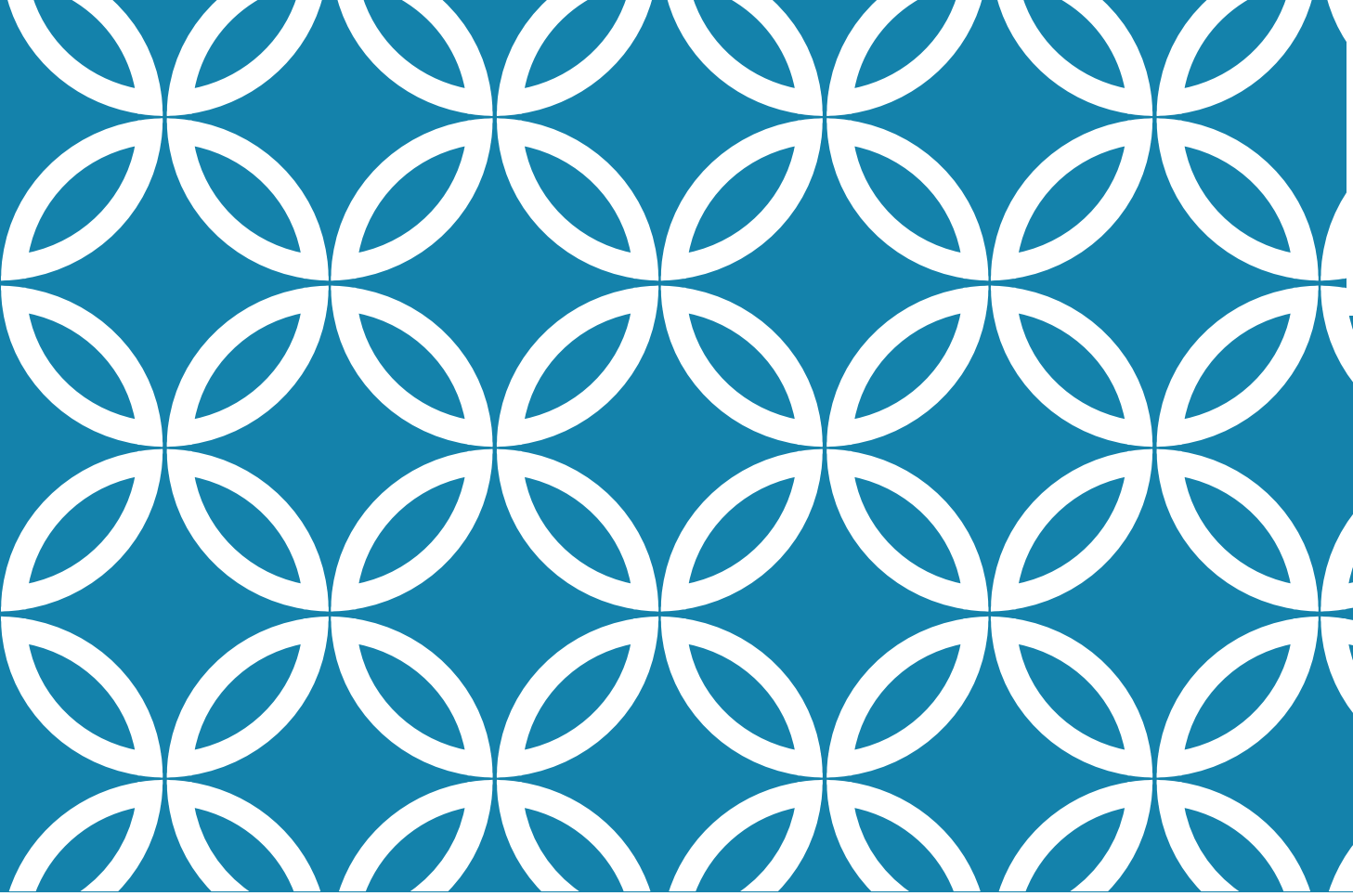


COURS 7 : HÉRITAGE ET ABSTRACTION

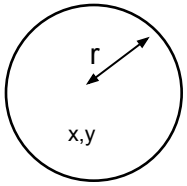
Par Aïcha El Golli

aicha.elgolli@essai.ucar.tn

LES CLASSES ABSTRAITES

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…

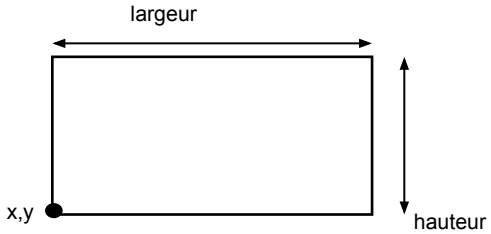


Attributs :

```
double x,y; //centre
double r;   // rayon
```

Méthodes :

```
déplacer(double
dx,double dy)
double surface()
double périmètre()
```

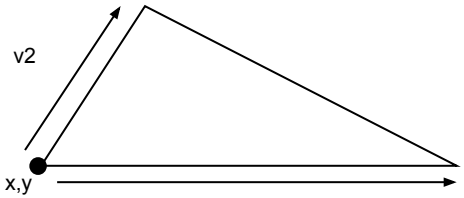


Attributs :

```
double     x,y;     //coin
inf-gauche
double largeur, hauteur;
```

Méthodes :

```
déplacer(double  dx,double
dy)
double surface()
double périmètre()
```



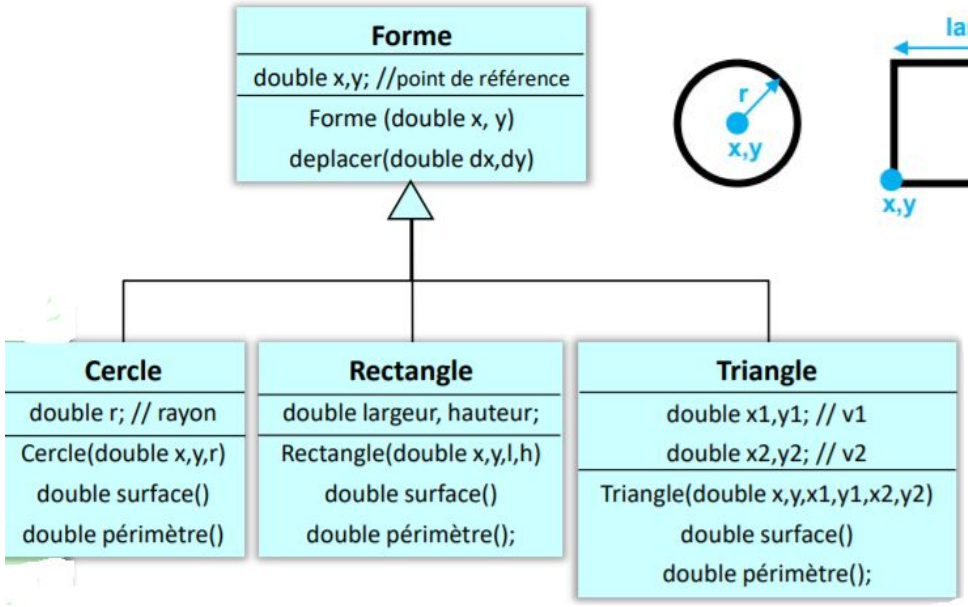
Attributs : v1

```
double x,y;   // 1 sommet
double x1,y1; // v1
Double x2,y2; // v2
```

Méthodes :

```
déplacer(double   dx,double
dy)
double surface()
double périmètre()
```

LES CLASSES ABSTRAITES




```
class Cercle extends Forme {
protected double r ;
public double surface(){
       return Math.PI * r* r;}
public double perimetre(){
        return 2 * Math.PI * r;}
}
```

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

Les classes abstraites

```
public class ListeDeFormes {

public static final int NB_MAX = 30 ;

private Forme [] tabForme= new Forme[NB_MAX];

private int nbFormes=0 ;


public void ajouter(Forme f){

if (nbFormes < NB_MAX)

       tabForme[nbFormes++]=f;

}

public void toutDeplacer(double dx, double dy){

       for (int i=0 ; i< nbFormes ; i++)

       tabFormes[i].deplacer(dx,dy);}

public double perimetreTotal(){

double pt = 0.;

for(int i=0 i< nbFormes; i++)

       pt += tabFormes[i].perimetre();

return pt;}

}
```

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

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

Définir une méthode perimetre dans Forme ?

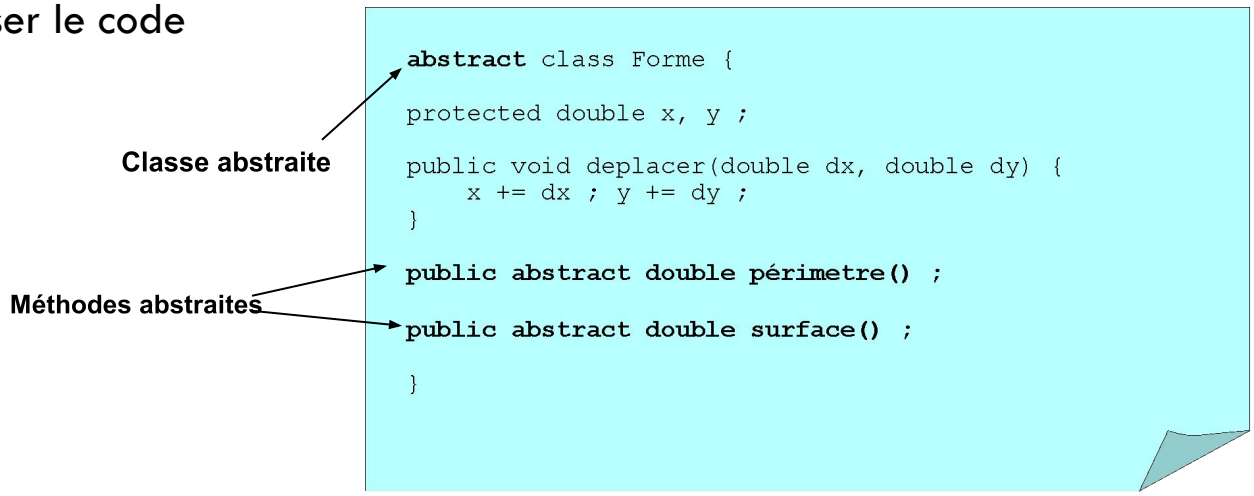
```
public double perimetre(){

      return 0.0;//ou -1. ??

}
```

Une solution propre et élégante : les classes abstraites

UTILITÉ DES CLASSES ABSTRAITES

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 !!!)*

UTILITÉ DES 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

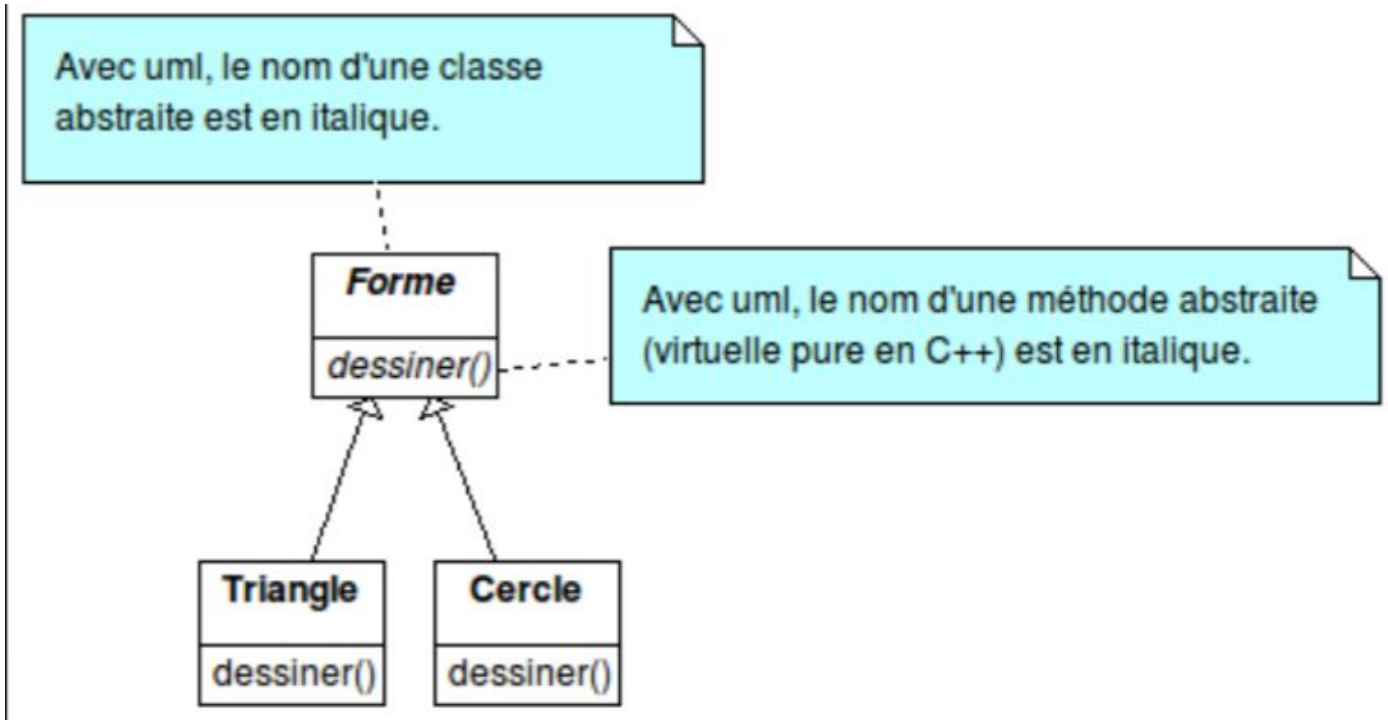
```
public abstract  class ClasseA {

   ...


}
```

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

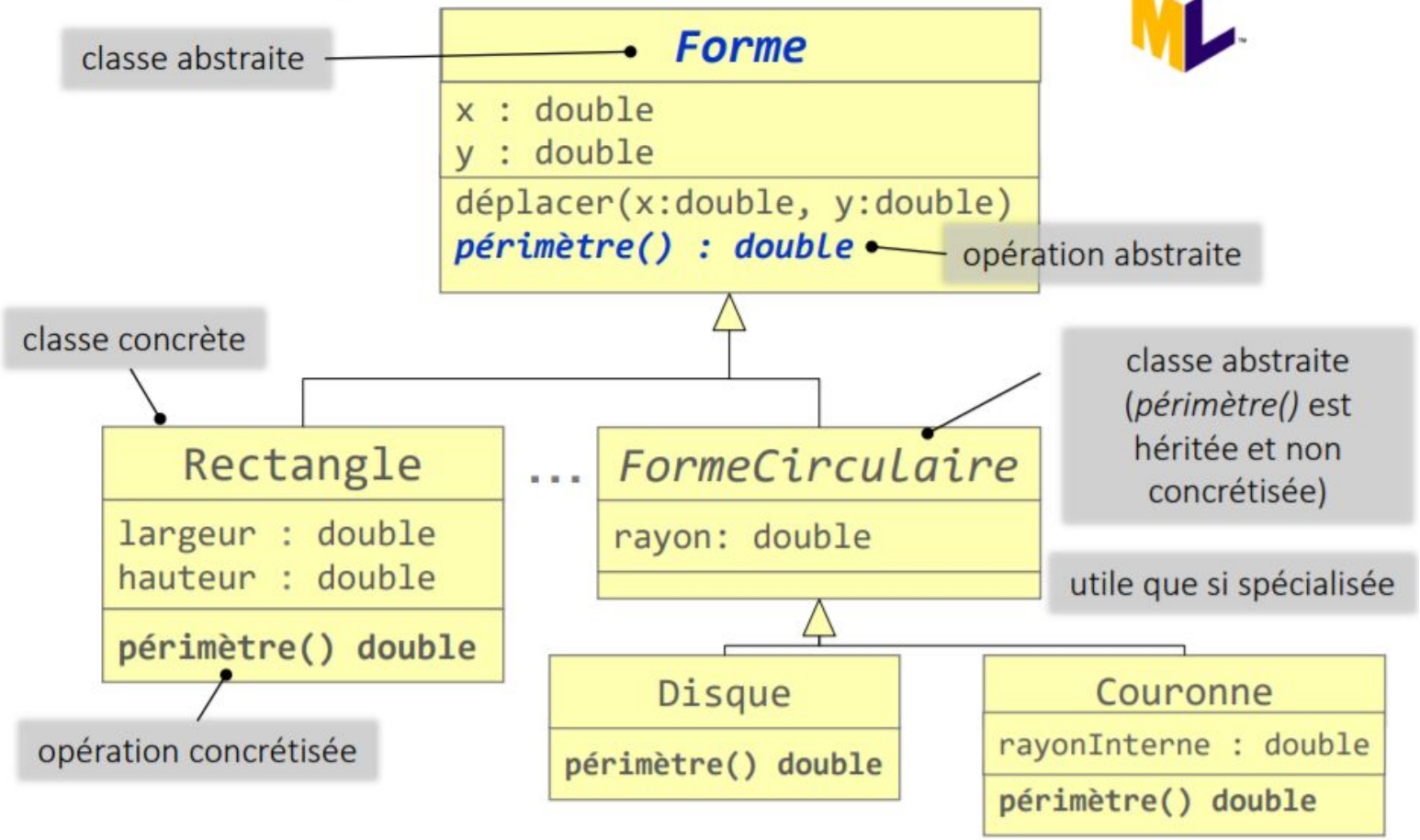
UTILITÉ DES CLASSES ABSTRAITES

- Une classe abstraite est une description d'objets. Elle est 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.
- 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 le plus souvent l'utilisation de classes abstraites.

UML ET LES CLASSES ABSTRAITES



Classes abstraites et diagrammes de classes UML



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

```
public class ListeDeFormes {
public static final int NB_MAX = 30 ;
private Forme [] tabForme= new Forme[NB_MAX];
private int nbFormes=0 ;

public void ajouter(Forme f){
if (nbFormes < NB_MAX)
tabForme[nbFormes++]=f;
}
public void toutDeplacer(double dx, double
dy){
for (int i=0 ; i< nbFormes ; i++)
tabForme[i].deplacer(dx,dy);
}
public double perimetreTotal(){
double pt = 0.;
for(int i=0 ;i< nbFormes; i++)
pt += tabForme[i].perimetre();
return pt; }…}
```

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

LES INTERFACES

- Une interface est constituée d'un ensemble de déclarations de méthodes sans implantation
- On n'y trouve uniquement, pour chaque méthode, que la définition de son profil, c'est-à-dire son en-tête suivi de ;
- Les interfaces sont définies par le mot clé ***interface***
- Si les interfaces permettent de déclarer des variables de référence portant leur type, elles ne sont pas, par contre, instanciables. En particulier une interface ne possède pas de constructeur
- Une classe peut implanter une ou plusieurs interface(s) : la définition de la classe comporte alors le mot clé ***implements*** suivi d'une liste de noms d'interfaces (les noms des interfaces implantées y sont séparés par une virgule).

LES INTERFACES

- Du point de vue syntaxique, la définition d'une interface est proche de celle d'une classe abstraite.
- Il suffit de remplacer les mots-clés abstract class par le mot clé interface :
-
```
public interface Printable {
      public void print(); }
```
- Une fois l'interface déclarée, il est possible de déclarer des variables de ce (nouveau) type :

```
Printable document;
Printable[]  printSpool;
```

MOTIVATION

- Types plus abstraits que les classes : plus réutilisables
- Technique pour masquer l'implémentation: découplage public/privé : type/implémentation
- Favorise l'écriture de code plus général : écrit sur des types plus abstraits
- Relations de spécialisation multiple
  - entre les interfaces
  - entre les classes et les interfaces

LES INTERFACES: DÉCLARATION

Le corps d'une interface peut contenir :

- Des constantes qui sont implicitement publiques
  - **Les modificateurs static et final ne sont pas nécessaires (implicites).**
- Des méthodes abstraites qui sont obligatoirement publiques
  - **Le modificateur abstract n'est pas nécessaire (implicite).**
  - **Le modificateur public n'est pas nécessaire (implicite).**

Des méthodes avec une implémentation par défaut

- **Le modificateur default doit être utilisé.**

LES INTERFACES: DÉCLARATION

Une interface non publique n'est accessible que dans son package

```
package m2pcci.dessin;
import java.awt.Graphics;

public interface Dessinable {
    public void dessiner(Graphics g);
    void effacer(Graphics g);
}
```

Dessinable.java

Possibilité d'implémentation par défaut avec Java8

Toutes les méthodes sont abstraites

Elles sont implicitement publiques

Une interface publique doit être définie dans un fichier .java de même nom

interface

opérations abstraites

LES INTERFACES: DÉCLARATION

- Possibilité de définir des attributs à condition qu'il s'agisse d'attributs de type primitif
- Ces attributs sont implicitement déclarés comme static final

```
import java.awt.Graphics;
public interface Dessinable {
   public static final int MAX_WIDTH = 1024;
   int MAX_HEIGHT = 768;
   public void dessiner(Graphics g);
   void effacer(Graphics g);
 }
```

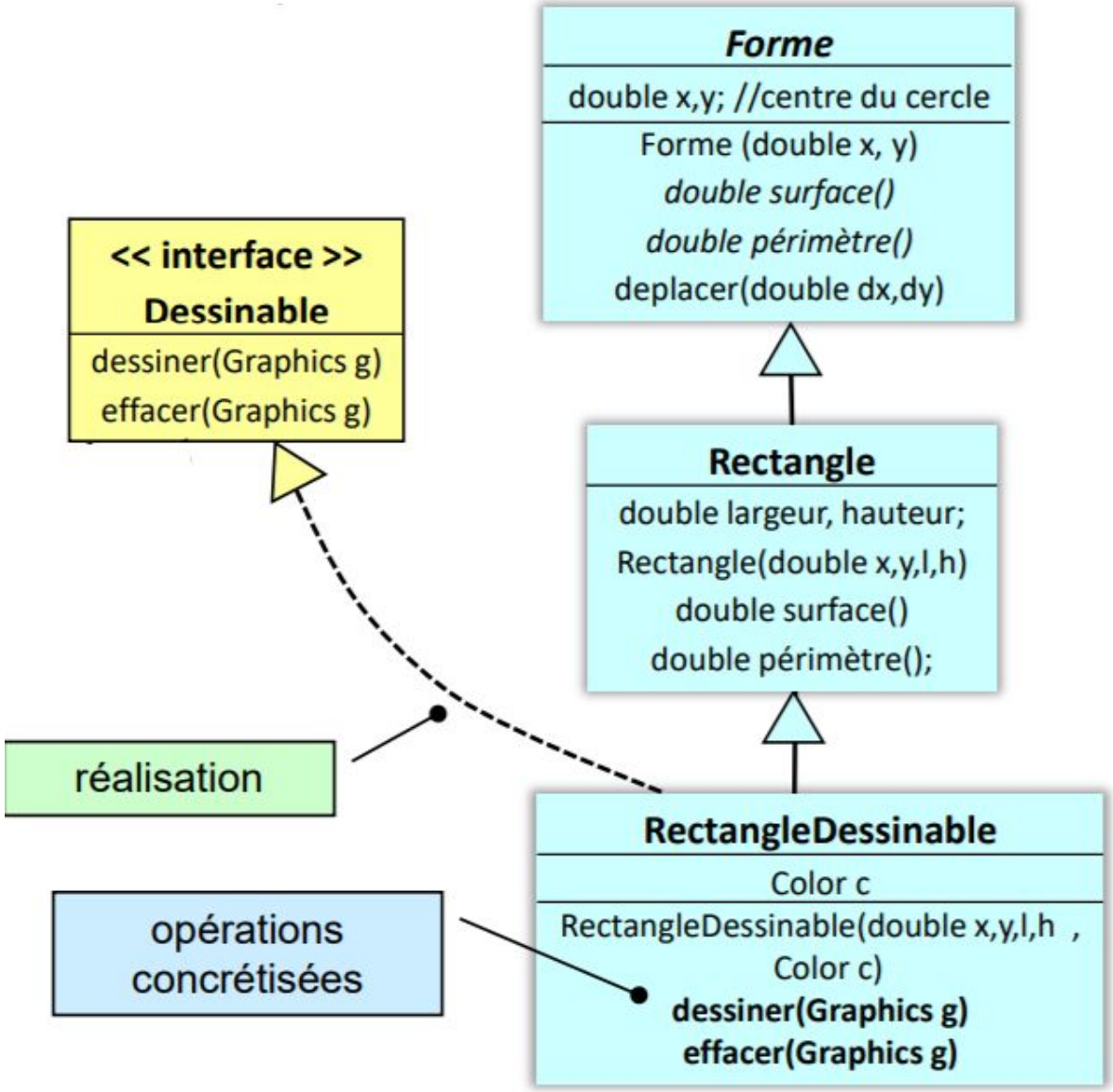
Dessinable.java

LES INTERFACES: RÉALISATION

- Une classe peut indiquer dans sa définition qu'elle implante une ou plusieurs interfaces. Elle doit alors à fournir une définition pour les méthodes de l'interface.
- L'intérêt des interfaces est de pouvoir rendre des classes, compatibles au niveau des types, sans forcément effectuer une liaison d'héritage.
- D'autre part le fait de pouvoir implanter plusieurs interfaces permet de simuler l'héritage multiple.
- Néanmoins les interfaces ne peuvent pas implanter de méthodes, ce qui rendra plus difficile la réutilisation de code que peut offrir un véritable héritage multiple.

LES INTERFACES

- Une interface est une collection d'opérations utilisée pour spécifier un service offert par une classe.
- Une interface peut être vue comme une classe abstraite sans attributs et dont toutes les opérations sont abstraites.
- Une interface est destinée à être “réalisée” (implémentée) par d’autres classes (celles-ci en héritent toutes les descriptions et concrétisent les opérations abstraites).
- Les classes réalisantes s'engagent à fournir le service spécifié par l'interface
- L’implémentation d’une interface est libre.

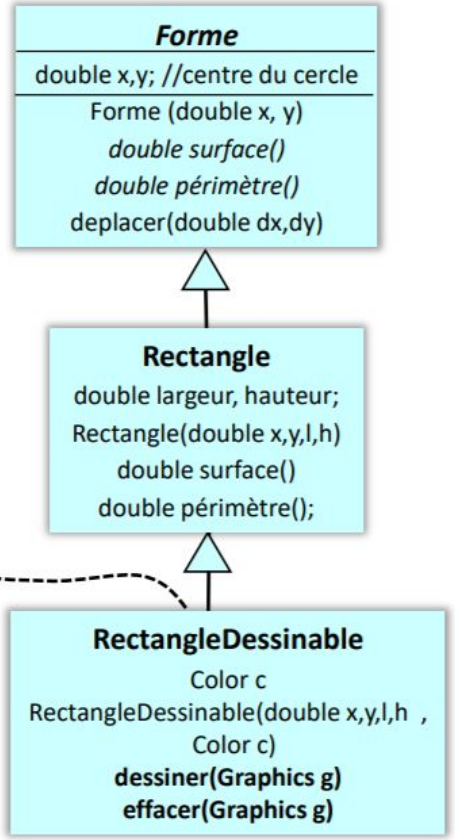

Forme
double x,y; //centre du cercle
Forme (double x, y)
double surface()
double périmètre()
deplacer(double dx,dy)
<< interface >>
Dessinable
dessiner(Graphics g)
effacer(Graphics g)
Rectangle
double largeur, hauteur;
Rectangle(double x,y,l,h)
double surface()
double périmètre();
réalisation
RectangleDessinable
Color c
RectangleDessinable(double x,y,l,h , Color c)
dessiner(Graphics g)
effacer(Graphics g)
opérations concrétisées

RÉALISATION D'UNE INTERFACE

De la même manière qu'une classe étend sa super-classe elle peut de manière **optionnelle** implémenter une ou plusieurs interfaces

- *dans la définition de la classe, après la clause* `extends` *`nomSuperClasse`, faire apparaître explicitement le mot clé* `implements` *suivi du nom de l'interface implémentée*

```
class RectangleDessinable extends Rectangle  implements Dessinable {
    private Color c;

    public RectangleDessinable(double x, double y,
                double l, double h, Color c) {
         super(x,y,l,h);
         this.c = c;
    }
    public void dessiner(Graphics g){
      g.drawRect((int) x, (int) y, (int) largeur, (int) hauteur);
    }
    public void effacer(Graphics g){
      g.clearRect((int) x, (int) y, (int)largeur, (int) hauteur);
    }
}
```



- *si la classe est une classe concrète* ***elle doit*** *fournir une implémentation (un corps) à chacune des méthodes abstraites définies dans l'interface (qui doivent être déclarées publiques)*

RÉALISATION D'UNE INTERFACE

Une classe JAVA peut implémenter **simultanément plusieurs** interfaces

- *la liste des noms des interfaces à implémenter séparés par des virgules doit suivre le mot clé* **`implements`**

```
class RectangleDessinable extends Rectangle
                          implements Dessinable , Enregistrable {
   private Color c;

    public RectangleDessinable(double x, double y,
              double l, double h, Color c) {
         super(x,y,l,h);
         this.c = c;
    }
    public void dessiner(Graphics g){
      g.drawRect((int) x, (int) y, (int) largeur, (int) hauteur);
    }
    public void effacer(Graphics g){
      g.clearRect((int) x, (int) y, (int)largeur, (int) hauteur);
    }

    public void enregistrer(File f) {
       ...
    }
}
```