

# Fms Academy 2022

## Formation Java Spring Angular

### Module 5 : Programmation Orientée Objet

# Sommaire

2

- C'est quoi la POO ou OOP ?
- Concrètement !
- L'intérêt d'utiliser UML avec la POO
- Les classes & objets, attributs et méthodes, constructeurs
- Encapsulation
- Utilité de l'héritage
- Polymorphisme
- Diagramme de classe
- Classe & méthode abstraite
- Interface
- Instanceof & Cast
- Consignes pour Exo et Tp

# POO

3

- **La programmation orientée objet** est un modèle de programmation informatique. Plutôt que d'organiser la conception logicielle autour de fonctions ou de logique, elle consiste à l'organiser autour des données ou des « objets »
- **Un objet** représente un concept, une idée ou toute entité du monde physique, comme une voiture, une personne ou encore une page d'un livre.

# Concrètement



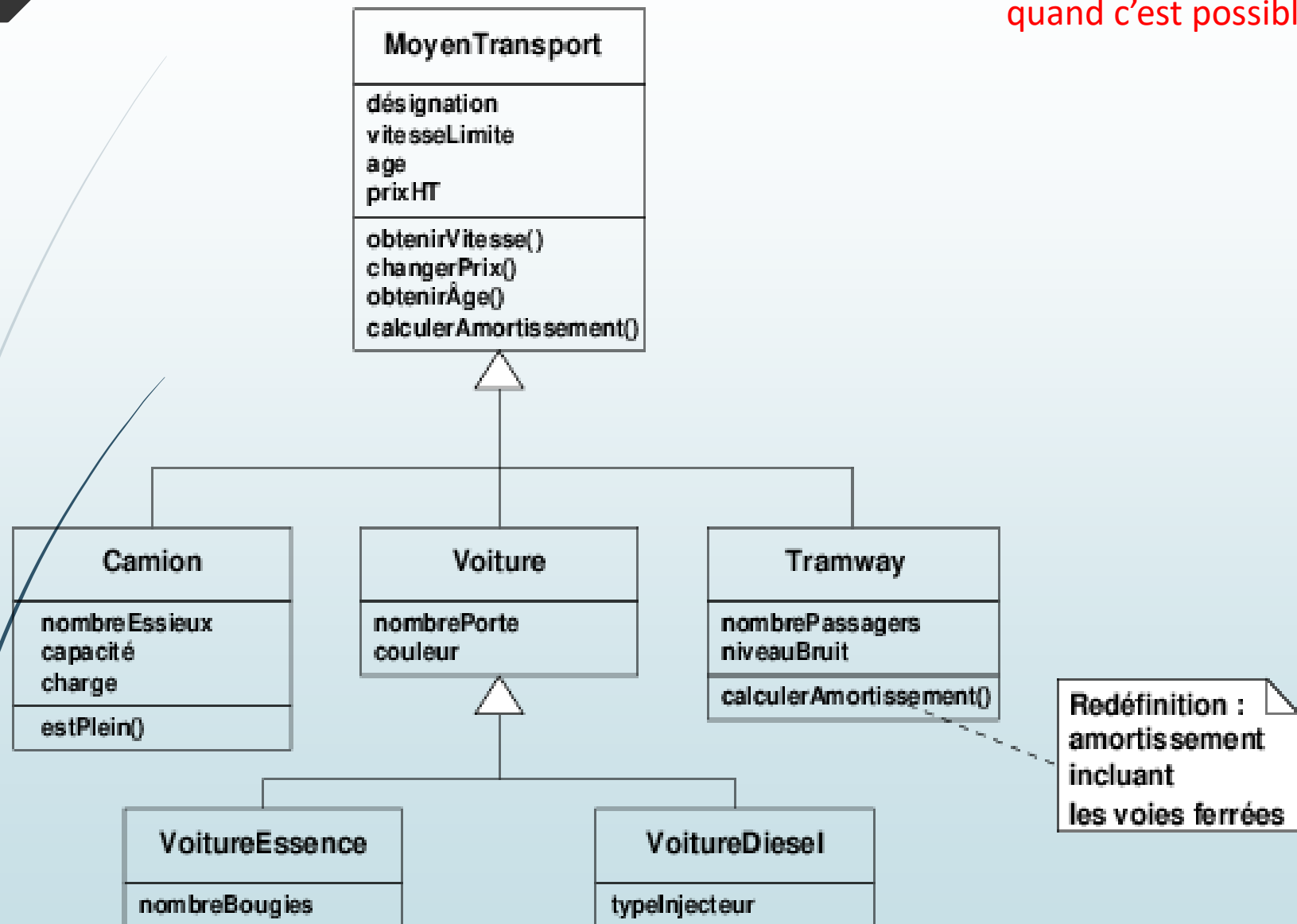
- On souhaite **représenter le monde réel virtuellement** à l'aide d'objet, pour ce faire on se base sur les principales caractéristiques d'un objet :
  - → Attributs ou **propriétés**
  - → Comportements ou **méthodes**



# L'intérêt d'utiliser UML avec la POO

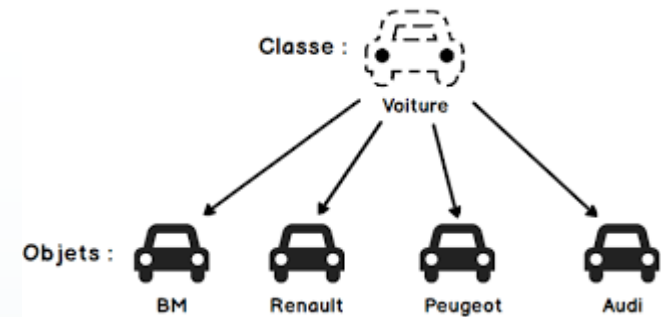
5

Uml contribue ici à organiser ou classier nos objets pour en faciliter l'utilisation notamment en factorisant quand c'est possible.





# Les classes & objets, attributs & méthodes, constructeurs



Package Expl...

Poo

JRE System Library [Java]

src

fr.lidnr.poo

Car.java

TestCar.java

ProjetHello

ProjetStatic

StockSwingApp

Car.java

```

1 package fr.lidnr.poo;
2
3 public class Car {
4     //attributs
5     public String brand; //marque ex Renault
6     public String type; //type ex Clio
7     public double price; //prix
8
9     final double MIN_CAR_PRICE = 1000; //constante
10
11     //constructeurs
12     public Car(String brand, String type, double price) {
13         this.brand = brand;
14         this.type = type;
15         this.price = price;
16     }
17     public Car(String brand) {
18         this.brand = brand;
19         this.type = "unknown";
20         this.price = MIN_CAR_PRICE;
21     }
22     public Car() {
23         this("unknown");
24     }
25
26     //méthodes
27     public void display() {
28         System.out.println("brand : " + this.brand + "\t" +
29                             "type : " + type + "\t" +
30                             "price : " + this.price);
31     }
32
33     // Est-ce possible ?
34     public static void main(String[] args) {
35         Car car = new Car("Renault", "Clio", 5000);
36         car.display();
37     }
38 }
39

```

TestCar.java

```

1 package fr.lidnr.poo;
2
3 public class TestCar {
4     public static void main(String[] args) {
5         Car car1 = new Car();
6         Car car2 = new Car("Renault", "Clio", 5000);
7         Car car3 = new Car("Peugeot", "206", 7000);
8         Car car4 = new Car("BM");
9
10        car1.display();
11        car2.display();
12        car3.display();
13        car4.display();
14
15        System.out.println();
16        //comment résoudre ce problème ?
17        car2.price = -4500;
18        car2.display();
19    }
20 }
21

```

Console

```

<terminated> TestCar [Java Application] C:\Program Files\Java\j
brand : unknown type : unknown price : 1000.0
brand : Renault type : Clio price : 5000.0
brand : Peugeot type : 206 price : 7000.0
brand : BM type : unknown price : 1000.0

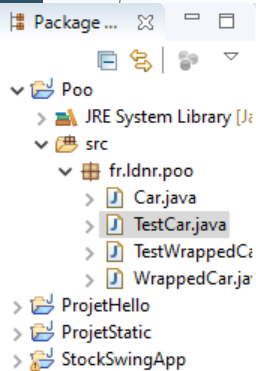
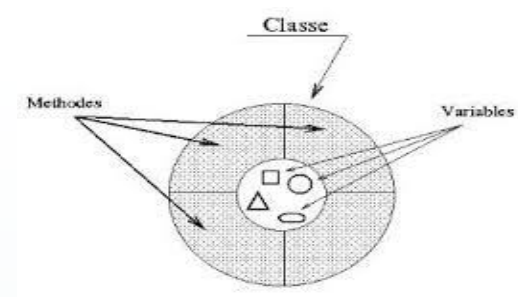
brand : Renault type : Clio price : -4500.0

```

**NB : Afin de faciliter l'utilisation de la Poo et apporter plus de lisibilité, le système de package permet d'organiser une application en couches distinctes qui peuvent communiquer entre elles selon la notion de portée.**

**Delors, on peut trouver le même nom de méthodes dans 2 packages différents !**

# Encapsulation !



```
WrappedCar.java
1 public class WrappedCar {
2     //attributs
3     private String brand; //marque ex Renault
4     private String type; //type ex Clio
5     private double price; //prix
6
7     public static final double MIN_CAR_PRICE = 1000; //constante
8
9     //constructeurs
10    public WrappedCar(String brand, String type, double price) {
11        setBrand(brand);
12        setType(type);
13        setPrice(price);
14    }
15    public WrappedCar(String brand, String type) {
16        this(brand, type, MIN_CAR_PRICE);
17    }
18
19    //accesseurs
20    public String getBrand() {
21        return brand;
22    }
23    public void setBrand(String brand) {
24        this.brand = brand;
25    }
26    public String getType() {
27        return type;
28    }
29    public void setType(String type) {
30        this.type = type;
31    }
32    public double getPrice() {
33        return price;
34    }
35    public void setPrice(double price) {
36        if (price < 0)
37            throw new RuntimeException("Le prix ne peut être négatif !");
38        this.price = price;
39    }
40
41    //méthodes
42    public String toString() {
43        return "brand : " + getBrand() + "\t" +
44            "type : " + getType() + "\t" +
45            "price : " + this.getPrice();
46    }
47 }
```

```
TestWrappedCar.java
1 package fr.lidnr.poo;
2
3 public class TestWrappedCar {
4     public static void main(String[] args) {
5
6         WrappedCar car = new WrappedCar("Renault", "Clio", 5000);
7
8         System.out.println(car + "\n");
9         //car.price = -4500; //execution Impossible : le champ n'est pas visible
10        car.setPrice(-3000);
11        System.out.println(car);
12    }
13 }
```

```
Console
<terminated> TestWrappedCar [Java Application] C:\Program Files\Java\jre1.8.0_191\bin\javaw.exe (22 avr. 2021 à 15:
brand : Renault type : Clio price : 5000.0

Exception in thread "main" java.lang.RuntimeException: Le prix ne peut être négatif !
    at fr.lidnr.poo.WrappedCar.setPrice(WrappedCar.java:39)
    at fr.lidnr.poo.TestWrappedCar.main(TestWrappedCar.java:11)
```

# Utilité de l'héritage

8

Q1 : Que pensez vous de ces 2 classes ?

Q2 : Que va afficher la console à l'exécution ?

```

Square.java
1 package fr.ldnr.entities;
2
3 public class Square {
4     private int x;
5     private int y;
6     private int side;
7
8     public Square(int side,int x, int y) {
9         setSide(side);
10        setX(x);
11        setY(y);
12    }
13
14    public Square(int side,Point center) {
15        setSide(side);
16        setX(0);
17        setY(0);
18    }
19
20    public int getX() {
21        return x;
22    }
23    public void setX(int x) {
24        this.x = x;
25    }
26    public int getY() {
27        return y;
28    }
29    public void setY(int y) {
30        this.y = y;
31    }
32    public int getSide() {
33        return side;
34    }
35    public void setSide(int side) {
36        if(side < 0) side = 1;
37        else this.side = side;
38    }
39 }
40

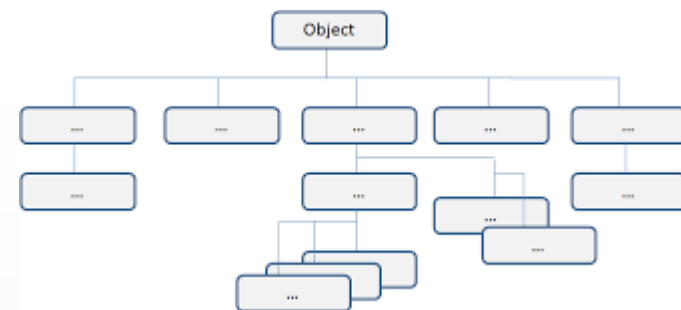
Circle.java
1 package fr.ldnr.entities;
2
3 public class Circle {
4     private int x;
5     private int y;
6     private double radius;
7
8     public Circle(double radius, int x, int y) {
9         setRadius(radius);
10        setX(x);
11        setY(y);
12    }
13    public Circle() {
14        setRadius(1);
15        setX(0);
16        setY(0);
17    }
18
19    public int getX() {
20        return x;
21    }
22    public void setX(int x) {
23        this.x = x;
24    }
25    public int getY() {
26        return y;
27    }
28    public void setY(int y) {
29        this.y = y;
30    }
31    public double getRadius() {
32        return radius;
33    }
34    public void setRadius(double radius) {
35        if(radius < 0 ) radius = 1;
36        else this.radius = radius;
37    }
38 }
39

TestShapes.java
1 package fr.ldnr.entities;
2
3 public class TestShapes {
4     public static void main(String[] args) {
5         Circle c1 = new Circle(2.0,3,7);
6         System.out.println(c1);
7
8         Square s1 = new Square(3,7,10);
9         System.out.println(s1);
10    }
11 }
```



# Utilisation de l'héritage

9



```
Shape.java
1 package fr.l2nr.entities;
2 //import java.awt.Point;
3
4 public class Shape {
5     private Point center;
6
7     public Shape(int x, int y) {
8         this.center = new Point(x,y);
9     }
10
11     public Shape(Point center) {
12         this.center = new Point(center.getX(),center.getY());
13     }
14
15     @Override
16     public String toString() {
17         return "Shape [center = " + center + "]";
18     }
19 }
20
```

```
Point.java
1 package fr.l2nr.entities;
2
3 public class Point {
4     private int x;
5     private int y;
6
7     public Point(int x, int y) {
8         this.x = x;
9         this.y = y;
10    }
11    public int getX() {
12        return x;
13    }
14    public void setX(int x) {
15        this.x = x;
16    }
17    public int getY() {
18        return y;
19    }
20    public void setY(int y) {
21        this.y = y;
22    }
23 }
```

```
Square.java
1 package fr.l2nr.entities;
2
3 public class Square extends Shape {
4     private int side;
5
6     public Square(int side,int x, int y) {
7         super(x, y);
8         this.setSide(side);
9     }
10
11     public Square(int side,Point center) {
12         super(center);
13         this.setSide(side);
14     }
15
16     public int getSide() {
17         return side;
18     }
19
20     public void setSide(int side) {
21         if(side < 0) side = 1;
22         else this.side = side;
23     }
24
25     public double area() {
26         return this.side * this.side;
27     }
28
29     public double perimeter() {
30         return 4 * side;
31     }
32
33     @Override
34     public String toString() {
35         return "Square side : " + side + super.toString();
36     }
37 }
38
```

```
Circle.java
1 package fr.l2nr.entities;
2
3 public class Circle extends Shape {
4     private double radius;
5
6     public Circle(double radius, int x, int y) {
7         super(x,y);
8         setRadius(radius);
9     }
10
11     public Circle(double radius, Point center) {
12         super(center);
13         setRadius(radius);
14     }
15
16     public Circle() {
17         super(0,0);
18         setRadius(1);
19     }
20
21     public double getRadius() {
22         return radius;
23     }
24
25     public void setRadius(double radius) {
26         if(radius < 0 ) radius = 1;
27         else this.radius = radius;
28     }
29
30     public double area() {
31         return Math.PI * this.radius * this.radius;
32     }
33
34     public double perimeter() {
35         return 2 * Math.PI * this.radius;
36     }
37
38     @Override
39     public String toString() {
40         return "Circle : radius = " + radius + super.toString();
41     }
42 }
43
```

Diagram illustrating the relationships and code structure for three Java classes: Shape, Circle, and Square, along with a test class TestShapes.

**Shape.java**

```
1 package fr.lidnr.entities;
2
3 public class Shape {
4     private Point center;
5
6     public Shape(int x, int y) {
7         this.center = new Point(x,y);
8     }
9     public Shape(Point center) {
10         this.center = new Point(center.getX(),center.getY());
11     }
12
13     @Override
14     public String toString() {
15         return "Shape [center = " + center + "]";
16     }
17 }
18
```

**Circle.java**

```
1 package fr.lidnr.entities;
2
3 public class Circle extends Shape {
4     private double radius;
5
6     public Circle(double radius, int x, int y) {
7         super(x,y);
8         setRadius(radius);
9     }
10
11     public Circle(double radius, Point center) {
12         super(center);
13         setRadius(radius);
14     }
15
16     public Circle() {
17         super(0,0);
18         setRadius(1);
19     }
20
21     public double getRadius() {
22         return radius;
23     }
24
25     public void setRadius(double radius) {
26         if(radius < 0 ) radius = 1;
27         else this.radius = radius;
28     }
29
30     public double area() {
31         return Math.PI * this.radius * this.radius;
32     }
33
34     public double perimeter() {
35         return 2 * Math.PI * this.radius;
36     }
37
38     @Override
39     public String toString() {
40         return "Circle : radius = " + radius + super.toString();
41     }
42 }
43
```

**Square.java**

```
1 package fr.lidnr.entities;
2
3 public class Square extends Shape {
4     private int side;
5
6     public Square(int side,int x, int y) {
7         super(x, y);
8         this.setSide(side);
9     }
10
11     public Square(int side,Point center) {
12         super(center);
13         this.setSide(side);
14     }
15
16     public int getSide() {
17         return side;
18     }
19
20     public void setSide(int side) {
21         if(side < 0) side = 1;
22         else this.side = side;
23     }
24
25     public double area() {
26         return this.side * this.side;
27     }
28
29     public double perimeter() {
30         return 4 * side;
31     }
32
33     @Override
34     public String toString() {
35         return "Square side : " + side + super.toString();
36     }
37 }
38
```

**TestShapes.java**

```
1 package fr.lidnr.entities;
2
3 public class TestShapes {
4     public static void main(String[] args) {
5         Circle c1 = new Circle(2.0,3,7);
6         System.out.println(c1);
7
8         Square s1 = new Square(3,7,10);
9         System.out.println(s1);
10
11         Point p = new Point(3,15);
12         Circle c2 = new Circle(1.5,p);
13         System.out.println(c2);
14     }
15 }

```

**Console Output:**

```
<terminated> TestShapes [Java Application] C:\Program Files\Java\jre1.8.0_191\bin\javaw.exe (22
Circle : radius = 2.0 Shape [center = x= 3, y=7]
Square side : 3 Shape [center = x= 7, y=10]
Circle : radius = 1.5 Shape [center = x= 3, y=15]
```

**Diagram Annotations:**

- 1: Arrow from `new Circle(2.0,3,7);` in TestShapes.java to `Circle(double radius, int x, int y)` in Circle.java.
- 2: Arrow from `this.center = new Point(x,y);` in Shape.java to `super(x,y);` in Circle.java.
- 3: Arrow from `setRadius(radius);` in Circle.java to `setRadius(double radius)` in Circle.java.
- 4: Arrow from `new Square(3,7,10);` in TestShapes.java to `Square(int side,int x, int y)` in Square.java.
- 5: Arrow from `new Circle(1.5,p);` in TestShapes.java to `Circle(double radius, Point center)` in Circle.java.

# Polymorphisme

Caractère de ce qui peut avoir ou adopter plusieurs formes différentes.



POURQUOI FAIRE ?

On souhaite parcourir une liste de figures géométriques et connaître leur nature originelle.

Shape.java

```
1 package fr.l2nr.entities;
2 //import java.awt.Point; -> Retirons le commentaire
3
4 public class Shape {
5     private Point center;
6
7     public Shape(int x, int y) {
8         this.center = new Point(x,y);
9     }
10
11     public Shape(Point center) {
12         this.center =
13             new Point(center.getX(),center.getY());
14     }
15
16     @Override
17     public String toString() {
18         return "Shape [center = " + center + "]";
19     }
20 }
21
```

TestShapes.java

```
1 package fr.l2nr.entities;
2
3 import java.util.ArrayList;
4
5 public class TestShapes {
6     public static void main(String[] args) {
7         Shape[] shapes = new Shape[5];
8         shapes[0] = new Circle(1.5,new Point(5,5));
9         shapes[1] = new Square(3.2,new Point(1,2));
10        shapes[2] = new Circle();
11        for( Shape s : shapes ) {
12            System.out.println(s);
13            //System.out.println(s.getRadius()); OOPS !
14        }
15        /*-----*/
16        System.out.println("-----");
17
18        Circle c1 = new Circle(2.0,3,7);
19        Shape s1 = new Square(3.2,7,10);
20        Point p = new Point(3,15);
21        Object c2 = new Circle(1.5,p);
22
23        ArrayList<Object> objects = new ArrayList<>();
24        objects.add(c1);
25        objects.add(s1);
26        objects.add(c2);
27        objects.add(p);
28        objects.add(new Shape(2,3));
29        for( Object o : objects ) {
30            if(o instanceof Point) System.out.print("Point ");
31            System.out.println(o);
32        }
33    }
34 }
35
```

Circle.java

```
1 package fr.l2nr.entities;
2
3 public class Circle extends Shape {
4     private double radius;
5
6     public Circle(double radius, int x, int y) {
7         super(x,y);
8         setRadius(radius);
9     }
10
11     public Circle(double radius, Point center) {
12         super(center);
13         setRadius(radius);
14     }
15
16     public Circle() {
17         super(1,1);
18         setRadius(1);
19     }
20
21     public double getRadius() {
22         return radius;
23     }
24
25     public void setRadius(double radius) {
26         if(radius < 0 ) radius = 1;
27         else this.radius = radius;
28     }
29
30     @Override
31     public String toString() {
32         return "Circle : radius = " + radius + " " + super.toString();
33     }
34 }
35
```

Square.java

```
1 package fr.l2nr.entities;
2
3 public class Square extends Shape {
4     private double side;
5
6     public Square(double side, int x, int y) {
7         super(x,y);
8         setSide(side);
9     }
10
11     public Square(double side, Point center) {
12         super(center);
13         setSide(side);
14     }
15
16     public Square() {
17         super(1,1);
18         setSide(1);
19     }
20
21     public double getSide() {
22         return side;
23     }
24
25     public void setSide(double side) {
26         if(side < 0 ) side = 1;
27         else this.side = side;
28     }
29
30     @Override
31     public String toString() {
32         return "Square : side = " + side + " " + super.toString();
33     }
34 }
35
```

Point.java

```
1 package fr.l2nr.entities;
2
3 public class Point {
4     private int x;
5     private int y;
6
7     public Point(int x, int y) {
8         this.x = x;
9         this.y = y;
10    }
11
12    @Override
13    public String toString() {
14        return "[x=" + x + ",y=" + y + "]";
15    }
16}
17
```

Console

```
<terminated> TestShapes [Java Application] C:\Program Files\Java\jre1.8.0_191\bin
Circle : radius = 1.5 Shape [center = [x=5,y=5]]
Square : side = 3.2 Shape [center = [x=1,y=2]]
Circle : radius = 1.0 Shape [center = [x=1,y=1]]
null
null
-----
Circle : radius = 2.0 Shape [center = [x=3,y=7]]
Square : side = 3.2 Shape [center = [x=7,y=10]]
Circle : radius = 1.5 Shape [center = [x=3,y=15]]
Point [x=3,y=15]
Shape [center = [x=2,y=3]]

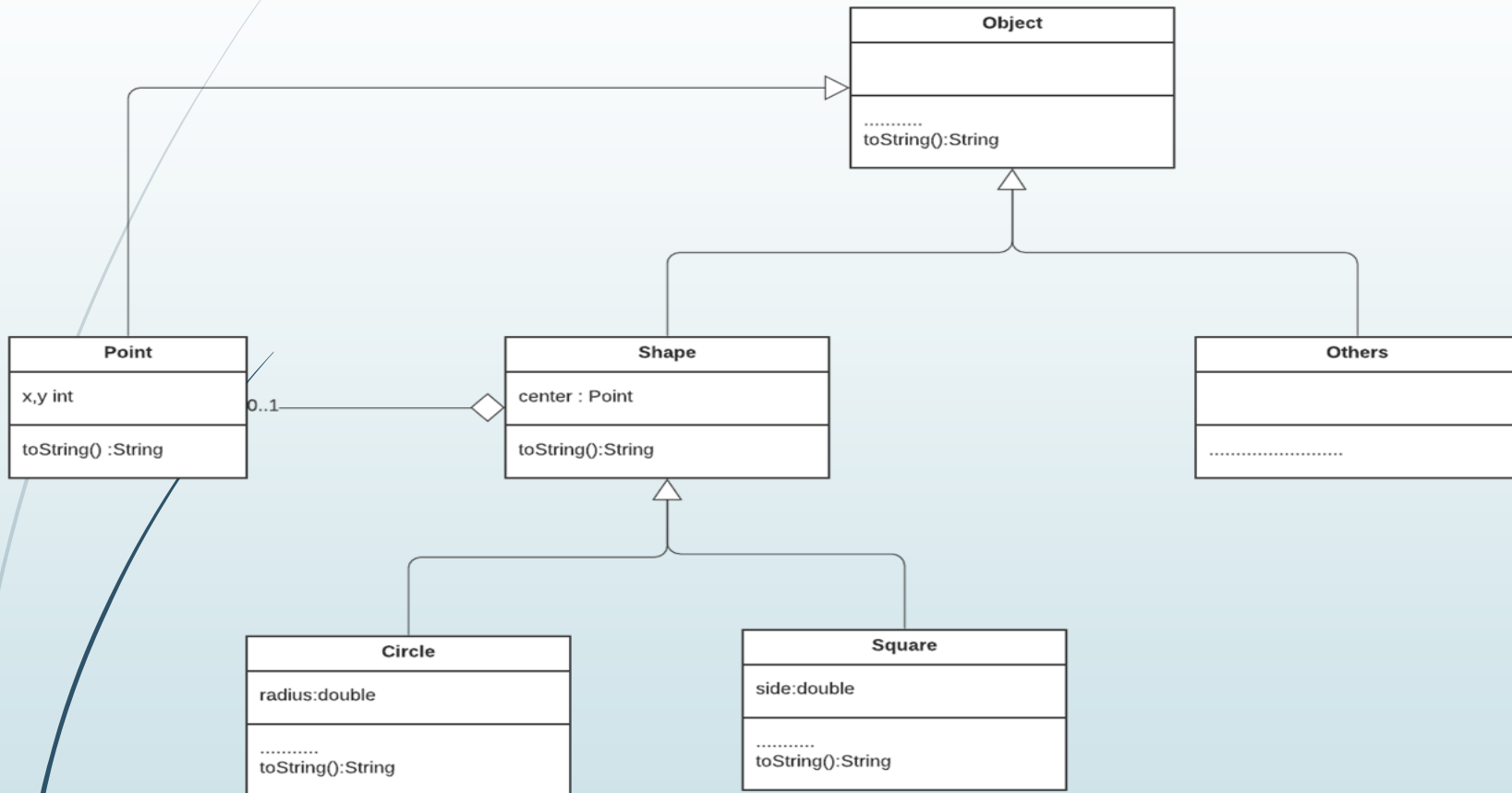
```

# Diagramme de classe

shape

Mohamed Elbabili | April 26, 2021

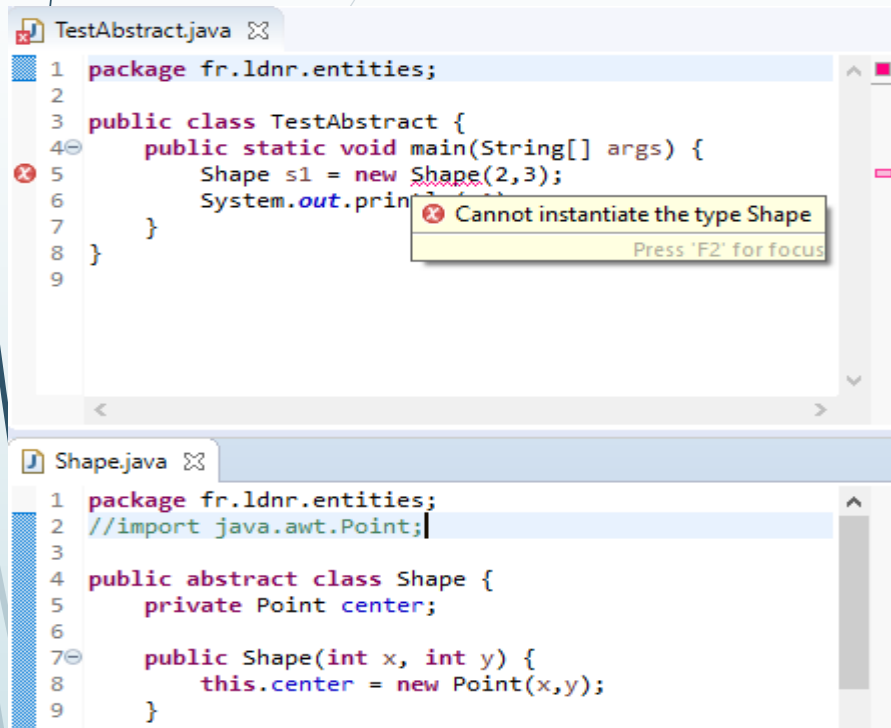
12



# Classe Abstraite

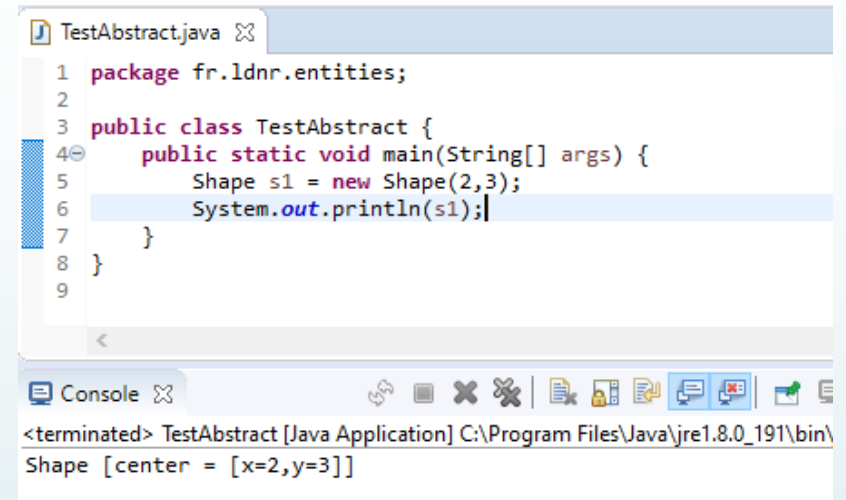
13

Pourquoi et Comment éviter cette situation ?



```
TestAbstract.java
1 package fr.ldnr.entities;
2
3 public class TestAbstract {
4     public static void main(String[] args) {
5         Shape s1 = new Shape(2,3);
6         System.out.println(s1);
7     }
8 }
9

Shape.java
1 package fr.ldnr.entities;
2 //import java.awt.Point;
3
4 public abstract class Shape {
5     private Point center;
6
7     public Shape(int x, int y) {
8         this.center = new Point(x,y);
9     }
10 }
```



```
TestAbstract.java
1 package fr.ldnr.entities;
2
3 public class TestAbstract {
4     public static void main(String[] args) {
5         Shape s1 = new Shape(2,3);
6         System.out.println(s1);
7     }
8 }
9

Console
<terminated> TestAbstract [Java Application] C:\Program Files\Java\jre1.8.0_191\bin\
Shape [center = [x=2,y=3]]
```



Maintenant, nous souhaitons appeler une méthode qui calcule l'air pour chaque forme géométrique d'une liste

```

5 public class TestShapes {
6     public static void main(String[] args) {
7         Shape[] shapes = new Shape[5];
8         shapes[0] = new Circle(1.5, new Point(5,5));
9         shapes[1] = new Square(3.2, new Point(1,2));
10        shapes[2] = new Circle();
11        for( Shape s : shapes ) {
12            System.out.println( s );
13            System.out.println( s.area());
14        }
15        /*-----
16        System.out.println("-----
17
18        Circle c1 = new Circle(2.0
19        Shape s1 = new Square(3.2,
20        Point p = new Point(3,15);
21        Object c2 = new Circle(1.5,p);
22
23        ArrayList<Object> objects = new ArrayList<>();
24        objects.add(c1);
25        objects.add(s1);
26        objects.add(c2);
27        objects.add(p);
28        //objects.add(new Shape(2,3));
    
```

The method area() is undefined for the type Shape

2 quick fixes available:

- Create method 'area()' in type 'Shape'
- Add cast to 's'

```

9    }
10
11    public Circle(double radius, Point center) {
12        super(center);
13        setRadius(radius);
14    }
15
16    public Circle() {
17        super(1,1);
18        Radius(1);
19
20    double getRadius() {
21        return radius;
22    }
23
24    public void setRadius(double radius) {
25        if(radius < 0 ) radius = 1;
26        else this.radius = radius;
27    }
28
29
30    public double area() {
31        return Math.PI * this.radius * this.radius;
32    }
    
```

```

4    private double side;
5
6    public Square(double side,int x, int y
7        super(x, y);
8        this.setSide(side);
9    }
10
11    public Square(double side,Point center
12        super(center);
13        this.setSide(side);
14    }
15
16    public double getSide() {
17        return side;
18    }
19
20    public void setSide(double side) {
21        if(side < 0)    this.side = 1;
22        else this.side = side;
23    }
24
25    public double area() {
26        return this.side * this.side;
27    }
    
```

Comment résoudre ce problème ?

Il suffit de la définir dans la classe mère et la redéfinir (Override) dans la classe fille (ne pas confondre avec la surcharge de méthodes).

```

TestAbstract.java
1 package fr.ldnr.entities;
2
3 public class TestAbstract {
4     public static void main(String[] args) {
5         Shape[] shapes = new Shape[3];
6         shapes[0] = new Circle(1.5, new Point(5,5));
7         shapes[1] = new Square(3.2, new Point(1,2));
8         shapes[2] = new Circle();
9         for( Shape s : shapes ) {
10             System.out.print( s );
11             System.out.println(" area = " + s.area());
12         }
13     }
14 }
15

Shape.java
1 package fr.ldnr.entities;
2 //import java.awt.Point;
3
4 public abstract class Shape {
5     private Point center;
6
7     public Shape(int x, int y) {
8         this.center = new Point(x,y);
9     }
10
11     public Shape(Point center) {
12         this.center =
13             new Point(center.getX(), center.getY());
14     }
15
16     public double area() {
17         return 0;
18     }
19
20     @Override
21     public String toString() {
22         return "Shape [center = " + center + "];"
23     }
24 }

Circle.java
15
16 public Circle() {
17     super(1,1);
18     setRadius(1);
19 }
20
21 public double getRadius() {
22     return radius;
23 }
24
25 public void setRadius(double radius) {
26     if(radius < 0 ) radius = 1;
27     else this.radius = radius;
28 }
29
30 public double area() {
31     return Math.PI * this.radius * this.radius;
32 }
33
34 @Override
35 public String toString() {
36     return "Circle : radius = " + radius + " " +
37 }
38

```

Néanmoins, nous avons 2 problèmes :

- ça n'a pas de sens de calculer l'air de Shape, encore moins de dire que c'est toujours 0 !
- que se passe-t-il si dans les classes filles, les méthodes area() renvoient vers celle de Shape ?

# Méthode Abstraite

NB : une méthode abstraite est obligatoirement dans une classe abstraite

16

```
TestAbstract.java
1 package fr.lidnr.entities;
2
3 public class TestAbstract {
4     public static void main(String[] args) {
5         Shape[] shapes = new Shape[3];
6         shapes[0] = new Circle(1.5, new Point(5,5));
7         shapes[1] = new Square(3.2, new Point(1,2));
8         shapes[2] = new Circle();
9         for( Shape s : shapes ) {
10             System.out.print( s );
11             System.out.println(" area = " + s.area());
12         }
13     }
14 }
15

Shape.java
1 package fr.lidnr.entities;
2 //import java.awt.Point;
3
4 public abstract class Shape {
5     private Point center;
6
7     public Shape(int x, int y) {
8         this.center = new Point(x,y);
9     }
10
11     public Shape(Point center) {
12         this.center =
13             new Point(center.getX(),center.getY());
14     }
15
16     public abstract double area();
17
18     @Override
19     public String toString() {
20         return "Shape [center = " + center + "];"
21     }
22 }

Circle.java
13     super(center);
14     setRadius(radius);
15 }
16
17 public Circle() {
18     super(1,1);
19     setRadius(1);
20 }
21
22 public double getRadius() {
23     return radius;
24 }
25
26 public void setRadius(double radius) {
27     if(radius < 0 ) radius = 1;
28     else this.radius = radius;
29 }
30
31 @Override
32 public double area() {
33     return Math.PI * this.radius * this.radius;
34     //return super.area();
35 }
36
37 @Override
38 public String toString() {
39     return "Circle : radius = " + radius + " " +
40 }
```

Console

```
<terminated> TestShapes [Java Application] C:\Program Files\Java\jre1.8.0_191\bin\javaw.exe (26 avr. 2021 à 13:26:07)
Circle : radius = 1.5 Shape [center = [x=5,y=5]] area = 7.0685834705770345
Square : side = 3.2 Shape [center = [x=1,y=2]] area = 10.240000000000002
Circle : radius = 1.0 Shape [center = [x=1,y=1]] area = 3.141592653589793
```

@Override signifie qu'on redéfinit une méthode présente dans la classe mère  
dans notre cas ici, on dit même qu'on implémente la méthode area() de la classe Shape  
Attention ici, si on refuse d'implémenter une méthode abstraite, notre classe fille de Shape devient abstraite

# Interface

Def : Une interface est constituée que de méthode(s) abstraite(s)



Souvent on l'utilise pour fixer les fonctionnalités d'une application : Elle représente la **couche métier ou business**

17

The screenshot shows an IDE with three Java files open: `IJob.java`, `IJobImpl.java`, and `TestJob.java`. The `IJob.java` file defines an interface with five methods: `addShape`, `deleteShape`, `getShapeById`, `moveShape`, and `drawShape`. The `IJobImpl.java` file implements these methods using a `HashMap` to store shapes. The `TestJob.java` file contains a `main` method that creates an `IJobImpl` instance and performs various operations on it, including adding, deleting, moving, and displaying shapes. The console output shows the results of these operations, displaying the details of each shape (Circle or Square) and its center coordinates.

```
package fr.ldnr.job;

import fr.ldnr.entities.Shape;

public interface IJob {
    public void addShape(int id, Shape shape);
    public void deleteShape(int id);
    public Shape getShapeById(int id);
    public void moveShape(int id, int x, int y);
    public void drawShape();
    public void displayAll();
}
```

```
import fr.ldnr.entities.Shape;

public class IJobImpl implements IJob {
    private Map<Integer, Shape> shapes;

    public IJobImpl() {
        shapes = new HashMap<>();
    }

    @Override
    public void addShape(int id, Shape shape) {
        shapes.put(id, shape);
    }

    @Override
    public void deleteShape(int id) {
        shapes.remove(id);
    }

    @Override
    public Shape getShapeById(int id) {
        return shapes.get(id);
    }

    @Override
    public void moveShape(int id, int x, int y) {
        Shape shape = shapes.get(id);
        shape.getCenter().setX(x);
        shape.getCenter().setY(y);
    }

    @Override
    public void drawShape() {
        // TODO Auto-generated method stub
    }

    @Override
    public void displayAll() {
        for(Shape s : shapes.values()) {
            System.out.println(s);
        }
    }
}
```

```
package fr.ldnr.job;

import fr.ldnr.entities.Circle;
import fr.ldnr.entities.Point;
import fr.ldnr.entities.Square;

public class TestJob {
    public static void main(String[] args) {
        IJobImpl job = new IJobImpl();
        job.addShape(1, new Circle(1.5, new Point(5,5)));
        job.addShape(2, new Square(3.2, new Point(2,3)));
        job.addShape(3, new Circle(1.6, 5, 2));
        job.addShape(4, new Square(3.2, 4, 8));
        job.addShape(5, new Circle(1.5, new Point(10,15)));

        job.displayAll();
        System.out.println("-----");

        job.deleteShape(1);
        job.moveShape(3, 20, 30);
        job.displayAll();
    }
}
```

Console Output:

```
<terminated> TestJob [Java Application] C:\Program Files\Java\jre1.8.0_191\bin\javaw.exe
Circle : radius = 1.5 Shape [center = [x=5,y=5]]
Square : side = 3.2 Shape [center = [x=2,y=3]]
Circle : radius = 1.6 Shape [center = [x=5,y=2]]
Square : side = 3.2 Shape [center = [x=4,y=8]]
Circle : radius = 1.5 Shape [center = [x=10,y=15]]
-----
Square : side = 3.2 Shape [center = [x=2,y=3]]
Circle : radius = 1.6 Shape [center = [x=20,y=30]]
Square : side = 3.2 Shape [center = [x=4,y=8]]
Circle : radius = 1.5 Shape [center = [x=10,y=15]]
```

# Instanceof & cast

Comment calculer le périmètre uniquement des carrés d'une liste de formes géométriques ?

18

```
TestJob.java
11 job.addShape(2,new Square(3.2,new Point(2,3)));
12 job.addShape(3,new Circle(1.6,5,2));
13 job.addShape(4,new Square(5.2,4,8));
14 job.addShape(5,new Circle(1.5,new Point(10,15)));
15
16 job.displayAll();
17 System.out.println("-----");
18
19 job.deleteShape(1);
20 job.moveShape(3, 20, 30);
21 job.displayAll();
22 System.out.println("-----");
23
24 job.squaresPerimeter();
```

```
Console
<terminated> TestJob [Java Application] C:\Program Files\Java\jre1.8.0_191\bin\javaw.exe (11 mai 2021 à 16:35:07)
Circle : radius = 1.6 Shape [center = [x=5,y=2]]
Square : side = 5.2 Shape [center = [x=4,y=8]]
Circle : radius = 1.5 Shape [center = [x=10,y=15]]
-----
Square : side = 3.2 Shape [center = [x=2,y=3]]
Circle : radius = 1.6 Shape [center = [x=20,y=30]]
Square : side = 5.2 Shape [center = [x=4,y=8]]
Circle : radius = 1.5 Shape [center = [x=10,y=15]]
-----
Square : side = 3.2 Shape [center = [x=2,y=3]] périmètre = 12.8
Square : side = 5.2 Shape [center = [x=4,y=8]] périmètre = 20.8
```

```
UobImpl.java
49 }
50
51 public void squaresPerimeter() {
52     Iterator<Shape> it = shapes.values().iterator();
53     while(it.hasNext()) {
54         Shape shape = it.next();
55         if(shape instanceof Square) {
56             System.out.println(shape + " périmètre = " + ((Square)shape).perimeter());
57         }
58     }
59 }
60 }
```

```
Circle.java
1 package fr.lidnr.entities;
2
3 public class Circle extends Shape {
4     private double radius;
5
6     public Circle(double radius, int x, int y) {
7         super(x,y);
8         setRadius(radius);
9     }
10
11     public Circle(double radius, Point center) {
12         super(center);
13         setRadius(radius);
14     }
15
16     public Circle() {
17         super(1,1);
18         setRadius(1);
19     }
20
21     public double getRadius() {
22         return radius;
23     }
24
25     public void setRadius(double radius) {
26         if(radius < 0 ) radius = 1;
27         else this.radius = radius;
28     }
29
30     @Override
31     public double area() {
32         return Math.PI * this.radius * this.radius;
33         //return super.area();
34     }
35
36     @Override
37     public String toString() {
38         return "Circle : radius = " + radius + " " +
39     }
40 }
41
```

```
Square.java
1 package fr.lidnr.entities;
2
3 public class Square extends Shape {
4     private double side;
5
6     public Square(double side,int x, int y) {
7         super(x, y);
8         this.setSide(side);
9     }
10
11     public Square(double side,Point center) {
12         super(center);
13         this.setSide(side);
14     }
15
16     public double getSide() {
17         return side;
18     }
19
20     public void setSide(double side) {
21         if(side < 0) this.side = 1;
22         else this.side = side;
23     }
24
25     public double area() {
26         return this.side * this.side;
27     }
28
29     public double perimeter() {
30         return this.side * 4;
31     }
32
33     @Override
34     public String toString() {
35         return "Square : side = " + side + " " +
36     }
37 }
38
```



# Toujours pas convaincu de l'intérêt de la poo !

- Comment alors exploiter l'API Java ainsi que toutes autres bibliothèques externes, Frameworks...
- Indispensable aussi pour comprendre les design patterns (bonnes réponses aux mêmes problèmes complexes)
- Possibilité de faire des applications robustes et évolutives à condition que tout le monde joue le jeu
- Permet le découpage d'une application volumineuse, exemple :
  - **Architecte** : utilise la poo et uml pour réaliser l'architecture globale
  - **Concepteurs** : se concentrent sur une partie précise de l'appli (couche) et définissent les fonctionnalités à l'aide d'interfaces.
  - **Développeurs** : les petites mains qui codent, ce sont les maçons !

# Consignes pour Exo & Tp

- **Vos programmes doivent précisément répondre aux besoins exprimé**
- **En cas de doute sur l'énoncé, n'hésitez pas à solliciter le formateur**
- **Prendre le temps de l'analyse et la compréhension avant de coder**
- **Entre faire simple ou compliqué, il ne doit pas y avoir de doute**
- **Respecter les règles d'écriture de code (maj-min)**
- **Utiliser les outils d'Eclipse pour générer du code**
- **Eviter trop de boucles, trop de tests, le mot d'ordre : factoriser pour optimiser**
- **Utiliser la Poo dans tous ses aspects vus**
- **Les noms de classes, attributs, méthodes et variables doivent être explicites, compréhensibles de tous, adaptés et en ANGLAIS**
- **Attention à l'indentation, votre code doit rester lisible pour tous notamment votre binôme**
- **Commenter vos codes afin d'en faciliter la relecture par des pairs, utiliser la javadoc !**
- **Utiliser Git avec les bonnes pratiques vues : une branche par fonctionnalité...**

# Next Steps



- Java SE 8
- Algo
- Git
- **La POO avec Java**
- Programmation avancée