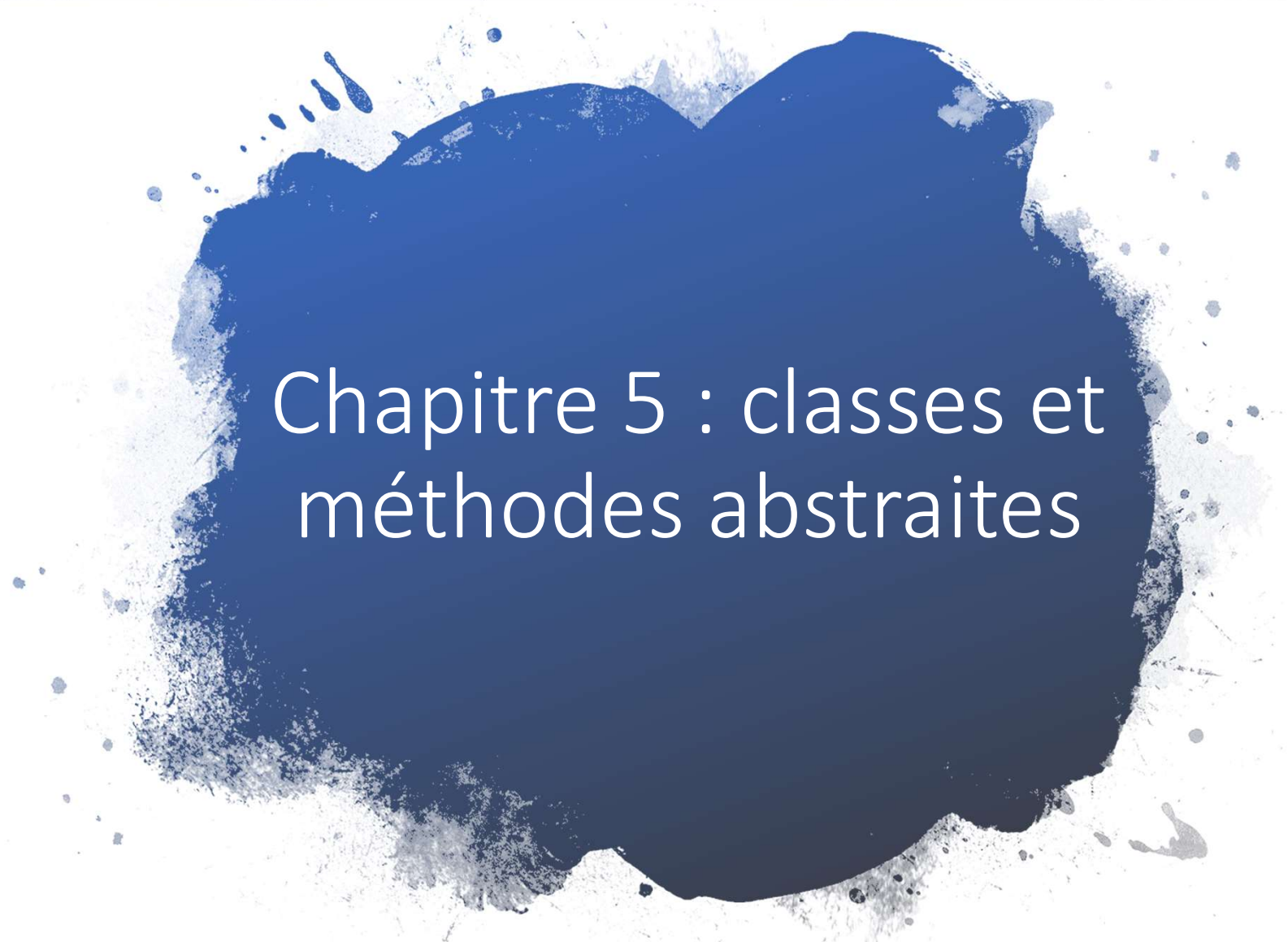


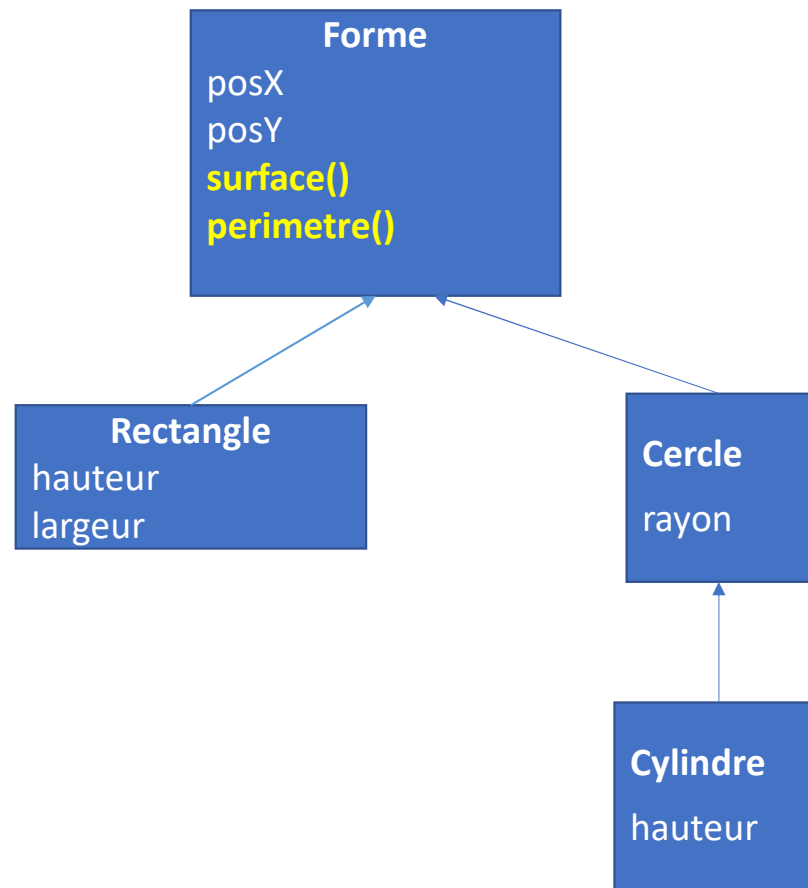
---



# Chapitre 5 : classes et méthodes abstraites

---

# Exemple

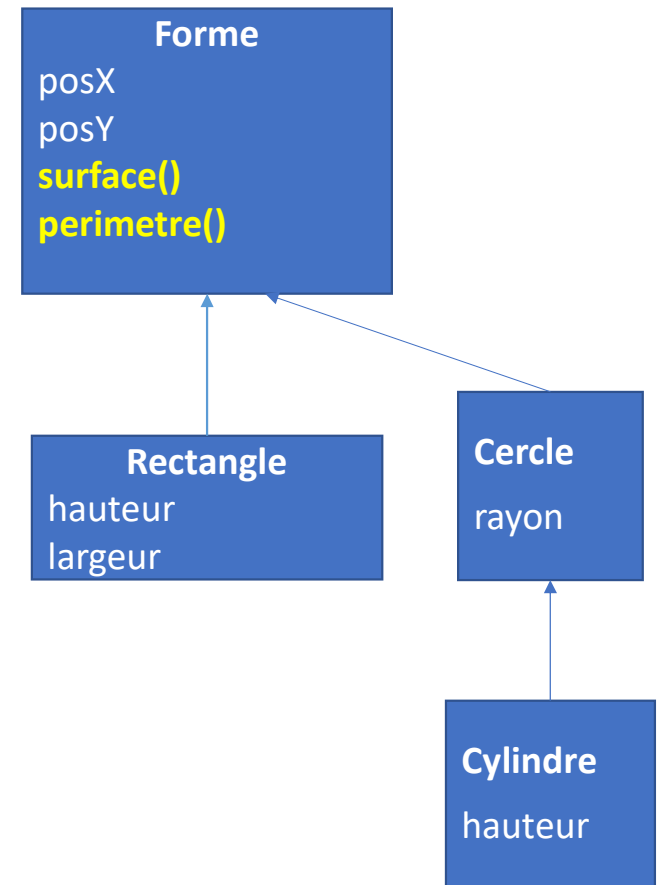


# Problématique

Avec l'**héritage**, il est possible de **généraliser** des concepts qui sont présents dans plusieurs classes du programme en les incorporant à une super-classe.

On construit, alors, une structure arborescente reliant différents éléments à un niveau plus abstrait.

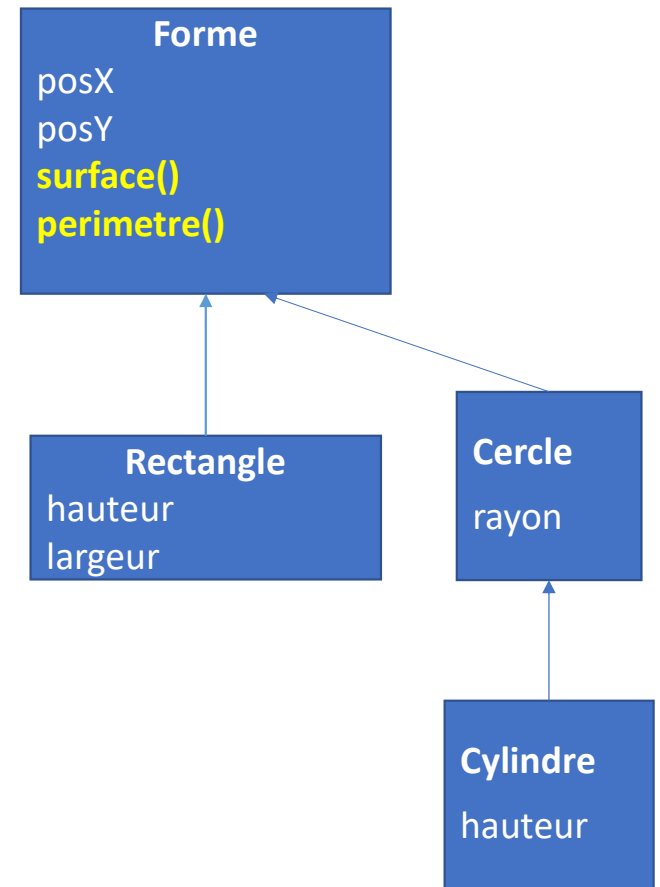
➡ fait partie des mécanismes d'**abstraction** de l'orienté objet.



# Problématique

Mais, au niveau le plus élevé d'une hiérarchie de classes, il est parfois impossible de définir une méthode générale qui devra pourtant exister dans toutes les sous-classes.

**Exemple : *calculer la surface*** d'une Forme quelconque se révèle difficile, tandis qu'à un niveau plus bas on peut le mettre en œuvre.



# Exemple :

```
public class Forme {
    protected double posX=0;
    protected double posY=0;
    protected static int nbObjets = 0;

    public Forme(){
        setPosX(0); setPosY(0);
        nbObjets++;
    }
    public Forme(double x, double y){
        setPosX(x); setPosY(y);
        nbObjets++;
    }
    public void deplaceDe(double dx, double dy){
        setPosX(posX+dx); setPosY(posY+dy);
    }
    public double surface() {return 0.0;}
    public double perimetre() {return 0.0;}
    public void afficher() {}

    // et les getters et les setters ...
}
```

```
public class Rectangle extends Forme {
    private double largeur = 3.0;
    private double hauteur = 4.0;
    public Rectangle() {
        super();
        setHauteur(0);
        setLargeur(0);
    }
    public Rectangle(double h, double l, double x, double y) {
        super(x,y);
        setHauteur(h);
        setLargeur(l);
    }
    public void coinDS() {
        System.out.println("Le coin sup droit est : " + (posX
        + largeur) + ", " + posY);
    }
    public double surface() {
        return largeur*hauteur;
    }
    public double perimetre() {
        return (largeur+hauteur)*2;
    }
    public void afficher() {
        System.out.println("[Rectangle] largeur = " + largeur
        + " et hauteur = " + hauteur);
    }
}
// et les getters et setters
```

```

public class Cercle extends Forme {
    protected double rayon;
    public Cercle(double x, double y, double r){
        super(x, y);
        setRayon(r);
    }

    public void setRayon(double r)
    {
        rayon = r;
    }
    public double getRayon() {
        return rayon;
    }

    double perimetre(){
        return 2*Math.PI*rayon;
    }

    double surface(){
        return Math.PI*rayon*rayon;
    }
    public void afficher() {
        System.out.println("[Cercle] rayon = "
+ rayon);
    }
    // et les getters et setters
}

```

```

public class Cylindre extends Cercle {
    private double longueur;

    public Cylindre(double x, double y, double rayon, double l)
    {
        super(x,y,rayon);
        setLongueur(l);
    }

    public double surface() {
        return 2 * super.surface() + (2 * Math.PI * getRayon())
* Longueur;
    }

    public double volume()
    {
        return super.surface() * longueur;
    }
    public void afficher() {
        System.out.println("[Cylindre] rayon = " + rayon + " et
Longueur = " + longueur);
    }

    // et les getters et setters
}

```

# Problématique

- Une telle méthode **surface()** au niveau de la super-classe **Forme** pourrait être nécessaire pour comparer deux surfaces (qui est générale) ou simplement par souci d'abstraction et d'unification des données.
- On veut comparer la surface de deux formes , mais on ne pourra savoir si c'est un Cercle ou Rectangle qu'en cours d'exécution.
- Le programme ne compilera pas si la méthode **surface** n'est pas déclarée au niveau de la classe **Forme**.

```
public class TestPolymorphisme {
    public static void main(String[] args)
    {
        // déclaration et initialisation de deux Formes
        Forme forme1 = new Cercle(5, 3, 6);
        Forme forme2 = new Rectangle(5, 3, 2, 5);

        System.out.println("Les 2 Formes ont La même surface ? " +
            surfaceCompare(forme1, forme2));

        // afficher le cercle
        afficherForme(forme1);

        // afficher le rectangle
        afficherForme(forme2);
    }

    // méthode pour comparer les surfaces de deux Formes
    static boolean surfaceCompare(Forme objet1, Forme objet2)
    {
        return objet1.surface() == objet2.surface();
    }

    // méthode pour afficher des infos sur la forme géométrique
    static void afficherForme(Forme objet)
    {
        System.out.println();
        objet.afficher();
        System.out.println("La surface est " + objet.surface());
        System.out.println("Le périmètre est " +
            objet.perimetre());
    }
}
```

# Solution: Méthode abstraite

- C'est une méthode qui dispose de la **signature** (type de retour et paramètres) mais **sans implémentation**.
- Elle est déclarée avec le modificateur **abstract**

## Exemple:

```
public abstract double surface();
```

- La méthode sera implémentée par les classes enfants.



# Définition : classes Abstraites

Une classe abstraite est une classe **incomplète**. Elle regroupe un ensemble de variables et de méthodes mais **certaines de ses méthodes ne contiennent pas d'instructions (méthodes abstraites)**

- Elle est déclarée avec le modificateur **abstract**

**Exemple:**

```
public abstract class Forme
```

# Classes Abstraites

- **Règle 1** : Une classe est automatiquement abstraite si une de ses méthodes est abstraite.
- **Règle 2** : Une classe abstraite peut contenir des méthodes non abstraites et des déclarations de variables ordinaires.
- **Règle 3** : On déclare qu'une classe est abstraite avec le mot clé **abstract**.
- **Règle 4** : Une classe abstraite n'est pas instanciable (on ne peut pas utiliser les constructeurs d'une classe abstraite et donc on ne peut pas créer d'objet de cette classe.)
- **Règle 5** : Une classe qui hérite d'une classe abstraite ne devient concrète que si elle implémente toutes les méthodes abstraites de la classe dont elle hérite.

# Exemple

```
public abstract class Forme {  
    protected double posX=0;  
    protected double posY=0;  
    protected static int nbObjets = 0;  
  
    public Forme(){  
        setPosX(0); setPosY(0); nbObjets++;  
    }  
    public Forme(double x, double y){  
        setPosX(x); setPosY(y); nbObjets++;  
    }  
    public void deplaceDe(double dx, double dy){  
        setPosX(posX+dx); setPosY(posY+dy);  
    }  
    public abstract double surface();  
    public abstract double perimetre();  
    public abstract void afficher();  
  
    // et les getters et les setters ...  
}
```