M2.23.3 Algorithmique

Jean-François Berdjugin, Jean-François Remm
IUT1, Département SRC, L'Isle d'Abeau

Référence

- Cours de Jean-François Remm
- http://www.pps.jussieu.fr/~rifflet/JAVA

M2.23.3

- CM : 3 x 1,5h
- TD : 6 x 1,5h
- TP : 6 x 3h
- DS :
  - Table semaine 12 : 1,5h
  - Machine semaine 20 : 1,5h

Ce nous savons

Programmation fonctionnelle :
- Données
- Fonctions

Programme = fonctions + données

Pb les données ne sont pas corrélées aux fonctions => appliquer les fonctions sur les mauvaises données, accéder au fonctions lorsque leur nombre augmente, pas de nouveau types, …

=>

Données
int x
Int y

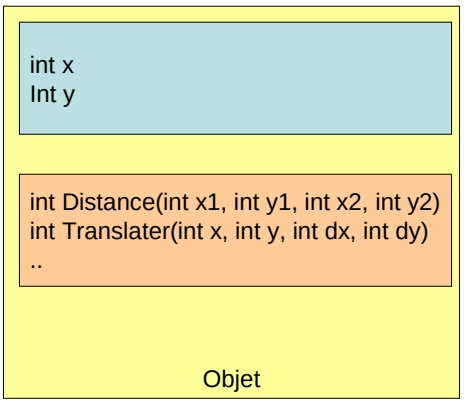
Fonctions
int Distance(int x1, int y1, int x2, int y2)
int Translater(int x, int y, int dx, int dy)
..

Programmation orientée objet

Encapsuler les fonctions et les données dans une boîte : l'objet

=>

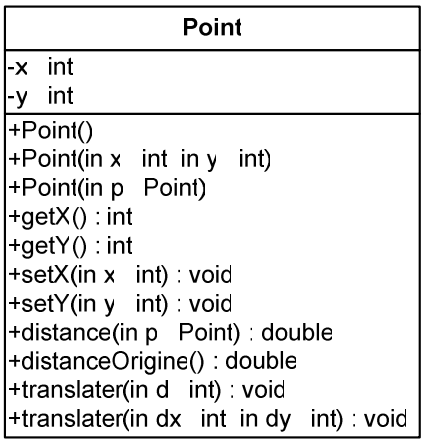
1. Regroupement des données avec leur fonction de traitement
2. Les fonctions modifie l'objet lui même
3. Nouveaux types objets
4. Autre moyen de factoriser le code (Héritage)
5. Nouvelles approches de conception



Classe

Classe : description STATIQUE d'une famille d'objets (un type) ayant même structure et même comportement (le moule des objets)

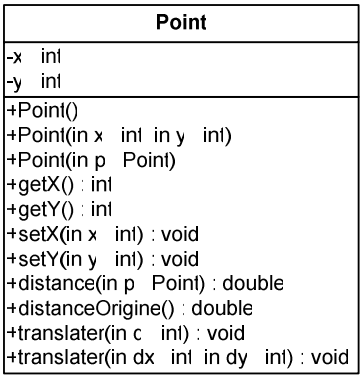
- Objet : entité créée DYNAMIQUEMENT, en respectant les plans de construction donnés par sa classe, une instance d'une classe.



Class Point

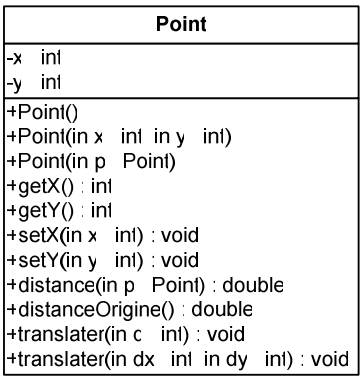
Définir une classe

1. Décrire l'**ETAT** (la structure) d'une instance : variables d'instances (ou propriétés ou attributs) : données caractérisant l'état d'une instance à un moment donné
2. D´écrire son **COMPORTEMENT** : Méthodes : fonctions spécialisées dans la manipulation des instances
3. **CONSTRUCTEURS** : méthodes spéciales construisant les objets



Définir une classe

1. **ETAT** : un point est défini par ces coordonnées x et y
2. **COMPORTEMENT** : les méthodes permettant de modifier ou d'interroger un état
3. **CONSTRUCTEURS** : le moyen de définir un état initial



Définir une classe

```
//code de la class Point
public class Point{
    int x;
    int y;
    Point() {
        x = 0;
        y = 0;
    }
}
```

```
//code de la class Point plus mieux
public class Point{
    private int x;
    private int y;
    Point() {
        this.x = 0;
        this.y = 0;
    }
}
```

Accessibilité aux membres d'un objet

accès depuis .... sur un membre	Private	rien	Protected	Public
la classe elle-même	Oui	Oui	Oui	Oui
sous-classe même package	Non	Oui	Oui	Oui
pas une sous-classe, même package	Non	Oui	Oui	Oui
une sous-classe d'un package différent	Non	Non	Oui	Oui
pas une sous-classe d'un package différent	Non	Non	Non	Oui

Accessibilité aux membres d'un objet

- Pour ce semestre toutes nos variables d'instances seront private (-)
- Nous utiliserons des méthodes public (+) pour les manipuler.

Point
-x : int -y : int
+Point() +Point(in x : int, in y : int) +Point(in p : Point) +getX() : int +getY() : int +setX(in x : int) : void +setY(in y : int) : void +distance(in p : Point) : double +distanceOrigine() : double +translater(in d : int) : void +translater(in dx : int, in dy : int) : void

Accessibilité aux membres d'un objet

this : permet de désigner l'objet dans lequel on se trouve.

Nous l'utiliserons systématiquement pour accéder aux variables d'instance

```
//code de la class Point plus mieux
public class Point{
    private int x;
    private int y;
    Point() {
        this.x = 0;
        this.y = 0;
    }
}
```

Constructeur

Des méthodes particulière de même nom que la classe qui ne retourne rien (même pas void)

Un mécanisme d'instanciation qui instancie les variables d'instance (« les initialisent »).

Sur notre exemple le point est créé en zéro (x=0) zéro (y=0)

```
//code de la class Point plus jolie
public class Point{
    private int x;
    private int y;
    Point() {
        this.x = 0;
        this.y = 0;
    }
}
```

Surcharge

Permet de définir plusieurs méthodes de même nom différentiées par leurs arguments (nombre, type et position)

```
public class Point{
    private int x;
    private int y;

    public Point() {
        this.x = 0;
        this.y = 0;
    }

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public Point(Point p) {
        this.x = p.getX();
        this.y = p.getY();
    }

}
```

surcharge

void maMethode() //Ok

int maMethode() //pas Ok type de retour non pris en compte

int maMethode(int a) //Ok

int maMethode(int a, int b) //Ok

int maMethode(double a, int b) //Ok

Int ma Mathode(int a, double b) //Ok

Méthodes

Les méthodes fixent le comportement, elles permettent d'obtenir ou de modifier l'état d'un objet.

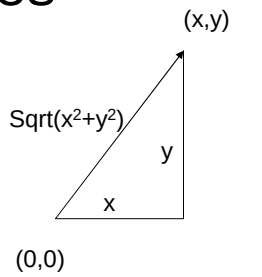
Méthodes

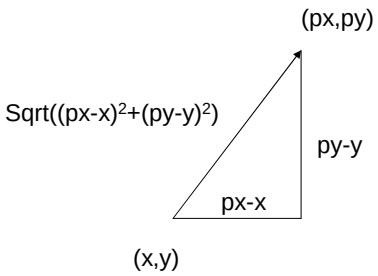
Ce n'est qu'une convention mais pour accéder aux variable d'instance, surtout si elles sont privées, l'on défini des « getters » ans « setters »

```
public int getX() {
        return this.x;
    }

    public int getY() {
        return this.y;
    }

    public void setX(int x) {
        this.x = x;
    }

    public void setY(int y) {
        this.y = y;
    }
```

Méthodes

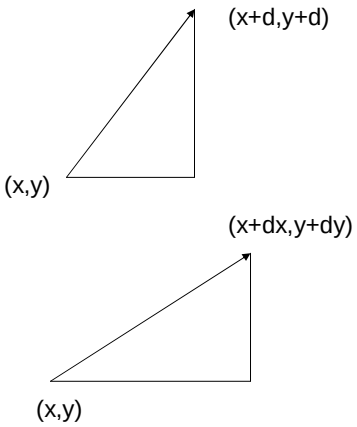
```
public double distanceOrigine() {
        double res;
        res = Math.sqrt((this.x *
    this.x) + (this.y*this.y));
        return res;
    }

    public double distance(Point p) {
        int px = p.getX();
        int py = p.getY();
        int dx = this.x - px;
        int dy = this.y - py;
        double res =
    Math.sqrt(dx*dx+dy*dy);
        return res;
    }
```





Méthodes

```
public void translater(int d)
   {
        this.x +=d;
        this.y +=d;
   }

   public void translater(int
   dx, int dy)
   {
        this.x +=dx;
        this.y +=dy;
   }
```



Méthodes

Nous allons réécrire notre en réutilisant au maximum nos propres méthodes.

Un constructeur peut faire appel à un autre constructeur en utilisant **this**(pramètres).

Une méthode peut faire appel à une autre méthode de la classe par son nom. Pour prendre des bonnes habitudes nous utiliserons **this.**nom(paramètres)

Méthodes

- Nous avons un constructeur « plus généralise » que les autres, nous pouvons le réutiliser

```
public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
```

Méthodes

```
public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
```

```
public Point() {
        this(0,0);
    }

public Point(Point p) {
this(p.getX(),p.getY());
    }
```

Méthodes

Nous pouvons faire de même avec la translation

```
public void translater(int dx, int dy)
{
    this.x +=dx;
    this.y +=dy;
}
```

Méthodes

```
public void translater(int dx, int dy)
{
    this.x +=dx;
    this.y +=dy;
}
```

```
public void translater(int d)
{
this.translater(d,d);
}
```

Objets

- Les classes ont des instances les objets (ils sont créés dynamiquement).

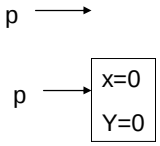
- Les objet sont créés en utilisant un des constructeurs de la classe
- Le mot clef est **new**

```
public class TestPoint() {
    public static void main(String[] args) {
            Point p  //p est déclaré comme étant de type point
            p = new Point(); //p est instancié grâce au constructeur par défaut

            Point p2 = new Point(p); //déclaration est instanciation de p2 au même
    coordonnées que p

            Point p3 = new Point(10,20); // déclaration de p3 en 10,20
            Point p4 = new Point(10,20); //déclaration de p4 en 10,20
    }
}
```

Objets

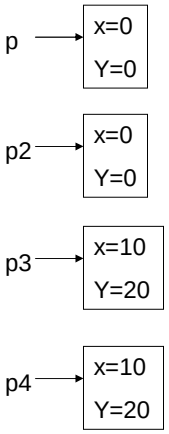
Point p //p est déclaré comme étant de type point

p = new Point(); //p est instancié grâce au constructeur par défaut



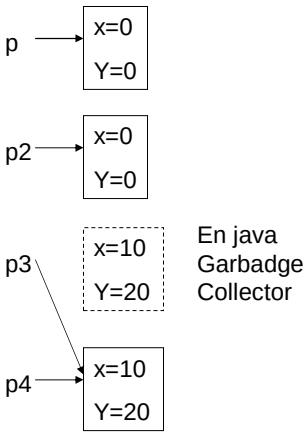
Objets

```
Point p
p = new Point();

Point p2 = new Point(p);

Point p3 = new
    Point(10,20);

Point p4 = new
    Point(10,20);

p3=p4 ?
```



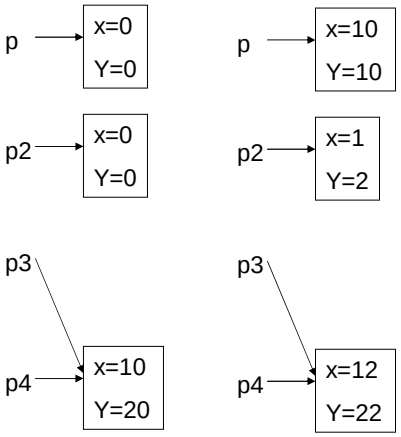
Objets

```
Point p
p = new Point();

Point p2 = new Point(p);

Point p3 = new
    Point(10,20);

Point p4 = new
    Point(10,20);

p3=p4 !
```



Objets

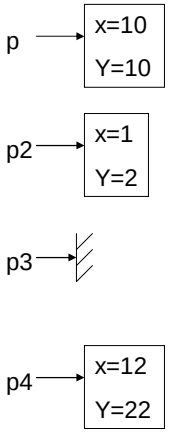
p.tanslater(10);
p2.translater(1,2);
p3.translater(1,1);
p4.translater(1);



Objets

p3=null

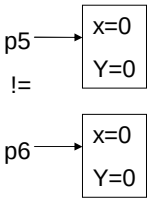
P3 ne référence plus rien



Objets

Point p5 = new Point();
Point p6 = new Point();

P6==p5 ?
Non



Méthodes (suite)

- La distance à l'origine peut être déduite de la distance entre deux points. Il suffit de prendre le point origine

```
public double distance(Point
    p) {
        int px = p.getX();
        int py = p.getY();
        int dx = this.x - px;
        int dy = this.y - py;
        double res =
    Math.sqrt(dx*dx+dy*dy);
        return res;
    }
```

Méthodes (suite)

```
public double distance(Point p) {
        int px = p.getX();
        int py = p.getY();
        int dx = this.x - px;
        int dy = this.y - py;
        double res = Math.sqrt(dx*dx+dy*dy);
        return res;
    }
```

```
public double distanceOrigine()
{
return this.distance(new Point());
}
```

Reste à voir

- Lors du cours de programmation nous compléterons par :
  - L'héritage : un moyen de factoriser du code
  - Le polymorphisme : un changement de type à l'exécution
  - Les variable et méthodes de classe
  - La généricité : la possibilité de passer un type en paramètre

  …
  - Swing