



# PROGRAMMATION ORIENTÉE OBJET JAVA

LICENCE 2 INGÉNIERIE - INFORMATIQUE

2020– 2021

Marie NDIAYE



# LA CLASSE OBJECT

# LA SUPER CLASSE

- En Java, toute classe (définies par l'utilisateur ou dans l'API) qui n'étend pas une autre en utilisant le mot réservé `extends` étend implicitement la classe `Object` (qui se trouve dans le package `java.lang`).
- Toutes les classes Java ont comme premier ancêtre la classe `Object`.
- Quelques méthodes très utiles :
  - `toString`
  - `equals`
  - `clone` (à voir plus tard)

# LA MÉTHODE TOSTRING

- **public String toString(){...}**

- Retourne une chaîne de caractères qui décrit l'objet qui l'appelle.

- Chaîne constituée ...

- du nom de la classe concernée;

- de l'adresse de l'objet en hexadécimal (précédée de @).

- Exemple :

```
Point point = new Point(1,4);
```

```
System.out.println("point = " + point); // point =  
Point@42e816
```

- Doit être redéfinie dans chaque classe pour décrire fidèlement les objets de la classe.

# EXEMPLE : REDÉFINITION DE toString

```
//Dans la classe Point
public String toString(){
    return(super.toString()+" abscisse : "+abscisse+" ordonnée : "+ordonnee);
}

//Dans le main
Point point = new Point(1,4);
System.out.println("point = " + point);

// Affichage
// point = Point@9304b1 abscisse : 1 ordonnée : 4
```

# LA MÉTHODE EQUALS

## ■ **public boolean equals (Object o)**

■ Comparer les adresses des deux objets concernés.

■ Exemple :

```
Point p, p1,p2;  
p1=new Point(1,2);  
p2=new Point(1,2);  
System.out.println(p1.equals(p2)); // false  
  
p=p1;  
p.affiche(); // Je suis en 1 2  
System.out.println(p.equals(p1)); // true
```

■ La méthode equals doit être redéfinie si on souhaite comparer les propriétés des objets d'une classe et non leur référence.

## EXEMPLE : REDÉFINITION DE EQUALS

```
//Dans la classe Point
public boolean equals (Object o){
    if(!(o instanceof Point))
        return false;
    Point p=(Point)o;
    return this.abscisse==p.abscisse && this.ordonnee==p.ordonnee;}

//Dans le main
Point p, p1,p2;
p1=new Point(1,2);
p2=new Point(1,2);
System.out.println(p1.equals(p2)); // true
p=p1;
p.affiche(); // Je suis en 1 2
System.out.println(p.equals(p1)); // true
```

# LA MÉTHODE EQUALS : LE PIÈGE À ÉVITER

```
public class Object {  
    ...  
    public boolean equals(Object o)  
        return this == o  
    }  
    ...  
}
```

De manière générale, il vaut mieux éviter de surcharger des méthodes en spécialisant les arguments

```
public class Point {  
  
    private double x;  
    private double y;  
  
    ...  
}
```

```
public boolean equals(Point pt) {  
    return this.x == pt.x && this.y == pt.y;  
}
```

surcharge (overloads) la méthode equals(Object o) héritée de Object

invokevirtual ... <Method equals(Object)>

Le choix de la méthode à exécuter est effectué statiquement à la compilation

en fonction du type déclaré de l'objet récepteur du message et du type déclaré du (des) paramètre(s)

```
Point p1 = new Point(15, 11);  
Point p2 = new Point(15, 11);  
p1.equals(p2)    --> true
```

```
Object o = p2;  
p1.equals(o)    --> false ☹️
```

```
o.equals(p1)    --> false
```



# L'OPÉRATEUR INSTANCEOF

- Retourne une valeur booléenne indiquant si un objet est une instance d'une classe particulière ou d'une fonction construite.
- Syntaxe : **objet instanceof Classe**
- Exemple :

```
PointCol x=new PointCol(1,2,(byte)3);  
if (x instanceof Point) System.out.println("x est un Point");  
System.out.println((new Point(1,1)) instanceof Object); // true  
System.out.println((new Object()) instanceof Object); // true  
System.out.println((new Object()) instanceof Point); // false
```



# CLASSE ABSTRAITES ET INTERFACES

# MÉTHODE ABSTRAITE : L'INTUITION (1/2)

- Soient la classe `Point` et les sous-classes `PointCol` et `PointNom`.
- On peut effectuer des opérations "communes" sur l'ensemble des objets de la classe `Point` (y compris ceux de ses sous-classes).
- Pour tout objet `p` de type `Point`, on peut écrire : `p.affiche(...)`
  - Deux conditions :
    - `Affiche` est définie dans la classe mère `Point` (pour que `p.affiche()` ne génère pas une erreur de compilation).
    - `Affiche` est héritée ou redéfinie dans les sous-classes `PointCol` et `PointNom` (le polymorphisme assure que la méthode adéquate sera appelée à l'exécution).

# MÉTHODE ABSTRAITE : L'INTUITION (2/2)

- Soit la classe **Forme** et les sous-classes **Rectangle** et **Cercle**
  - On ne peut pas définir la méthode **dessine()** dans la classe la classe mère **Forme** mais cela est possible dans les sous-classes **Rectangle** et **Cercle**.
  - On souhaite quand même pouvoir utiliser l'écriture **f.dessine()** où **f** est un objet de type **Forme**.
    - Problème : cet appel de méthode ne passe pas à la compilation
    - Solution : Déclarer dans la classe **Forme** la méthode **dessine()** comme **méthode abstraite**.

# QU'EST-CE QU'UNE MÉTHODE ABSTRAITE ?

- Une **méthode abstraite** est une méthode qui ne possède pas de définition.
- Exemple :
  - **public abstract void dessine();**
- Une méthode abstraite doit **obligatoirement être déclarée public**.

# QU'EST-CE QU'UNE CLASSE ABSTRAITE ?

- Une classe comportant une ou plusieurs méthodes abstraites est une classe abstraite (**l'inverse n'est pas vrai**).

- Exemple :

```
public abstract class Forme {  
    public abstract void dessine();  
    public void affiche(){  
        System.out.println("Je suis une  
        forme géométrique");}}}
```

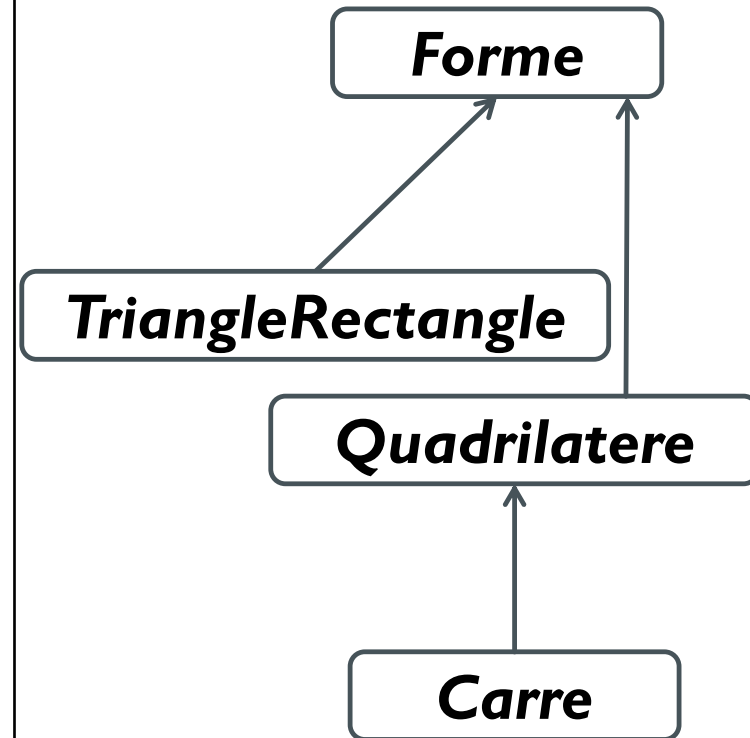
- Une classe abstraite **ne peut pas être instanciée** (new).

# CLASSE ABSTRAITE ET DÉRIVATION

- Une classe dérivée d'une classe abstraite ne redéfinissant pas toutes les méthodes abstraites est elle-même abstraite.
- Toute classe concrète sous-classe d'une classe abstraite doit concrétiser toutes les opérations abstraites de cette dernière.
- Une classe dérivée d'une classe non abstraite peut être déclarée abstraite.

# EXEMPLE : LA CLASSE FORME

```
public abstract class Forme {  
    public Forme(){  
        System.out.println("Constructeur de la classe Forme");  
    }  
  
    public abstract double perimetre();  
  
    public abstract double surface();  
  
    public void affiche(){  
        System.out.println("Je suis une forme géométrique");  
    }  
}
```





# EXEMPLE : LA CLASSE TRIANGLERECTANGLE

```
public class TriangleRectangle extends Forme {  
    protected double b,h,c;  
  
    public TriangleRectangle(double b, double h, double c){  
        super();  
        System.out.println("Constructeur de la classe  
TriangleRectangle");  
        this.b=b; this.h=h; this.c=5; }  
  
    public double perimetre() { return(b+h+c); }  
    public double surface() { return b*h/2; }  
  
    public void affiche(){  
        super.affiche();  
        System.out.println("Triangle rectangle de base  
"+b+" de hauteur "+ h+" et de coté : "+c); }  
}
```

# EXEMPLE : LA CLASSE QUADRILATÈRE

```
public abstract class Quadrilatre extends Forme {  
    protected double c1,c2,c3,c4;  
  
    public Quadrilatre(double c1,double c2,double c3,double c4){  
        super();  
        this.c1=c1; this.c2=c2; this.c3=c3; this.c4=c4;  
        System.out.println("Constructeur de la classe  
        Quadrilatère");  
    }  
  
    public double perimetre(){ return(c1+c2+c3+c4); }  
  
    public void affiche(){  
        super.affiche();  
        System.out.println("de la famille des quadrilatères");  
    }  
}
```

# EXEMPLE : LA CLASSE CARRE

```
public class Carre extends Quadrilatere {  
  
    public Carre(double c){  
        super(c,c,c,c);  
        System.out.println("Constructeur de la classe Carré");  
    }  
  
    public double surface(){  
        return (c1*c1);  
    }  
  
    public void affiche(){  
        super.affiche();  
        System.out.println("Carré de coté "+c1);  
    }  
}
```

# LA MÉTHODE MAIN (I/2)

```
public static void main(String[] args) {  
    int n=10;  
    ArrayList<Forme> formes = new ArrayList<Forme>();  
    for(int i=1;i<=n;i++){  
        if(i%2==0)  
            formes.add(new Carre(i));  
        else  
            formes.add(new TriangleRectangle  
                (i,i+1,Math.sqrt((i*i)+((i+1)*(i+1)))));  
        System.out.println();  
    }  
  
    for(int i=0;i<n;i++)  
        formes.get(i).affiche();  
}
```

## LA MÉTHODE MAIN (2/2)

```
...  
double sommeSurfaces=0; double moyennePerimetres=0;  
  
for(int i=0;i<n;i++){  
    sommeSurfaces+=formes.get(i).surface();  
    moyennePerimetres+=formes.get(i).perimetre();  
}  
  
System.out.println("Somme des surfaces : "+sommeSurfaces);  
  
System.out.println("Moyenne des périmètres :  
    "+moyennePerimetres/10);  
}
```

# QU'EST-CE QU'UNE INTERFACE?

■ Une interface correspond à une classe dans laquelle

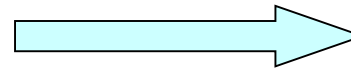
■ Toutes les méthodes sont abstraites

■ Il n'existe pas de variables d'instance.

■ Définition

```
abstract class I{  
    public abstract void f(int n);  
    public abstract void g();  
}
```

*public abstract  
facultatifs*



```
interface I{  
    void f(int n);  
    void g();  
}
```

■ Implémentation

```
class A implements I{  
    public void f(int n){. . .}  
    public void g(){. . .}  
}
```

*La non définition des méthodes f  
et g génère une erreur à la compilation.*

# CARACTÉRISTIQUES DES INTERFACES

- Une interface définit les en-têtes d'un certain nombre de méthodes ainsi que des constantes.

```
interface I{static final int i=10; void f(int n);}
```

- Les interfaces n'ont jamais de champs d'instance

```
interface I{ int i; /* 😞 ERREUR A LA COMPILATION*/ }
```

- Une interface peut hériter (**extends**) de plusieurs interfaces.

```
interface I1{...}  
interface I2{...}  
interface I extends I1,I2{...}
```

# HÉRITAGE / IMPLÉMENTATION

- Une classe peut implémenter (**implements**) une ou plusieurs interfaces tout en héritant (**extends**) d'une classe.

```
interface I1{...}  
interface I2{...}  
class A{...}  
class B extends A, implements I1,I2{...}
```

- La notion d'interface ne se substitue pas à celle de dérivation mais elle se superpose à celle-ci.



# EXERCICE I : A TESTER

```
class A{
    public void m1(){ System.out.println("m1 de A"); } }

class B extends A{
    public void m1(){ System.out.println("m1 de B"); } }

public class Test{
    public static void main(String[] args){
        A objet; double hasard;
        for (int i = 0; i < 10; i++){
            hasard = Math.random();
            if ( hasard < 0.5) objet = new A();
            else objet = new B();
            objet.m1();
        }
    } }
```