

# CONSTRUIRE UNE CLASSE

Martine GAUTIER - Université de Lorraine  
[martine.gautier@univ-lorraine.fr](mailto:martine.gautier@univ-lorraine.fr)

# Ce qu'on sait faire

- Lire une documentation
- Ecrire une séquence d'instructions qui crée et utilise des objets
- Compiler, exécuter un programme (avec entrées/sorties)
- Maîtriser son environnement de développement en ligne de commandes

Il reste à apprendre à construire une nouvelle classe, pour décrire une catégorie d'objets.

# Description d'une classe

- Toutes les instances d'une classe disposent de données propres et réagissent aux mêmes fonctions.
- Dans une classe, on décrit :
  - les données propres à chaque objet
  - les fonctions applicables aux objets (profil + corps)
- ... sans oublier de préciser le nom du package et le nom de la classe

```
package geometrie ;

/** @author MG
    @version 2020
 */

public class Segment {

    // Description des données

    // Définition des constructeurs

    // Définition des fonctions d'observation

    // Définition des fonctions de transformation

}
```

```
package geometrie ;
```

```
/** @author MG  
    @version 2020  
*/
```

```
public class Segment {
```

```
    // Description des données
```

```
    // Définition des constructeurs
```

```
    // Définition des fonctions d'observation
```

```
    // Définition des fonctions de transformation
```

```
}
```

**Commentaires normalisés  
utilisés pour construire la doc.  
HTML**

```
package geometrie ;
```

```
/** @author MG
```

```
    @version 2020
```

```
*/
```

```
public class Segment {
```

```
    // Description des données
```

```
    // Définition des constructeurs
```

```
    // Définition des fonctions d'observation
```

```
    // Définition des fonctions de transformation
```

```
}
```

Aucun ordre imposé

# Description des données propres à chaque objet

Classe	Caractéristiques	Déclaration des champs/attributs
Point	les coordonnées cartésiennes	<code>private double abs ;</code> <code>private double ord ;</code>
Date	le jour, le mois et l'année	<code>private int jour ;</code> <code>private int mois ;</code> <code>private int annee ;</code>
Segment	les deux extrémités	<code>private Point ext1 ;</code> <code>private Point ext2 ;</code>
Triangle	les trois sommets	<code>private Point p1 ;</code> <code>private Point p2 ;</code> <code>private Point p3 ;</code>

```
package geometrie ;

/** @author MG
    @version 2020
 */

public class Segment {

    // Description des données
    private Point p1 ;

    private Point p2 ;

    // Définition des constructeurs

    // Définition des fonctions d'observation

    // Définition des fonctions de transformation

}
```

Les champs sont privés :  
protection des données



# Allocation mémoire lors de l'instanciation

Classe	Instruction de création	Déclaration des champs/attributs	Représentation mémoire				
Point	new Point(.....)	private double abs ; private double ord ;	<table><tr><td>Point</td></tr><tr><td></td></tr><tr><td></td></tr></table>	Point			
Point							
Date	new Date(...)	private int jour ; private int mois ; private int annee ;	<table><tr><td>Date</td></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr></table>	Date			
Date							
Segment	new Segment(...)	private Point p1 ; private Point p2 ;	<table><tr><td>Segment</td></tr><tr><td></td></tr><tr><td></td></tr></table>	Segment			
Segment							
Triangle	new Triangle	private Point p1 ; private Point p2 ; private Point p3 ;	<table><tr><td>Triangle</td></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr></table>	Triangle			
Triangle							

# Définition d'un constructeur

- L'allocation mémoire est réalisée par l'opérateur new.
- Le constructeur a seulement pour mission de remplir les champs.
- Les instructions diffèrent selon les constructeurs, en fonction des paramètres.

```
package geometrie ;  
public class Segment {  
    private Point p1 ;  
    private Point p2 ;
```

Commentaires normalisés

```
    /** Initialise un segment à partir de ses deux extrémités  
        @param pt1 première extrémité  
        @param pt2 seconde extrémité  
    */  
    public Segment(Point pt1, Point pt2) {  
        this.p1 = pt1 ;  
        this.p2 = pt2 ;  
    }  
    .....  
}
```

this désigne l'instance  
créée par l'opérateur new

```
package geometrie ;  
public class Segment {  
    private Point p1 ;  
    private Point p2 ;  
    public Segment(Point pt1, Point pt2) {  
        this.p1 = pt1 ;  
        this.p2 = pt2 ;  
    }  
    public Segment(Segment s) {  
        this.p1 = s.getP1() ;  
        this.p2 = s.getP2() ;  
    }  
    .....  
}
```

Les deux constructeurs sont publics.


Les instructions du constructeur dépendent de la nature des paramètres.

Pour consulter les extrémités du segment *s*, on utilise les fonctions adéquates.

```
package geometrie ;  
public class Segment {  
    private Point p1 ;  
    private Point p2 ;  
    public Segment(Point pt1, Point pt2) {  
        this.p1 = pt1 ;  
        this.p2 = pt2 ;  
    }  
    public Segment(Segment s) {  
        this.p1 = s.p1 ;  
        this.p2 = s.p2 ;  
    }  
    .....  
}
```

Pour consulter les extrémités du segment *s*, on utilise les fonctions adéquates ou bien les champs, même privés.

```
package geometrie ;  
public class Segment {  
    private Point p1 ;  
    private Point p2 ;  
    ...  
}
```



```
public Segment(Segment s) {  
    this.p1 = s.getP1() ;  
    this.p2 = s.getP2() ;  
}
```

```
public Segment(Segment s) {  
    this(s.getP1(), s.getP2()) ;  
}
```

Si on décide de changer la représentation mémoire d'un segment, il n'y aura qu'un seul constructeur à modifier.

```
package geometrie ;  
  
public class Triangle {  
    private Point p1 ;  
    private Point p2 ;  
    private Point p3 ;  
    public Triangle(Point pt1, Point pt2, Point pt3) {  
        this.p1 = pt1 ;  
        this.p2 = pt2 ;  
        this.p3 = pt3 ;  
    }  
    public Triangle(Segment s, Point p) {  
        Point a = s.getP1() ;  
        Point b = s.getP2() ;  
        this(a, b, p) ;  
    }  
}
```

Les instructions du constructeur dépendent des paramètres.

```
package geometrie ;  
  
public class Triangle {  
    private Point p1 ;  
    private Point p2 ;  
    private Point p3 ;  
    private Segment segment ;  
    Point point ;  
  
    public Triangle(Point pt1, Point pt2, Point pt3) {  
        this.p1 = pt1 ; this.p2 = pt2 ; this.p3 = pt3 ;  
    }  
  
    public Triangle(Segment s, Point p) {  
        this.segment = s ;  
        this.point = p ;  
    }  
}
```

*Ceci n'a aucun sens.*



# Définition des méthodes

- Une fonction d'observation utilise les champs et les paramètres pour calculer une valeur de retour (instruction `return` obligatoire).
- Une fonction de transformation modifie un ou plusieurs champs, en faisant attention de laisser l'objet dans un état cohérent.
- Sauf exception, une fonction d'observation ne modifie pas l'état de l'objet.

```
package geometrie ;  
public class Segment {  
    private Point p1 ;  
    private Point p2 ;  
    /** @return le premier point */  
    public Point getP1(){  
        return this.p1 ;  
    }  
    public Point milieu() {  
        double ab = (p1.getAbscisse()+p2.getAbscisse())/2 ;  
        double or= (p1.getOrdonnee()+p2.getOrdonnee())/2 ;  
        return new Point(ab, or) ;  
    }  
    .....  
}
```

Une nouvelle  
balise à retenir

Ou bien  
return p1 ;

# Gestion de la mémoire

- Impossible de programmer efficacement sans avoir une idée précise de ce qui se passe en mémoire.
- Questions auxquelles il faut savoir répondre :
  - que contient une variable de type `int` ?
  - que contient une variable de type `Point` ?
  - comment la machine virtuelle exécute-t-elle une affectation ?
  - comment réalise-t-elle un appel de fonction, avec passage de paramètres ?
  - certains objets sont-ils inaccessibles ?
- Logiciel artEoz ([artEoz.loria.fr](http://artEoz.loria.fr))
  - visualisation de schémas mémoire

# artEoz

Déclaration/affectation de variables de type primitif

Déclaration/affectation de variables de type Point, Segment, Triangle

Déclaration/affectation de variables de type String (représentation simplifiée)

Exécution pas à pas

Pile (this, paramètres)



```
public Segment(Point pt1, Point pt2) {  
    this.p1 = pt1 ;  
    this.p2 = pt2 ;  
}
```

OU

```
public Segment(Point pt1, Point pt2) {  
    this.p1 = new Point(pt1) ;  
    this.p2 = new Point(pt2) ;  
}
```

PARTAGE  
DES OBJETS

PAS DE PARTAGE  
DES OBJETS

```
/** Copie superficielle (Shallow copy) */  
public Segment(Segment s) {  
    this.p1 = s.getP1() ;  
    this.p2 = s.getP2() ;  
}
```

OU

```
/** Copie profonde (Deep copy) */  
public Segment(Segment s) {  
    this.p1 = new Point(s.getP1()) ;  
    this.p2 = new Point(s.getP2()) ;  
}
```

PARTAGE

DES OBJETS

PAS DE PARTAGE

DES OBJETS

# Maîtriser la conception d'un logiciel

- ★ Le nombre de classes peut vite devenir important :
  - plusieurs centaines
- ★ Conception/développement réalisés par une équipe
  - Une même personne ne peut pas connaître en détail chaque classe.
- ★ Trouver un support de dialogue entre concepteurs/développeurs
- ★ Utiliser la documentation HTML ?
  - indispensable pour pouvoir utiliser une classe
  - inutilisable pour avoir une vue d'ensemble de l'application

# Unified Modeling Language

- ★ Langage de modélisation graphique à base de pictogrammes permettant de visualiser la conception d'un système.
  - né en 1995, issu de travaux d'origines diverses
  - Grady Booch, James Rumbaugh et Ivar Jacobson
  - couramment utilisé en développement logiciel
- ★ Standard adopté par l'Object Management Group
- ★ Au moins neuf sortes de diagrammes
- ★ Diagramme de classes : modéliser les relations entre classes





# Diagramme de classes



Classe

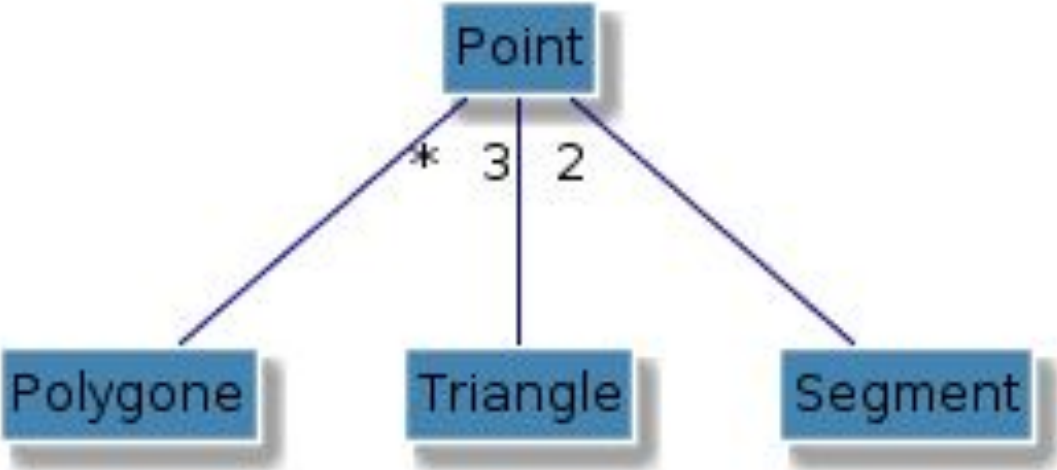
- champs/attributs primitifs
- profils des constructeurs/fonctions



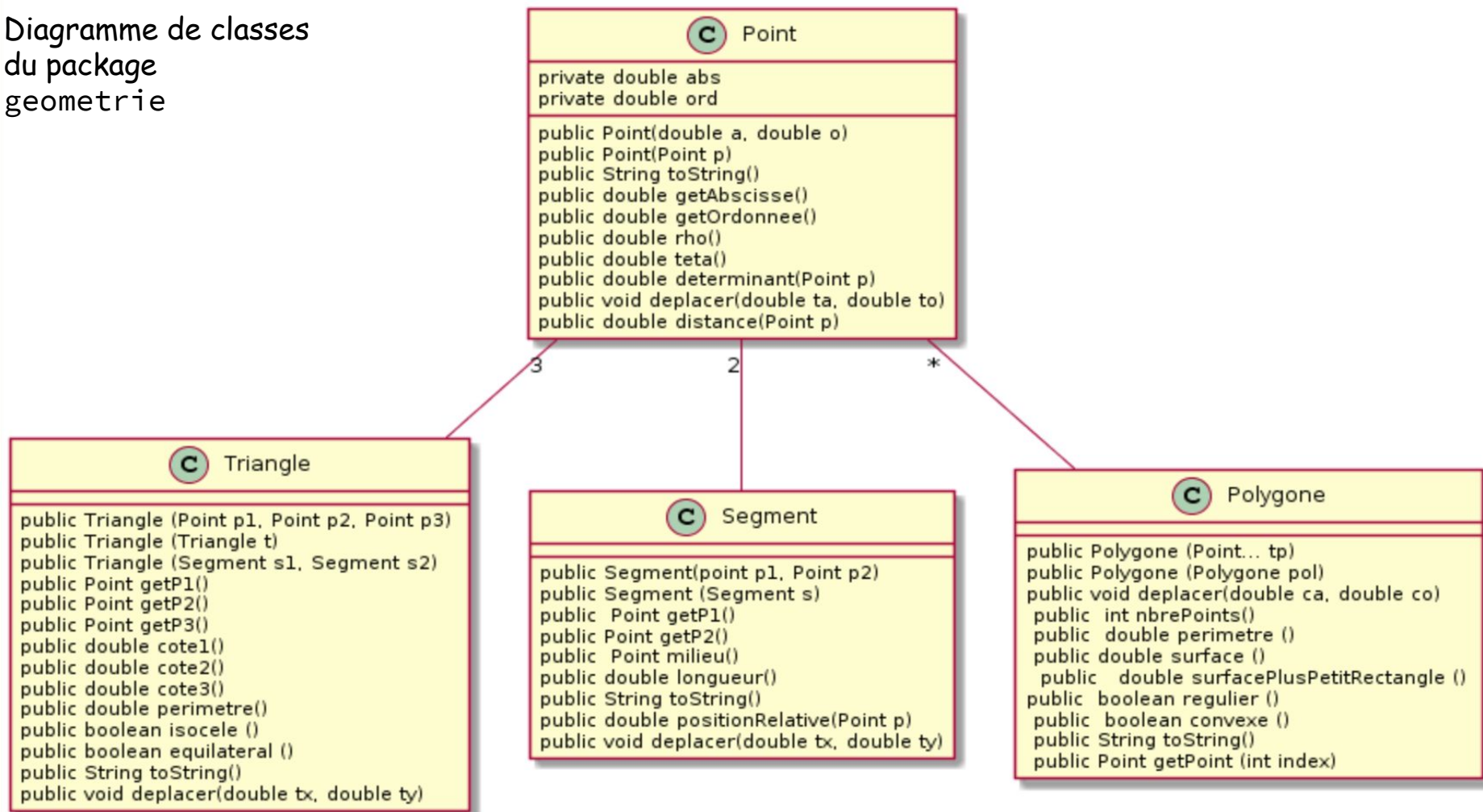
Relation entre deux classes : association

- Une instance de la classe C1 connaît  $x$  instances de la classe C2

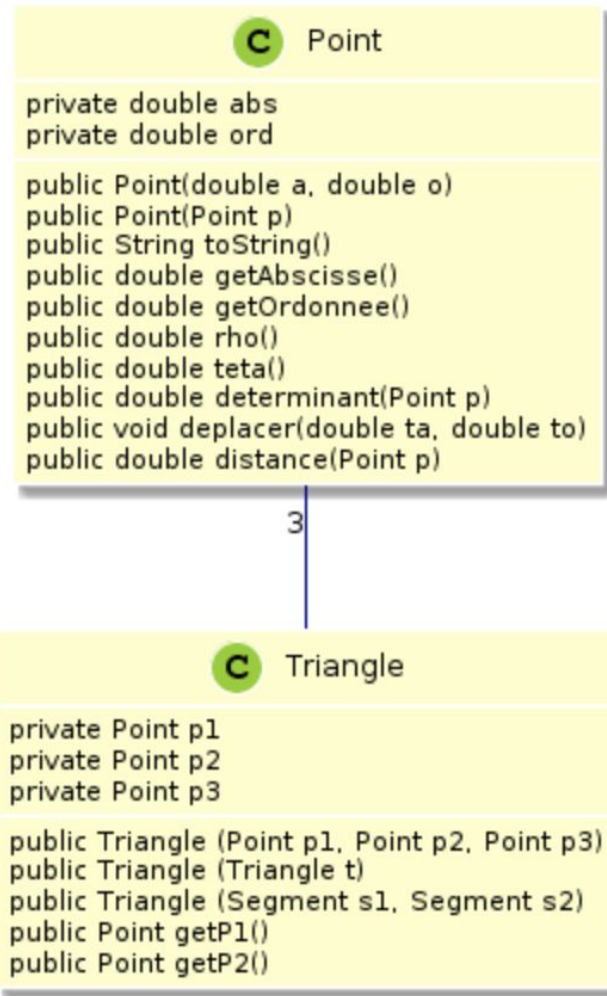
Diagramme de classes  
**simplifié** du package  
geometrie



# Diagramme de classes du package geometrie



**Diagramme incorrect**  
**Il dit qu'un Triangle a 6 points ....**



# Les énumérations

```
package geometrie ;

public enum Couleur {

    private String francais, anglais, code ;
    private Couleur(String francais, String anglais, String code) { .... }
    public String getFrancais() { .... }
    public String getAnglais() { .... }
    public String getCode() { .... }
    ....
}
```

# Les énumérations

L'apparence  
d'une classe

```
package geometrie ;

public enum Couleur {

    private String francais, anglais, code ;
    private Couleur(String francais, String anglais, String code) { .... }
    public String getFrancais() { .... }
    public String getAnglais() { .... }
    public String getCode() { .... }
    ....
}
```

# Les énumérations

```
package geometrie ;

public enum Couleur {

    private String francais, anglais, code ;
    private Couleur(String francais, String anglais, String code) { .... }
    public String getFrancais() { .... }
    public String getAnglais() { .... }
    public String getCode() { .... }
    ....
}
```

**Avec un  
constructeur  
privé**

# Les énumérations

```
package geometrie ;

public enum Couleur {

    BLEU("Bleu", "Blue", "#318ce7") ,
    ROUGE("Rouge", "Red", "#850606") ,
    VERT("Vert", "Green", "#b0f2b6") ;

    private String francais, anglais, code ;
    private Couleur(String francais, String anglais, String code) { .... }
    public String getFrancais() { .... }
    public String getAnglais() { .... }
    public String getCode() { .... }
    ....
}
```



# Les énumérations

Trois objets  
prédéfinis

```
package geometrie ;  
public enum Couleur {  
    BLEU("Bleu", "Blue", "#318ce7") ,  
    ROUGE("Rouge", "Red", "#850606") ,  
    VERT("Vert", "Green", "#b0f2b6") ;  
    private String francais, anglais, code ;  
    private Couleur(String francais, String anglais, String code) { .... }  
    public String getFrancais() { .... }  
    public String getAnglais() { .... }  
    public String getCode() { .... }  
    ....  
}
```

# Les énumérations

```
Couleur coulv = Couleur.VERT ;  
String codev = coulv.getCode() ;
```

```
package geometrie ;  
  
public enum Couleur {  
    BLEU("Bleu", "Blue", "#318ce7") ,  
    ROUGE("Rouge", "Red", "#850606") ,  
    VERT("Vert", "Green", "#b0f2b6") ;  
    private String francais, anglais, code ;  
    private Couleur(String francais, String anglais, String code) { .... }  
    public String getFrancais() { .... }  
    public String getAnglais() { .... }  
    public String getCode() { .... }  
    ....  
}
```