CLASSES ABSTRAITES ET INTERFACES

DJAAFRI LYES

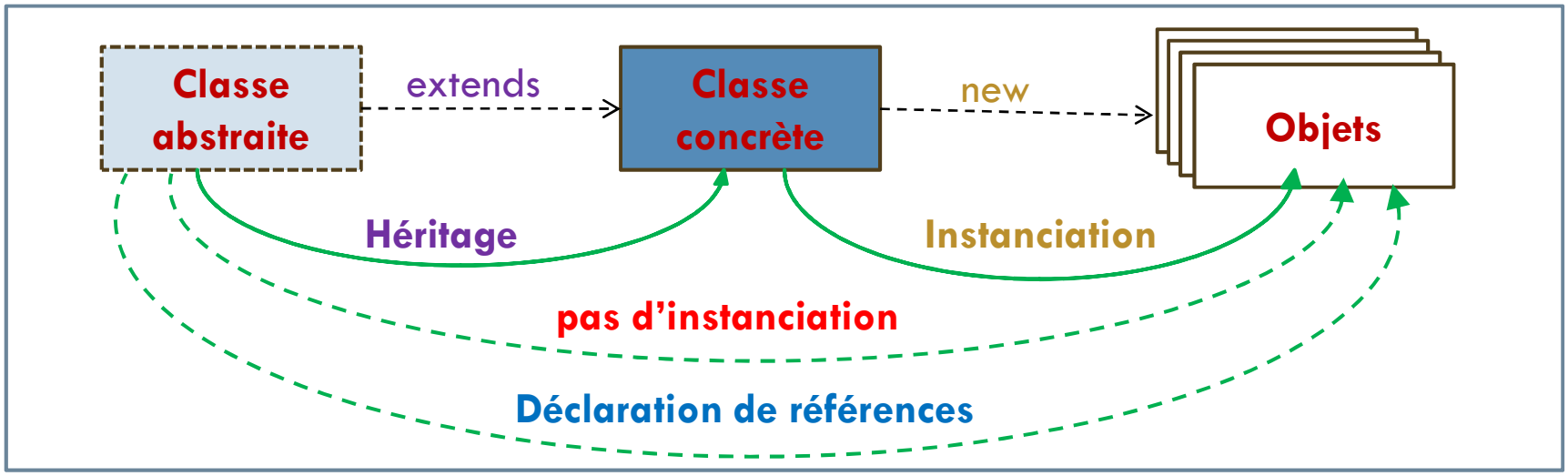
Université Alger1 /Faculté de sciences/Département MI

Classes abstraites (1)

- Une **classe abstraite** est une classe qui **ne peut pas instancier des objets.**
- Elle a seulement pour rôle d'être une classe de base pour une dérivation.
- Une classe abstraite doit être qualifier, obligatoirement, par le modificateur **abstract**
- Une classe abstraite est une classe qui a une ou plusieurs **méthodes abstraites**
- Une **méthode abstraite** est une méthode sans corps, elle est définie par sa signature et un point-virgule à la fin.
- Une méthode abstraite doit être déclaré avec le mot clé **abstract**

Classes abstraites (2)

- Une classe concrète qui **hérite** d'une classe abstraite, doit donner un corps (**implémenter**) toutes les méthodes abstraites
- Les classe abstraite **ne peut pas être instanciée**. Ce sont les classes concrètes qui seront **instanciée**
- La classe abstraite est considérée comme un type et on peut **déclarer des référence** avec des **classe abstraite.**

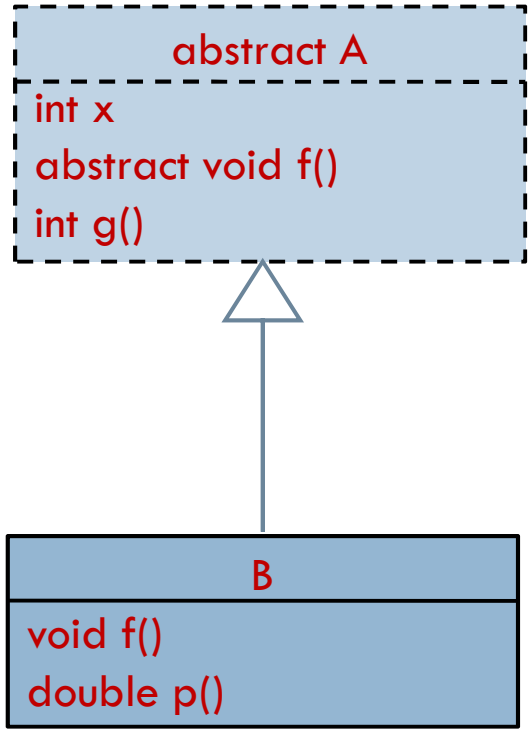


Classes abstraites: les règles

- Si une méthode est déclarée abstraite alors la classe où elle est déclarée doit être une classe abstraite
- Java permet de déclarer une classe abstraite même si elle ne contient aucune méthode abstraite
- Si une sous-classe n'implémente pas toutes les méthodes abstraite dont elle hérite, cette sous doit être déclarée abstraite (et aussi elle ne peut pas être instanciée)
- Une méthode déclarée abstraite, ne peut pas utiliser les modificateur: final, static ou private
- Les modificateurs final et abstract sont **incompatible** avec une classe

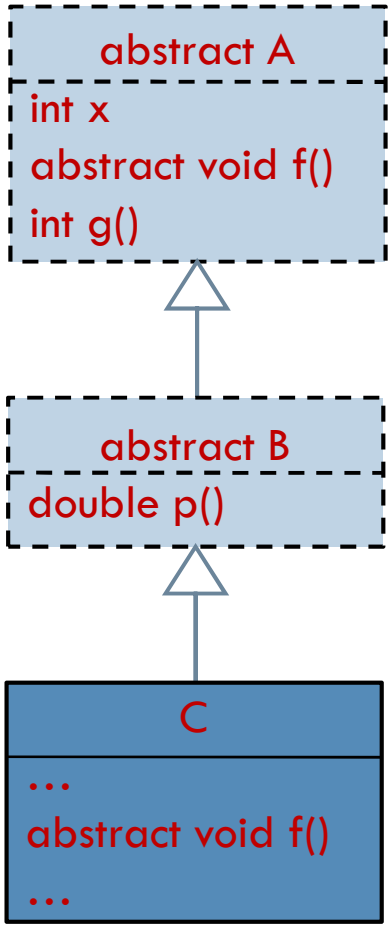
Classe abstraite: exemple 1


```
abstract class A { //déclaration d'une classe abstraite
     int x;
     public abstract void f(); //f() est une méthode abstraite
     int g() {...}
}
class B extends A {
     …
     public void f() {…} // implémentation de la méthode f()
     public double p() {…}
…
}

     …
     A a; // correcte, La classe A est un type
     a = new A(…); // instanciation interdite, A est abstraite
     B b1=new B(…);
     A b2=new B(…);
```



Classe abstraite: exemple 2


```
abstract class A { //déclaration d'une classe abstraite
      int x;
      public abstract void f(); //f() est une méthode abstraite
      int g() {...}
}

// B est abstrait parce qu'elle n'a pas implémenté f()
abstract class B extends A {
       …
        // f() n'est pas implémentée dans B
       public double p() {…}
      …
}
class C extends B {
       …
      public void f(){…} //f() est implémentée dans C
       …
}
       …
       A a; // correcte, La classe A est un type
       a = new A(…); // erreur, instanciation interdite, A est abstraite
       B b1=new B(…);
       A b2=new B(…);
```

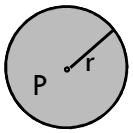


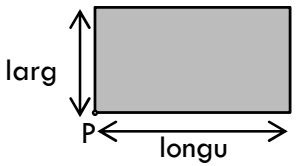
Utilité des classes abstraites

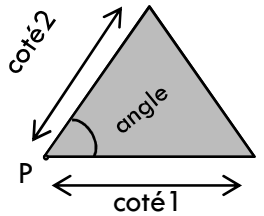
- définir des concepts incomplets qui devront être implémentés dans les sous classes: utiliser les méthode abstraites
- factoriser le code : regrouper toutes les propriétés et méthodes implémentées (non abstraites) communes à toutes les sous-classes dans la classe abstraite
- Polymorphisme : l'utilisation d'une classe abstraite comme type pour les références+ implémentation des méthodes abstraites dans les sous classe ➔ polymorphisme efficace

Exemple : Formes Géométriques(1)

- On veut définir une application permettant de manipuler des formes géométriques (triangles, rectangles, cercles…).



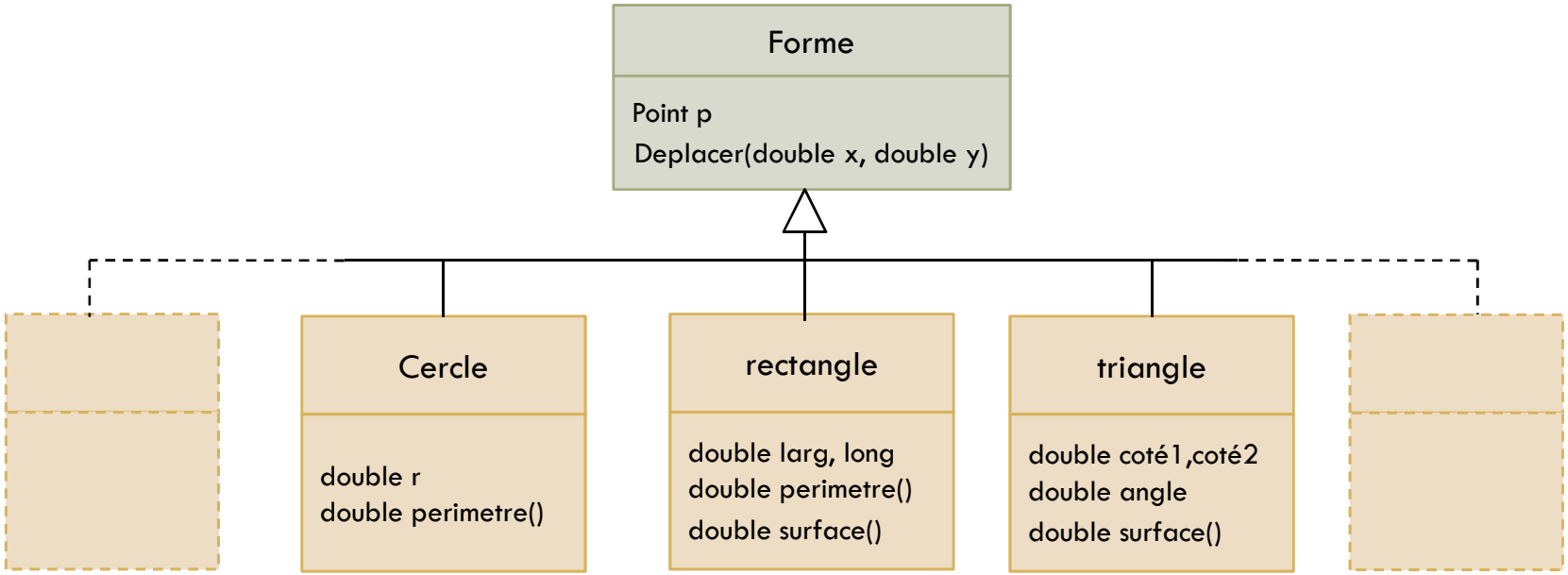




- Chaque forme est définie par sa position dans le plan
- Chaque forme peut être déplacée (modification de sa position), peut calculer son périmètre, sa surface
- Est-ce qu'on peut factoriser le code?

Exemple : Formes Géométriques(2)

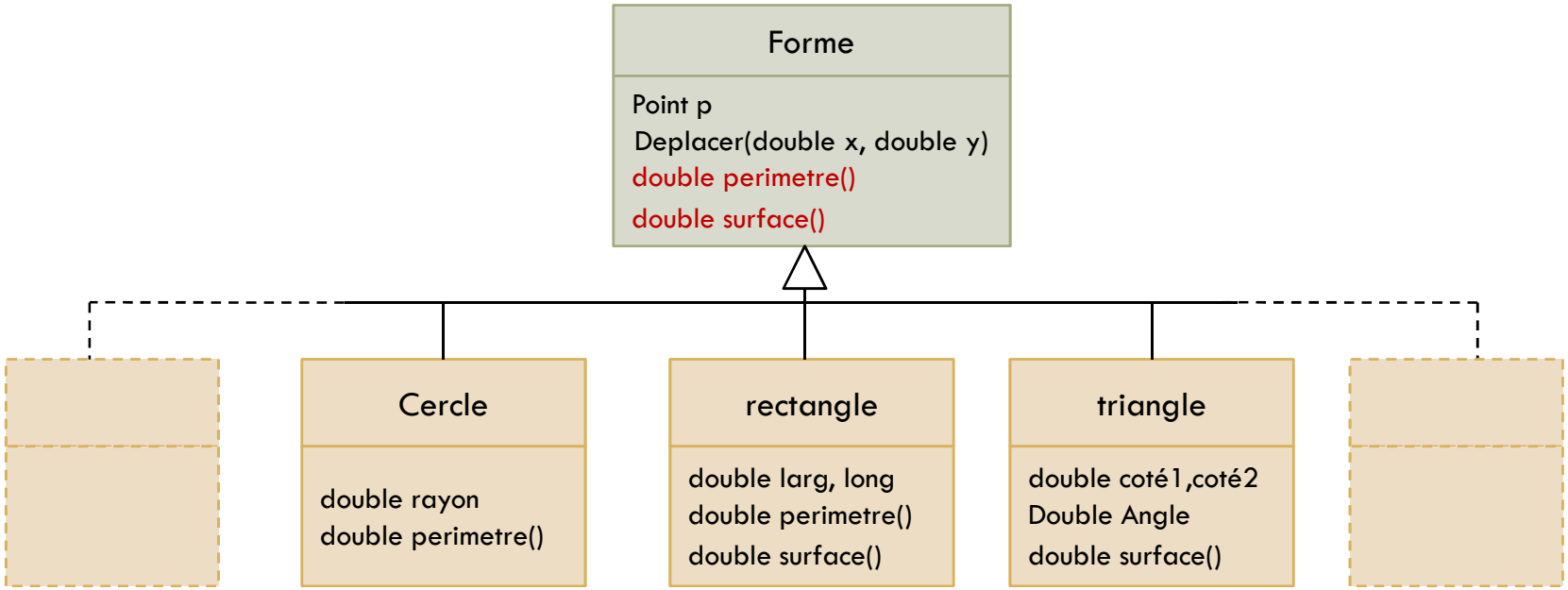
- Une 1ère solution :



- Pour factoriser le code, on définit dans la classe forme: p de type Point et la méthode déplacer().
- mais cette solution **ne garantie pas** que les sous-classes définissent les méthodes perimetre() et surface().

Exemple : Formes Géométriques (3)

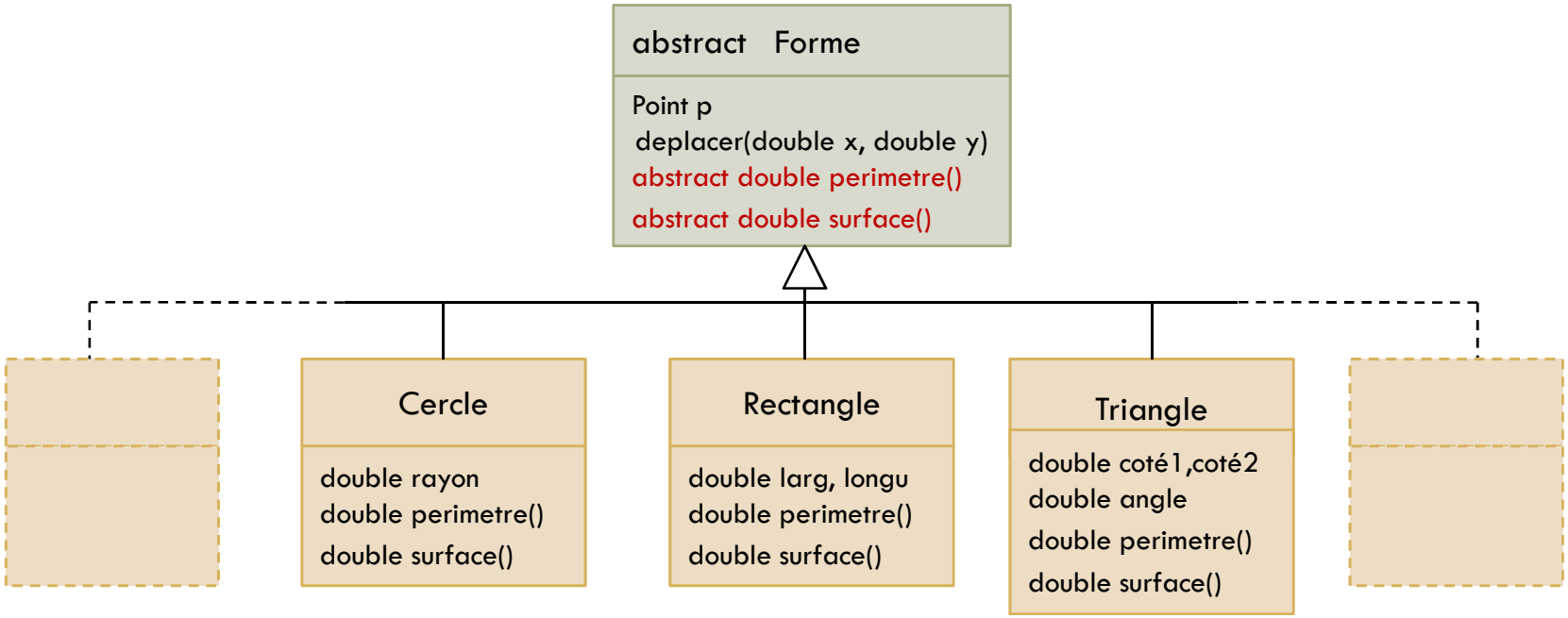
- Une 2ème solution :



- On ajoutera les méthodes perimetre() et surface() à la classe Forme en leur donnant un code par défaut. Mais comment calculer le périmètre ou la surface d'une forme quelconque ?
- Une sous-classe peut ne redéfinir les méthodes perimetre() et surface() !

Exemple : Formes Géométriques (4)

Une solution propre et élégante : les classes abstraites



- On ajoutera les méthodes **perimetre()** et **surface()** à la classe Forme en tant que méthodes abstraites.
- Toutes les sous-classe sont obligées d'implémenter les méthodes **perimetre()** et **surface()**.

Exemple : Formes Géométriques(5)


```
abstract class Forme {
  static final double PI=3.14;
  protected Point p;
  public Forme (double x, double y) { p=new Point(x,y); }
  public void deplacer(double dx,double dy) {
    p.setX(p.getX()+dx);
    p.setY(p.getY()+dy);
  }
  public abstract double perimetre();
  public abstract double surface();
}
```

```
class Cercle extends Forme {
  private double r;
  Cercle(Point centre, double r) {
       super(centre.getX(), centre.getY()); this.r=r;
  }
  public double perimetre() {return 2*PI*r;}
  public double surface() {return PI*r*r;}
}
```

```
class Rectangle extends Forme {
  private double longu,  larg;
  public Rectangle(Point p, double longu, double larg) {
       super(p.getX(), p.getY()); this.longu=longu; this.larg=larg;
  }
  public double perimetre() {return 2*(larg+longu);}
  public double surface(){return larg*longu;}
}
```

```
class Triangle extends Forme {
  private double coté1,  coté2; private double angle;
  public Triangle(Point p, double coté1, double coté2, double angle)
       {
        super(p.getX(), p.getY()); this.coté1=coté1; this.coté2=coté2;
       this.angle=angle;
  }
  public double perimetre() {
       double coté3=Math.sqrt(coté1*coté1+coté2*coté2-
       2*coté1*coté2*Math.sin(angle*PI/180));
       return coté1+coté2+coté3;
  }
 public double surface(){…}
}
```

Exemple : Formes Géométriques (6)


```
public class TestFormesGeo {
  public static void toutDeplacer(Forme[] tabForme ,double dx,double dy) {
  for (int i=0; i < tabForme.length; i++)
      if(tabForme[i]!=null) tabForme[i].deplacer(dx,dy);//polymorphisme
  }
  public static  double périmetreTotal(Forme[] tabForme){
      double pt = 0.0;
      for (int i=0; i < tabForme.length; i++)
                  if(tabForme[i]!=null) pt += tabForme[i].perimetre();//polymorphisme
      return pt;
  }
    public static void main(String[] args){
      Forme[] tf = new Forme[3];
      tf[0] = new Rectangle(new Point(1,2), 2.5, 6.8);   // Conversion implicite, tout rectangle est une forme
      tf[1] = new Cercle(new Point(-1,-2), 4.66); // Conversion implicite, tout cercle est une forme
      tf[2] = new Triangle(new Point(1,2), 2.0, 3.0,45); // Conversion implicite, tout triangle est une forme
      toutDeplacer(tf,6,-4);
      System.out.println("le périmètre total de tous les formes est:"+ périmetreTotal(tf));
    }
}
```

- Il reste les interfaces …