sur les attributs:

penser à l'égalité structurelle (au lieu de ==)

```
1  public boolean egalite(Voiture v) {
2    if (! position==v.position)
3      return false;
4    return true;
5  }
```

```
1  public boolean egalite(Voiture v) {
2    if (! position.egalite(v.position))
3      return false;
4    return true;
5  }
```

⇒ Evidemment, il faut une méthode `egalité` dans `Point`

# Egalité structurelle: attention aux attributs

- Structure standard classique...
- jusqu'au moment du test sur les attributs:

penser à l'égalité structurelle (au lieu de ==)

```
1  public boolean egalite(Voiture v) {
2    if (! position==v.position)
3      return false;
4    return true;
5  }
```

```
1  public boolean egalite(Voiture v) {
2    if (! position.egalite(v.position))
3      return false;
4    return true;
5  }
```

⇒ Evidemment, il faut une méthode `egalité` dans `Point`

⇒ Plus tard, on ne fera que du `equals`... Mais c'est plus tard

- La compilation n'est pas plus difficile que pour un `main` (utilisant lui aussi d'autres classes)...

```
» javac *.java
OU
» javac Point.java Segment.java TestSegment.java
PUIS
» java TestSegment
```