Algorithmique et Programmation

Introduction aux classes, épisode 2

IUT Informatique de Bordeaux

Plan du cours

Dans l'épisode précédent...

Nous avons vu la notion de classe, avec :

- les attributs,
- le passage de référence.

Classe : définir, initialiser

Définir une classe

```
1 class Point {
2   double abs;
3   double ord;
4 }
```

Définir des variables

```
1 Point p1;
2 Point p2;
```

Initialiser des variables avec des instances

```
1 p1 = new Point();
2 p2 = new Point();
```

Accès aux attributs

```
1 p1.abs = 30.f;
2 p1.ord = 53.2f;
```

```
1 p2.abs = 0.4f;
2 p2.ord = 3.8f;
```

Classe : constructeurs

Définir un nouveau constructeur

```
1  Point(double abscisse, double ordonnee) {
2    abs = abscisse;
3    ord = ordonnee
4  }
```

```
1  Point p1 = new Point(30.f, 53.2f);
```

Redéfinir le constructeur par défaut (si on veut)

Si, par exemple, on connaît des valeurs par défaut pour tous les attributs :

```
1  Point() {
2    abs = 0.f;
3    ord = 0.f;
4  }
```

```
1  Point p1 = new Point();
2  p1.abs = 30.f; // on ecrase
```

Classe : constructeurs

Où placer un constructeur ?

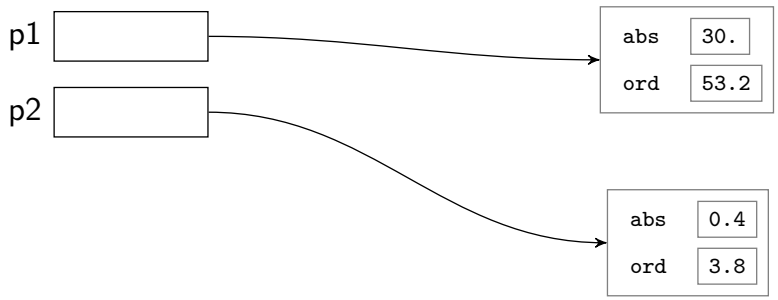
→ dans la classe !

```
1  class Point {
2
3    // Attributs
4    double abs;
5    double ord;
6
7    // Un premier constructeur
8    Point() {
9      abs = 0.f;
10     ord = 0.f;
11   }
12
13   // Un deuxieme constructeur
14   Point(double abscisse, double ordonnee) {
15     abs = abscisse;
16     ord = ordonnee
17   }
18 }
```

Variables, références, instances

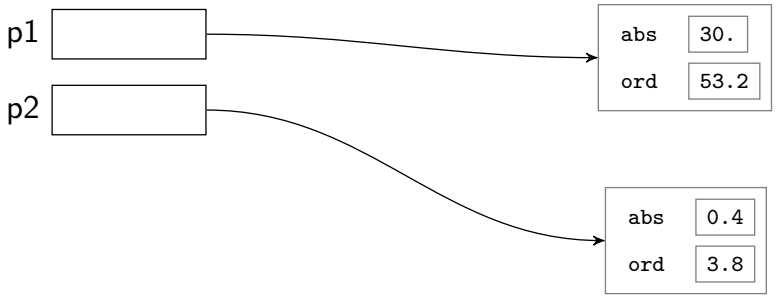
Deux variables peuvent désigner la même instance. Le contenu d'une variable (de type objet) est une référence vers une instance.

```
1  Point p1 = new Point(30.f, 53.2f);
2  Point p2 = new Point(0.4f, 3.8f);
```

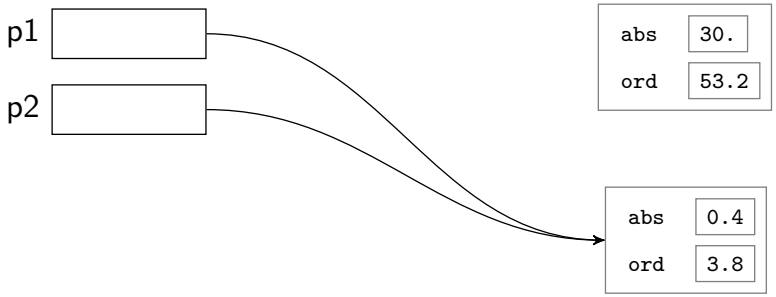


Variables, références, instances

```
1  Point p1 = new Point(30.f, 53.2f);
2  Point p2 = new Point(0.4f, 3.8f);
```




```
1  p1 = p2;
```



Remarque : un attribut peut être de n'importe quel type...

... y compris un objet, un tableau d'objets, etc.

```
1  class Polygone {
2    int      nbPoints;
3    Point[] sommets; // tableau de points
4
5    public static void main(String[] args) {
6      Polygone poly = new Polygone();
7      poly.sommets = new Point[30];
8      poly.nbPoints = 0;
9      ...
10   }
11 }
```

```
1  class Point {
2    double abs;
3    double ord;
4
5    ...
6  }
```

```
1  class Dessin {
2    Polygone[] polygones;
3    ...
4  }
```

Plan du cours

Méthodes

Dans une classe, on peut trouver :

- des attributs (déjà vu),
- des constructeurs (déjà vu), et
- des méthodes.

Une méthode est simplement une fonction déclarée dans une classe.

Méthodes

Dans une classe, on peut trouver :

- des attributs (déjà vu),
- des constructeurs (déjà vu), et
- des méthodes.

Une méthode est simplement une fonction déclarée dans une classe.

Ainsi le rôle de chacun est clair :

les attributs	stockent les données
les constructeurs	initialisent les instances
les méthodes	contiennent les traitements

Méthodes et attributs de classe OU d'instance

On distingue les méthodes :

- d'instance (= non statiques) : elles sont appliquées sur l'instance courante. On peut alors désigner cette instance avec le mot-clé "`this`".

Méthodes et attributs de classe OU d'instance

On distingue les méthodes :

- d'instance (= non statiques) : elles sont appliquées sur l'instance courante. On peut alors désigner cette instance avec le mot-clé "`this`".
- de classe (= statiques) : elles ne sont pas liées à une instance, mais à la classe. Ainsi, `this` est interdit, ainsi que l'accès aux attributs non statiques.

Méthodes et attributs de classe OU d'instance

On distingue les méthodes :

- d'instance (= non statiques) : elles sont appliquées sur l'instance courante. On peut alors désigner cette instance avec le mot-clé "`this`".
- de classe (= statiques) : elles ne sont pas liées à une instance, mais à la classe. Ainsi, `this` est interdit, ainsi que l'accès aux attributs non statiques.

C'est exactement la même chose pour les attributs.

On désigne les méthodes/attributs de classe avec le mot-clé "`static`".

Exemples...

Exemple Point : attributs

```
1  class Point {
2
3    // attributs d'instance :
4    double abs;
5    double ord;
6
7    // attributs de classe :
8    static int nbPoints = 0;
9
10   ...
11 }
```

→ `abs` est bien lié à un point (chaque point a une abscisse), mais pas `nbPoints`.

Exemple Point : constructeurs

```
1   ...
2
3   // Un premier constructeur
4   Point() {
5     abs = 0.f;
6     ord = 0.f;
7     nbPoints++; // attribut statique
8   }
9
10  // Un deuxieme constructeur
11  Point(double abscisse, double ordonnee) {
12    abs = abscisse;
13    ord = ordonnee;
14    nbPoints++; // attribut statique
15  }
16
17  ...
```

Exemple Point : méthode d'instance

```
1   ...
2
3   // methode d'instance :
4
5   /**
6    * Permet de calculer la distance a l'origine du point
7    * @return la distance a l'origine
8    */
9   double distance() {
10    return (double)Math.sqrt(abs*abs + ord*ord);
11  }
12
13  ...
14
15  public static void main(String[] args) {
16    Point p1 = new Point(10.f, 20.5f);
17    println("Distance entre p1 et l'origine " + p1.distance());
18    ...
19  }
20  ...
```

→ la distance entre un point et l'origine est bien liée à chaque point individuellement (à chaque instance).

Exemple Point : méthode de classe

```
1   ...
2
3   // methode de classe :
4
5   /**
6    * Nombre de points qui ont ete instancies au cours de l'execution du programme
7    * @return le nombre de points instancies
8    */
9   static int totalPoints() {
10    return nbPoints;
11  }
12  ...
13  public static void main(String[] args) {
14    Point p1 = new Point(10.f, 20.5f);
15    println("Nombre points instancies " + Point.totalPoints()); // affiche "Nombre ... 1"
16    Point p2 = new Point(10.f, 20.5f);
17    println("Nombre points instancies " + Point.totalPoints()); // affiche "Nombre ... 2"
18    ...
19  }
20
21 }
```

→ le nombre de `Point` instanciés lors de l'exécution du programme n'est pas lié à chaque `Point`.

Exercice

On souhaite écrire une méthode retournant le milieu de deux `Point`.

1. S'agit-il d'une méthode d'instance ou de classe ?
2. Implémentez la méthode.

Méthodes avec le même nom : la surcharge

Remarque :

on peut déclarer *plusieurs méthodes avec le même nom* (dans la même classe), pourvu qu'elles aient des signatures différentes.

```
1  double distance() {
2    return (double)Math.sqrt(abs*abs + ord*ord);
3  }
4
5  double distance(Point p) {
6    return ... ;
7  }
```

C'est la notion de surcharge.

Exercice

Écrire le corps de la méthode `double distance(Point p)`.

A vous de jouer : classe Polygone

Exercice

Écrire la classe `Polygone` telle que décrite ci-contre (sans le corps des méthodes).

Polygone
attributs : - nbPoints : int - sommets : Point[10]
méthodes (soulignée = méthode de classe) : + < *constructor* > Polygone() + ajouterPoint(p : Point) : void + perimetre() : int + carre(x : int, y : int, cote : int) : Polygone

Exercice

Ajouter un (ou plusieurs) attribut(s) de classe pour stocker les polygones instanciés et ajouter une méthode de classe retournant le polygone de plus grand périmètre.

NullPointerException

```
1  Point p = null;
2  ...
3  if (p.distance() > 0.f) { // leve un NullPointerException
4     ...
5  }
```

La version correcte :

```
1  Point p = new Point(...);
2  if (p.distance() > 0.f) {
3     ...
4  }
```

Plan du cours

Tester l'égalité de variables : types primitifs

Pour les types primitifs (`int`, `double`, `char`, etc), on utilise == :

```
1  int i, j;
2  char c, d;
3  ...
4  if (i == j || c == d) {
5     ...
6  }
```

Tester l'égalité de variables : objets

Quand deux variables de type objet sont-elles "égales" ?

- Si elles référencent la même instance, elles sont forcément égales. On peut le tester avec ==.
- Mais dans le cas général, il faut le préciser...

Par exemple pour les points on peut considérer que deux instances représentent le même point si elles ont les mêmes coordonnées.

Tester l'égalité de variables : objets

Pour cela tout objet possède une méthode `equals` que l'on peut redéfinir :

```
1  @Override
2  public boolean equals(Object obj) {
3      if (this == obj) { // meme instance => egaux
4          return true;
5      }
6      if (obj == null) { // this n'est jamais null, par definition...
7          return false;
8      }
9      if (getClass() != obj.getClass()) { // pas la meme classe => differents
10         return false;
11     }
12     final XXX other = (XXX) obj;     // on cast en XXX (a remplacer par le nom de la classe)
13
14     // On teste enfin les criteres (souvent sur les attributs) qui permettent d'affirmer
15     // que 2 points sont egaux.
16 }
```

On peut générer ce code automatiquement dans NetBeans : Alt+Insérer / equals() and hashcode() puis choisir les attributs discriminants.

Exercice

Écrire le corps de la méthode `public boolean equals(Object obj)` de la classe `Point`.

Tester l'égalité de variables : objets

```
1  Point p1 = new Point(...);
2  Point pointPlusProcheOrigine = Point.plusProcheOrigine();
3
4  if (p1.equals(pointPlusProcheOrigine)) {
5    ...
6  }
```

Si `Point.plusProcheOrigine()` renvoie une nouvelle instance de point, alors

```
1  p1 == pointPlusProcheOrigine
```

renvoie `false` même si

```
1  p1.equals(pointPlusProcheOrigine)
```

renvoie `true`.

Tester l'égalité de variables

Moralité :

Types primitifs : utiliser ==

```
1  int i, j;
2  ...
3  if (i == j) {
4     ...
5  }
```

Objets : utiliser equals()

```
1  Point p1, p2;
2  ...
3  if (p1.equals(p2)) {
4     ...
5  }
```

Seule exception : tester si une variable est `null` (car c'est bien la *référence* sur l'objet que l'on veut comparer à null).

```
1  if (p1 != null) { ... }
```

Plan du cours

Java

Comment s'exécute un programme Java ?

Dans les IDE comme NetBeans, c'est un peu caché : l'IDE s'occupe de la compilation et du lancement.

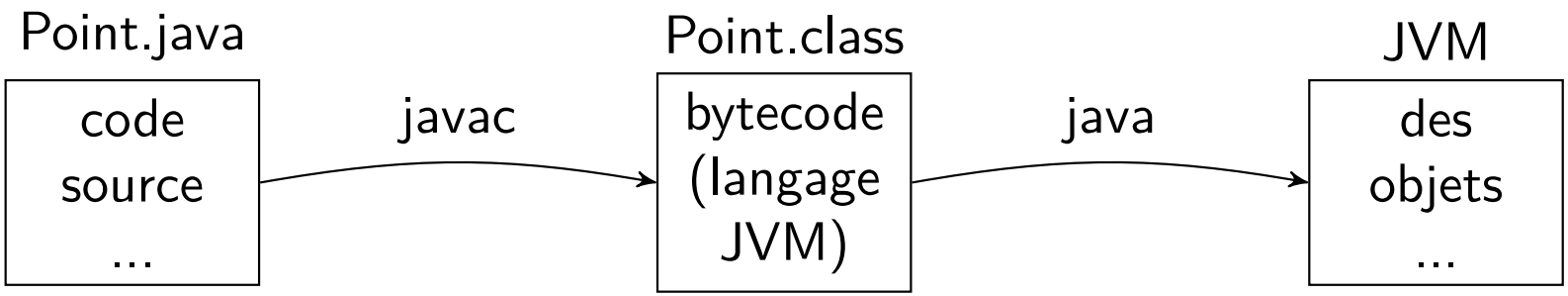
On vous montre les coulisses...

Java : compiler et exécuter

Un programme Java n'a pas besoin d'être recompilé pour chaque architecture d'ordinateur :

write once, run anywhere

Ceci est rendu possible grâce à la JVM : Java Virtual Machine.



Java : compiler et exécuter

En Java on exécute une classe. Elle doit posséder une méthode main() :

```
1  class Point {
2
3      ...
4
5      public static void main(String[] args) {
6          Point p1 = new Point(10.f, 12.f);
7          if (p1.distance() > 0.f) {
8              System.out.println("Point p1 (" + p1.abs + ", " + p1.ord + ") est a "
9                  + p1.distance() + " de l'origine.");
10         }
11     }
12     ...
13 }
```

C'est le point de départ de l'exécution.

Java : compiler et exécuter

1. Compiler le fichier Point.java (-cp = "classpath") :

```
$ tree .
.
├── bin
└── src
    └── gestionPoint
        └── Point.java

3 directories, 1 file
$ javac -cp src -d bin src/gestionPoint/Point.java
$ tree .
.
├── bin
│   └── gestionPoint
│       └── Point.class
└── src
    └── gestionPoint
        └── Point.java

4 directories, 2 files
```

2. Lancer la classe `gestionpoints.Point` :

```
$ java -cp bin gestionPoint.Point
Point p1 (10.0, 12.0) est a 15.6205 de l'origine.
```

Dernières remarques : public et import

public

```
1  public static void main(String[] args) {
2      ...
3  }
```

`public` signifie que la méthode (ou l'attribut) est *accessible en dehors de son package*.

Comme nous n'utilisons qu'un package, nous n'en avons pas besoin, sauf là où Java l'impose...

import

```
1  package gestionPoint;
2
3  import java.lang.Math;
4
5  class Point {
6    ...
7    double distance() {
8      // sqrt est une methode de classe
9      // de la classe Math
10     return (double)Math.sqrt(abs*abs + ord*ord);
11   }
12   ...
13 }
```

Ici, `import` sert à indiquer que la classe `Math` utilisée est celle du package `java.lang`.

Sinon il faudrait écrire : `java.lang.Math.sqrt` à la place de `Math.sqrt` partout.