

# 2i002 - Objets complexes, composition d'objet

Vincent Guigue



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

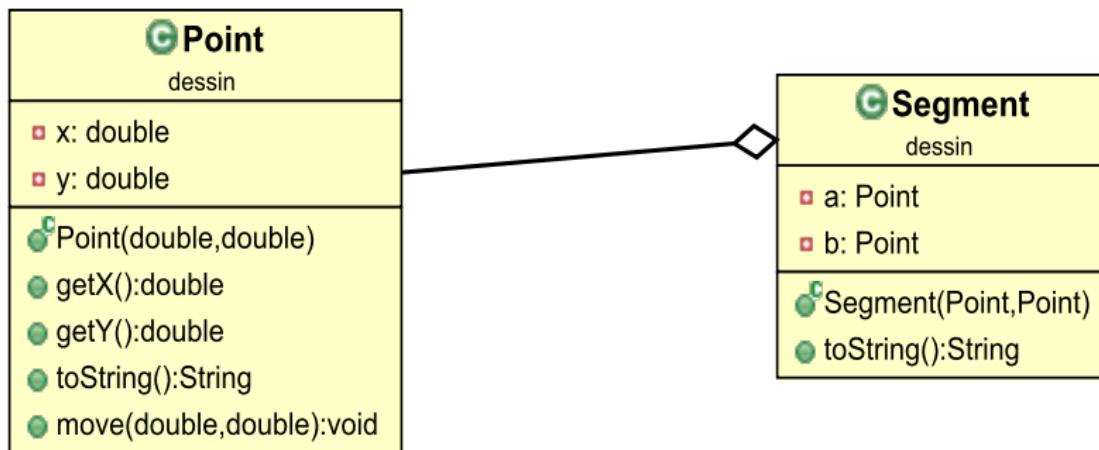
```
1 public class Segment{  
2     private Point a, b; // simple déclaration  
3  
4     public Segment( Point a, Point b) {  
5         this.a = a;  
6         this.b = b;  
7     }  
8     public String toString() {  
9         return "Segment [a=" + a + ", b=" + b + "]";  
10        // note " +a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

# REPRÉSENTATION DES LIENS UML

```
1 public class Segment{  
2     private Point a,b;  
3     ...
```

Deux représentations usuelles :

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



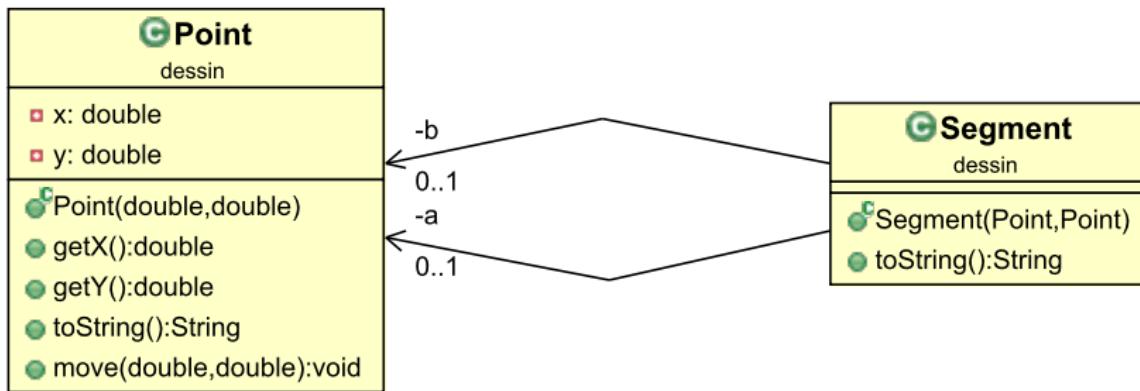
# REPRÉSENTATION DES LIENS UML

```
1 public class Segment{  
2     private Point a,b;  
3     ...
```

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



# CLONAGE D'OBJET COMPOSÉ : LE PIEGE

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

Proposition :

Voiture
-Point position
+ Voiture(Point p)
+ avancer() : void
+ 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();
```

# CLONAGE D'OBJET COMPOSÉ : LE PIEGE

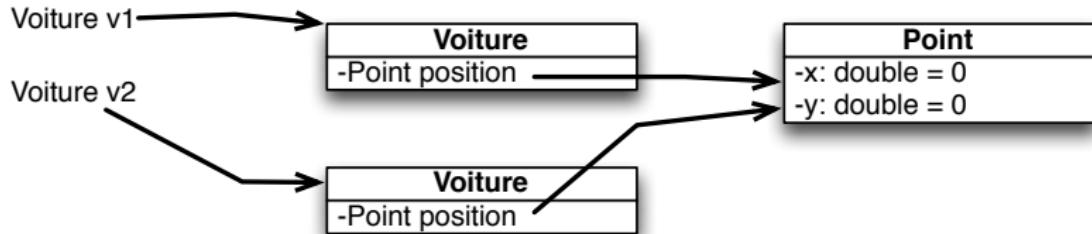
Cas classique : besoin de dupliquer une Voiture dont la position est définie par un attribut Point

Proposition :

Voiture
-Point position
+ Voiture(Point p)
+ avancer() : void
+ 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();
```

⚠ 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)

# CLONAGE D'OBJET COMPOSÉ : LE PIÈGE

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

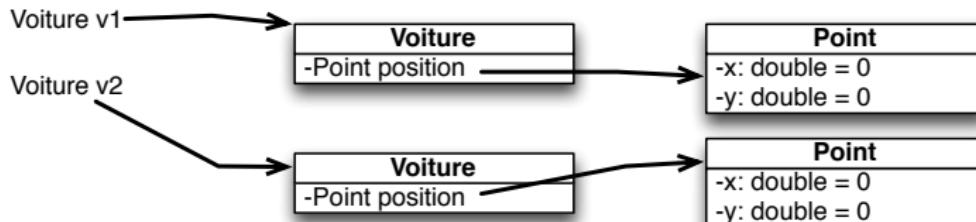
Proposition :

Voiture
-Point position
+ Voiture(Point p)
+ avancer() : void
+ 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();
```

Solution :

```
1 // Dans voiture
2 Voiture clone(){ // clonage recursif
3     return new Voiture(position.clone());
4 }
```



## ÉGALITÉ STRUCTURELLE : ATTENTION AU equals

- Structure standard classique...
- jusqu'au moment du test sur les attributs :  
penser au equals (au lieu de ==)

```
1  public boolean equals(Object obj) {  
2      if (this == obj) return true;  
3      if (obj == null) return false;  
4      if (getClass() != obj.getClass())  
         return false;  
5      Voiture other = (Voiture) obj;  
6      if (! position.equals(obj.position)) return false;  
7      return true;  
8  }  
9 }
```

# COMPILEATION DES OBJETS COMPOSÉS

- 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
```

# 2i002 - Tableaux

Vincent Guigue



# STRUCTURE DE DONNÉES

La structure de base de la programmation impérative : disponible sur les types de base et sur les objets.

## ① Tableau à taille fixe

- + Economie mémoire
- + Rapidité d'accès
- Peu flexible (taille fixe !)

## ② Tableau à taille variable

- Gourmand en mémoire
- (Un peu) moins rapide
- + Très flexible

## [TAILLE FIXE] SYNTAXE

- Déclaration : *type* [] nomVariable
- Instanciation : nomVariable = new *type* [*taille*]
- Accès à la case *i* (lecture ou écriture) : nomVariable[i]
- accès à la longueur du tableau : nomVariable.length

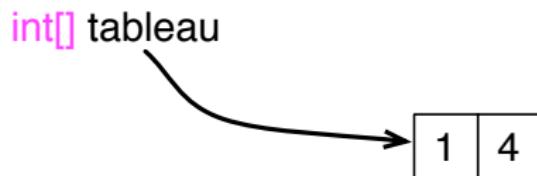
```
1 class TableauA {  
2     public static void main(String[] argv) {  
3         int[] tableau; // déclaration  
4  
5         tableau = new int[2]; // instanciation  
6         tableau[0] = 1; // utilisation (écriture)  
7         tableau[1] = 4;  
8  
9         int i = tableau[0]; // utilisation en lecture  
10        // accès à la longueur du tableau  
11        System.out.println("Longueur: "+tableau.length);  
12    }  
13 }
```

- tableau est une variable de type int[] (ie tableau d'entiers)
- tableau[i] : chaque case de tableau est de type int

# REPRÉSENTATION MÉMOIRE

```
1 int [] tableau = new int [2];  
2 tableau[0] = 1;  
3 tableau[1] = 4;
```

tableau = création d'un **ensemble de variables**...  
... Facilement accessibles dans les boucles.



Attention : bien différencier **variables** et **instances**...

- instantiation d'un tableau = création de **variables**
- créer les **instances** dans un second temps
- ⇒ passage aux objets (un peu) piégeux

# TABLEAU D'OBJETS

Soit la classe Point (vue dans les cours précédent)

Déclaration d'une variable `tabP`    `1 Point [] tabP ;`  
de type `Point[]` (tableau de  
points)

 Le tableau n'existe pas ! (il  
n'est **pas instancié**)

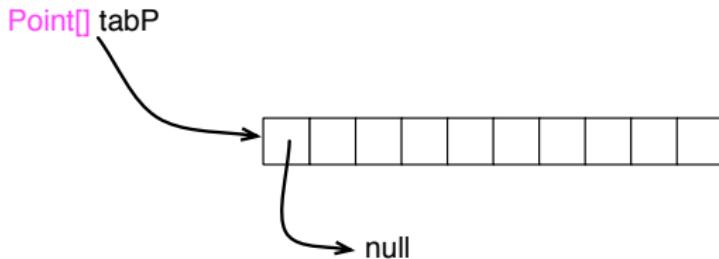
`Point[] tabP`

# TABLEAU D'OBJETS

La variable `tabP` référence un tableau de 10 cases

**⚠ 10 cases = 10 variables...  
... 0 instances de Point !**

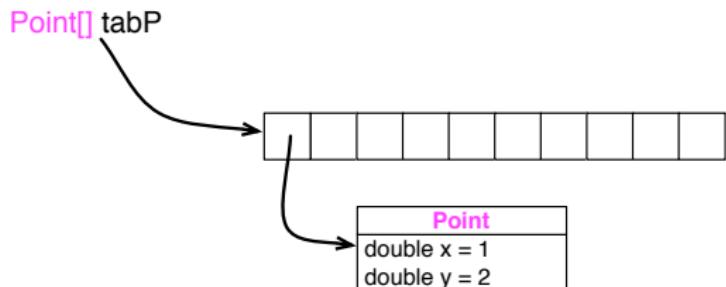
```
1 Point [] tabP;  
2 tabP = new Point [10];
```



# TABLEAU D'OBJETS

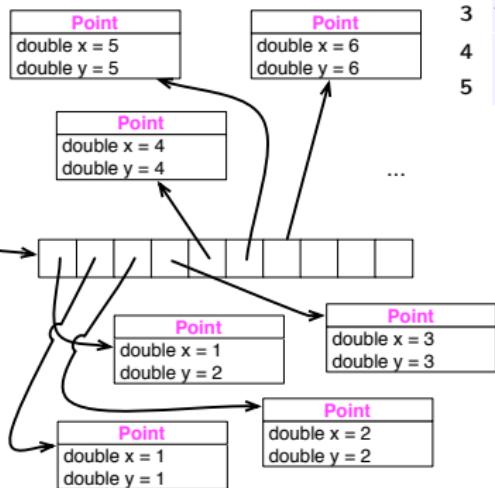
Chaque case (=variable) doit être instanciée

```
1 Point [] tabP;  
2 tabP = new Point [10];  
3 tabP [0] = new Point (1,2);
```



# TABLEAU D'OBJETS

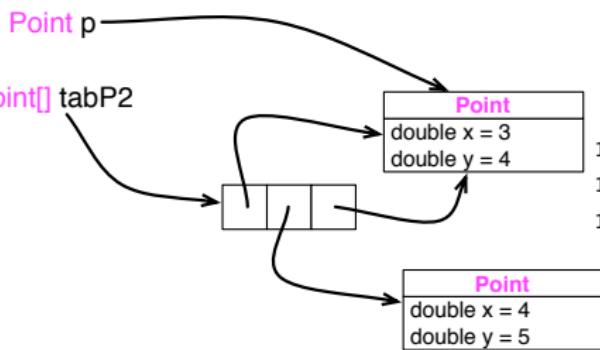
Usage parfait en combinaison des boucles :



```
1 Point [] tabP;  
2 tabP = new Point [10];  
3 tabP [0] = new Point (1,2);  
4 for (int i=1; i<tabP.length; i++)  
5     tabP [i] = new Point (i,i);
```

# TABLEAU D'OBJETS

Les cases se comportent vraiment comme des variables : on peut **jouer avec les références**



```
1 Point [] tabP;
2 tabP = new Point [10];
3 tabP[0] = new Point(1,2);
4 for( int i=1; i<tabP.length; i++)
5     tabP[i] = new Point(i,i);
6 // second tableau
7 // + jeu de références
8 Point [] tabP2 = new Point [3];
9 Point p = new Point(3,4);
10 tabP2[0] = p;
11 tabP2[1] = new Point(4,5);
12 tabP2[2] = tabP2[0];
```

## VARIANTES DE SYNTAXE

Création Syntaxe simplifiée : {value,value,...}

 Ne marche que sur la ligne de déclaration

```
1 boolean [] tableau={true , false , true };
```

# VARIANTES DE SYNTAXE

Création Syntaxe simplifiée : {value,value,...}

 Ne marche que sur la ligne de déclaration

```
1 boolean [] tableau={true , false , true };
```

Création Syntaxe intermédiaire (marche partout) :

new type [] {value,value,...}

```
1 boolean [] tableau;
```

```
2 tableau = new boolean []{ true , false , true };
```

# VARIANTES DE SYNTAXE

Boucle Parcours des éléments du tableau (sans référence d'indice) :

```
for(type var : nomTableau) ...
```

*var* prend successivement toutes les valeurs des éléments du tableau

```
1 for(boolean b: tableau) // affichage de tous les éléments  
2 System.out.println(b);
```

⚠ Pas de référence aux indices : usage possible, ou pas, en fonction des algorithmes

# TABLEAUX ET BOUCLES

Code robuste = pas de duplication de l'information

Attention aux conditions de fin de boucles

```
1 int [] tab = {2, 3, 4, 5, 6};
```

Code robuste = pas de duplication de l'information

Attention aux conditions de fin de boucles

```
1 int [] tab = {2, 3, 4, 5, 6};
```

Besoin de faire une boucle...

```
2 for (int i=0; i<5; i++) // FAUX dans le cadre de 2i002  
3     ...
```

# TABLEAUX ET BOUCLES

Code robuste = pas de duplication de l'information

Attention aux conditions de fin de boucles

```
1 int [] tab = {2, 3, 4, 5, 6};
```

Besoin de faire une boucle...

```
2 for(int i=0; i<5; i++) // FAUX dans le cadre de 2i002  
3     ...
```

## length

La taille du tableau tab est définie lors de la **création** (implicitement ou explicitement).

Utiliser `tab.length` pour y faire référence

```
4 // OK: modifier le tableau = modifier la boucle !  
5 for(int i=0; i<tab.length; i++)  
6     ...
```

## [TABLEAU DYNAMIQUE] ARRAYLIST

Usage dans 2 cas (imbriqués) :

- **Taille finale inconnue** lorsque l'on commence à utiliser le tableau (e.g. lecture d'un fichier...)
- **Taille variable** en cours d'utilisation (e.g. pile d'objets à traiter de taille variable)
- Objet JAVA à déclarer avant utilisation :

```
1 import java.util.ArrayList; // en entête
```
- Syntaxe objet classique + approche générique (hors prog.) :
  - la variable sera de type : `ArrayList<type>`
  - `type` est forcément un objet ( $\neq$  type de base) : `Integer`, `Double`, `Point`...
- Même représentation mémoire que les tableaux de taille fixe

# ARRAYLIST : SYNTAXE DÉTAILLÉE

- Exemple sur un tableau de Point
- Méthodes principales : constructeur, add, get, remove, size
- Plus d'informations sur la javadoc (beaucoup de d'autres méthodes disponibles) :

<https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html>

```
1 // Construction comme un objet classique
2 ArrayList<Point> tabArr = new ArrayList<Point>();
3 tabArr.add(new Point(1,2)); // ajout
4 for(int i=0; i<9; i++) tabArr.add(new Point(i,i));
5
6 // accesseur sur le deuxième élément (index = 1)
7 Point p = tabArr.get(1);
8 tabArr.remove(0); // suppression du premier élément
9
10 // accesseur sur la taille courante
11 System.out.println(tabArr.size());
```

## SORTIE DE TABLEAU [FIXE OU DYNAMIQUE]

⚠ Tableau... ⇒ possibilité de sortir du tableau

- Cas classique :
    - Mélange entre taille  $n$  et dernier indice du tableau ( $n - 1$ )
    - Tentative d'accès à un index négatif
    - Erreur de boucle...
  - Symptôme : **ArrayIndexOutOfBoundsException**
    - Echec lors de l'exécution du code (compilation OK)
- ```
1 Point [] tab = {new Point(), new Point()};
2 System.out.println(tab[2]);
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 2
at test.Point.main(Point.java:118)
```

- Attention aux **NullPointerException** : après instanciation d'un tableau, aucune instance n'est disponible :
- ```
1 Point [] tab = new Point[2];
2 System.out.println(tab[0].getX()); // => NullPointerException
```

# FONCTIONS AVANCÉES

## o ArrayList

- Dans la classe même :

```
1 ArrayList<Double> arr = new ArrayList<Double>();  
2 for (int i = 0; i < 10; i++)  
3     arr.add((double) i);  
4 if (arr.contains(2.))  
5     System.out.println("Trouvé!");
```

- Dans la classe Collections

- Tris, min, max, mélange, renversement...

## o Tableau

- Dans la classe Arrays : recherche, copie, remplissage

```
1 int[] tab = {3, 5, 1, 8, 9};  
2 System.out.println(Arrays.binarySearch(tab, 1));
```

# TABLEAU À DEUX DIMENSIONS

Comment gérer les matrices ?

Comme des tableaux de tableaux

- Déclaration des variables : type [] []

```
1 int [][] matrice;
```

- Instanciation

```
2 matrice = new int [2][3]; // 2 lignes , 3 colonnes
```

- Usage

```
3 matrice[0][0] = 0; matrice[0][1] = 1; matrice[0][2] = 2;
```

```
4 matrice[1][0] = 3; matrice[1][1] = 4; matrice[1][2] = 5;
```

- Syntaxe alternative d'instanciation/initialisation

```
6 int [][] matrice = {{0, 1, 2},{3, 4, 5}}
```

- Accès aux dimensions :

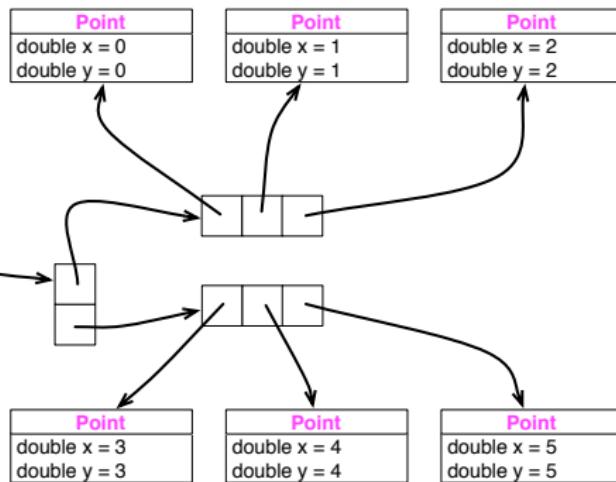
```
1 matrice.length // nb lignes
```

```
2 matrice[0].length // nb de colonnes de la première ligne
```

# TABLEAU À DEUX DIMENSIONS : VISION AVANCÉE

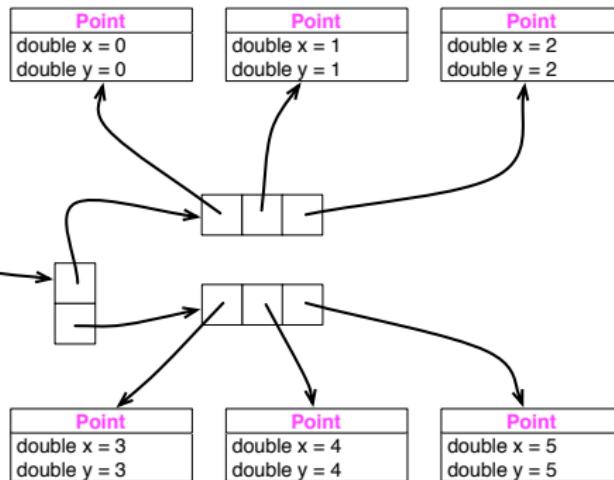
```
1 Point [][] matriceP = new Point [2][3];
```

Point[] matP

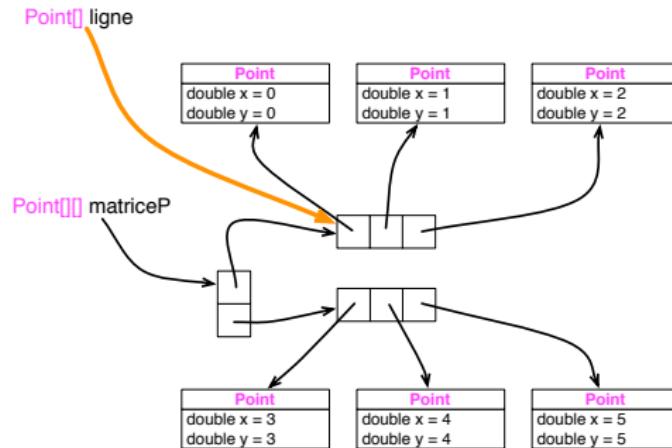


# TABLEAU À DEUX DIMENSIONS : VISION AVANCÉE

```
1 Point [][] matriceP = new Point [2][3];  
2 // est équivalent à :  
3 // création d'un tableau de tableau  
4 // de taille 2  
5 Point [][] matriceP2 = new Point [2][];  
6 // création de 2 tableaux de 3 cases  
7 for(int i=0; i<matriceP2.length; i++)  
8     matriceP2[i] = new Point [3];
```



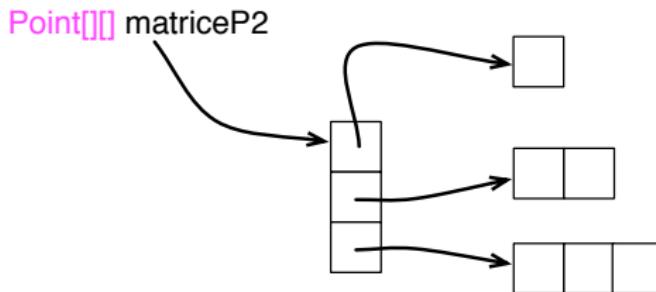
# TABLEAU À DEUX DIMENSIONS : VISION AVANCÉE (2)



- Possibilité de manipuler les lignes de la matrice de manière indépendante

```
1 Point [][] matriceP = new Point [2][3];  
2 Point [] ligne = matriceP [0];  
3 // Affichage du premier point:  
4 System.out.println(ligne [0]);
```

# TABLEAU À DEUX DIMENSIONS : VISION AVANCÉE (2)



- Possibilité de manipuler les lignes de la matrice de manière indépendante

```
1 Point [][] matriceP = new Point [2][3];  
2 Point [] ligne = matriceP [0];  
3 // Affichage du premier point:  
4 System.out.println(ligne [0]);
```

- Construction de matrice triangulaire

```
1 Point [][] matriceP2 = new Point [3][];  
2 for (int i=0; i<matriceP2.length; i++)  
3     matriceP2[i] = new Point [i+1];
```

# REMARQUE SUR LA CONVERSION

- Conversion sur les types de base : OK

```
1 double d = 2.4;  
2 int i = (int) d; // conversion explicite
```

- Conversion sur les tableaux : KO, impossible

```
1 double[] tab = {2.5, 5, 8.};  
2 // aucune conversion possible  
3 // seule option: reconstruction complète:  
4 int[] tabInt = new int[tab.length];  
5 for(int i=0; i<tab.length; i++)  
6   tabInt[i] = (int) tab[i];
```

Même fonctionnement avec les tableaux ou les ArrayList

Tableau = ensemble de variables

Pas de flexibilité à ce niveau là...

... A voir avec l'héritage

# 2i002 - Javadoc, déboggage... Vers plus d'autonomie

Vincent Guigue



# LES BONS REFLEXES...

- ① Lire les messages d'erreur dans la console
- ② Savoir corriger les erreurs les plus courantes
- ③ Savoir chercher dans la documentation officielle JAVA...
- ④ ... Et éventuellement documenter votre propre code

# COMPILATION ET EXÉCUTION

## Exemple d'instructions de compilation/exécution :

```
1 javac Point.java  
2 javac MainPoint.java  
3 java MainPoint
```

- Les deux premières instructions concernent la compilation...  
Même s'il n'y a pas de message d'erreur, il faut encore vérifier le bon fonctionnement du programme !
- La troisième ligne exécute le code compilé

# COMPILATEUR, JVM ET GARBAGE COLLECTOR

## ○ Compilateur

- **syntaxe** ( ;, parenthèses, ...)
- vérifie le **type des variables**,
- l'existence des méthodes/attributs et les niveaux d'accès :
  - les méthodes/attributs existent-elles dans l'objet,
  - les accès sont-ils permis (**public/private**)

## ○ JVM

- gestion dynamique des liens (cf redéfinition avec l'héritage)
- gestion des erreurs d'utilisation des objets
  - problème d'instanciation,
  - dépassement dans les tableaux,
  - gestion des fichiers...
- garbage collector (cf cycle de vie des objets)

# ERREURS USUELLES À CORRIGER SOIT MÊME

## Point

dessin

- x: double
- y: double
- Point(double,double)
- getX():double
- getY():double
- toString():String
- move(double,double):void

## Syntaxe

```
1 Point p = new Point(1,2)
2 p.move(1, 0);
3 // Syntax error, insert ";" to complete BlockStatements
```

## Compilation (les plus faciles !) :

Toujours bien regarder la ligne de l'erreur (elle est donnée). Trouver le raccourci de votre éditeur permettant d'aller à la ligne fautive

Il y a souvent plusieurs erreurs : toujours regarder la première, les autres sont peut-être simplement des conséquences de la première

# ERREURS USUELLES À CORRIGER SOIT MÊME

## Point

dessin

- x: double
- y: double
- Point(double,double)
- getX():double
- getY():double
- toString():String
- move(double,double):void

## Niveau d'accès

```
1 Point p = new Point(1,2);
2 p.x = 3;
3 // The field Point.x is not visible
```

### Compilation (les plus faciles !) :

Toujours bien regarder la ligne de l'erreur (elle est donnée). Trouver le raccourci de votre éditeur permettant d'aller à la ligne fautive

Il y a souvent plusieurs erreurs : toujours regarder la première, les autres sont peut-être simplement des conséquences de la première

# ERREURS USUELLES À CORRIGER SOIT MÊME

## Point

dessin

▪ x: double

▪ y: double

• Point(double,double)

• getX():double

• getY():double

• toString():String

• move(double,double):void

Compilation (les plus faciles !) :

Toujours bien regarder la ligne de l'erreur (elle est donnée). Trouver le raccourci de votre éditeur permettant d'aller à la ligne fautive

## Existence des méthodes

```
1 Point p = new Point(1,2);
2 p.mover(1,3);
3 // The method mover(int, int) is undefined for the type Point
4 p.move(1, 2, 3);
5 // The method move(double, double) in the type Point is
6 // not applicable for the arguments (int, int, int)
```

Il y a souvent plusieurs erreurs : toujours regarder la première, les autres sont peut-être simplement des conséquences de la première

# ERREURS USUELLES À CORRIGER SOIT MÊME

## Point

dessin

- x: double
  - y: double
- 
- Point(double,double)
  - getX():double
  - getY():double
  - toString():String
  - move(double,double):void

Compilation (les plus faciles !) :

Toujours bien regarder la ligne de l'erreur (elle est donnée). Trouver le raccourci de votre éditeur permettant d'aller à la ligne fautive

## Anticipation des erreurs de la JVM

```
1 Point p;  
2 p.move(1, 0);  
3 // The local variable p may not have been initialized
```

Il y a souvent plusieurs erreurs : toujours regarder la première, les autres sont peut-être simplement des conséquences de la première

# ERREURS USUELLES À CORRIGER SOIT MÊME

## Execution (JVM) : Toujours vérifier la ligne également

- **NullPointerException**

```
1 Point p = null;  
2 p.move(1, 0);  
3 // Exception in thread "main" java.lang.NullPointerException  
4 //      at cours1.TestPoint.main(TestPoint.java:11)
```

- Cette erreur arrive souvent dans des cas plus complexe de composition d'objet

- **IndexOutOfBoundsException**

```
1 int[] tab = new int[3];  
2 tab[3] = 2;  
3 // Exception in thread "main"  
4 //      java.lang.ArrayIndexOutOfBoundsException: 3  
5 //      at cours1.TestPoint.main(TestPoint.java:11)
```

- Vérifier la ligne + l'index !
- Souvent dans les boucle for

# DOCUMENTATION

Java est un langage très bien documenté et plein d'outils :

<https://docs.oracle.com/javase/8/docs/api/index.html>

The screenshot shows a web browser displaying the Java API Specification for Java Platform Standard Edition 8. The URL is https://docs.oracle.com/javase/8/docs/api/index.html. The page has a header with tabs for OVERVIEW, PACKAGE, CLASS, USE, TREE, DEPRECATED, INDEX, and HELP. The OVERVIEW tab is selected. The page content includes an introduction, a 'See' section, a 'Profiles' section, and a 'Packages' section. The 'Packages' section lists various Java packages with their descriptions.

Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
java.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.
java.awt.font	Provides classes and interface relating to fonts.
java.awt.geom	Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.
java.awt.im	Provides classes and interfaces for the input method framework.
java.awt.im.spi	Provides interfaces that enable the development of input methods that can be used with any Java runtime environment.
java.awt.image	Provides classes for creating and modifying images.
java.awt.image.renderable	Provides classes and interfaces for producing rendering-independent images.
java.awt.print	Provides classes and interfaces for a general printing API.
java.beans	Contains classes related to developing beans -- components based on the JavaBeans™ architecture.

# DOCUMENTATION

Java est un langage très bien documenté et plein d'outils :

<https://docs.oracle.com/javase/8/docs/api/index.html>

## ○ Cas 1 : objet connu, usage inconnu

- Quels sont les constructeurs d'une ArrayList ?
- Comment faire un cosinus (en se doutant que c'est dans la classe Math) ?
- ...

⇒ Chercher directement dans la classe au niveau de la javadoc

All Methods	Static Methods	Concrete Methods
Modifier and Type	Method and Description	
static double	<code>abs(double a)</code> Returns the absolute value of a double value.	
static float	<code>abs(float a)</code> Returns the absolute value of a float value.	
static int	<code>abs(int a)</code> Returns the absolute value of an int value.	
static long	<code>abs(long a)</code> Returns the absolute value of a long value.	
static double	<code>acos(double a)</code> Returns the arc cosine of a value; the returned angle is in the range 0.0 through pi.	
static int	<code>addExact(int x, int y)</code> Returns the sum of its arguments, throwing an exception if the result overflows an int.	
static long	<code>addExact(long x, long y)</code> Returns the sum of its arguments, throwing an exception if the result overflows a long.	

# DOCUMENTATION

Java est un langage très bien documenté et plein d'outils :

<https://docs.oracle.com/javase/8/docs/api/index.html>

- **Cas 1 : objet connu, usage inconnu**

- Quels sont les constructeurs d'une ArrayList ?
- Comment faire un cosinus (en se doutant que c'est dans la classe Math) ?
- ...

- **Cas 2 : recherche d'une fonctionnalité sans point d'entrée**

ex : Comment créer une image en JAVA ?

- Google...
  - recherche d'un point d'entrée ⇒ BufferedImage ⇒ retour à la javadoc
  - recherche d'un tutoriel (idéalement officiel) :

<https://docs.oracle.com/javase/tutorial/2d/images/>

# DOCUMENTATION... SUITE

De manière générale, **on programme pour les autres...**

⇒ documenter son code pour le rendre utilisable

- ① premier niveau : choisir des noms de classes, méthodes et variables **explicites**.
- ② deuxième niveau : faire des classes et des méthodes courtes, utiliser des méthodes privées... Eviter à tout prix les copier-coller.
- ③ troisième niveau : ajouter des commentaires pour créer une documentation.
  - Outil intégré dans JAVA : commentaires spéciaux + création automatique d'une page web

# CRÉATION D'UNE DOCUMENTATION

```
1 /**
2 * @author Vincent Guigue
3 * Cette classe permet de gérer des points en 2D
4 */
5 public class Point {
6 /**
7 * Attributs correspondant aux coordonnées du point
8 */
9 private double x, y;
10 /**
11 * Constructeur standard à partir de 2 réels
12 * @param x : abscisse du point
13 * @param y : coordonnée du point
14 */
15 public Point(double x, double y) {
16     this.x = x;
17     this.y = y;
18 }
19 /**
20 * @return l'abscisse du point
21 */
22 public double getX() {
23     return x;
24 }
25 }
```

```
$ javadoc Point.java
```

## JAVADOC : QUELQUES OPTIONS UTILES

- De manière générale : vérifier la documentation

```
$ javadoc -h
```

- Pour gérer les accents :

```
$ javadoc -encoding utf8 -docencoding utf8 -charset utf8 [fichier.java]
```

- Pour sélectionner le répertoire de stockage du html :

```
$ javadoc -d <directory> [fichier.java]
```

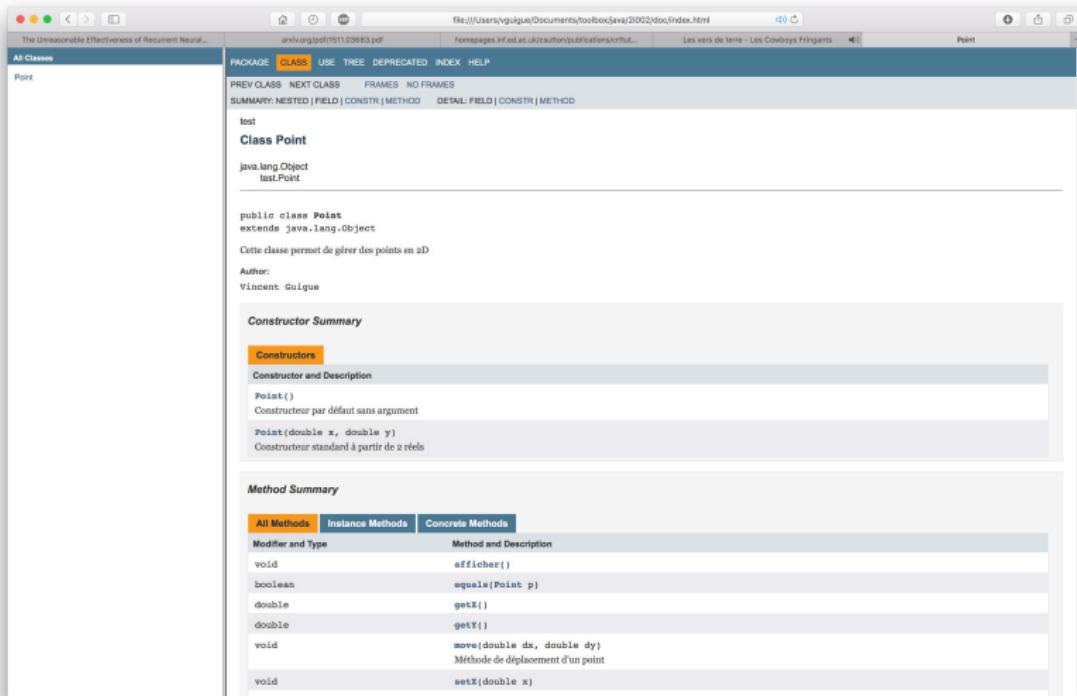
- Représentation public/private

(par défaut, représentation de la partie public seulement)

```
$ javadoc -public/-private [fichier.java]
```

# JAVADOC : RÉSULTATS OBTENUS

- Classe Point, présentation conforme à la javadoc standard (présence des liens hypertextes...)



# JAVADOC : RÉSULTATS OBTENUS

- Classe Point, description des méthodes (entrées, sorties...)

The screenshot shows a JavaDoc interface with the following details:

- Title Bar:** The Unreasonable Effectiveness of Recurrent Neural...
- Toolbar:** Standard Mac-style window controls.
- Address Bar:** arxiv.org/pdf/1511.03683.pdf
- File Path:** file:///Users/vguigue/Documents/toolbox/java/...
- Left Sidebar:** All Classes, Point
- Content Area:**
  - toString:** public java.lang.String toString()  
Overrides:  
toString in class java.lang.Object
  - move:** public void move(double dx,  
double dy)  
Méthode de déplacement d'un point  
Parameters:  
dx: - déplacement en x  
dy: - déplacement en y
  - afficher:** public void afficher()
  - setX:** (Method description not visible)

# JAVADOC : RÉSULTATS OBTENUS

## ○ Classe Point, représentation privée

The screenshot shows a Java Javadoc viewer window. The title bar indicates the file is located at `file:///Users/vguigue/Documents/toolbox/java/2i002/doc/index.html`. The browser address bar shows `arxiv.org/pdf/1511.03683.pdf`. The page content is for the `Point` class in the `test` package.

**Page Headers:** PACKAGE, CLASS (highlighted), TREE, DEPRECATED, INDEX, HELP  
PREV CLASS, NEXT CLASS, FRAMES, NO FRAMES  
SUMMARY: NESTED | FIELD | CONSTR | METHOD    DETAIL: FIELD | CONSTR | METHOD

**Class Summary:** `test`  
**Class Point**  
java.lang.Object  
    test.Point

**Code Snippet:**

```
public class Point
extends java.lang.Object
```

Cette classe permet de gérer des points en 2D

**Field Summary**

**Fields**

Modifier and Type	Field and Description
private double	<code>x</code> Attributs correspondant aux coordonnées du point
private double	<code>y</code> Attributs correspondant aux coordonnées du point

**Constructor Summary**

**Constructors**

# 2i002 - Exercices d'application

Vincent Guigue



# CYCLE DE VIE

- Le programme suivant compile-t-il ? Dans la négative, corriger la ou les erreurs.

```
1 Point p;
2 Point [] tab = {new Point(1,2), new Point(2,3), new Point(3,4)};
3 for (int i=0; i<3; i++){
4     p = tab[i];
5 }
6 System.out.println(i);
7 System.out.println(p);
8 tab[0] = tab[2];
```

## CYCLE DE VIE

- Le programme suivant compile-t-il ? Dans la négative, corriger la ou les erreurs.
- Une fois le programme fonctionnel, donner le nombre d'instance de Point en mémoire à la fin de l'exécution des lignes de code.

```
1 Point p;
2 Point [] tab = {new Point(1,2), new Point(2,3), new Point(3,4)};
3 for (int i=0; i<3; i++){
4     p = tab[i];
5 }
6 System.out.println(i);
7 System.out.println(p);
8 tab[0] = tab[2];
```

# CYCLE DE VIE

- Le programme suivant compile-t-il ? Dans la négative, corriger la ou les erreurs.
- Une fois le programme fonctionnel, donner le nombre d'instance de Point en mémoire à la fin de l'exécution des lignes de code.
- Comment faire pour garder un accès sur l'instance [1,2] ?

```
1 Point p;
2 Point [] tab = {new Point(1,2), new Point(2,3), new Point(3,4)};
3 for (int i=0; i<3; i++){
4     p = tab[i];
5 }
6 System.out.println(i);
7 System.out.println(p);
8 tab[0] = tab[2];
```

## CLONAGE D'OBJET COMPOSÉ

Soit une classe Segment composée (=ayant pour attributs) de deux Point.

- En imaginant qu'une classe de test permet de valider le bon fonctionnement du programme, combien de classe y a-t-il dans mon projet ?
- Donner le code de la classe (avec un constructeur de base) et les instructions de compilation.
- Donner le code de la méthode de clonage.
- Votre code est-il robuste au cas suivant ?

```
1 Point p1 = new Point(1,2); Point p2 = new Point(3, 4);
2 Segment s1 = new Segment(p1,p2);
3 Segment s2 = s1.clone();
4 p1.move(1,1);
```

- Donner le code de la méthode de test de l'égalité structurelle entre Segment.