Bases de la POO / Java

Classes abstraites et interfaces

CLASSE ABSTRAITE
INTERFACE

Classe abstraite

- Une classe abstraite n'est pas **instanciable**
  - On ne peut pas créer d'objets de cette classe
- Une classe abstraite permet de :
  - définir des constantes et/ou des services utilisables par d'autres classes
  - de définir des méthodes abstraites (sans code)
- On peut définir une classe abstraite sans méthode abstraite

Définition d'une classe abstraite

- Utilisation du modificateur **abstract**
- public **abstract** class C {…}

Définition d'une méthode abstraite

- Utilisation du modificateur **abstract**
- public **abstract** void m ( ) **;**

- La méthode est déclarée dans la classe mais pas implémentée.
- Une classe qui contient au moins une méthode abstract doit être déclarée abstract.

Intérêt d'une classe abstraite (1)

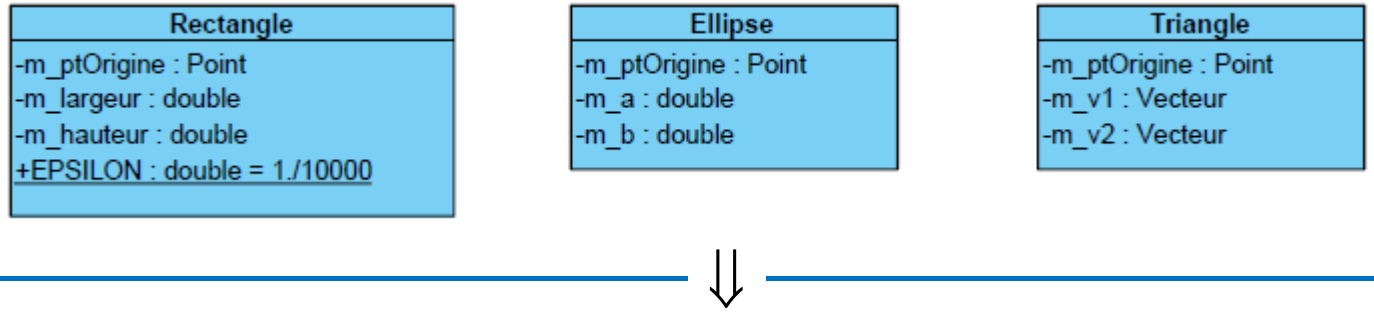
- Définir des constantes et/ou des services utilisables par d'autres classes

```
public abstract class Arithm
{
        public static double EPSILON = 0.0001;
        public static int pgcd (int n, int m) {…}
        public static int ppcm (int n, int m){…}
}
```

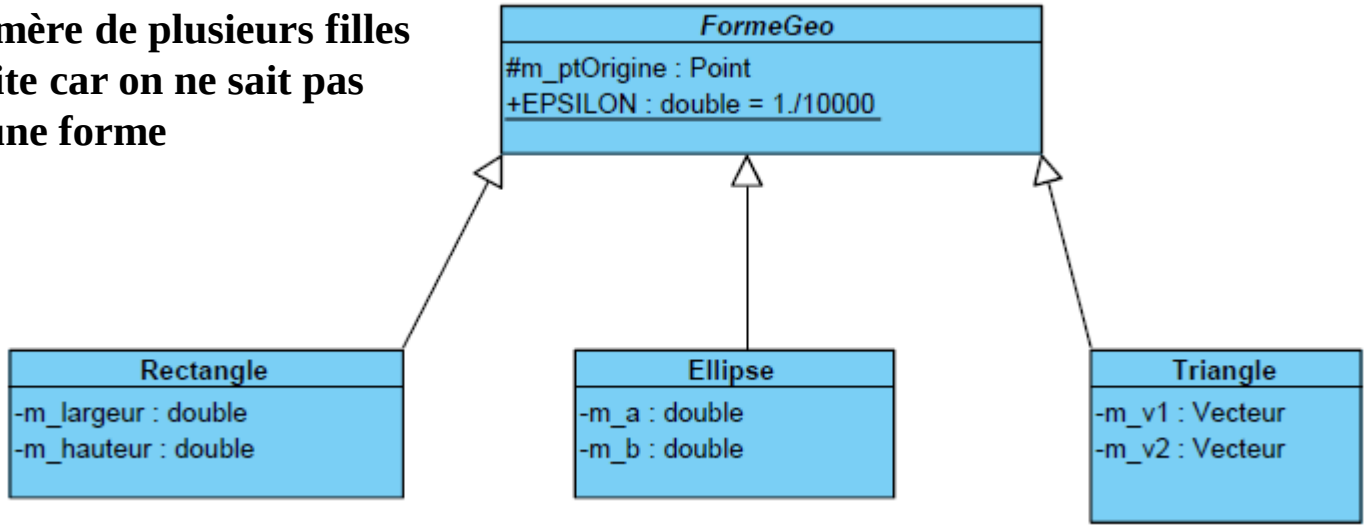
Intérêt d'une classe abstraite (2)

- Définir des membres (attributs et méthodes) communs à une hiérarchie de classes
  - Factorisation d'attributs et de méthodes
- Si la classe abstraite comporte des méthodes abstraites, les classes dérivées peuvent les implémenter
  - complètement, partiellement, ou pas du tout
- S'il reste au moins une méthode non implémentée dans la classe dérivée, celle-ci est obligatoirement abstract.

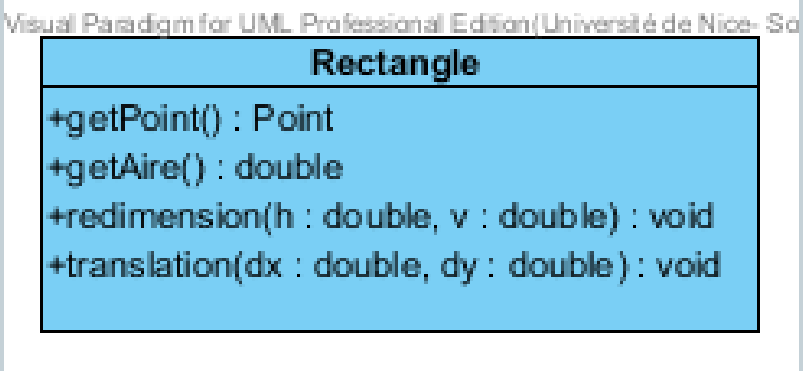
Factorisation d'attributs

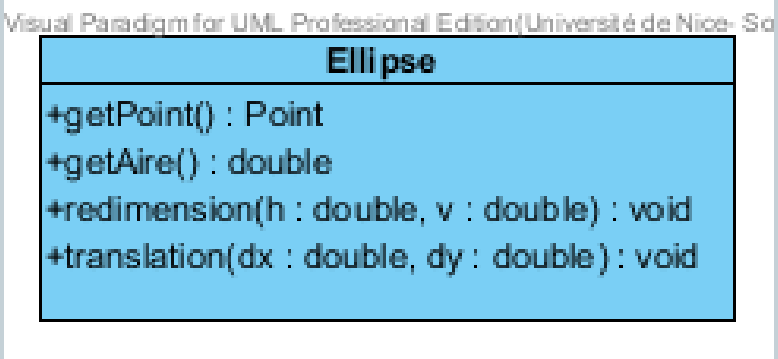


- **classe mère de plusieurs filles**
- **abstraite car on ne sait pas créer une forme**



Factorisation de méthodes (1)





- public Point getPoint(){return m_ptOrigine;}

→ FormeGeo

- public void translation (double dx, double dy) throws Exception

```
{     if (dx <0 || dy<0) throw new
Exception();
        m_ptOrigine.translation(dx, dy);
}
```

→ FormeGeo

Factorisation de méthodes (2)

Rectangle

```
public void redimension (double h, double v) throws Exception
 {
        if ((h==0) || (v==0)) throw new Exception();
        if (h<0) h=1/Math.abs(h);
        if (v<0) v=1/Math.abs(v);
        m_largeur *= h;
        m_hauteur *= v;
 }
public double getAire() {return m_hauteur*m_largeur;}
```

Ellipse

```
public void redimension (double h, double v) throws Exception
 {
        if ((h==0) || (v==0)) throw new Exception();
        if (h<0) h=1/Math.abs(h);
        if (v<0) v=1/Math.abs(v);
        m_a *= h;
        m_b *= v;
 }
public double getAire() {return Math.PI*m_a*m_b;}
```

Factorisation de méthodes (3)

- on crée une méthode abstraite dans FormeGeo
- redéfinie dans Rectangle, Ellipse,…

  - **abstract** public void redimension (double h, double v) throws Exception;

  - **abstract** public double getAire() ;

Classe abstraite FormeGeo


```
abstract public class FormeGeo
{
    private Point m_ptOrigine ;

    public FormeGeo () { m_ptOrigine = new Point () ; }
    public FormeGeo (double x, double y) { m_ptOrigine =  new Point(x, y) ; }
    public FormeGeo (Point p) { m_ptOrigine = (Point) p.clone(); }

    public Point getPoint() { return m_ptOrigine ;}

    public void translation (double dx, double dy) throws Exception
    {       if (dx <0 || dy<0) throw new Exception();
            m_ptOrigine.translation(dx,dy);
    }

    abstract public double getAire() ;
    abstract public void redimension (double h, double v) throws Exception;
}
```

Modification dans Rectangle


```
public class Rectangle extends FormeGeo
{
    private double m_largeur ;
    private double m_hauteur ;

    public Rectangle(Point p) {
        super(p);
        m_largeur = 0.;
        m_hauteur = 0.;
    }
     public double getAire() {return m_hauteur*m_largeur;}
    // ...
}
```

Utilisation (1)


```
public class T_FormeGeo_N1 {

   public static void main (String[] args) throws Exception{
      Tests.Begin("FormeGeo", "V 0.0.0");
         Tests.Design("Essai", 3); {

            Tests.Case("Construction d'un tableau de formes"); {

               Rectangle r= new Rectangle(2.,3.);
               Ellipse e= new Ellipse(4.,2.);

               Tests.Unit(6.0,  r.getAire());
               Tests.Unit(25.132741228718345,  e.getAire());
        }
      }

      Tests.End();
   }
}
```

Utilisation (2)

- On ne peut pas créer d'instances d'une classe abstraite C

```
FormeGeo f = new FormeGeo();
error: FormeGeo is abstract; cannot be instantiated
```

- Mais on peut déclarer des variables de type C et les instancier avec un objet d'une classe concrète (non abstraite) dérivée de C

```
FormeGeo f = r;
Tests.Unit("0.0, 0.0)",  f.getPoint());
Tests.Unit(6.0,  f.getAire());
```

Utilisation (3)


```
    FormeGeo tab[] = new FormeGeo[2];
    tab[0] = r;
    tab[1]= e;

    double aireTotale = 0.;
    for (int i=0; i<tab.length; i++) {
           aireTotale += tab[i].getAire();
}

    Tests.Unit(31.132741228718345,  aireTotale);
```

Interface

- C’est une classe "entièrement" abstraite
  - sans attribut d'instance,
  - avec uniquement des méthodes abstraites et/ou des constantes
- Une interface est implicitement publique
  - Ainsi que toutes ses méthodes
  - Dans le package dans lequel elle est définie
- Permet de définir un « contrat »
  - que doivent remplir toutes les classes qui en « héritent »
  - Une ou plusieurs méthodes

Déclaration d'une interface

- Mot-clé «**interface**»

➔ ***interface*** *I {…}*

Exemples


```
interface Affichable
{
    String toString ();
 }

interface Deplacable
{
    void translation (double dx, double dy) throws Exception;
 }

interface Pivotable
{
    static final double PI=3.14159 ;
    void rotation (double angle);
 }
```

Interface

→Une interface permet de décrire un ensemble de comportements

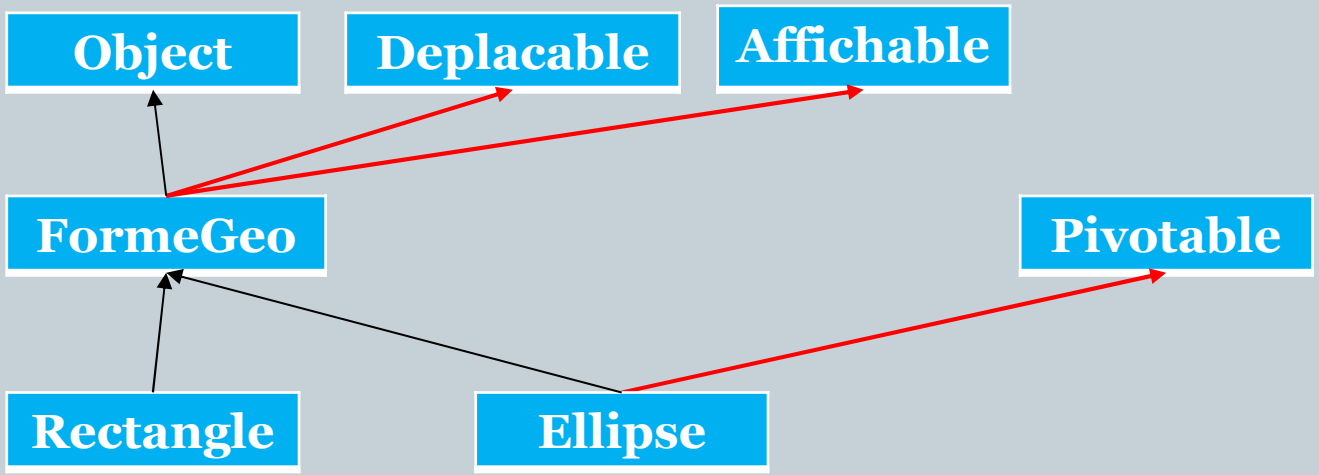
→Une classe « implémente » une interface

→ Moyen de mettre en œuvre l'héritage multiple

- **Une classe peut n'hériter que d'une seule classe, mais implémenter plusieurs interfaces**

Héritage multiple (1)

- toutes les formes géométriques sont affichables et déplaçables
- parmi les formes, seule l'ellipse est pivotable



Héritage multiple (2)


```
abstract public class FormeGeo implements Deplacable, Affichable
{
    // doit implémenter la méthode translation
    public void translation (double dx, double dy) throws Exception {...}

    // doit implémenter la méthode toString
    //(ou dans les classes dérivées)
}

public class Ellipse extends FormeGeo implements Pivotable
{
    ...
    public String toString() {...}
    public void rotation(double angle) {...}

}
```