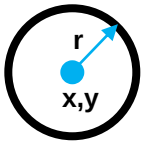
Héritage et abstraction
classes abstraites

Philippe Genoud

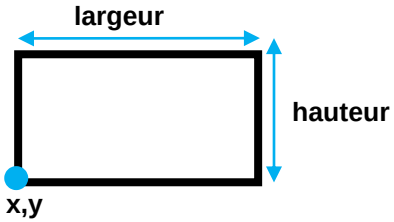
Classes Abstraites

Exemple introductif

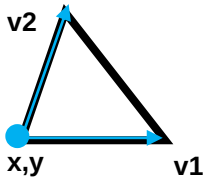
- un **grand classique** les formes géométriques
  - *on veut définir une application permettant de manipuler des formes géométriques (triangles, rectangles, cercles…).*
  - *chaque forme est définie par sa position dans le plan*
  - *chaque forme peut être déplacée (modification de sa position), peut calculer son périmètre, sa surface*



Attributs :
double x,y; //centre du cercle
double r; // rayon
Méthodes :
deplacer(double dx, double dy)
double surface()
double périmètre()



Attributs :
double x,y; //coin inférieur gauche
double largeur, hauteur;
Méthodes :
deplacer(double dx, double dy)
double surface()
double périmètre();



Attributs :
double x,y; //1 des sommets
double x1,y1; // v1
double x2,y2; // v2
Méthodes :
deplacer(double dx, double dy)
double surface()
double périmètre();

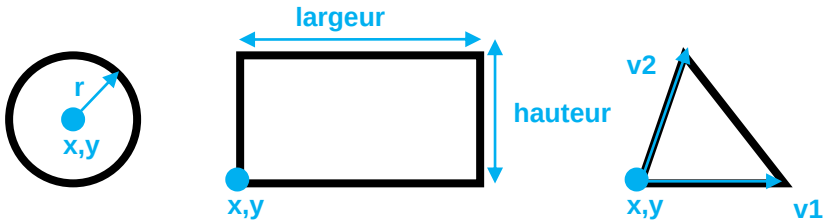
Factoriser le code ?

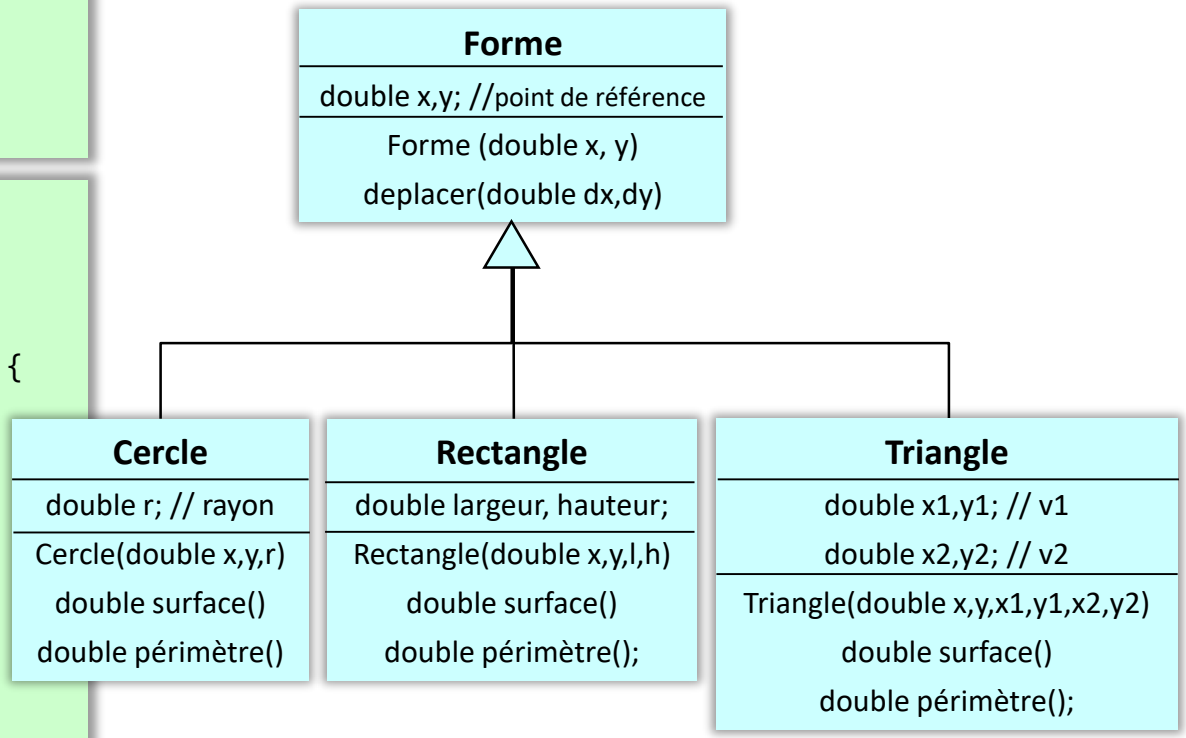
Classes abstraites

```
public class Forme {
   protected double x,y;

   public Forme(double x, double y) {
     this.x = x;
     this.y = y;
   }

   public void deplacer(double dx,
       double dy) {
      x += dx; y += dy;
   }
}
```

```
public class Cercle extends Forme {
    protected double r;

    public Cercle(double x, double y,
                            double r) {
       super(x,y);
       this.r = r;
     }

    public double surface(){
       return Math.PI * r * r;
    }

    protected double périmetre(){
       return 2 * Math.PI * r;
    }
}
```

Exemple introductif





Classes abstraites

Exemple introductif

```
public class ListeDeFormes {

  public static final int NB_MAX = 30;
  private Forme[] tabFormes = new Forme[NB_MAX];
  private int nbFormes = 0;

  public void ajouter(Forme f) {
    if (nbFormes < NB_MAX)
       tabFormes[nbFormes++] =f
  }

  public void toutDeplacer(double dx,double dy) {
    for (int i=0; i < NbFormes; i++)
       tabFormes[i].deplacer(dx,dy);
  }


  public double périmètreTotal() {
    double perimTotal = 0.0;
    for (int i=0; i < nbFormes++; i++)
         perimTotal += tabFormes[i].périmètre();
    return perimTotal;
  }

}
```

erreur de compilation

On veut pouvoir gérer des listes de formes

On exploite le **polymorphisme** la prise en compte de nouveaux types de forme ne modifie pas le code

Appel non valide car la méthode **périmètre** n'est pas implémentée au niveau de la classe Forme

Définir une méthode **périmètre** dans **Forme** ?

```
public double périmètre() {
  return 0.0; // ou -1. ??
}
```

BOF

Une solution propre et élégante : les **classes abstraites**

Classes abstraites

Classe abstraite

```
public abstract class Forme {
   protected double x,y;

     public Forme(double x, double y) {
       this.x = x;
       this.y = y;
     }

   public void deplacer(double dx,
         double dy) {
     x += dx; y += dy;
   }
   public abstract double périmètre();

   public abstract double surface();

  }
```

la classe doit être déclarée comme étant explicitement abstraite

Méthodes abstraites

on spécifie qu'un objet de type Forme aura une méthode périmètre et une méthode surface

par contre on ne sait pas comment cela sera implémenté → méthodes sans corps : ; au lieu de { … }

- Utilité :
  - *définir des concepts incomplets qui devront être implémentés dans les sous classes*
  - *factoriser le code*
  - *les opérations abstraites sont particulièrement utiles pour mettre en œuvre le polymorphisme.*
    - *l'utilisation du nom d'une classe abstraite comme type pour une (des) référence(s) est toujours possible (et souvent souhaitable !!!)*

Classes abstraites

- ***classe abstraite*** : classe non instanciable, c'est à dire qu'elle n'admet pas d'instances directes.
  - *Impossible de faire* **`new ClasseAbstraite(…);`**
  - *mais une classe abstraite peut néanmoins avoir un ou des constructeurs*

- ***opération abstraite*** : opération n'admettant pas d'implémentation
  - *au niveau de la classe dans laquelle elle est déclarée, on ne peut pas dire comment la réaliser.*

- Une classe pour laquelle au moins une opération abstraite est déclarée est une classe abstraite (l'inverse n'est pas vrai).

```
public abstract class ClasseA {

   ...

   public abstract void methodeA();

   ...
}
```

la classe contient une méthode abstraite => elle **doit** être déclarée abstraite

```
public abstract  class ClasseA {

   ...



}
```

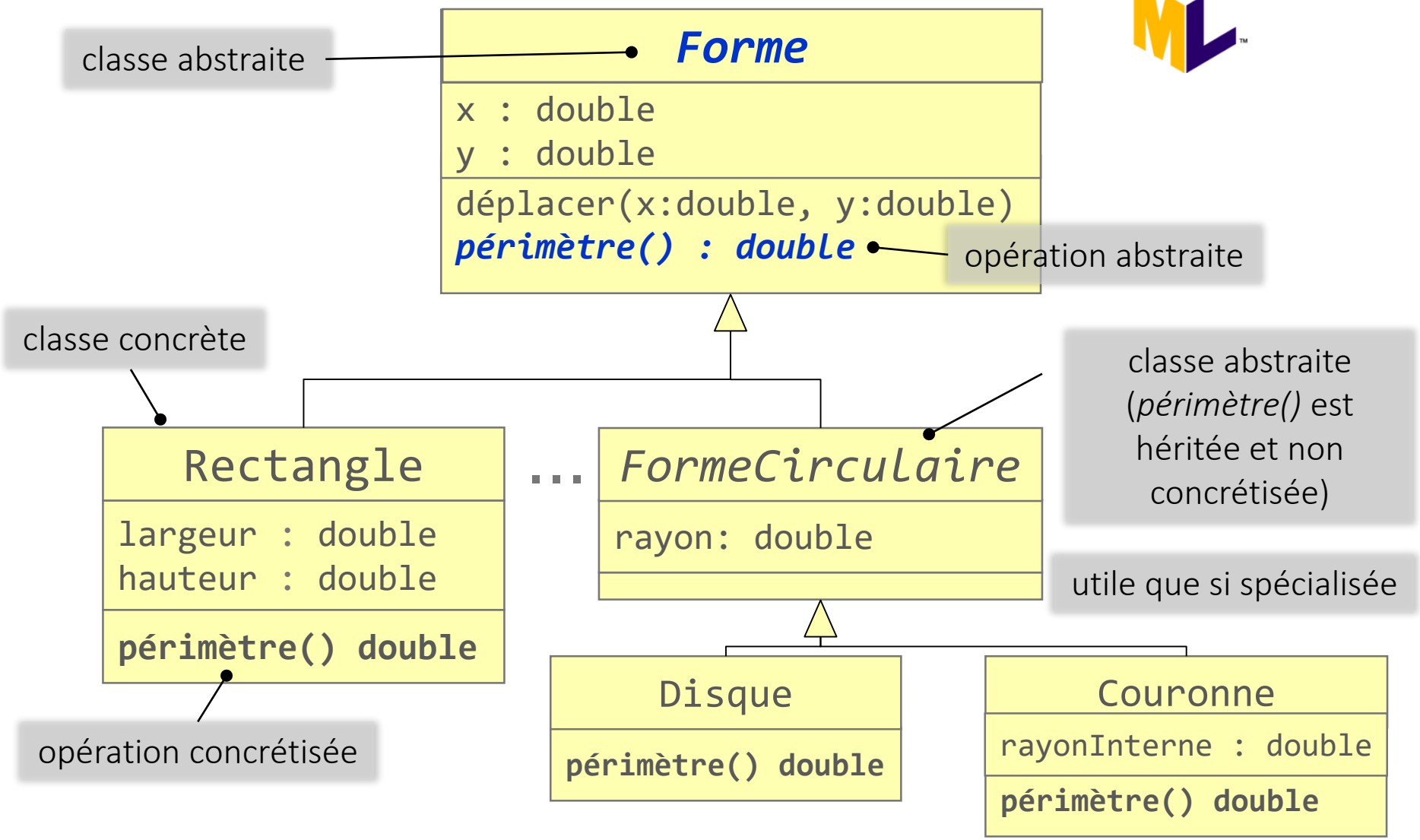
la classe ne contient pas de méthode abstraite => elle **peut** être déclarée abstraite

Classes abstraites

- Une classe abstraite est une description d'objets destinée à être héritée par des classes plus spécialisées.
- Pour être utile, une classe abstraite doit admettre des classes descendantes ***concrètes***.
- Toute classe **concrète** sous-classe d'une classe abstraite doit “concrétiser” toutes les opérations abstraites de cette dernière.
  - *elle doit implémenter toutes les méthodes abstraites*
- Une classe abstraite permet de regrouper certaines caractéristiques communes à ses sous-classes et définit un comportement minimal commun.
- La factorisation optimale des propriétés communes à plusieurs classes par généralisation nécessite souvent l'utilisation de classes abstraites.

Classes abstraites

Classes abstraites et diagrammes de classes UML



Classes abstraites

Exemple introductif

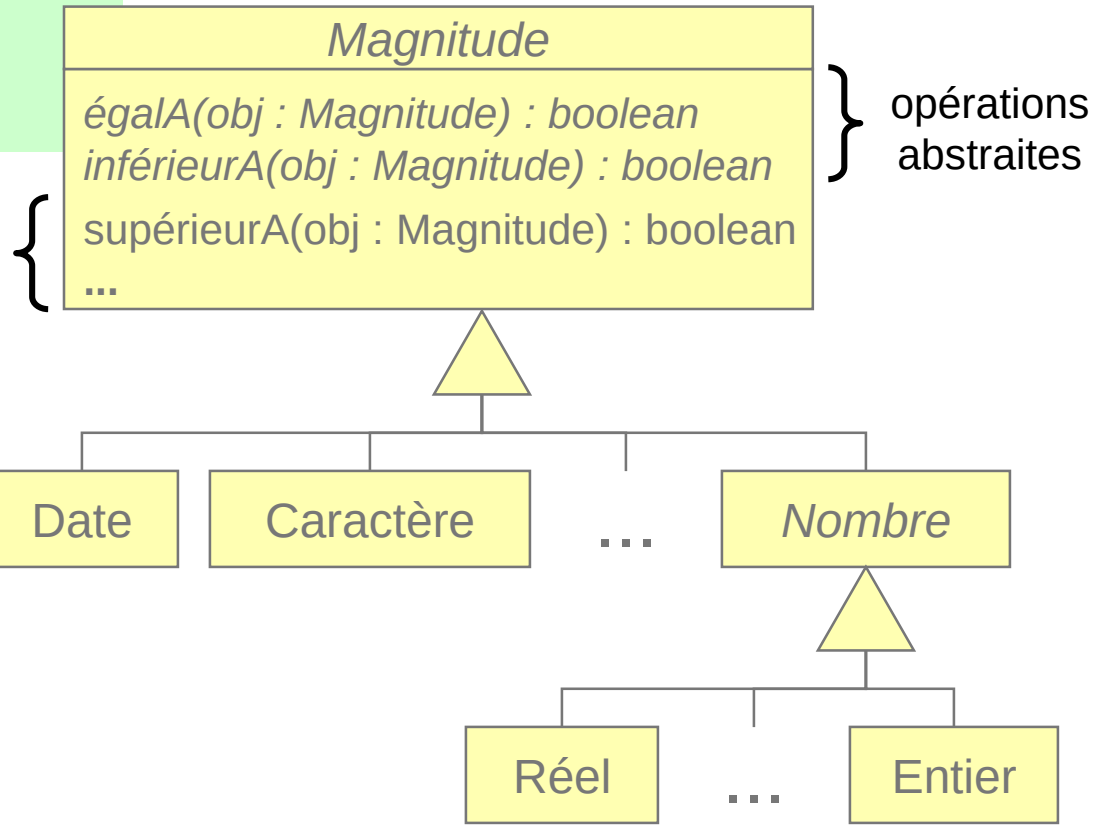
```
public class ListeDeFormes {

  public static final int NB_MAX = 30;
  private Forme[] tabFormes = new Forme[NB_MAX];
  private int nbFormes = 0;

  public void ajouter(Forme f) {
    if (nbFormes < NB_MAX)
       tabFormes[nbFormes++] =f
  }

  public void toutDeplacer(double dx,double dy) {
    for (int i=0; i < NbFormes; i++)
       tabFormes[i].deplacer(dx,dy);
  }


  public double périmètreTotal() {
    double perimTotal = 0.0;
    for (int i=0; i < nbFormes++; i++)
         perimTotal += tabFormes[i].périmètre();
    return perimTotal;
  }

}
```

```
public abstract class Forme {
  protected double x,y;

  public Forme(double x, double y) {
      this.x = x;
      this.y = y;
  }

  public void deplacer(double dx,
       double dy) {
     x += dx; y += dy;
  }

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

Le polymorphisme peut être pleinement exploité. Le compilateur sait que chaque objet **Forme** peut calculer son périmètre

Classes abstraites

```
public abstract class Magnitude {

   public abstract boolean egalA(Magnitude m) ;

   public abstract boolean inferieurA(Magnitude m) ;

   public boolean superieurA(Magnitude m) {
      return !egalA(m) && !inferieurA(m);
   }
   ...
```



chaque sous-classe concrète admet une implémentation différente pour `égalA()` et `inférieurA()`

(exemple inspiré du cours GL de D. Bardou, UPMF)