# INTERFACE ET POLYMORPHISME

- Une interface peut être utilisée comme un type
  - *A des variables (références) dont le type est une interface il est possible d'affecter des instances de toute classe implémentant l'interface, ou toute sous-classe d'une telle classe.*

```
public class ZoneDeDessin {
    private nbFigures;
    private Dessinable[] figures;
    ...
    public void ajouter(Dessinable d){
      ...
    }
    public void supprimer(Dessinable o){
      ...
    }

    public void dessiner() {
         for (int i = 0; i < nbFigures; i++)
             figures[i].dessiner(g);
    }

}
```

```
Dessinable d;
..
d = new RectangleDessinable(…);
...
d.dessiner(g);
d.surface();
```

permet de s'intéresser uniquement à certaines caractéristiques d'un objet

règles du polymorphisme s'appliquent de la même manière que pour les classes :

- vérification statique du code
- liaison dynamique

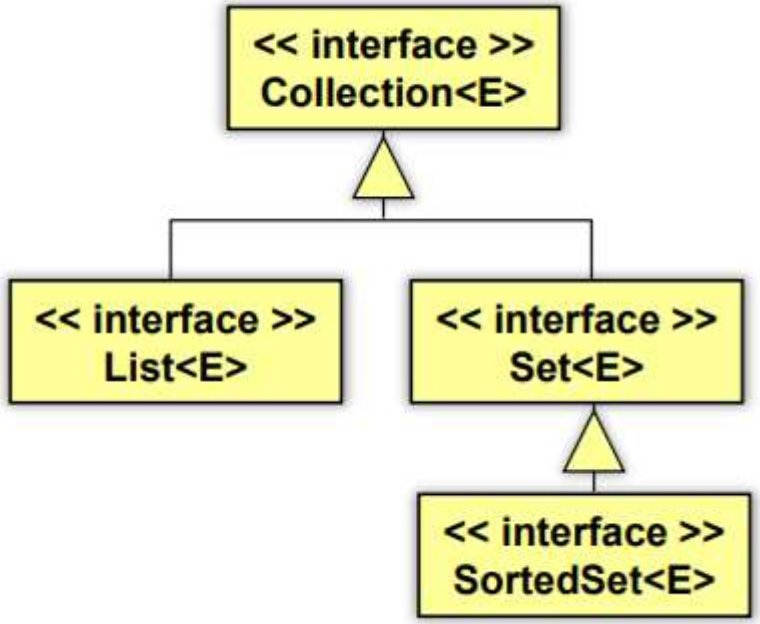
# HÉRITAGE D'INTERFACE

de la même manière qu'une classe peut avoir des sous-classes, une interface peut avoir des « sous-interfaces ».

une sous-interface

- hérite de toutes les méthodes abstraites et des constantes de sa « super-interface » ;
- peut définir de nouvelles constantes et méthodes abstraites

# HÉRITAGE D'INTERFACE

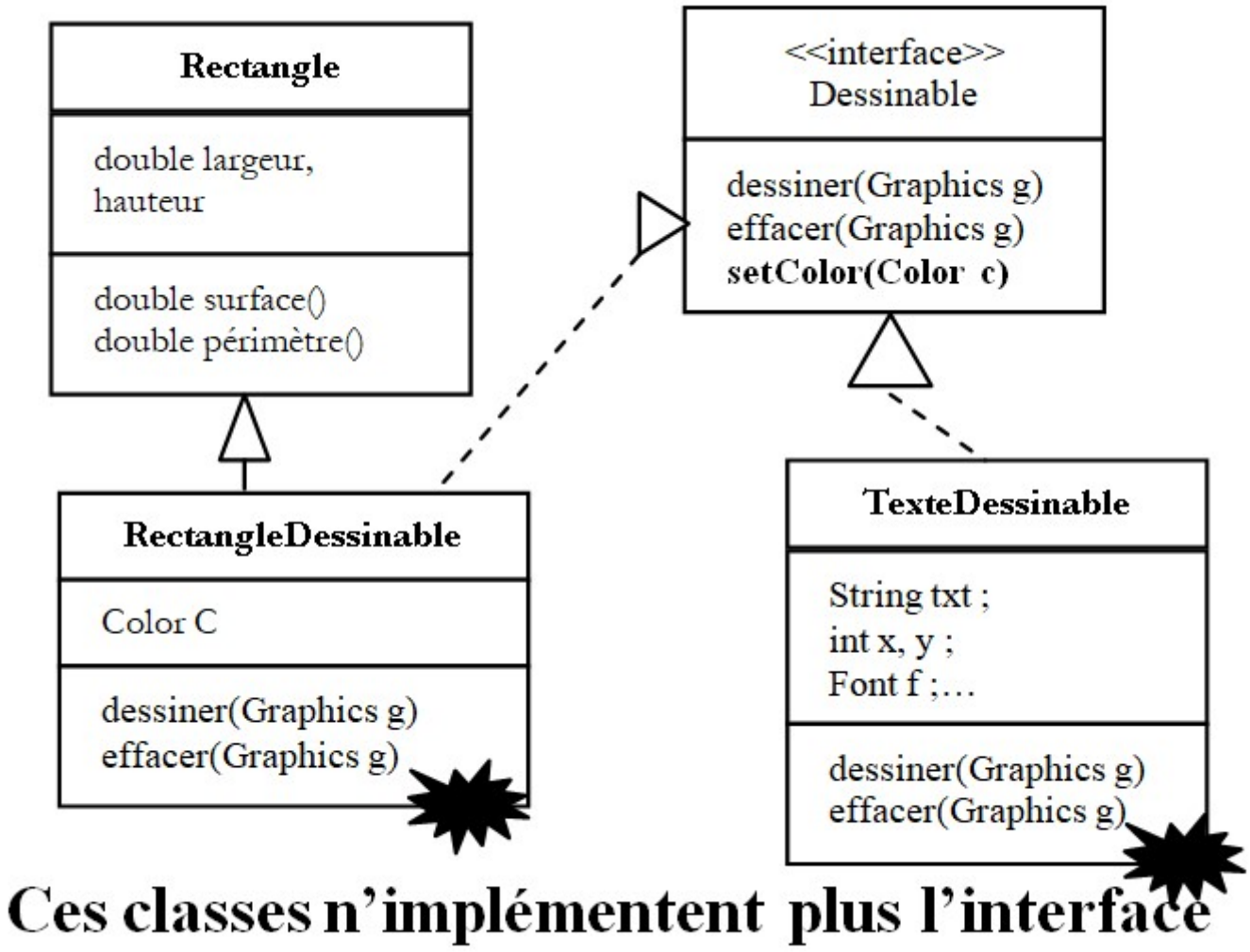



```
interface Set<E> extends Collection<E>
{
...
}
```

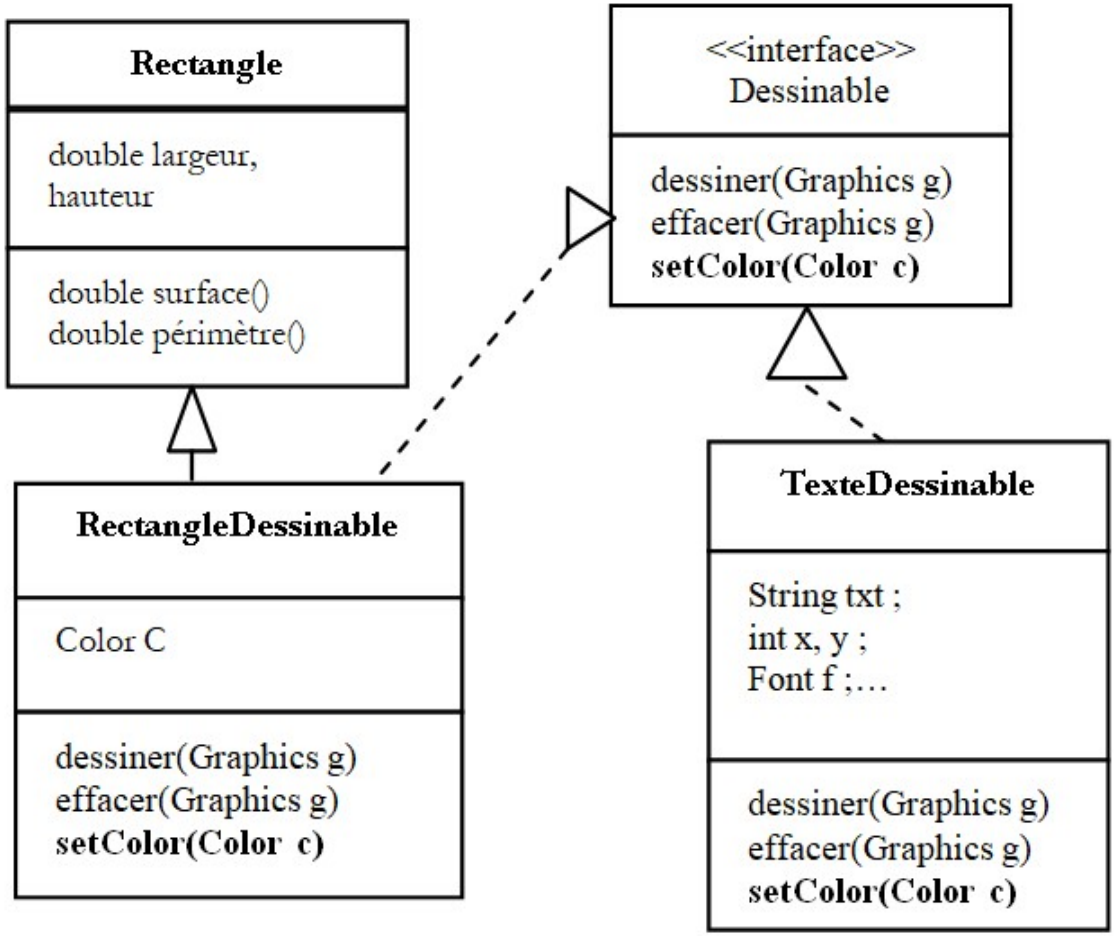
une classe qui implémente une interface doit implémenter toutes les méthodes abstraites définies dans l'interface et dans les interfaces dont elle hérite


# EVOLUTION DES INTERFACES

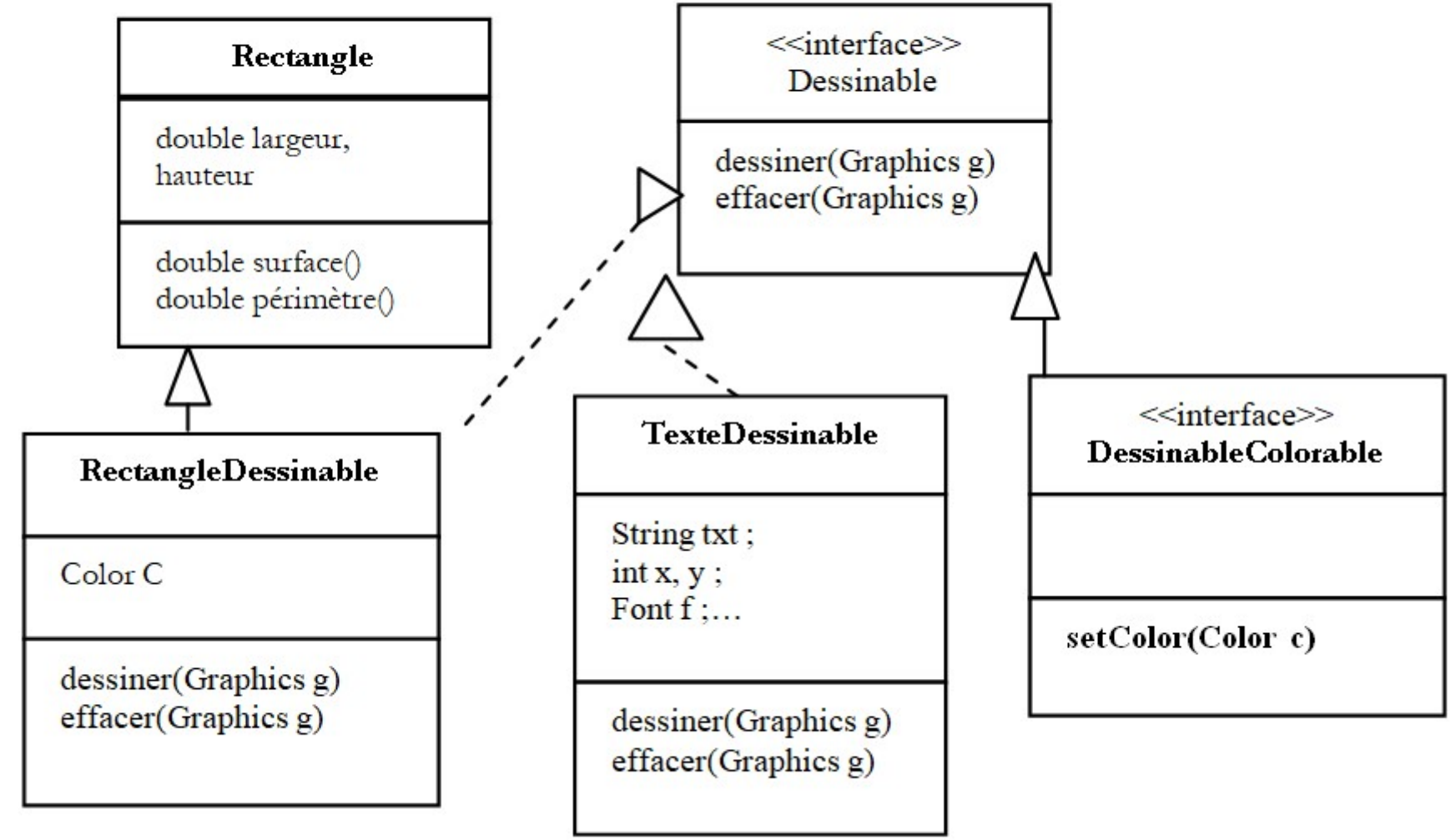
Quand on définit des interfaces il faut être prudent : tout ajout ultérieur « brise » le code des classes qui implémente l'interface.

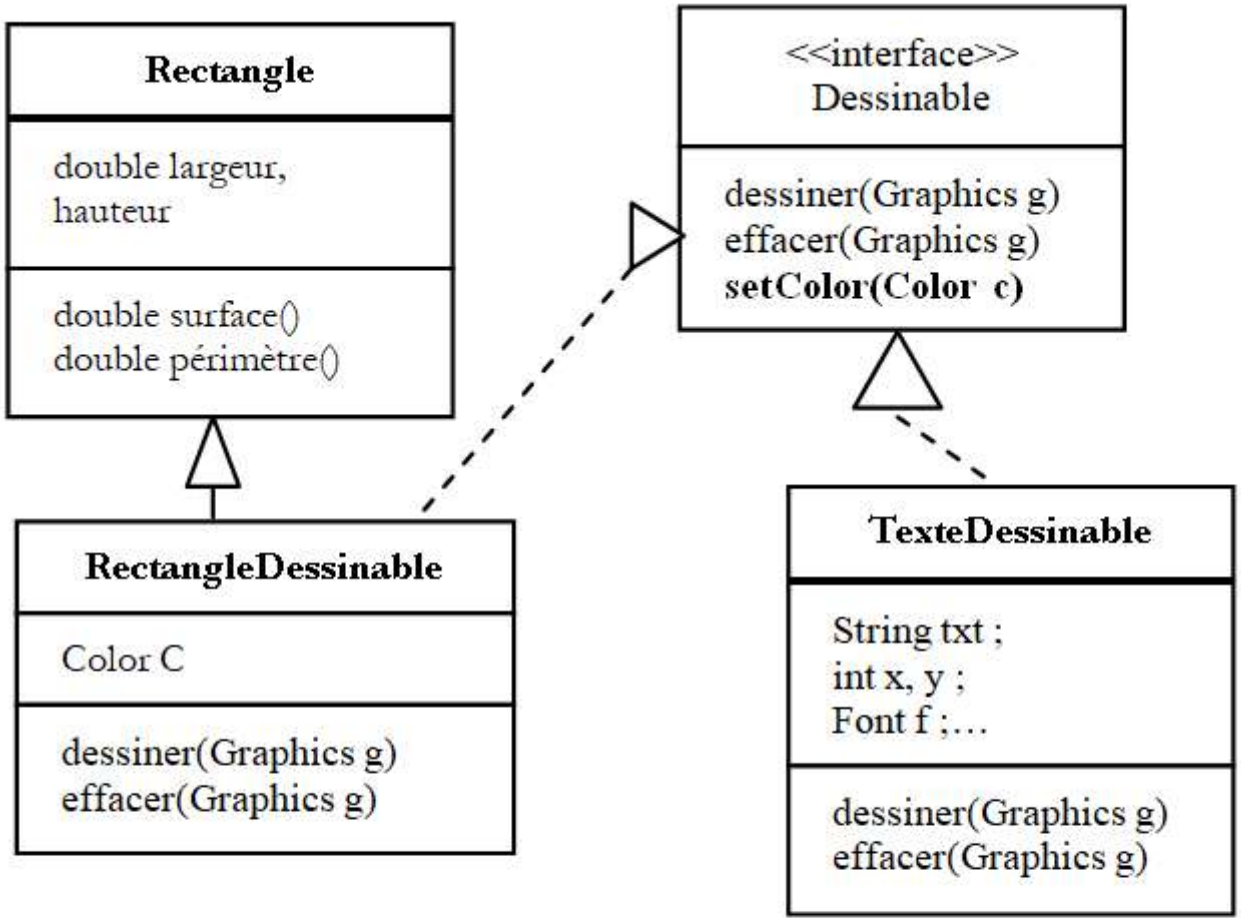

Rectangle
double largeur,
hauteur
double surface()
double périmètre()
<<interface>>
Dessinable
dessiner(Graphics g)
effacer(Graphics g)
setColor(Color c)
RectangleDessinable
Color C
dessiner(Graphics g)
effacer(Graphics g)
TexteDessinable
String txt ;
int x, y ;
Font f ;…
dessiner(Graphics g)
effacer(Graphics g)
Ces classes n'implémentent plus l'interface

**1ère Solution :** Modifier le code de toutes les classes implémentant l'interface (encore faut-il le pouvoir)



## 2ème solution : Définir une sous interface






**3ème solution : Définir une méthode par défaut**

- Java7-
  - *une méthode déclarée dans une interface ne fournit pas d'implémentation*
  - *Ce n'est qu'une signature, un contrat auquel chaque classe dérivée doit se conformer en fournissant une implémentation propre*

- Java 8 relaxe cette contrainte, possibilité de définir
  - *des méthodes statiques*
  - *des méthodes par défaut*
  - *des interface fonctionnelles*


```
public default void setColor(Color c){System.out.println("Default implementation of setColor()");  }
```

# CLASSE ABSTRAITE

**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

# INTERFACE

Une interface est une classe abstraite ayant les caractéristiques suivantes:

toutes les méthodes sont abstraites et public, alors qu'une classe abstraite peut avoir des méthodes non abstraites

toutes les variables sont **static** et constantes déclarées par le modificateur **final,** alors qu'une classe abstraite peut avoir des variables ordinaires

toute classe peut hériter d'**une seule classe** mais elle peut hériter de **plusieurs interfaces**

une interface peut hériter d'une ou plusieurs interfaces mais elle ne peut pas hériter d'une classe

il n'est pas nécessaire d'indiquer abstract pour les méthodes et static final pour les variables

une interface définie par le mot **interface**

si une classe hérite d'une interface: **implements**


```
class Petite {
public String n ;
public Petite(String x) { n = x ; }
public String toString() { return "Petite ! "; }
public void meth1() {
System.out.println("Petite.meth1 : " + this+ n );}
}
public class Grande extends Petite{
public int n ;
public Grande(String a, int b) { super(a) ; this.n = b ; }
public String toString() { return "Grande ! "; }
public void meth1() {
super.meth1() ;
System.out.println("Grande.meth1 : " + this + super.toString()+n+" "+
super.n);}
public void test() {  System.out.println("Grande.test : " + this);  }


public static void main(String args[]) {
Grande m = new Grande("POO", 2) ;
System.out.println(m) ;
m.meth1();
Petite n = new Petite("Java") ;
System.out.println(n) ;
n.meth1() ;
Petite v = new Grande("POO", 2) ;
v.meth1();
Grande maV=(Grande)v;
maV.test(); }
```

# EXERCICES

```
class A{
A()
{System.out.println("constructeur de A");}
}
class B extends A {
B()
{System.out.println("constructeur de B");}
B(int a)
{this();System.out.println("autre constructeur de B");}
}
class C extends B{
C()
{super(3);System.out.println("constructeur de C");}
}
public class Tst_const
{public static void main(String args[])
{ C c= new C();}
}
constructeur de A
constructeur de B
autre constructeur de B
constructeur de C
```