

# A journey through OOP

1<sup>e</sup> année Polytech NS

Sciences du logiciel

2022

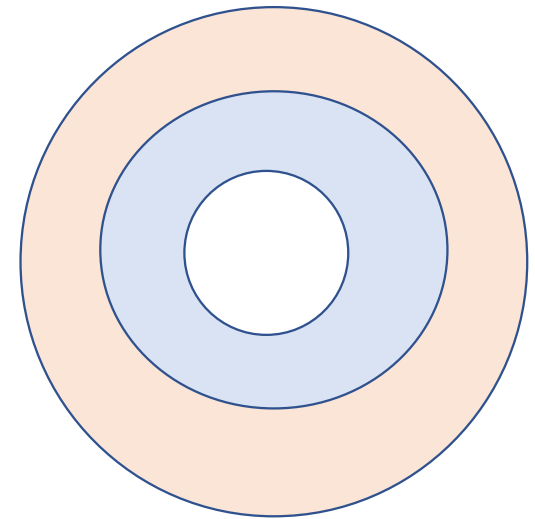
Mireille Blay-Fornarino

# Quick Introduction (Reminders)

# Class Point in java

**Point.java**

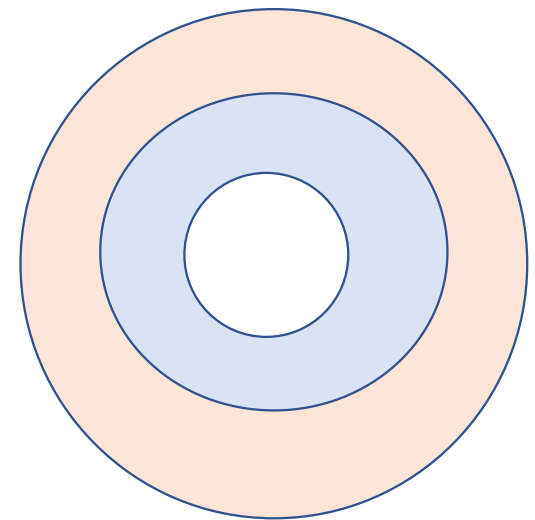
```
class Point { // class  
  
}
```



# Class Point in java and its constructor

## Point.java

```
class Point { // class
    Point() { // empty constructor, optionnal
    }
}
```

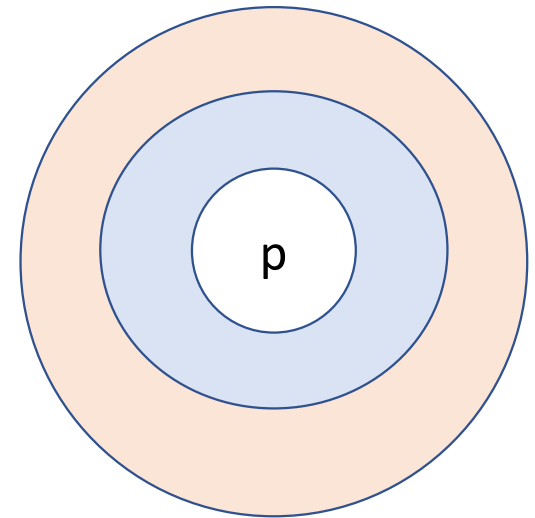


# An instance of Point in java

## Point.java

```
class Point { // class
    Point() { // empty constructor, optionnal
    }
}
```

Point p = new Point();



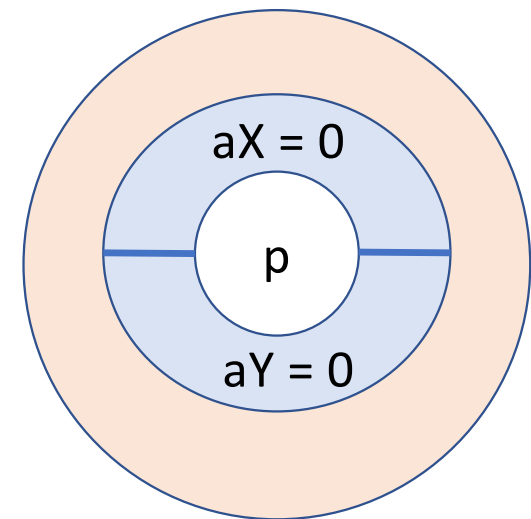
# Instance variables/Attributes/Fields

## Point.java

```
class Point { // class
    int aX; // attribute
    int aY; // attribute

    Point() { //Constructor
        aX= 0;
        aY= 0;
    }
}
```

Point p = new Point();



# Constructor Overload

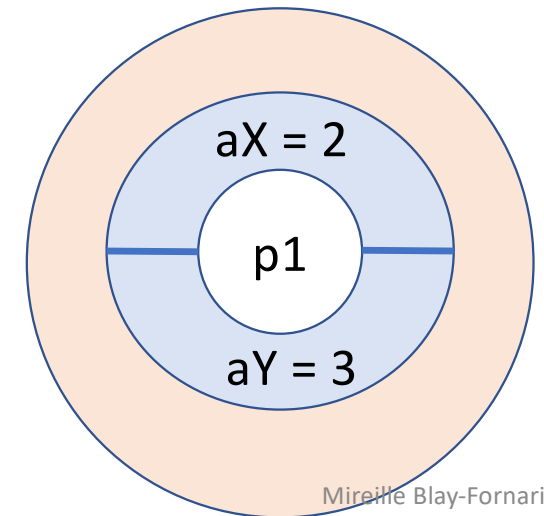
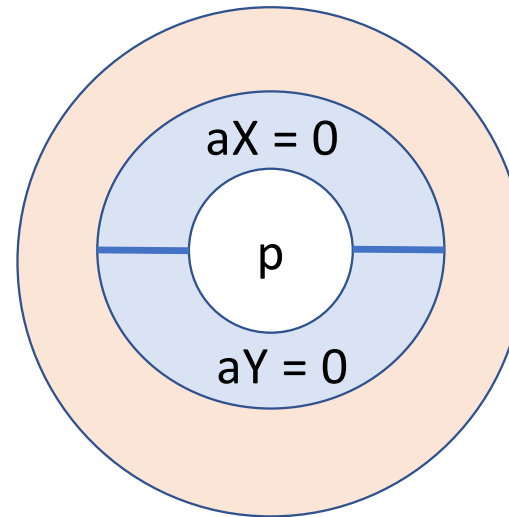
## Point.java

```
class Point { // class
    int aX; // attribute
    int aY; // attribute

    Point() { //Constructor
        aX= 0;
        aY= 0;
    }
    Point(int pX, int pY) { // overload
        aX= pX;
        aY= pY;
    }
}
```

@Based on Jean-Remi Falleri

Point p = new Point()  
Point p1 = new Point(2,3)



# Methods

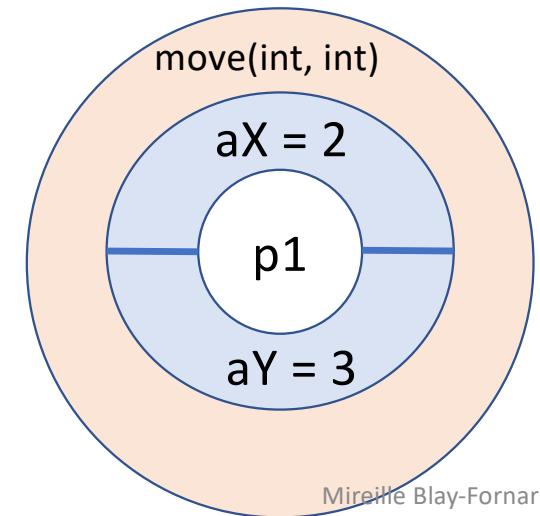
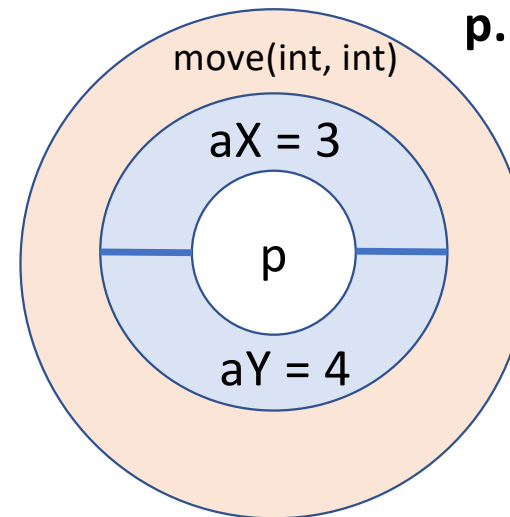
## Point.java

```
class Point { // class
    int aX; // attribut
    int aY; // attribut
    Point() { //Constructor
        aX= 0;
        aY= 0;
    }
    Point(int pX, int pY) { // overload
        aX= pX;
        aY= pY;
    }
    void move(int pX, int pY) { // method
        aX =pX ;
        aY= pY;
    }
}
```

Point p1 = new Point(2,3);

Point p = new Point();

**p.move(3,4);**





# Methods and parameters

- Java classes define **methods**
- Methods may have **parameters** to pass additional information needed to execute.
- Some methods don't **return** anything (**void** return type).
- NB: **Returning is not printing!**

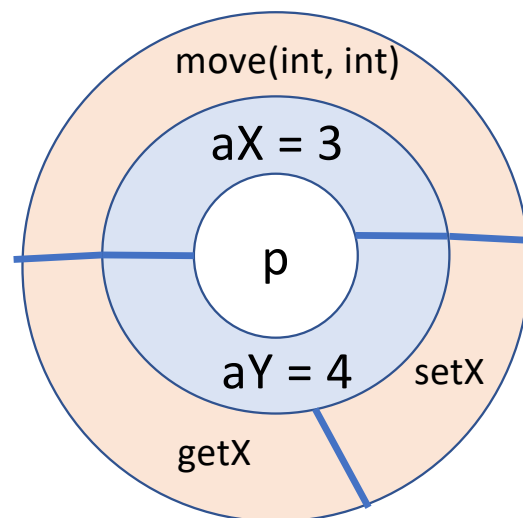
# Accessors & Mutators

## Point.java

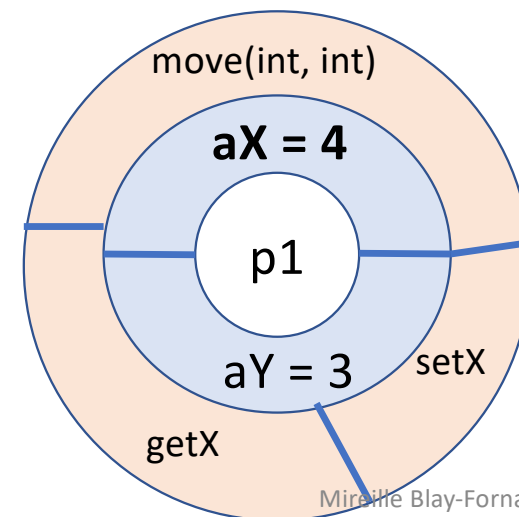
```
class Point { // class
    int aX; // attribut
    int aY; // attribut ...
    ...
    void move(int pX, int pY) { // methode
        aX = pX;
        aY = pY;
    }
    void setX ( int pX ){ // MUTATOR
        aX = pX;
    }
    int getX() { // ACCESSOR
        return aX;
    }
}
```

@Based on Jean-Remi Falleri

```
Point p = new Point();
p.move(3,4);
int x = p.getX();//x=3
```



```
Point p1 = new Point(2,3);
p1.setX(4);
```



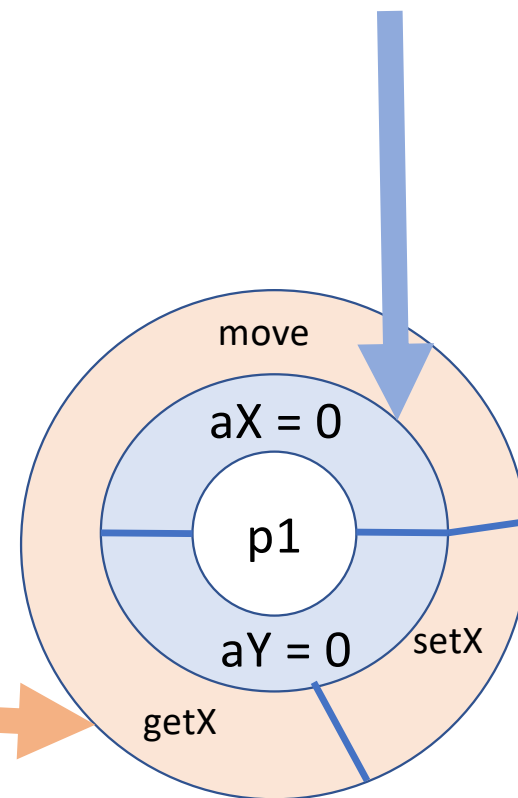
Mireille Blay-Fornarino

# Encapsulation

- Not all object methods are accessible.
- Accessible methods are its interface.

Interface methods

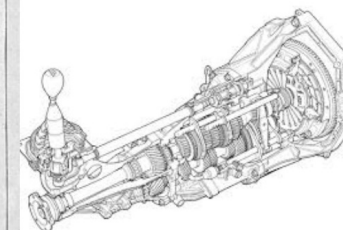
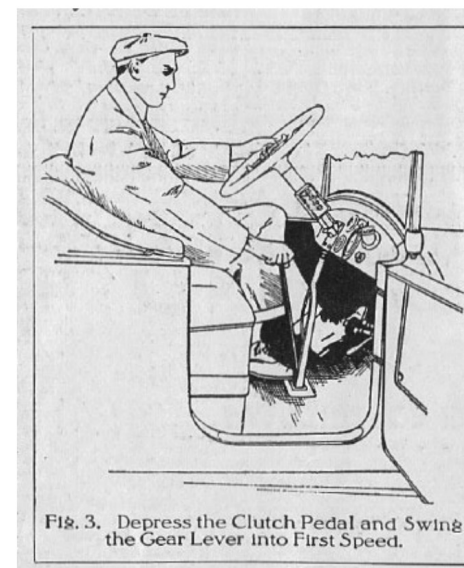
Object state



# Encapsulation

## First BIG IDEA in OOP

- AKA Data Hiding.
- Objects hide stuff from the outside.
- Interface exposes only what's
  - necessary to use the object.



# Class Segment

## Point.java

```
class Point { // class
    private int aX; // attribut
    private int aY; // attribut ...
    public void move(int pX, int pY) { // method
        aX = pX;
        aY = pY;
    }
    public Point() {
        aX = 0;
        aY = 0;
    }
    @Override
    public String toString() {
        return "Point{" + aX +
            "," + aY +
            '}';
    }
}
```

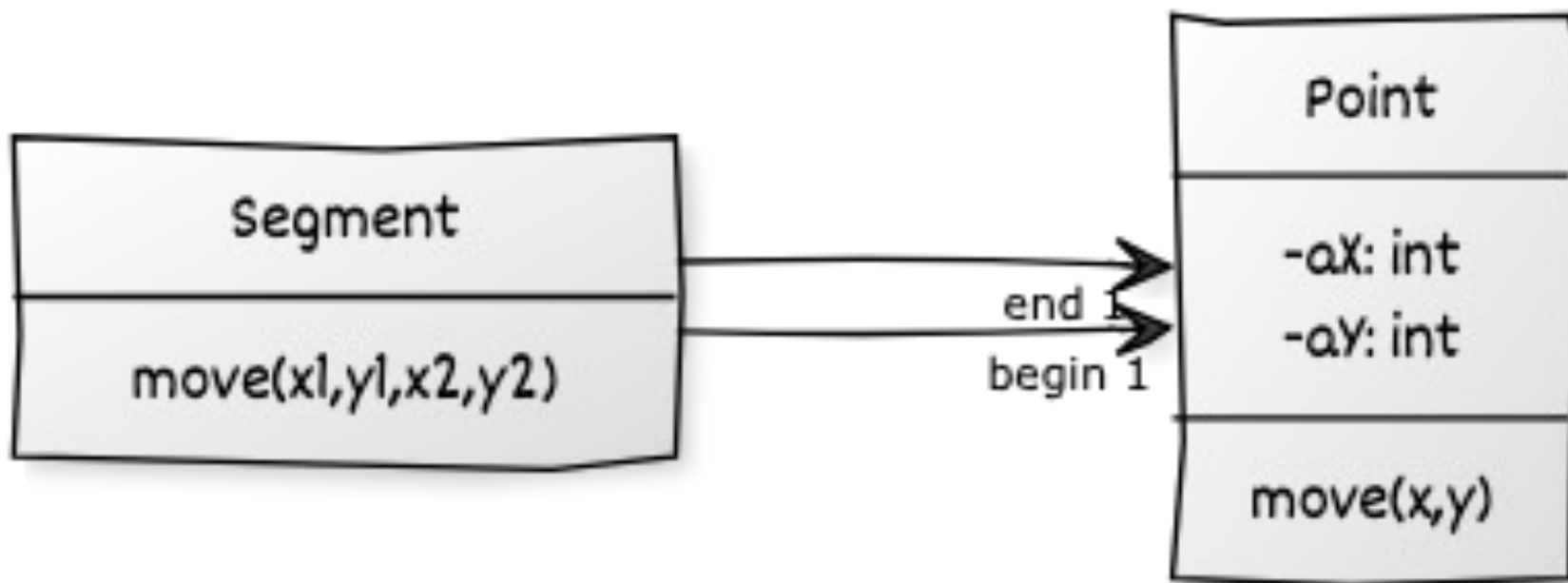
## Segment.java

```
public class Segment {

    private Point begin;
    private Point end;

    public Segment(Point begin, Point end) {
        this.begin = begin;
        this.end = end;
    }
    void move(int x1, int y1, int x2, int y2) {
        this.begin.move(x1,y1);
        this.end.move(x2,y2);
    }
}
```

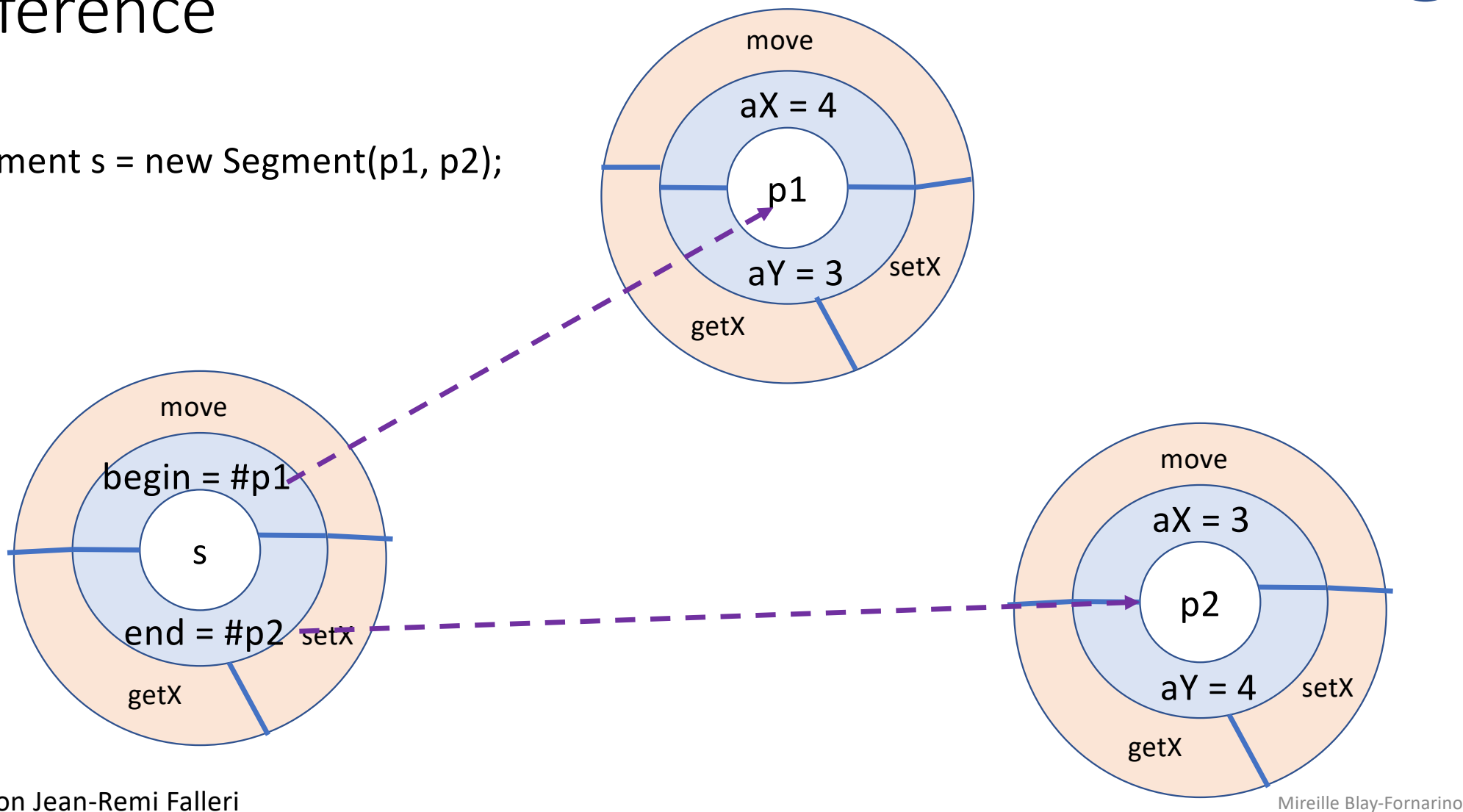
# UML Notation



CREATED WITH YUML

# Reference

Segment s = new Segment(p1, p2);

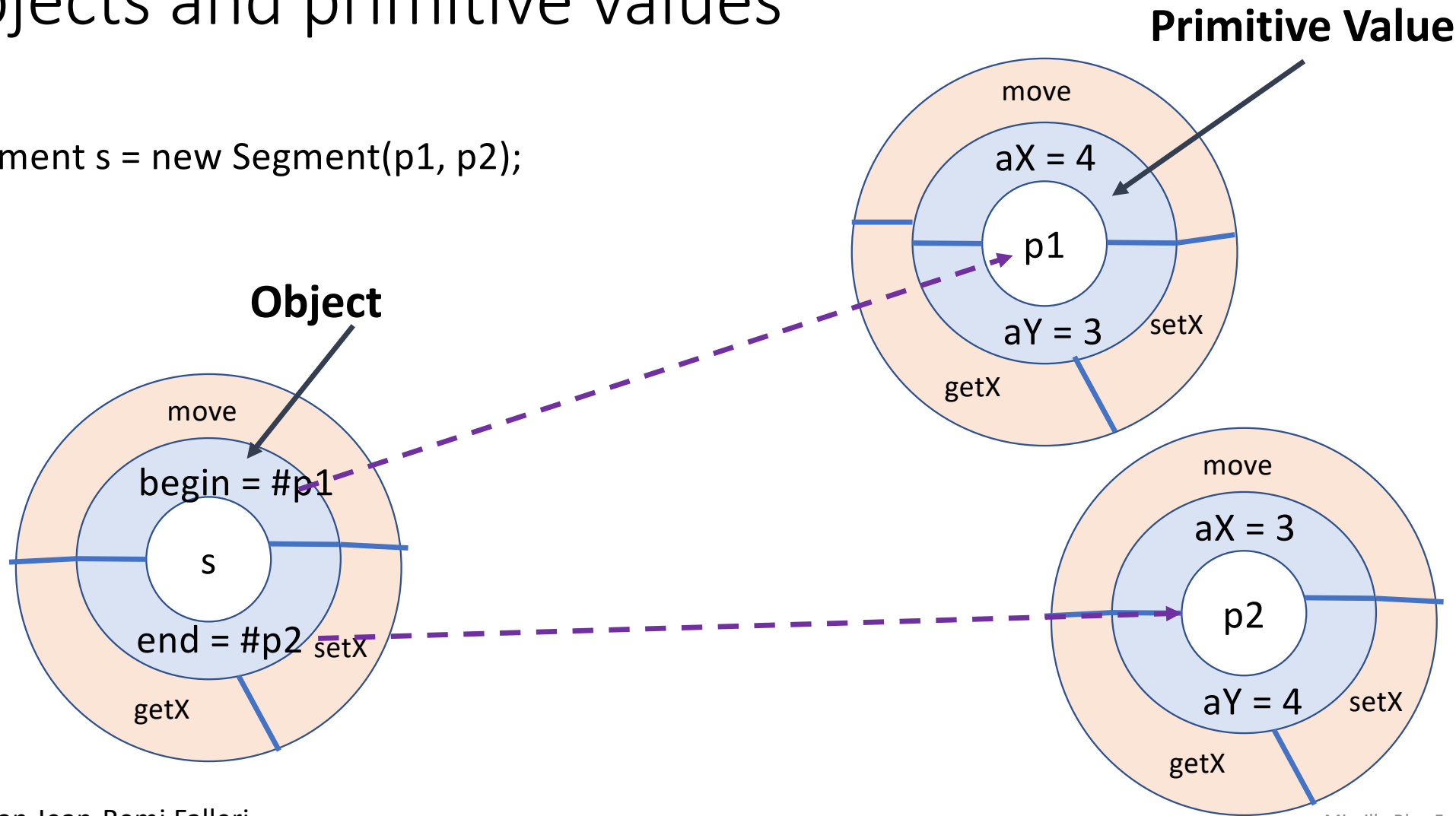


@Based on Jean-Remi Falleri

Mireille Blay-Fornarino

# Objects and primitive values

Segment s = new Segment(p1, p2);



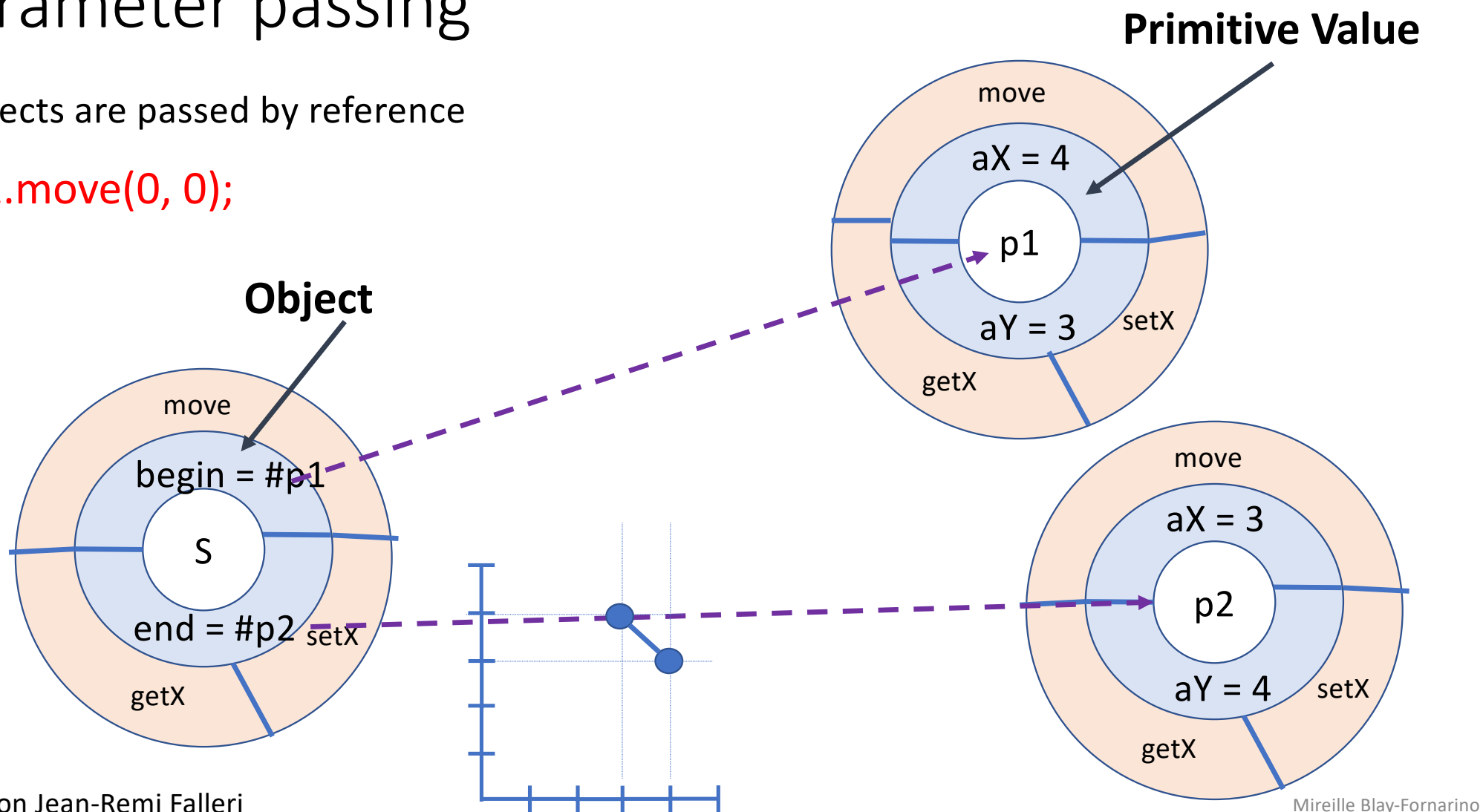


# Parameter passing

Primitive values are passed by copy

Objects are passed by reference

`p1.move(0, 0);`



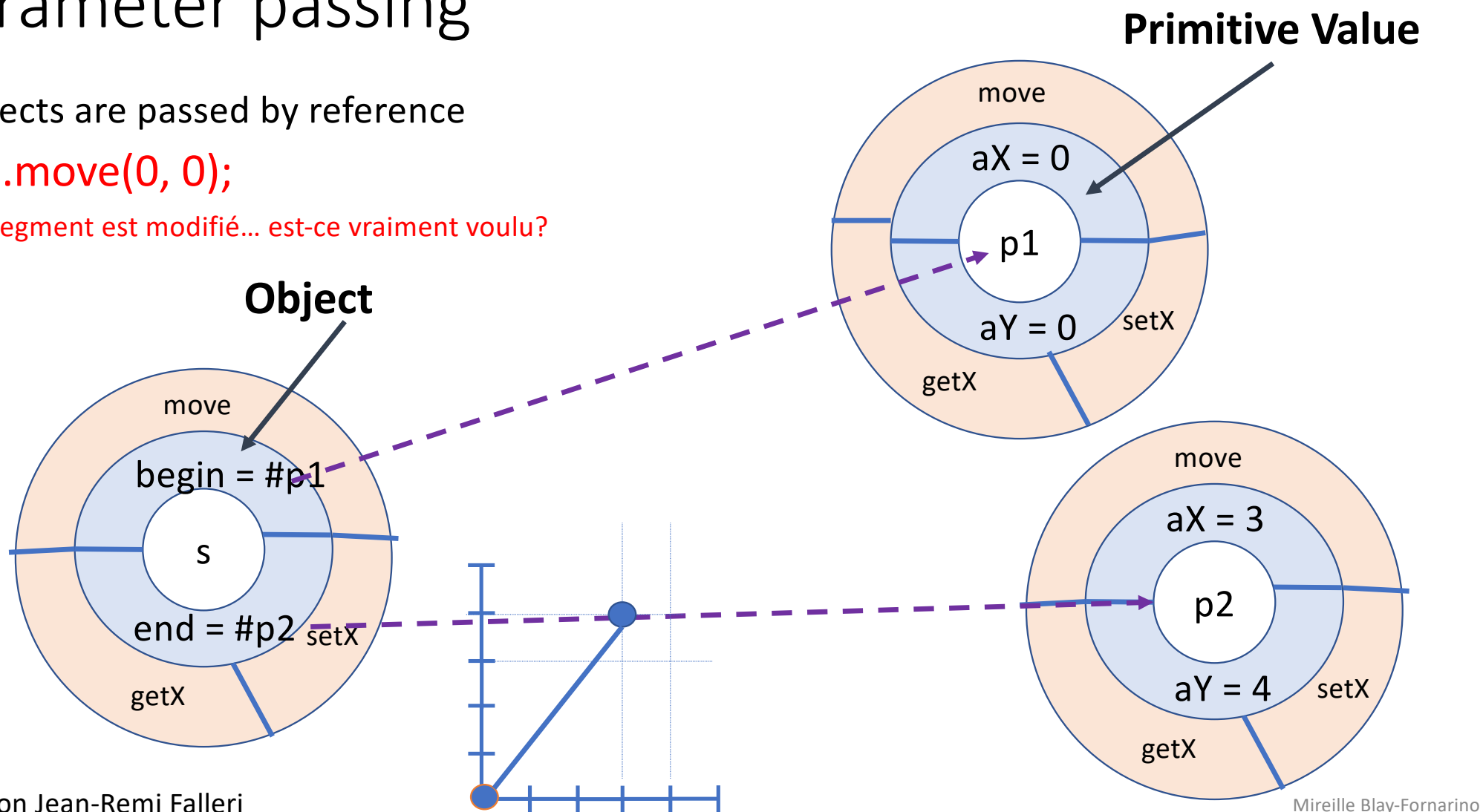
# Parameter passing

Primitive values are passed by copy

Objects are passed by reference

`p1.move(0, 0);`

Le segment est modifié... est-ce vraiment voulu?



# Passage de paramètres

Les objets sont passés par **référence**

```
void foo(Point p) {  
    p.move(0, 0);  
}
```

```
Point p = new Point(10, 10);  
foo(p); // après l'appel, p est à (0, 0)  
System.out.println(p);  
//p = Point{0,0}
```

Les valeurs primitives sont passées  
par **copie**

```
void bar(int i) {  
    i = 0; }
```

```
int i = 20;  
bar(i);  
System.out.println(i);  
//i = 20
```



# Apprenons au travers d'un exemple

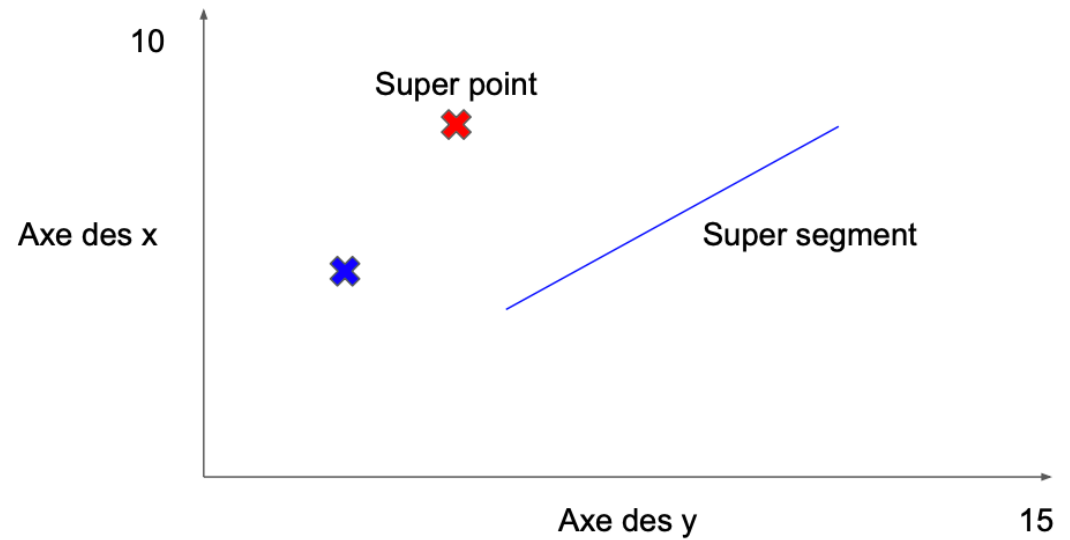
*Les éléments importants seront approfondis en TD et dans les cours suivants.*

D'après un exemple de JR Falléri. Les codes sont accessibles en ligne.

Mireille Blay-Fornarino

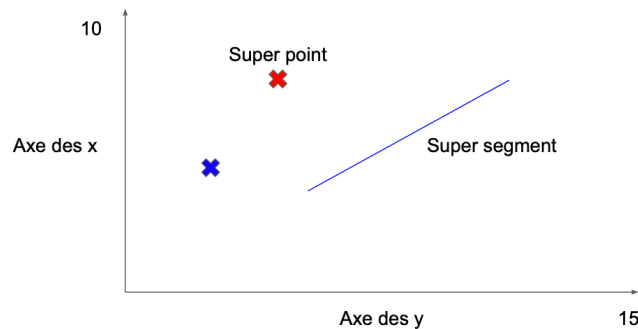
Nous souhaitons implémenter

- Spécifications



Nous souhaitons implémenter

- Spécifications plus précises (1)

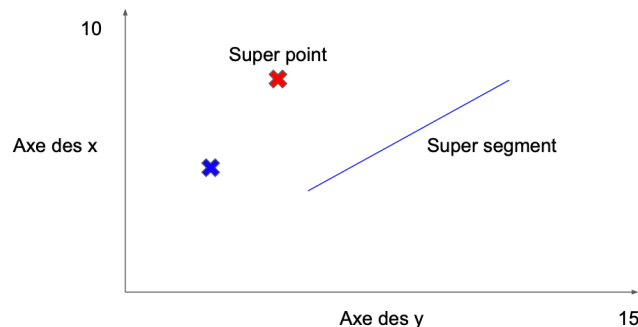


Tout objet de notre système doit pouvoir donner sa description :

- *Point* a pour description :  
Point (x,y) , couleur : (r,g,b), titre: titre
  - Exemple : *Point (3,5), color: (185,94,73), Title : A*
- *Segment* a pour description :  
Segment (x1,y2) -> (x2,y2), couleur : (r,g,b), titre: titre
  - Ex: *Segment (3,-2) -> (2,0), color: (0,0,0), titre : v1*
- *Axe* a pour description :  
Axis size : size, Title : title en majuscule
  - Exemple : *Axis size : 50, Title : AXE DES X*

Nous souhaitons implémenter

- Spécifications plus précises (2)



On veut pouvoir manipuler des ensembles d'éléments, c'est-à-dire des points et des segments :

- leur ajouter des points et des segments;
- obtenir la description de tous les éléments qu'ils contiennent;

*par exemple : un ensemble d'éléments*

1 : Point (3,-2), color: (231,193,148), Title : A

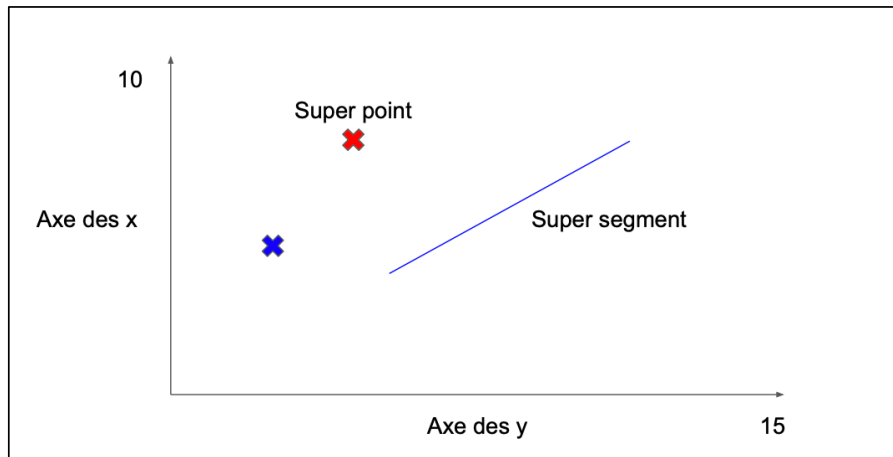
2 : Point (2,0), color: (213,171,93), Title : B

...

5 : Segment (2,0) -> (4,1), color: (0,0,0), titre : s2

6 : Segment (4,1) -> (3,-2), color: (0,0,0), titre : s3

- Retrouver des éléments de cet ensemble en fonction de son indice (ordre de création) : Le premier est à la position 0, le troisième à 2 etc.



Tout objet de notre système doit pouvoir donner sa description :

- Point a pour description : Point (x,y) , couleur : (r,g,b), titre: titre
  - *Exemple : Point (3,5), color: (185,94,73), Title : A*
- Segment a pour description : Segment (x1,y2) -> (x2,y2), couleur : (r,g,b), titre: titre
  - *Exemple : Segment (3,-2) -> (2,0), color: (0,0,0), titre : v1*
- Axe a pour description : Axis size : size, Title : title
  - *Exemple : Axis size : 50, Title : **AXE DES X***

On veut pouvoir manipuler des ensembles d'éléments, c'est-à-dire des points et des segments :

- leur ajouter des points et des segments;
- obtenir la description de tous les éléments qu'ils contiennent;

*par exemple :*

1 : Point (3,-2), color: (231,193,148), Title : A  
 2 : Point (2,0), color: (213,171,93), Title : B  
 ...  
 5 : Segment (2,0) -> (4,1), color: (0,0,0), titre : s2  
 6 : Segment (4,1) -> (3,-2), color: (0,0,0), titre : s3

# Identification des classes



# Identification des Concepts

## Point :

- les coordonnées d'un point sont nécessairement positives ou nulles.
- une couleur
- un titre

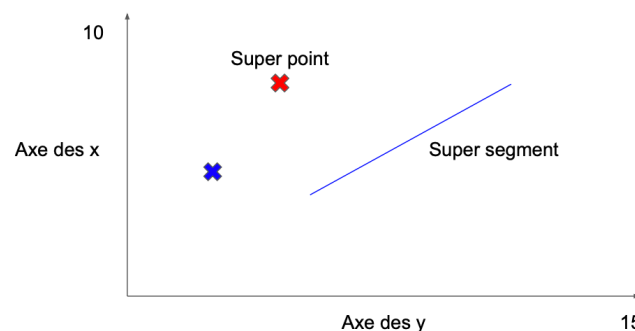
## Segment :

- un point de départ et d'arrivée
- une couleur
- un titre

**Couleur** : 3 composantes **r**, **g** et **b** comprises entre 0 et 255.

## Axe :

- Un titre en majuscule
- Une taille positive



## Ensemble des éléments du système :

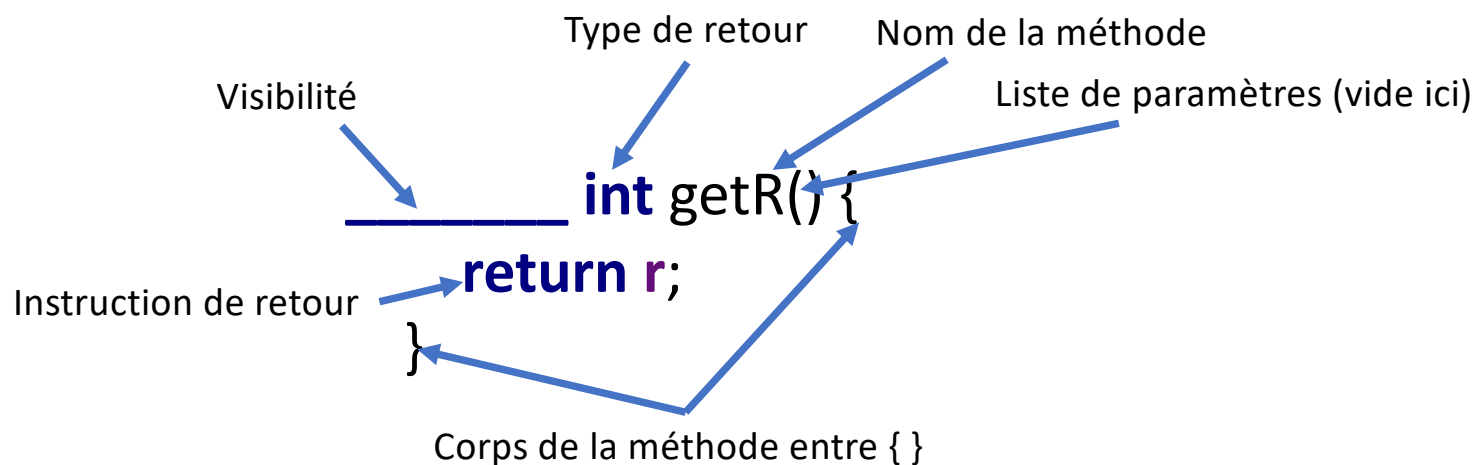
- Des points et des Segments
- On peut ajouter des éléments à cet ensemble et en retrouver un en fonction de son indice (ordre de création),  
Le premier est à la position 0, le troisième à 2 etc.

# Une classe Color

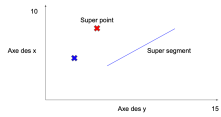
**Couleur** : 3 composantes **r**, **g** et **b**

```
class Color {  
    int r;  
    int g;  
    int b;  
  
    public int getR() {  
        return r;  
    }  
    ....
```

# Méthode Accesseur (get)



*On ne définit les accesseurs que pour les variables auxquelles on veut donner accès en lecture et on le fait avec précaution (cours futur)*



# Contrôler les valeurs des variables

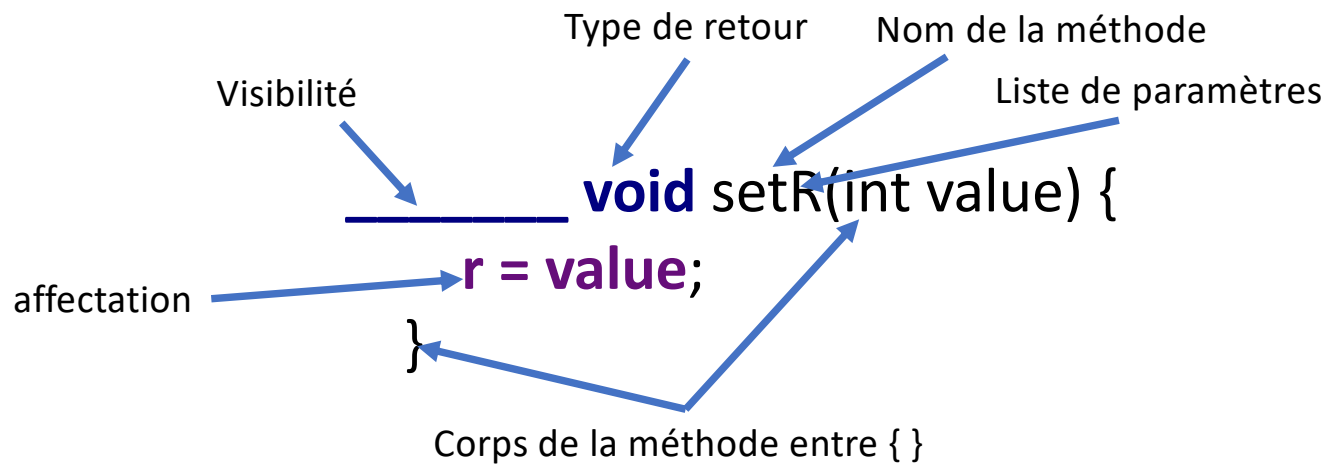
**Couleur** : 3 composantes r, g et b **comprises entre 0 et 255**.

Si une composante n'est pas comprise entre 0 et 255 alors il faut qu'elle soit ramenée dans cet intervalle (0 pour une valeur inférieure à 0 et 255 pour une valeur supérieure à 255).

```
class Color {
    int r;          ....
    int g;
    int b;

    int getR() {
        return r;
    }
    void setR(int value) {
        r = value;
    }
}
```

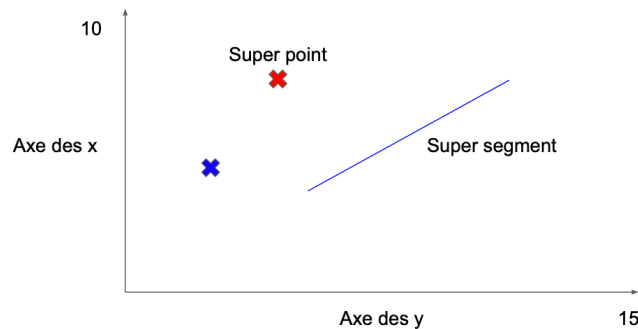
# Méthode Mutator (set)



*On définit les mutators que pour les variables auxquelles on veut donner accès en écriture, en contrôlant, et on le fait avec précaution (cours futur)*

Nous souhaitons implémenter

- Evolution des specifications (1)



**Couleur** : 3 composantes r, g et b comprises entre 0 et 255.

Si une composante n'est pas comprise entre 0 et 255 alors il faut quelle soit ramenée dans cet intervalle (0 pour une valeur inférieure à 0 et 255 pour une valeur supérieure à 255).



```
Color c = new Color();  
c.setR(-2);
```

# Condition

# Contrôler les valeurs des variables

**Couleur** : 3 composantes **r**, **g** et **b** comprises entre 0 et 255.

Si une composante n'est pas comprise entre 0 et 255 alors il faut quelle soit ramenée dans cet intervalle (0 pour une valeur inférieure à 0 et 255 pour une valeur supérieure à 255).

```
class Color {
```

```
    int r;  
    int g;  
    int b;
```

```
    public int getR() {  
        return r;  
    }
```

```
    .....
```

```
    void setR(int r) {  
        this.r = backToTheDomain(r);  
    }
```

```
int backToTheDomain(int x) {  
    if (x < 0) {  
        return 0;  
    }  
    if (x > 255) {  
        return 255;  
    }  
    return x;  
}
```



# Contrôler les valeurs des variables

**Couleur** : 3 composantes **r**, **g** et **b** comprises entre 0 et 255.

Si une composante n'est pas comprise entre 0 et 255 alors il faut quelle soit ramenée dans cet intervalle (0 pour une valeur inférieure à 0 et 255 pour une valeur supérieure à 255).

```

54
55
56
57
58
59
60
61
62
}

int backToTheDomain(int x) {
    if (x < 0) {
        return 0;
    }
    if (x > 255) {
        return 255;
    }
    return x;
}

```

(54) Une méthode qui renvoie un int +

(55, 57) Utilisation d'une condition

If (Condition) { FAIRE }

(56, 59, 61) Le **return** nous fait sortir de la méthode

- Ex : Un appel avec x = -2  
⇒ Return 0
- Ex : Un appel avec x = 6
  - x n'est pas inférieur à 0
  - x n'est pas supérieur à 255
  - c'est x qui est renvoyé

# Contrôler les valeurs des variables

**Couleur** : 3 composantes **r**, **g** et **b** comprises entre 0 et 255.

Si une composante n'est pas comprise entre 0 et 255 alors il faut quelle soit ramenée dans cet intervalle (0 pour une valeur inférieure à 0 et 255 pour une valeur supérieure à 255).

```
64 void setR(int r) {  
65     this.r = backToTheDomain(r);  
66 }  
67
```

(64) Protection de l'attribut **r** par son mutator qui ramène la valeur dans l'espace attendu.

(65) Appel à la méthode *backToTheDomain*

# Contrôler les valeurs des variables, y compris à la construction des objets

**Couleur** : 3 composantes **r**, **g** et **b** comprises entre 0 et 255.

Si une composante n'est pas comprise entre 0 et 255 alors il faut qu'elle soit ramenée dans cet intervalle (0 pour une valeur inférieure à 0 et 255 pour une valeur supérieure à 255).

```
18  Color() {  
19      this( r: 0, g: 0, b: 0);  
20  }  
    8 usages  
21  Color(int r, int g, int b) {  
22      setR(r);  
23      setG(g);  
24      setB(b);  
25  }
```

(18, 21) Constructor (même nom que la classe)

(19) Appel à l'autre constructeur (Cascade)

(22, 23, 24) Protection des composantes r , g et b par appel au mutator.

*Contrôler les valeurs  
contenus dans les  
attributs par les  
mutators et les  
constructeurs.*

Comme les objets sont passés par  
référence, ce n'est pas toujours suffisant.  
De même la visibilité donnée sur vos  
attributs devra aussi être contrôlée.

## Condition pour contrôler les valeurs des variables

# Encapsulation et Accesseurs & Mutators

- Ne perdez pas le contrôle
  - Les outils permettent de les générer, mais il est de votre responsabilité
    - a) de le faire ou non et
    - b) de leur donner la visibilité adaptée à votre problème!

# Concaténation de String

# Description d'une instance de Color

**Couleur** : Toute instance de Color doit pouvoir donner sa description comme suit : *color: (r,g,b) par exemple color: (45,231,8)*

```
class Color {  
  
    int r;  
    int g;  
    int b;  
  
    public int getR() {  
        return r;  
    }  
    ....  
    String description() {  
        return "color: (" + this.getR() + "," + this.getG() + "," + this.getB() + ")";  
    }  
}
```

# Description d'une instance de Color

**Couleur** : 3 composantes **r**, **g** et **b** comprises entre 0 et 255.

Si une composante n'est pas comprise entre 0 et 255 alors il faut qu'elle soit ramenée dans cet intervalle (0 pour une valeur inférieure à 0 et 255 pour une valeur supérieure à 255).

```
class Color {
```

```
    int r;
```

```
    int g;
```

```
    int b;
```

```
    public int getR() {
```

```
        return r;
```

```
    }
```

```
    ....
```

```
    String description() {
```

```
        return "color: (" + this.getR() + "," + this.getG() + "," + this.getB() + ")";
```

```
    }
```

Concaténation de String : +

Conversion automatique  
d'un entier en String



# Héritage par l'exemple

# Nous souhaitons modéliser...

## Quels sont les éléments communs?

### Point :

- les coordonnées d'un point sont nécessairement positives ou nulles.

- **une couleur**
- **un titre**

### Segment :

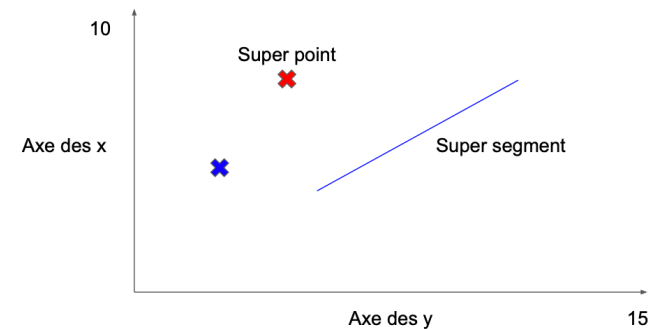
- un point de départ et d'arrivée

- **une couleur**
- **un titre**

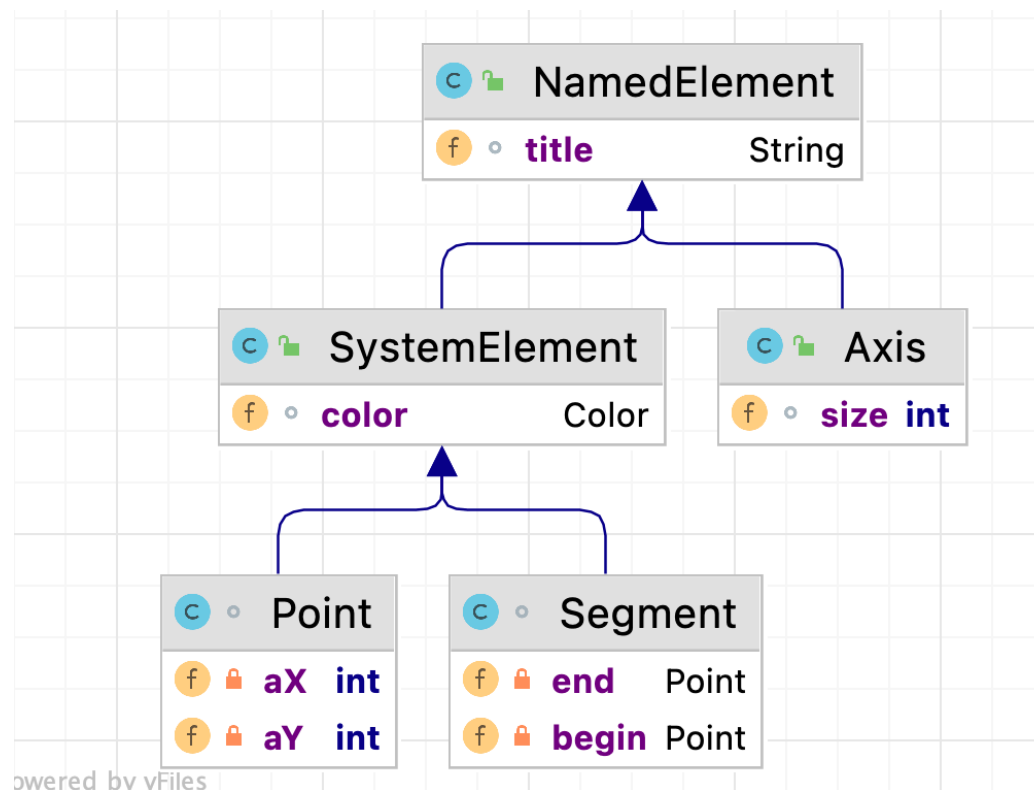
**Couleur** : 3 composantes **r**, **g** et **b** comprises entre 0 et 255.

### Axe :

- **un titre** en majuscule
- Une taille positive



# Héritage – Est-un

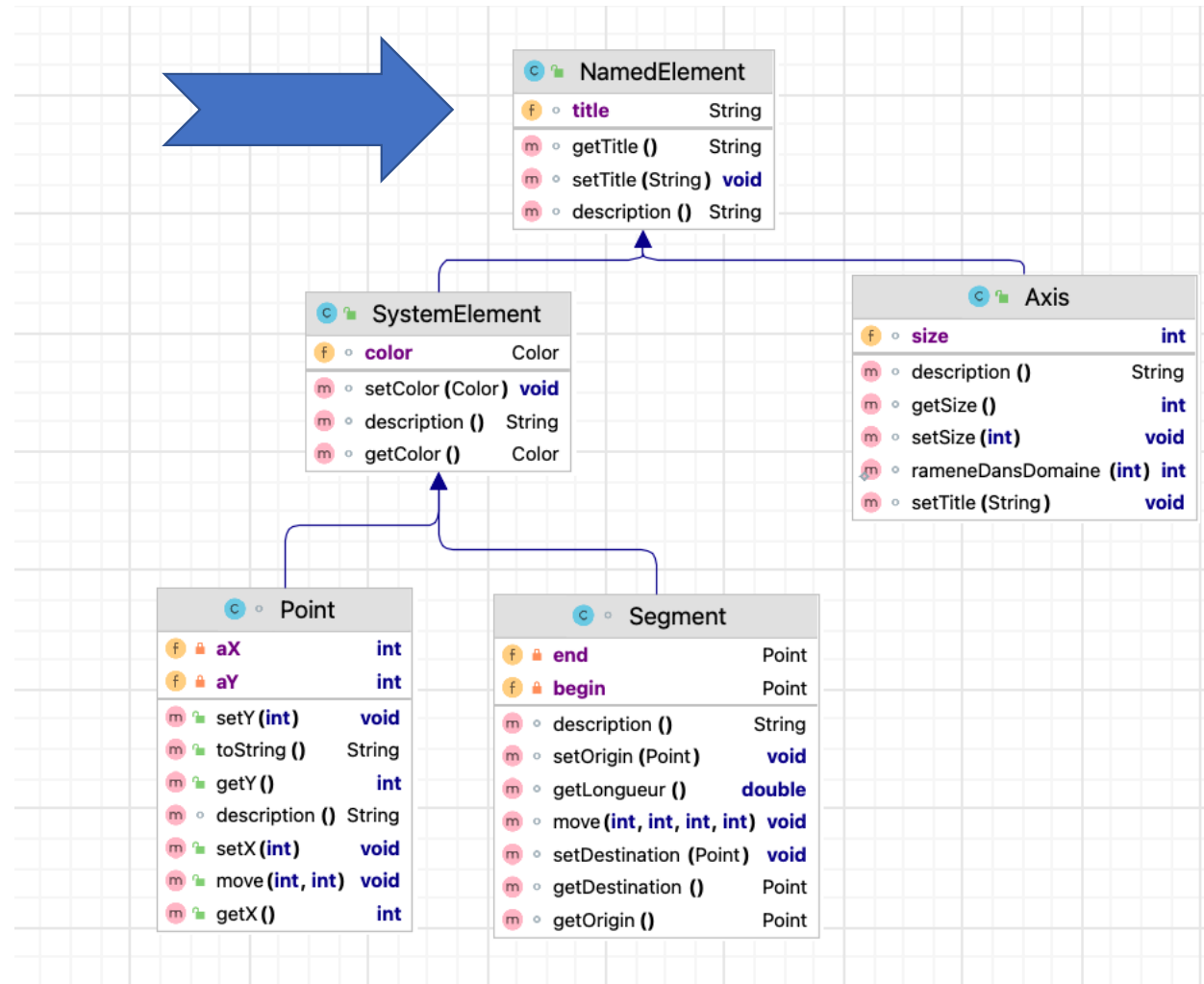


powered by yFiles

Généré dans IntelliJ

Mireille Blay-Fornarino

# Héritage – Est-un



Généré dans IntelliJ

Mireille Blay-Fornarino

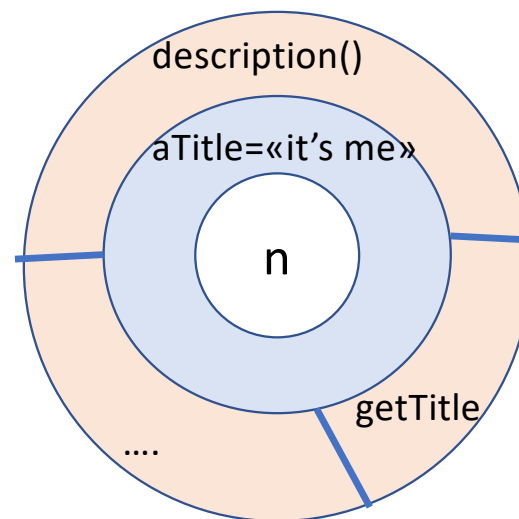
# Des éléments nommés en « Racine »

```
1 package cours;
2
3 public class NamedElement {
4     private String aTitle;
5     NamedElement(String titre) {
6         this.aTitle = titre;
7     }
8     String getTitle() {
9         return this.aTitle;
10    }
11    void setTitle(String title) {
12        this.aTitle = title;
13    }
14    String description() {
15        return "Title : " + this.getTitle();
16    }
17 }
18
19
```

# Créer une instance de NamedElement

```
1 package cours;
2
3 public class NamedElement {
4     private String aTitle;
5     NamedElement(String titre) {
6         this.aTitle = titre;
7     }
8     String getTitle() {
9         return this.aTitle;
10    }
11    void setTitle(String title) {
12        this.aTitle = title;
13    }
14    String description() {
15        return "Title : " + this.getTitle();
16    }
17 }
18
19
```

NamedElement n = new NamedElement(« It's me »);



# Tout *NamedElement* peut nous donner sa description

Tout objet de notre système doit pouvoir être affiché comme suit :

- Point affiche : Point (x,y), couleur : (r,g,b), **titre: titre**
- Segment affiche : Segment (x1,y2) -> (x2,y2), couleur : (r,g,b), **titre: titre**
- Axe affiche : Axe taille : taille, **titre : titre**

```

1  package cours;
2
3  public class NamedElement {
4      private String aTitle;
5      NamedElement(String titre) {
6          this.aTitle = titre;
7      }
8      String getTitle() {
9          return this.aTitle;
10     }
11     void setTitle(String title) {
12         this.aTitle = title;
13     }
14     String description() {
15         return "Title : " + this.getTitle();
16     }
17 }
18

```



... Préparation  
de la partie de  
description  
commune

# Retourner une string n'est pas l'afficher !

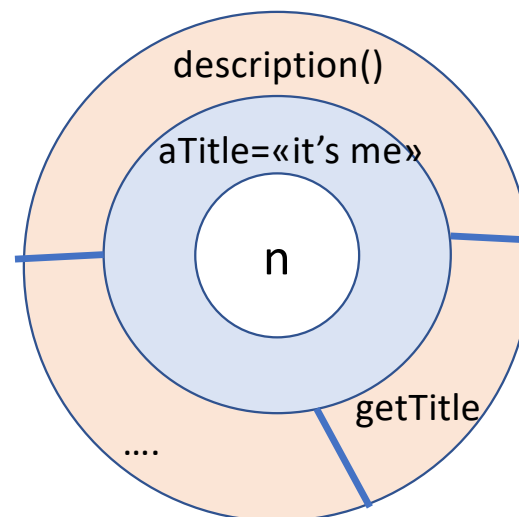
```

1 package cours;
2
3 public class NamedElement {
4     private String aTitle;
5     NamedElement(String titre) {
6         this.aTitle = titre;
7     }
8     String getTitle() {
9         return this.aTitle;
10    }
11    void setTitle(String title)
12        this.aTitle = title;
13    }
14    String description() {
15        return "Title : " + this.getTitle();
16    }
17 }
18
19

```

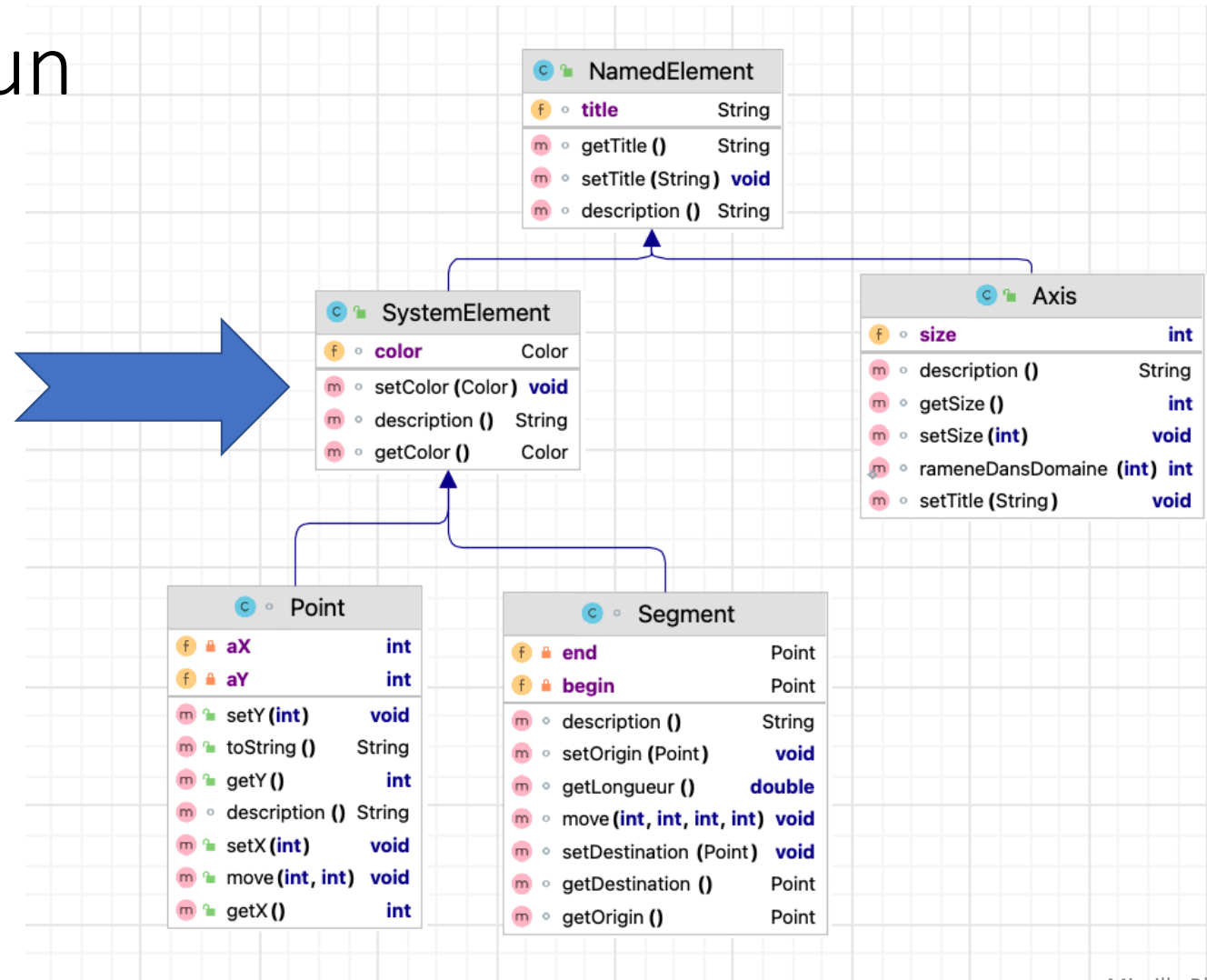
NamedElement n = **new** NamedElement("It's me");  
 System.**out**.println(n.description());

*Title : It's me*





# Héritage – Est-un



Généré dans IntelliJ

Un élément du système **est un** élément nommé qui a **en plus** une couleur

```
1  package cours;
2
3  public class SystemElement extends NamedElement {
4      private Color color;
5      SystemElement(String titre, Color color) {
6          super(titre);
7          this.color = color;
8      }
9      Color getColor() {
10         return this.color;
11     }
12     void setColor(Color color) {
13         this.color = color;
14     }
15     String description() {
16         return this.getColor().description() + ", " + super.description();
17     }
18 }
```

# Un élément du système **est un** élément nommé : **extends**

```
1  package cours;
2
3  public class SystemElement extends NamedElement {
4      private Color color;
5      SystemElement(String titre, Color color) {
6          super(titre);
7          this.color = color;
8      }
9      Color getColor() {
10         return this.color;
11     }
12     void setColor(Color color) {
13         this.color = color;
14     }
15     String description() {
16         return this.getColor().description() + ", " + super.description();
17     }
18 }
```

**Extends exprime l'héritage (est-un).**

Toute instance de *SystemElement* est aussi une instance de *NamedElement*

# Un élément du système **est un** élément nommé : **extends**

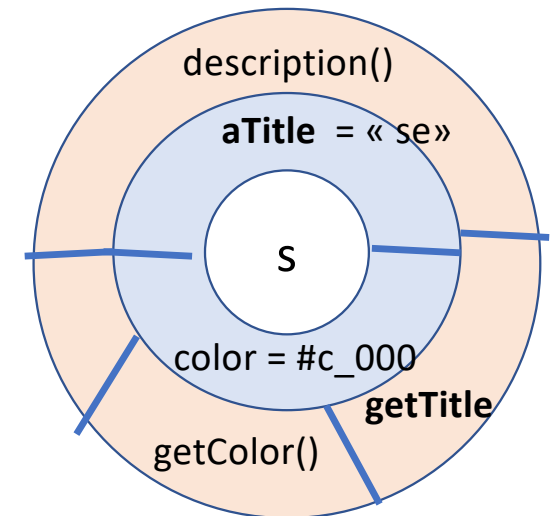
SystemElement s = **new** SystemElement("se", **new** Color());

*Héritage des attributs et des méthodes (pas des constructeurs)*

```

1  package cours;
2
3  public class SystemElement extends NamedElement {
4      private Color color;
5      SystemElement(String titre, Color color) {
6          super(titre);
7          this.color = color;
8      }
9      Color getColor() {
10         return this.color;
11     }
12     void setColor(Color color) {
13         this.color = color;
14     }
15     String description() {
16         return this.getColor().description() + ", " + super.description();
17     }
18 }

```



## Un élément du système **est un** élément nommé : **extends**

SystemElement s = **new** SystemElement("se", **new** Color());

```
1 package cours;
2
3 public class SystemElement extends NamedElement {
4     private Color color;
5     SystemElement(String titre, Color color) {
6         super(titre);
7         this.color = color;
8     }
```

(6) Super(...) appel le constructeur de la classe héritée donc ici NamedElement qui affecte le titre.

(7) Affectation de la couleur

*Super fait référence à une classe dont on hérite.*

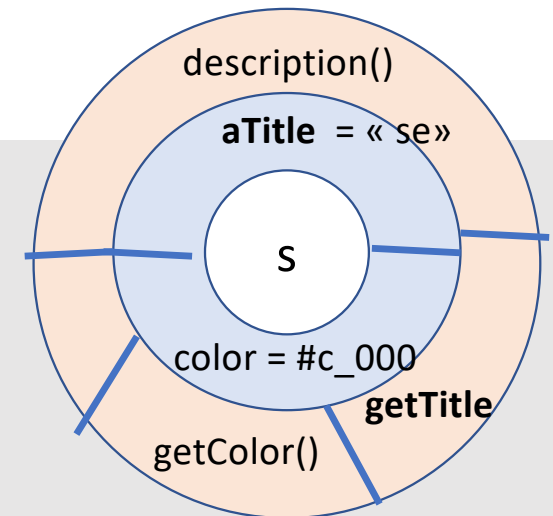
# Un élément du système **est un** élément nommé : **extends**

SystemElement s = **new** SystemElement("se", **new** Color());

```
1 package cours;
2
3 public class SystemElement extends NamedElement {
4     private Color color;
5     SystemElement(String titre, Color color) {
6         super(titre);
7         this.color = color;
8     }
9 }
```

(6) Super(...) appel le constructeur de la classe héritée donc ici NamedElement qui affecte le titre.

(7) Affectation de la couleur



Un élément du système **est un** élément nommé  
qui a **en plus** une couleur : **override**

*override : une méthode  
précédemment définie est redéfinie.*

```
SystemElement s =
    new SystemElement("se", new Color());

System.out.println(s.description());
```

=> color: (0,0,0), Title : se

```

14     }
15     String description() {
16         return this.getColor().description() + ", " + super.description();
17     }
18 }
```

Une instance d'une classe est aussi une instance de ces « super-classes »

*B extends A*

*Toute instance de B est aussi une instance de A*

```
NamedElement ne = new SystemElement("se1", new Color());  
System.out.println(ne.description());
```

*=> color: (0,0,0), Title : se1*

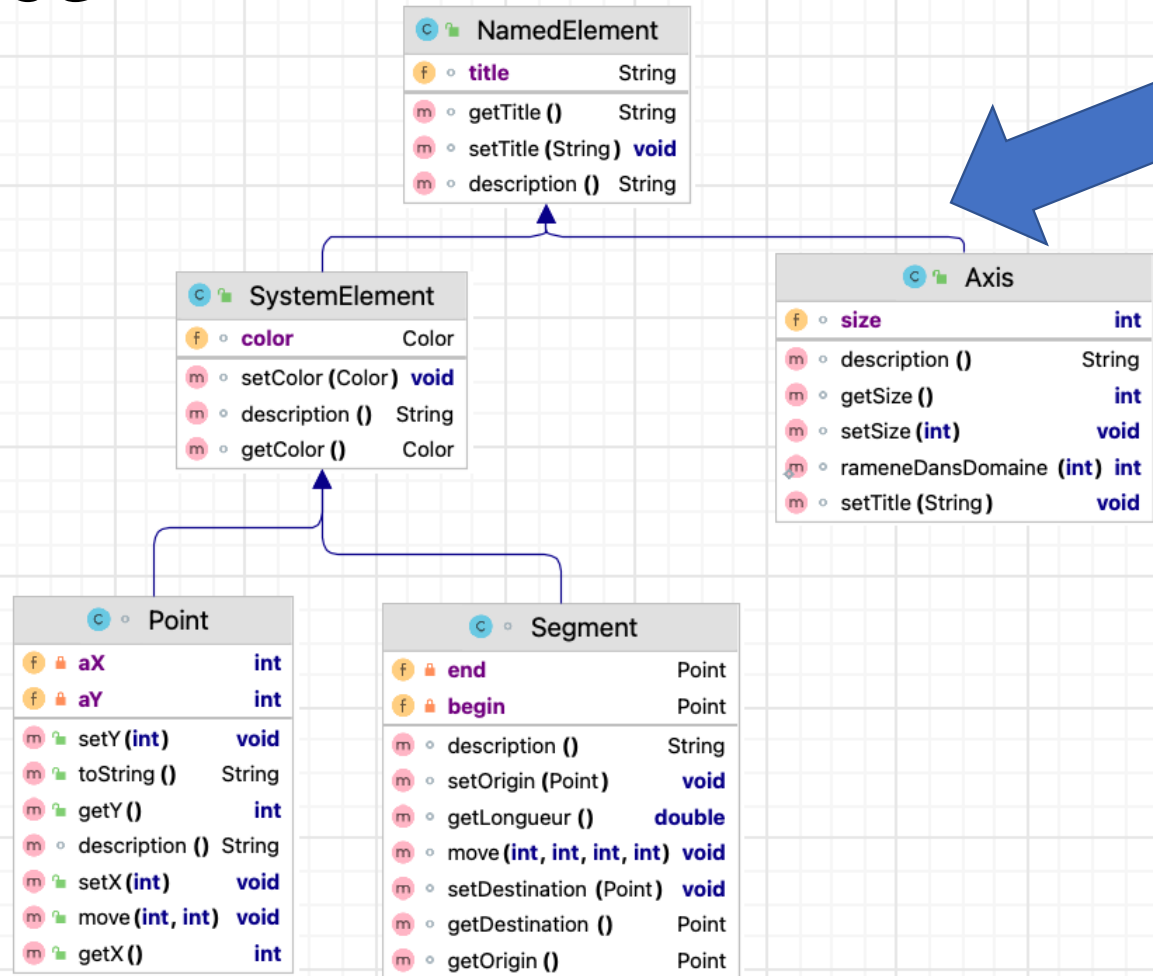


## Utilisation de l'héritage

- définir une superclasse : *namedElement*
- définir des sous-classes (**extends**): *SystemElement*, *Axis*, ..
- la superclasse définit les attributs et méthodes communs
- les sous-classes
  - héritent de la superclasse ses attributs et méthodes
  - ajoutent des attributs et méthodes
  - redéfinissent (**override**) des méthodes et peuvent faire
- référencer les méthodes ou accesseurs de la super-classe par **super**.

# Héritage

# Une autre sous-classe de NamedElement

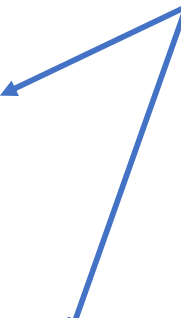


Généré dans IntelliJ

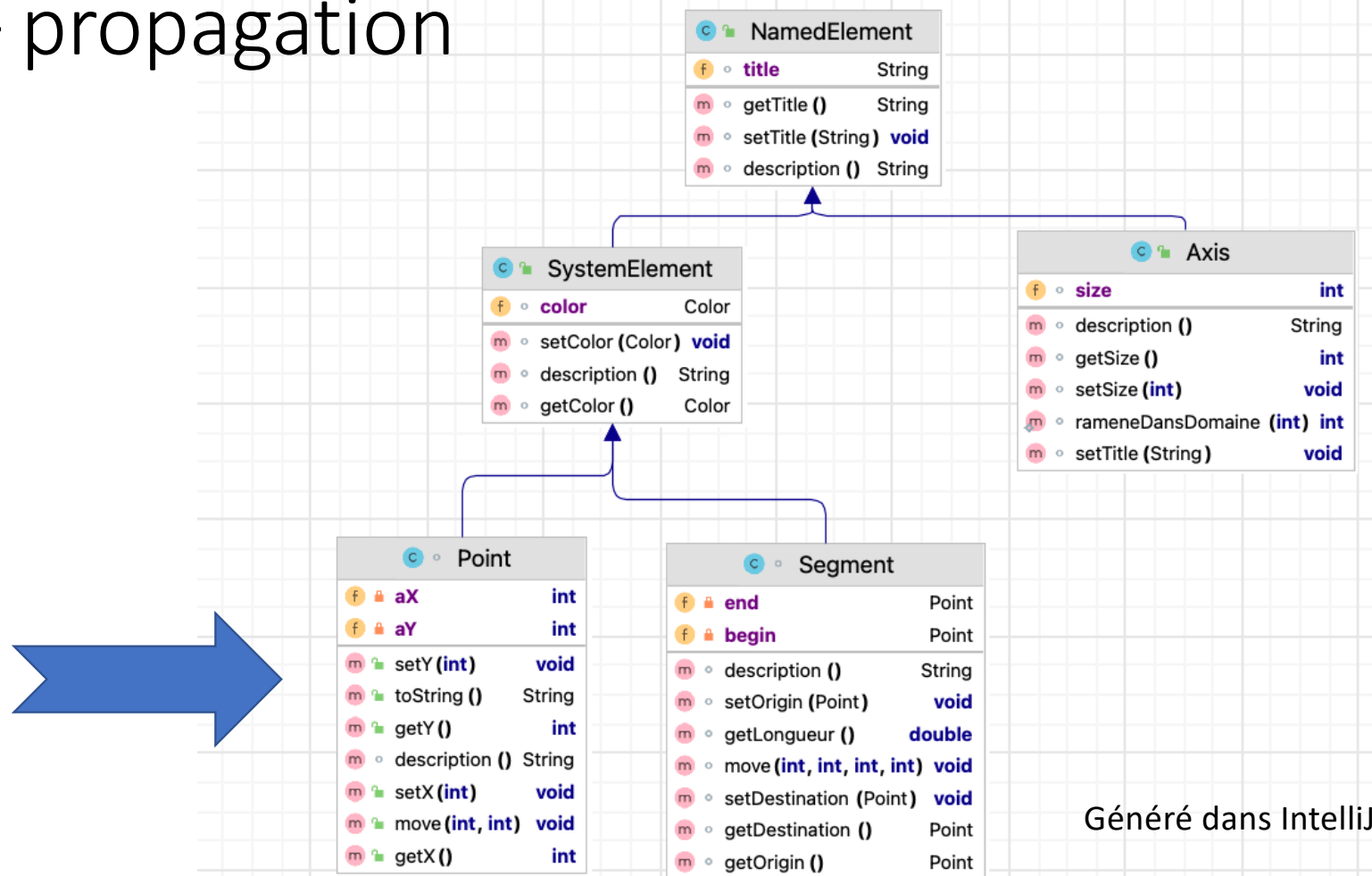
# Héritage – extends, surcharge, ...

```
public class Axis extends NamedElement {
6     int size;
7     Axis() {
8         this("AXIS", 0);
9     }
10    Axis(String title, int size) {
11        super(title.toUpperCase());
12        setSize(size);
13    }
14    @Override
15    void setTitle(String title) {
16        super.setTitle(title.toUpperCase());
17    }
18
19    @Override
20    String description() {
21        return "Axis size : " + this.getSize() + ", " + super.description();
22    }
}
```

La chaîne de caractères reçoit l'ordre de passer en majuscules



# Héritage – propagation



Généré dans IntelliJ

Un Point est un élément du système etc.

```
class Point extends SystemElement{
2     private int aX;
3     private int aY;
4     Point() {
5         this(0,0);
6     }
7     Point(int pX, int pY) {
8         this("point",Color.black(),pX,pY);
9     }
10    Point(String title, Color color, int x, int y) {
11        super(title, color);
12        setX(x);
13        setY(y);
14    }
15    String description() {
16        return "Point (" + this.getX() + "," + this.getY() + "), "
17            + super.description();
18    }
}
```

# Un Point est un élément du système : cascade de constructeurs

```
class Point extends SystemElement{
2   private int aX;
3   private int aY;
4   Point() {
5       this(0,0);
6   }
7   Point(int pX, int pY) {
8       this("point",Color.black(),pX,pY);
9   }
10  Point(String title, Color color, int x, int y) {
11      super(title, color);
12      setX(x);
13      setY(y);
14  }
15  String description() {
16      return "Point (" + this.getX() + "," + this.getY() + "), "
17          + super.description();
18  }
```

The diagram illustrates the cascade of constructors in the Point class. A blue arrow points from the `this(0,0);` call in the no-argument constructor (line 5) to the `this("point",Color.black(),pX,pY);` call in the two-argument constructor (line 8). Another blue arrow points from the `super(title, color);` call in the three-argument constructor (line 11) to the `super` call in the `description()` method (line 16). A third blue arrow points from the `super` call in the `description()` method (line 16) to the `super` call in the three-argument constructor (line 11).

# Un Point est un élément du système : héritage en action

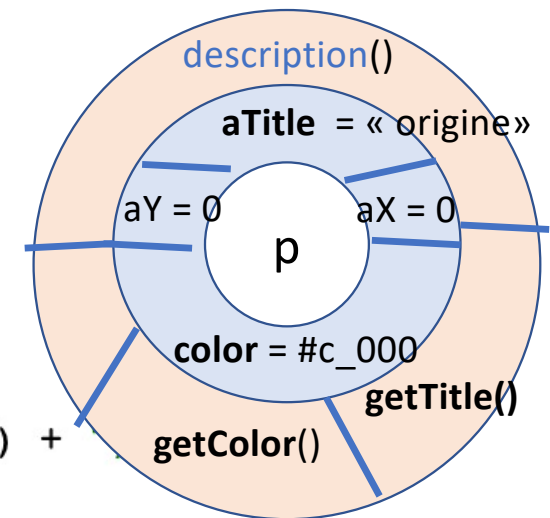
```

class Point extends SystemElement{
2     private int aX;
3     private int aY;
4     Point() {
5         this(0,0);
6     }
7     Point(int pX, int pY) {
8         this("point",Color.black(),pX,pY);
9     }
10    Point(String title, Color color, int x, int y) {
11        super(title, color);
12        setX(x);
13        setY(y);
14    }
15    String description() {
16        return "Point (" + this.getX() + "," + this.getY() +
17               + super.description();
18    }

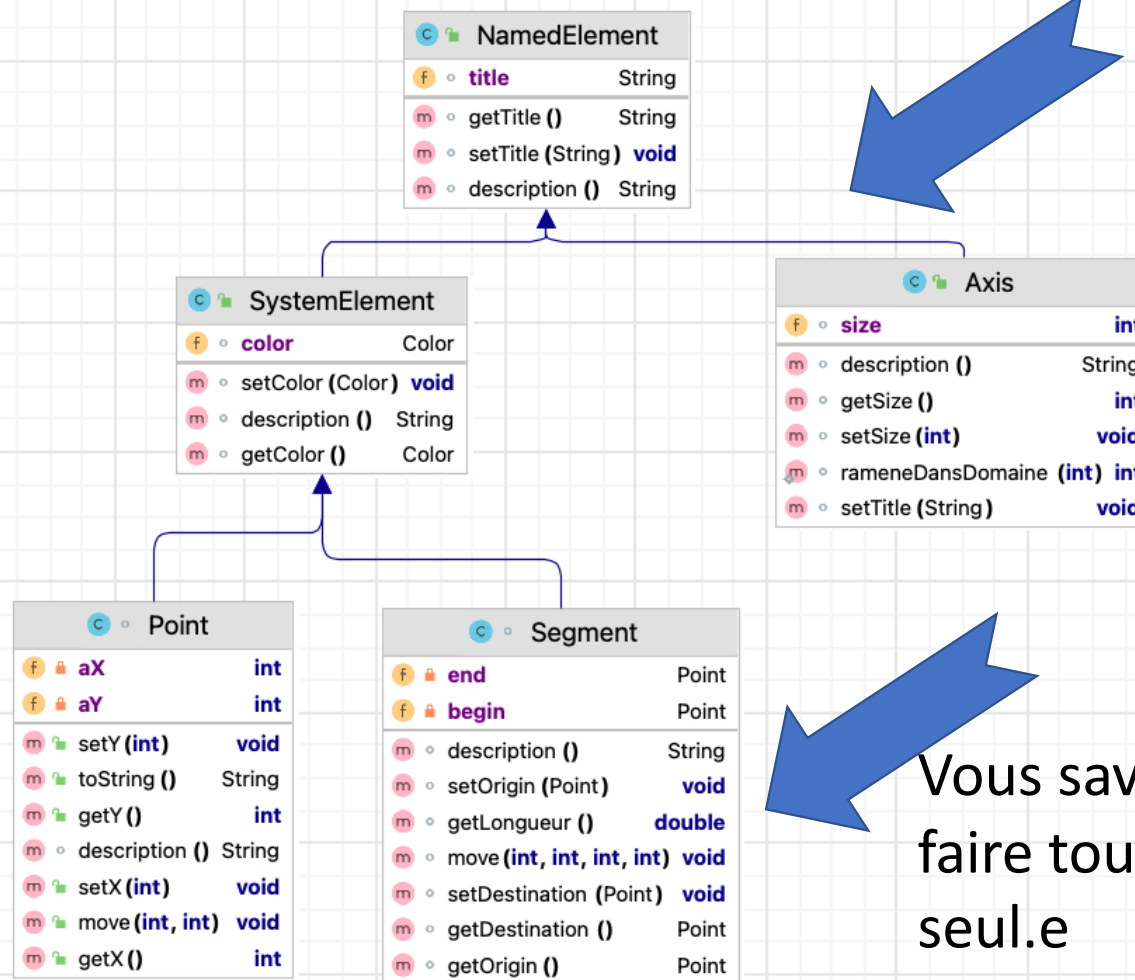
```

Point p = **new** Point("origine", **new** Color(), 0, 0);  
System.**out**.println(p.description());

=> Point (0,0), color: (0,0,0), Title : origine



# Héritage – Est-un



Généré dans IntelliJ



# Variables et méthodes de classe : « Static »

# Accès à une méthode depuis une classe

```
class Point extends SystemElement{
2     private int aX;
3     private int aY;
4     Point() {
5         this(0,0);
6     }
7     Point(int pX, int pY) {
8         this("point",Color.black(),pX,pY);
9     }
10    Point(String title, Color color, int x, int y) {
11        super(title, color);
12        setX(x);
13        setY(y);
14    }
15    String description() {
16        return "Point (" + this.getX() + "," + this.getY() + "), "
17            + super.description();
18    }
}
```

Color.black() ?

# Méthode de classe : Méthode statique (static)

```
class Color {  
3     int r;  
4     int g;  
5     int b;  
6     Color(int r, int g, int b) {  
7         setR(r);  
8         setG(g);  
9         setB(b);  
10    }  
11    static Color black() {  
12        return new Color(0, 0, 0);  
13    }  
14    static Color white() {  
15        return new Color(255, 255, 255);  
16    }  
17    static Color red() {  
18        return new Color(255, 0, 0);  
19    }  
}
```

**Appels :**  
Color.black();  
Color.white();

*Static : mot clef pour  
définir des méthodes de  
classe.*

On peut faire mieux en utilisant des constantes...  
A voir plus loin !!

## *Nous souhaiterions aussi*

Nous aimerions pouvoir non pas affecter une valeur par défaut (donc le noir dans notre exemple) , mais une couleur qui serait choisie au hasard.

Par exemple :

```
Point(int pX, int pY) {  
    this("point",Color.atRandom(),pX,pY);  
}
```

## *Une couleur au hasard ! La classe Random*

### **Color.atRandom();**



On s'adresse à la classe : méthode statique



Elle doit générer un nombre aléatoire pour chacune des composantes r, g et b

Regardons du côté de la classe **Random** pour générer un nombre aléatoire compris en 0 et 255

# La javadoc de Random

Accès à la javadoc (dans google : javadoc Random)

The screenshot shows the javadoc page for the `Random` class in the Java Platform Standard Ed. 8. The page includes navigation tabs (OVERVIEW, PACKAGE, CLASS, USE, TREE, DEPRECATED, INDEX, HELP) and a sub-navigation bar (PREV CLASS, NEXT CLASS, FRAMES, NO FRAMES, ALL CLASSES). The main content area displays the class hierarchy (java.lang.Object, java.util.Random), implemented interfaces (Serializable), and direct known subclasses (SecureRandom, ThreadLocalRandom). The source code snippet shows the class declaration: `public class Random extends Object implements Serializable`. The description states: "An instance of this class is used to generate a stream of pseudorandom numbers. The class uses a 48-bit seed, which is modified using a linear congruential formula. (See Donald Knuth, *The Art of Computer Programming, Volume 2*, Section 3.2.1.)" and "If two instances of Random are created with the same seed, and the same sequence of method calls is made for each, they will generate and return identical sequences".

int

[`nextInt`](#)(int bound) Returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.

# Utilisation de Random

```
static Color atRandom() {  
    Random generator = new Random();  
    return  
        new Color(    generator.nextInt(256),  
                    generator.nextInt(256),  
                    generator.nextInt(256));  
}
```

int

[nextInt](#)(int bound) Returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.

# Des Constantes (static final)

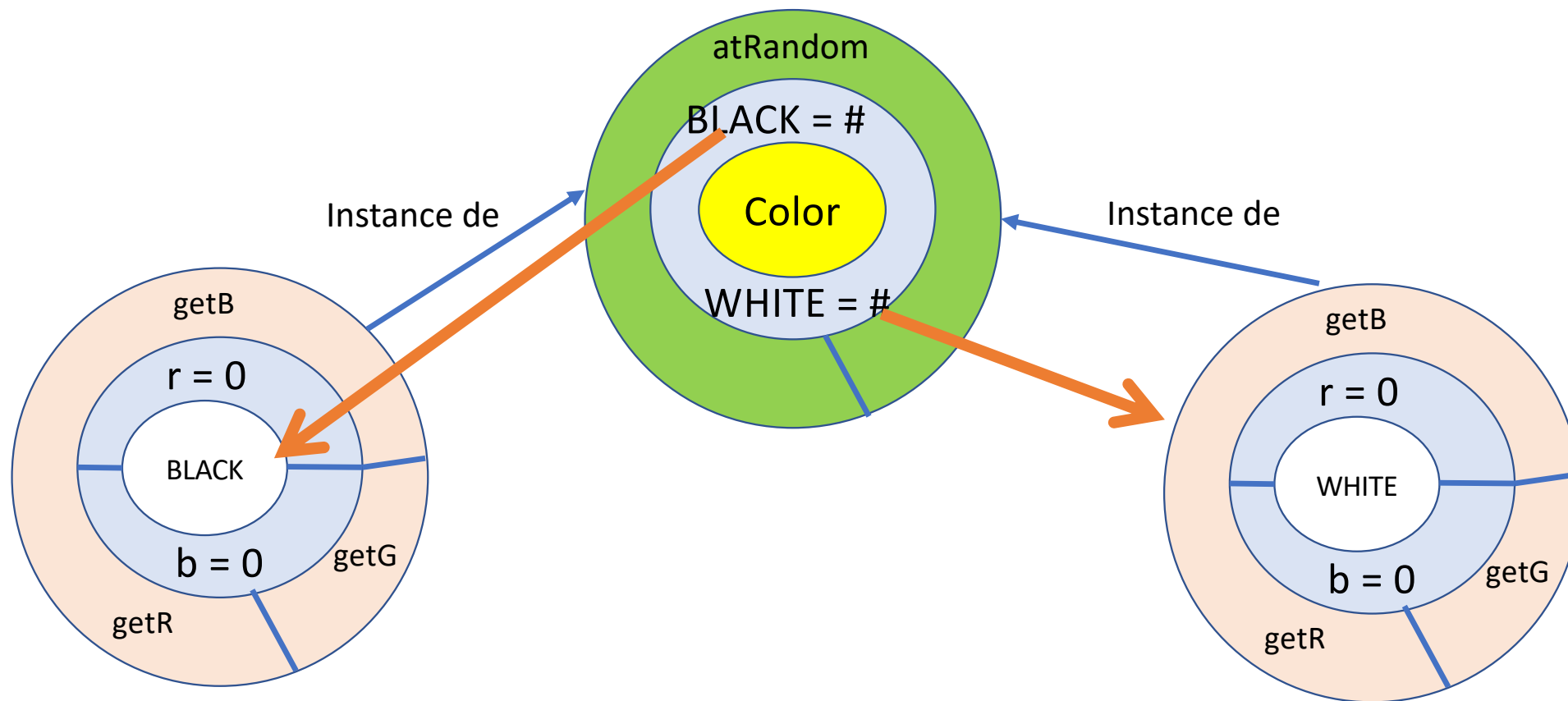
A chaque appel à black, white ou red, nous créons un nouvel objet.....  
Pourtant il s'agit toujours de la même couleur !

Définissons ces couleurs comme des constantes, puis utilisons les :

```
class Color {  
5  
6    static final Color BLACK = new Color(0,0,0);  
7    static final Color WHITE = new Color(255,255,255);  
8  
9    static Color black() {  
10        return BLACK;  
11    }  
12    static Color white() {  
13        return WHITE);  
14    }
```



# Classes, Objects, and static properties



# Des Constantes (static final)

Autre constante:

```
70
71     static final int MIN_VALUE = 0;
72     static final int MAX_VALUE = 255;
73     int backToTheDomain(int x) {
74         if (x < MIN_VALUE) {
75             return MIN_VALUE;
76         }
77         if (x > MAX_VALUE) {
78             return MAX_VALUE;
79         }
80         return x;
81     }
```

Les variables **constantes** sont introduites par **static final**. Elles doivent obligatoirement être initialisées lors de leur déclaration.

Par convention, leur nom est toujours en majuscules, chaque mot est séparé par un underscore '\_'.

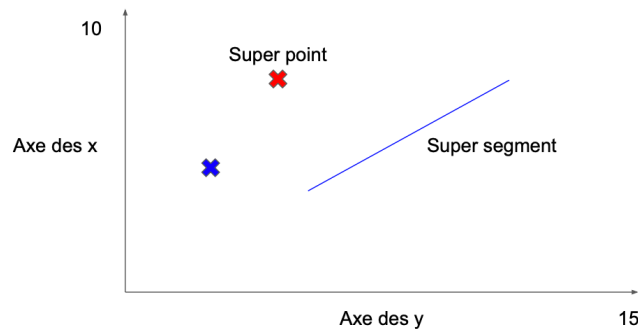
# Convention de nommage des Identificateurs :

## Je retiens

- Pour les classes, la première lettre est en majuscule puis les mots sont accolés avec des majuscules, pas de `_` :
  - exemples : `BeautifulClass`, `ArrayList`, ...
- Pour les méthodes (*devrait inclure un verbe*), les attributs et les variables qui ne sont pas « statiques », la première lettre est minuscule
  - exemples : `setLongueur`, `i`, `aStudent`
- Les constantes sont entièrement en majuscules et chaque mot est séparé par un underscore '`_`'.
  - exemple : `LONGUEUR_MAX`, `VAL_MIN`, `BLACK`
- Les packages tout en minuscules
  - exemple : `fr.pns.projet`

Nous souhaitons implémenter

- Evolution des specifications (2)



On veut pouvoir manipuler des **ensembles d'éléments**, c'est-à-dire des points et des segments :

- leur ajouter des points et des segments;
- obtenir la description de tous les éléments qu'ils contiennent;

*par exemple :*

1 : Point (3,-2), color: (231,193,148), Title : A

2 : Point (2,0), color: (213,171,93), Title : B

...

5 : Segment (2,0) -> (4,1), color: (0,0,0), titre : s2

6 : Segment (4,1) -> (3,-2), color: (0,0,0), titre : s3

- appliquer une translation à leurs éléments.

# TABLEAUX par l'exemple

# Une solution basée sur des tableaux

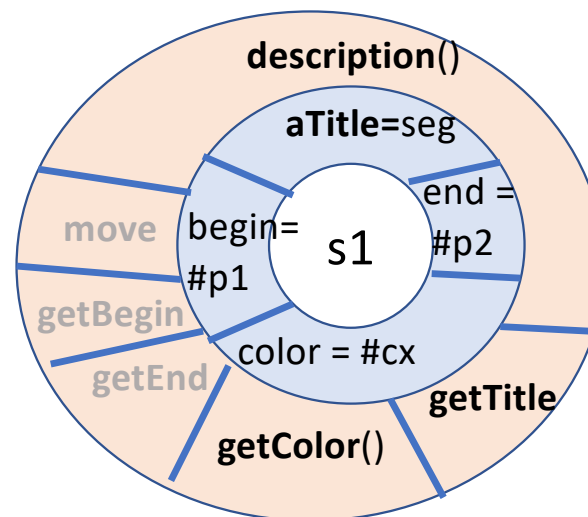
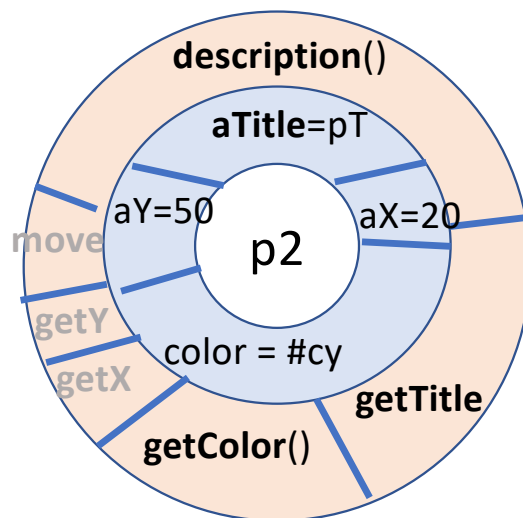
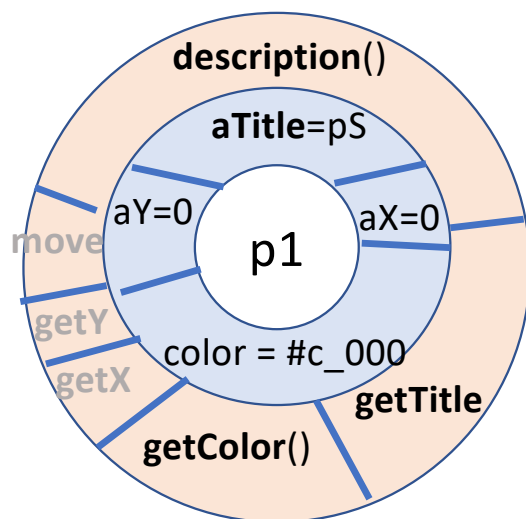
```
1  package cours;
2
3  /**
4   * @author JR Falleri
5   * @author MBF
6   */
7  class SystemElementSet {
8      SystemElement[] elements; //déclaration d'un tableau
9      int maxSize = 4; //Par défaut nous créons un tableau de dimension maxSize
10     int currentSize = 0; //Un tableau commence en case 0
11
12     SystemElementSet() {
13         elements = new SystemElement[maxSize]; //création du tableau
14     }
15
16     //Attention que se passe-t-il si index > currentSize ? index > maxSize
17     SystemElement get(int index) {
18         return elements[index];
19     }
19 }
```

```
SystemElement[] elements = new SystemElement[4];
```

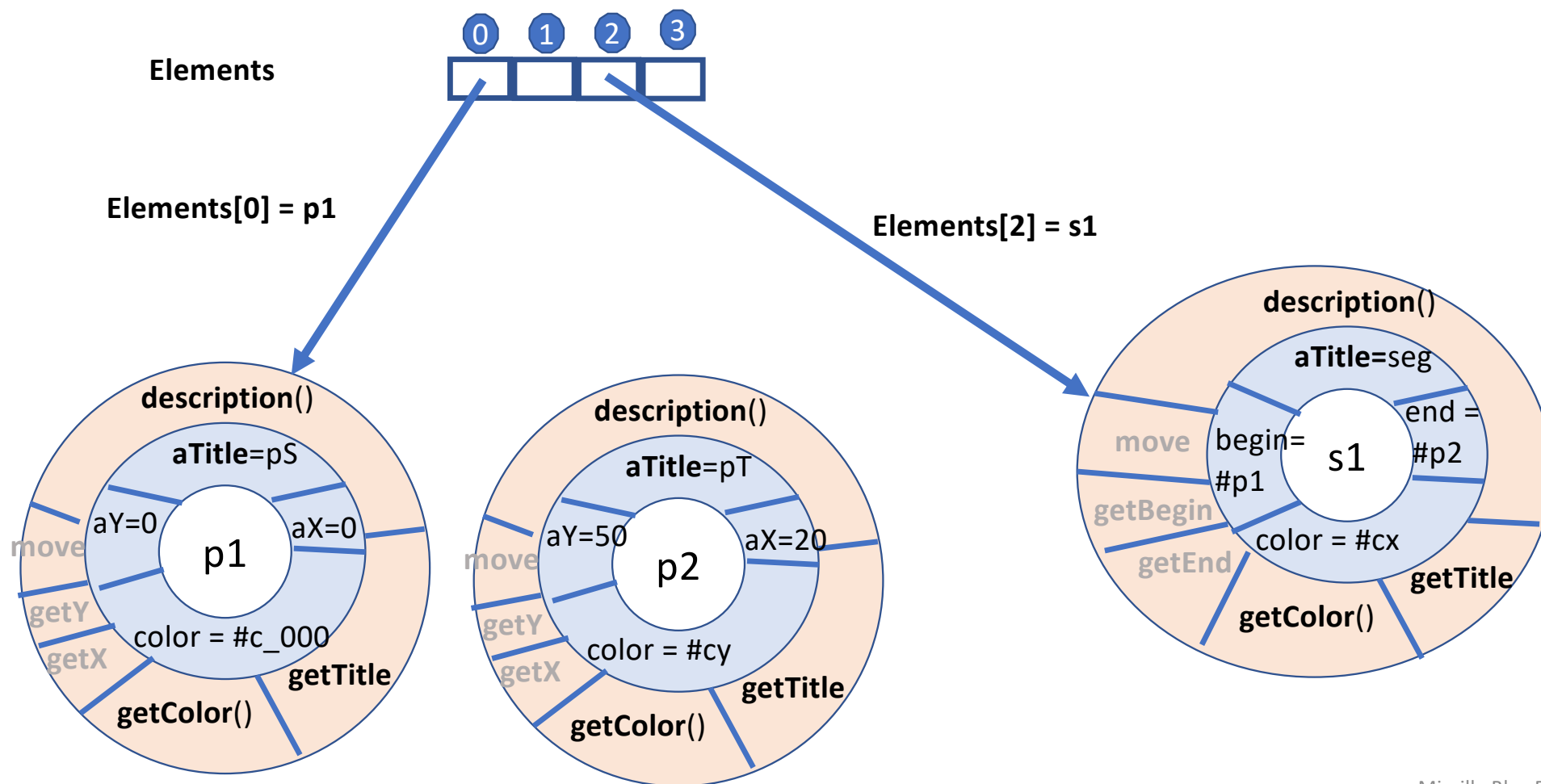
**elements**



elements.length == 4



```
SystemElement[] elements = new SystemElement[4];
```





# Manipulation des tableaux

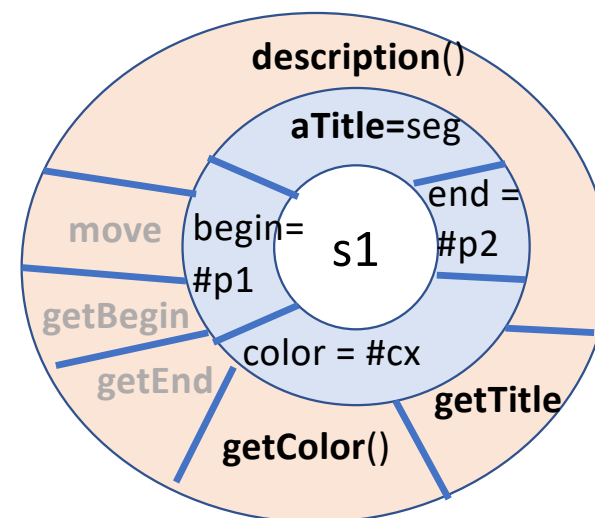
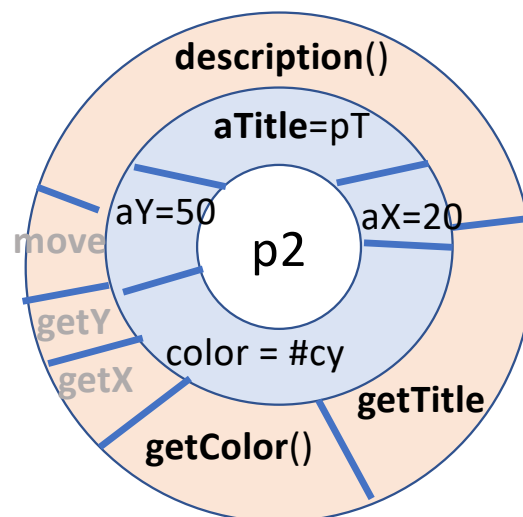
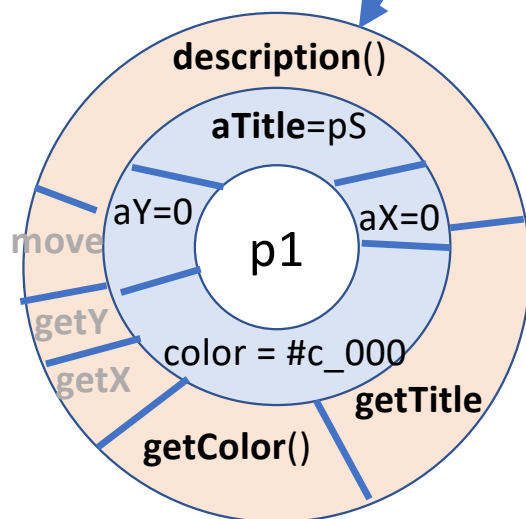
```
SystemElement[] elements = new SystemElement[4];
```



Elements[0] = p1

```
elements[elements.length] = new Point();
```

Exception in thread "main"  
java.lang.ArrayIndexOutOfBoundsException:  
Index 4 out of bounds for length 4



# Agrandir un tableau

Un tableau a une taille fixée à sa création (length) ; ses indices commencent à 0.

```
31
32 void addElement(SystemElement element) {
33     if (currentSize == maxSize) {
34         maxSize = maxSize * 2;
35         SystemElement[] tmp = new SystemElement[maxSize];
36         if (currentSize >= 0) {
37             elements = shallowCopyElementsIn(elements, tmp);
38             //Version using Library method
39             //System.arraycopy(elements, 0, tmp, 0, currentSize);
40         }
41     }
42     elements[currentSize] = element;
43     currentSize++;
44 }
45
46 private SystemElement[] shallowCopyElementsIn(SystemElement[] source, SystemElement[] target) {
47     for (int i = 0; i < currentSize; i++) {
48         target[i] = source[i];
49     }
50     return target;
51 }
```

# Boucle **for** pour recopier un tableau dans un autre

Utilisation d'une boucle for

```
for (statement 1; statement 2; statement 3) {  
    // code block to be executed  
}
```

Statement 1 is executed (one time) before the execution of the code block.

Statement 2 defines the condition for executing the code block.

Statement 3 is executed (every time) after the code block has been executed.

```
private SystemElement[] shallowCopyElementsIn(SystemElement[] source, SystemElement[] target  
    for (int i = 0; i < currentSize; i++) {  
        target[i] = source[i];  
    }  
    return target;  
}
```

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance. ... Inheritance lets us inherit attributes and methods from another class. **Polymorphism uses those methods to perform different tasks. This allows us to perform a single action in different ways.**



# POLYMORPHISME

[https://www.w3schools.com/java/java\\_polymorphism.asp](https://www.w3schools.com/java/java_polymorphism.asp)

# Description : un appel, de multiples formes

```

23     String description(){
24         StringBuilder bld = new StringBuilder();
25         for (int i = 0; i < currentSize; i++){
26             bld.append(String.format("%d : %s%n ", i + 1, elements[i].description()));
27         }
28         return bld.toString();
29     }

```

On a choisi de montrer les indices incrémentés de 1

Que les éléments soient des Points ou des Segments, on appelle uniquement la méthode *description*

## Code

```

47     SystemElementSet ses = new SystemElementSet();
48     ses.addElement(A);
49     ses.addElement(B);
50     ses.addElement(C);
51     ses.addElement(s1);
52     ses.addElement(s2);
53     ses.addElement(s3);
54     System.out.println(ses.description());

```

## Affichage

```

1 : Point (3,-2), color: (47,213,18), Title : A
2 : Point (2,0), color: (185,93,117), Title : B
3 : Point (4,1), color: (88,237,228), Title : C
4 : Segment (3,-2) -> (2,0), color: (0,0,0), titre : s1
5 : Segment (2,0) -> (4,1), color: (0,0,0), titre : s2
6 : Segment (4,1) -> (3,-2), color: (0,0,0), titre : s3

```

# Translations ?

On veut pouvoir manipuler des ensembles d'éléments, c'est-à-dire des points et des segments :

...

- appliquer une translation à leurs éléments.

Traduire un point ( $\Delta x$ ,  $\Delta y$ ) : OK ?

Traduire un segment ( $\Delta x$ ,  $\Delta y$ ) : OK ?

Q1 : Mais que se passe-t-il si je translate  $p_1$  qui est un point du segment  $s_1$  ?

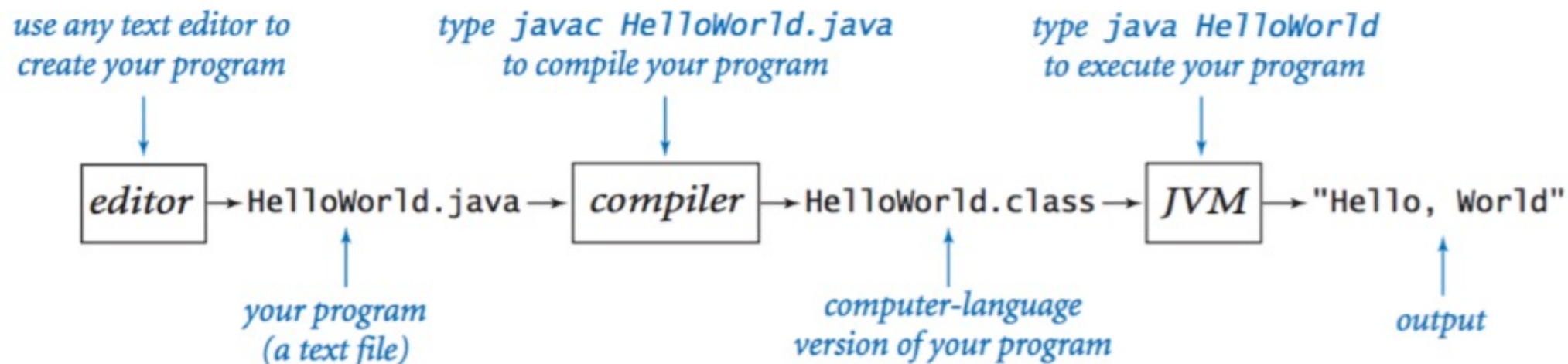
Q2 : Les `SystemElement` ne sont pas translatables... et on ne sait pas les traduire ...

Comment dire que les objets contenus dans un `SystemElementSet` doivent être translatables ?

## Editing, Compiling, and executing



# Editing, Compiling, and executing.



# Point d'entrée d'un programme Java

Pour qu'un programme soit exécutable, il doit y avoir une classe qui contient une méthode particulière, la méthode « main » : c'est le point d'entrée dans le programme

```
public static void main(String arg[ ])  
{  
.../...  
}
```

En TD, nous testerons nos classes, sans nécessairement utiliser un *main*.

# Exemple

Fichier MusicPlayerDemo.java

```
package chapter04.musicorganizer;
```

```
import java.util.Scanner;
```

```
/**
```

```
 * To use Player
```

```
 * @author Mireille Blay-Fornarino
```

```
 */
```

```
public class MusicPlayerDemo {
```

```
    public static void main(String[] args){
```

```
        MusicPlayer mp = new MusicPlayer();
```

```
        Scanner sc = new Scanner(System.in);
```

```
        System.out.println("Placez vos fichiers audio sous : "+ System.getProperty("user.dir"));
```

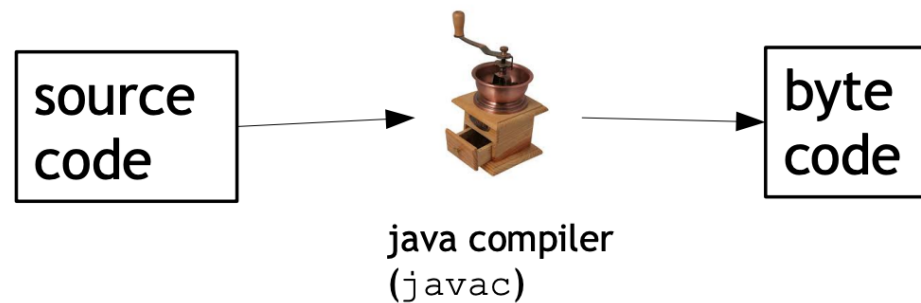
```
        boolean goOn = true;
```

```
....
```

```
}
```

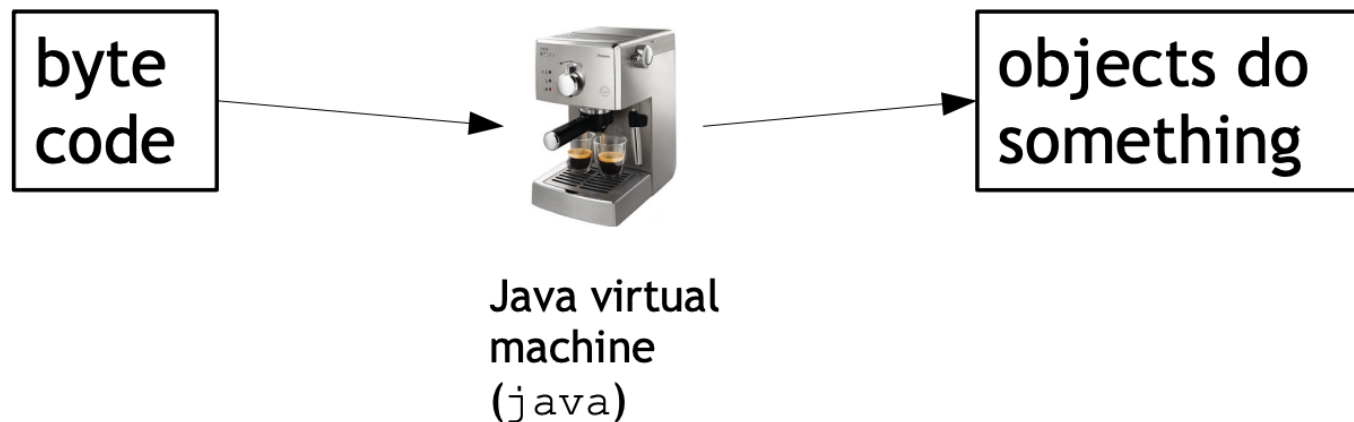
# Source code

- Source code classes are what you write.
- Class source code defines its details.
- Java *compiler* turns source code into byte code.



# Running code

- Byte code interpreted (executed) by *Java Virtual Machine (JVM)*.
- Classes become objects at execution.



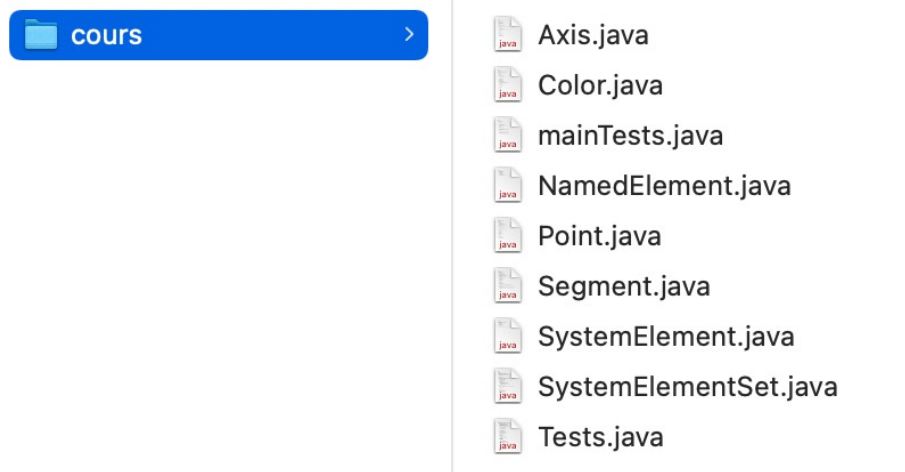
# De la conception des codes aux fichiers

# Vos codes source, une perspective fichiers

Structuration des fichiers (cours est le nom du package)

## Structuration dans IntelliJ

- ▼ src
  - ▼ cours
    - Axis
    - Color
    - mainTests
    - NamedElement
    - Point
    - Segment
    - SystemElement
    - SystemElementSet

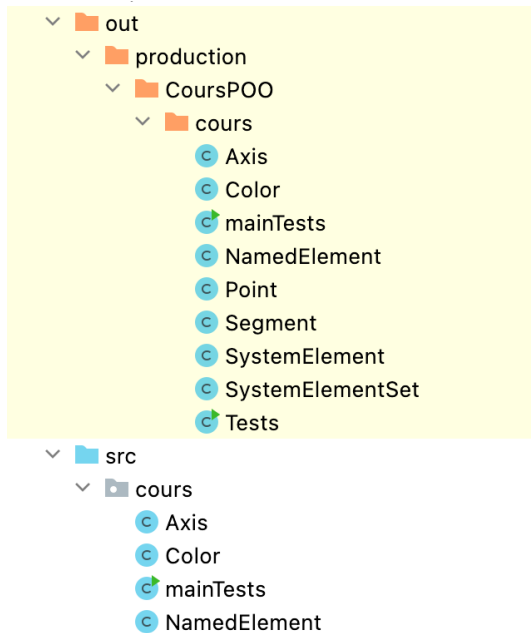


```
total 80
drwxr-xr-x@ 11 blay  staff   352  2 août 06:51 .
drwxr-xr-x  4 blay  staff   128  2 août 07:00 ..
-rw-r--r--  1 blay  staff   977  1 août 19:28 Axis.java
-rw-r--r--  1 blay  staff  1994  1 août 16:57 Color.java
-rw-r--r--  1 blay  staff   404  1 août 07:35 NamedElement.java
-rw-r--r--  1 blay  staff  1150  1 août 19:00 Point.java
-rw-r--r--  1 blay  staff  1557  1 août 19:44 Segment.java
-rw-r--r--  1 blay  staff   428  1 août 07:46 SystemElement.java
-rw-r--r--  1 blay  staff  2334  2 août 06:39 SystemElementSet.java
-rw-r--r--  1 blay  staff  5462  2 août 06:51 Tests.java
-rw-r--r--  1 blay  staff   896  30 jul 23:55 mainTests.java
mbp-de-mireille:cours blay$
```

Mireille Blay-Fornarino

# Java bytecode : une perspective fichier

## Structuration dans IntelliJ



## Structuration des fichiers (cours est le nom du package)

```
[mbp-de-mireille:cours blay$ ls
Axis.class          Point.class          SystemElementSet.class
Color.class          Segment.class         Tests.class
NamedElement.class   SystemElement.class   mainTests.class
mbp-de-mireille:cours blay$
```



# References

- <https://docs.oracle.com/javase/tutorial/java/TOC.html>
- <https://www.labri.fr/perso/falleri/perso/ens/pg220/>