Héritage et abstraction
interfaces

Philippe Genoud

Interfaces

Exemple introductif

Bill Venners *Designing with Interfaces*

One Programmer's Struggle to Understand the Interfaces

http://www.javaworld.com/article/2076841/core-java/designing-with-interfaces.html

```
abstract class Animal {
      ...
      abstract void talk();
  }
```

```
class Dog extends Animal {
    ...
    void talk() {
        System.out.println("Woof!");
    }
}
```

```
class Bird extends Animal {
    ...
    void talk() {
        System.out.println("Tweet");
    }
}
```

```
class Cat extends Animal {
    ...
    void talk() {
        System.out.println("Meow");
    }
}
```

Polymorphisme signifie qu'une référence d'un type (classe) donné peut désigner un objet de n'importe quelle sous classe et selon la nature de cet objet produire un comportement différent

```
Animal animal = new Dog();
...
animal = new Cat();
```

animal **peut être** un **Chien**, un **Chat** ou n'importe quelle sous classe d'**Animal**

En JAVA le polymorphisme est rendu possible par la **liaison dynamique** (*dynamic binding*)

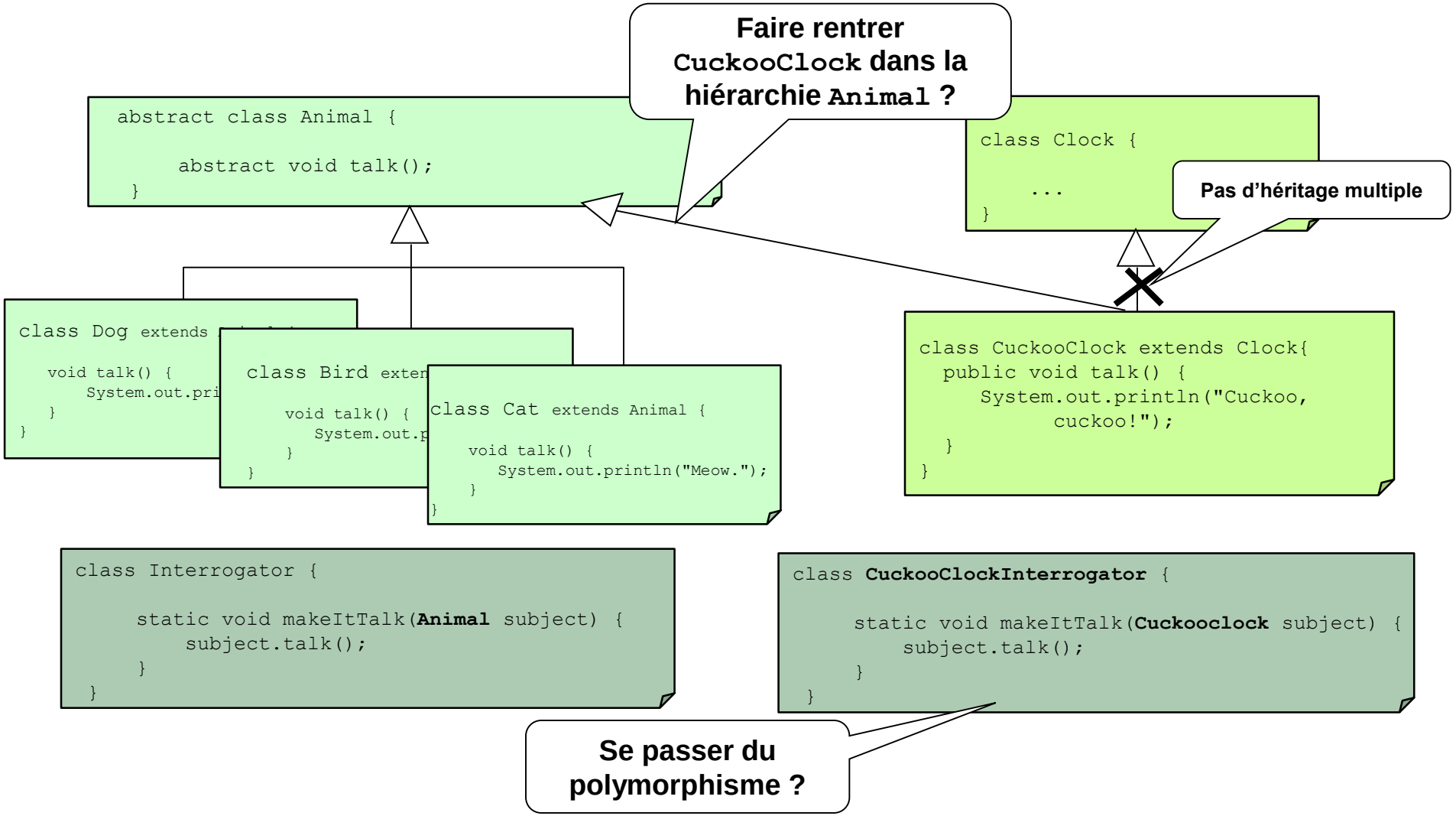
```
class Interrogator {

    static void makeItTalk(Animal subject) {
        subject.talk();
    }
}
```

JVM **décide à l'exécution** (*runtime*) quelle méthode invoquer en se basant sur la classe de l'objet

Interfaces

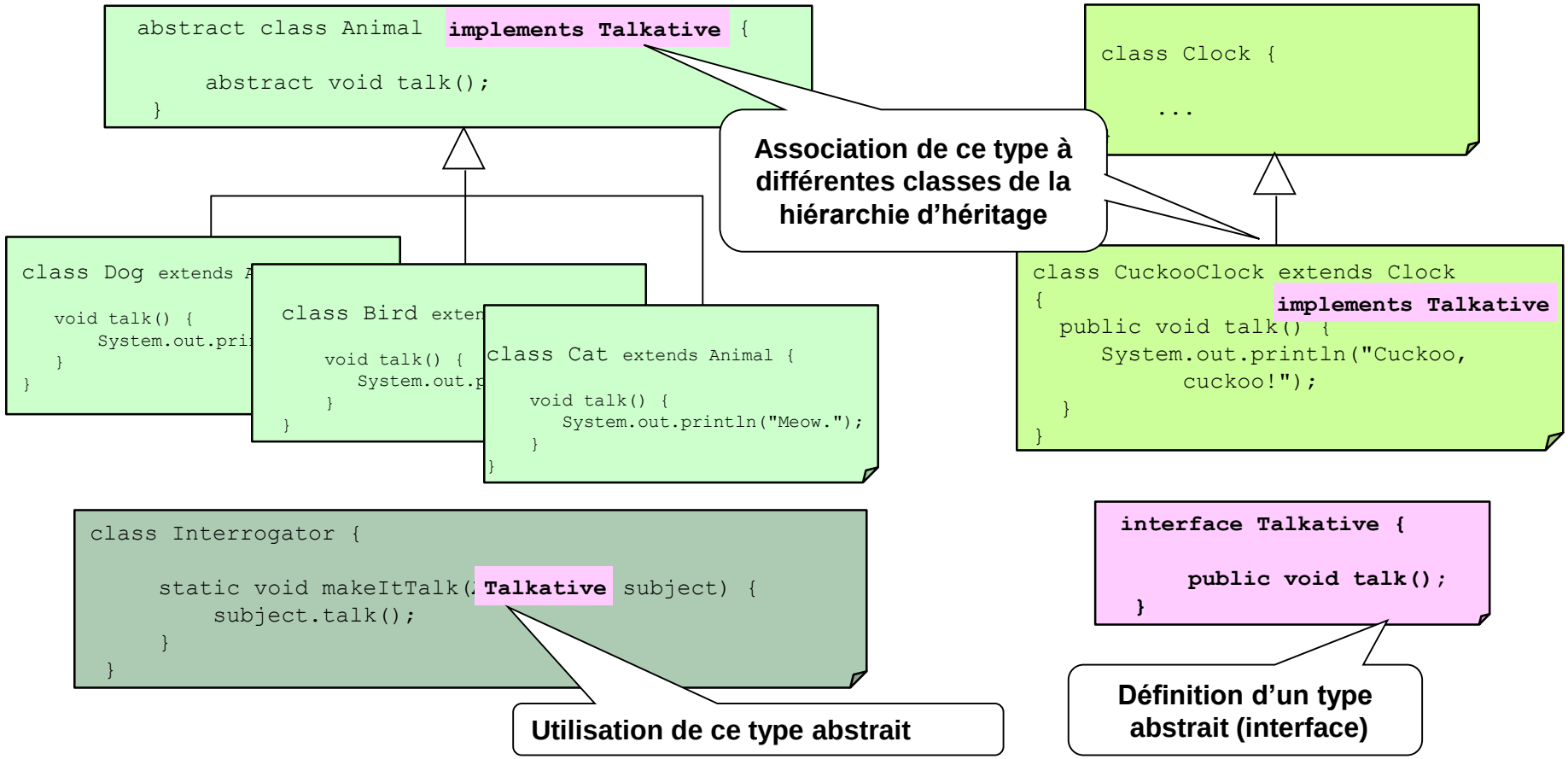
Exemple introductif

Comment utiliser **`Interrogator`** pour faire parler aussi un **`CuckooClock`** ?



Interfaces

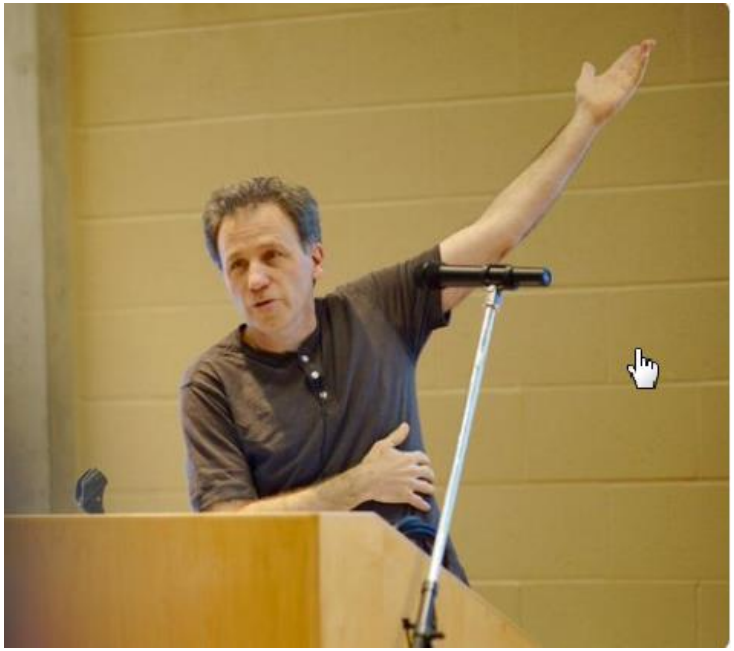
Exemple introductif



- Les interfaces permettent **plus de polymorphisme** car avec les interfaces il n'est pas nécessaire de tout faire rentrer dans une seule famille (hiérarchie) de classes

Interfaces

- Java's interface gives you more polymorphism than you can get with singly inherited families of classes, without the "burden" of multiple inheritance of implementation.



Bill Venners *Designing with Interfaces –*
One Programmer's Struggle to Understand the Interface
http://www.atrima.com/designtechniques/index.html

Interfaces

déclaration d'une interface

- Une *interface* est une collection d'opérations utilisée pour spécifier un service offert par une classe.
- Une interface peut être vue comme une classe 100% abstraite sans attributs et dont toutes les opérations sont abstraites.

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);
}
```

Possibilité d'implémentation par défaut avec Java 8

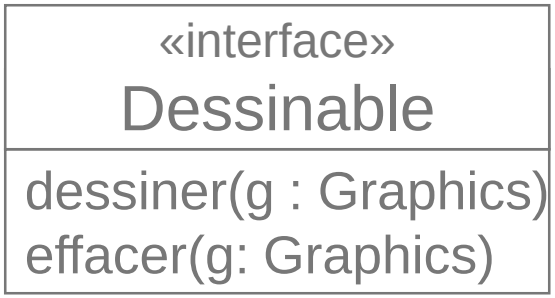
Toutes les méthodes sont abstraites

Elles sont implicitement publiques

Dessinable.java

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

opérations abstraites



interface

Interfaces

déclaration d'une interface

- 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);
 }
```

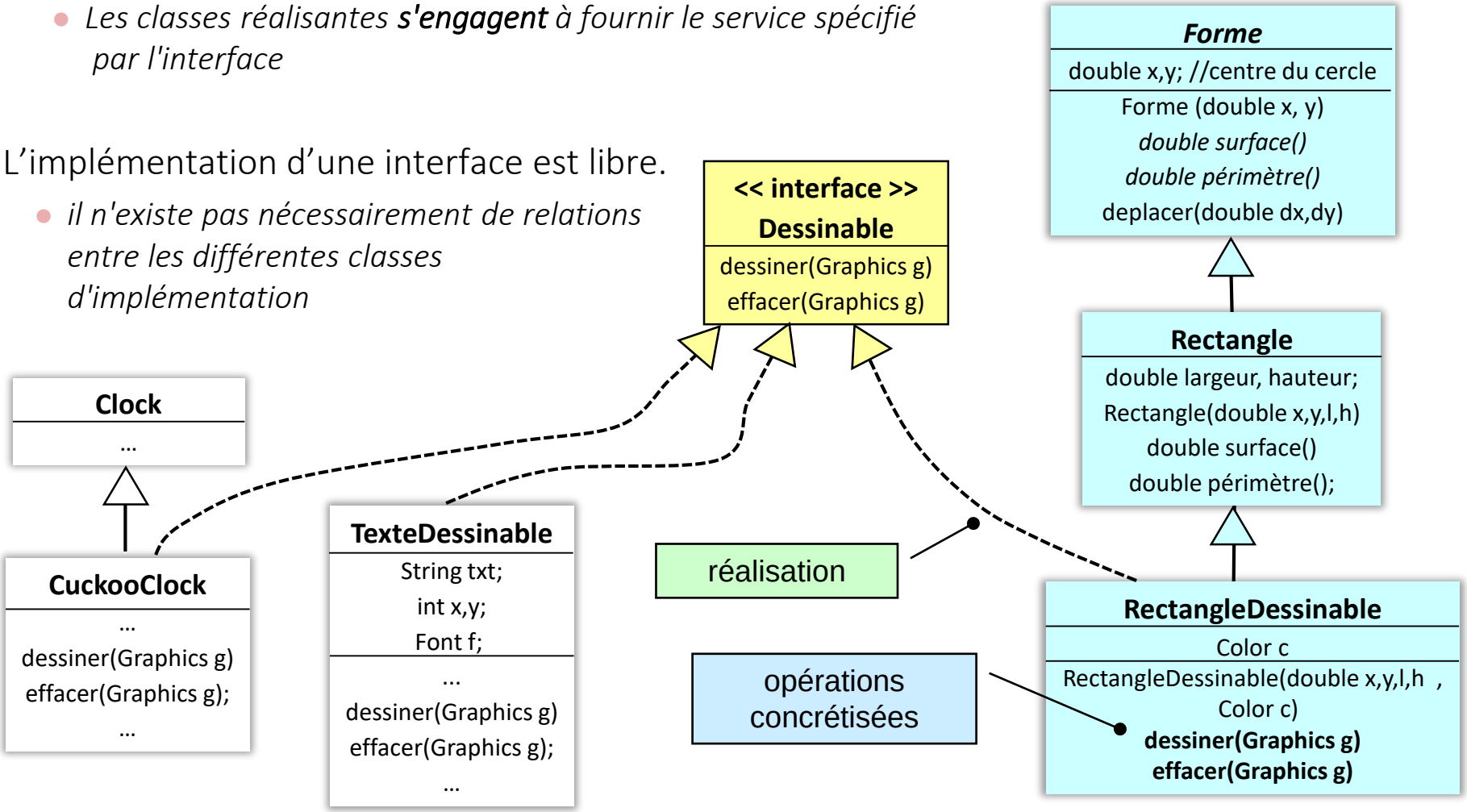
Dessinable.java

Interfaces

"réalisation" d'une interface

- 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.
  - *il n'existe pas nécessairement de relations entre les différentes classes d'implémentation*

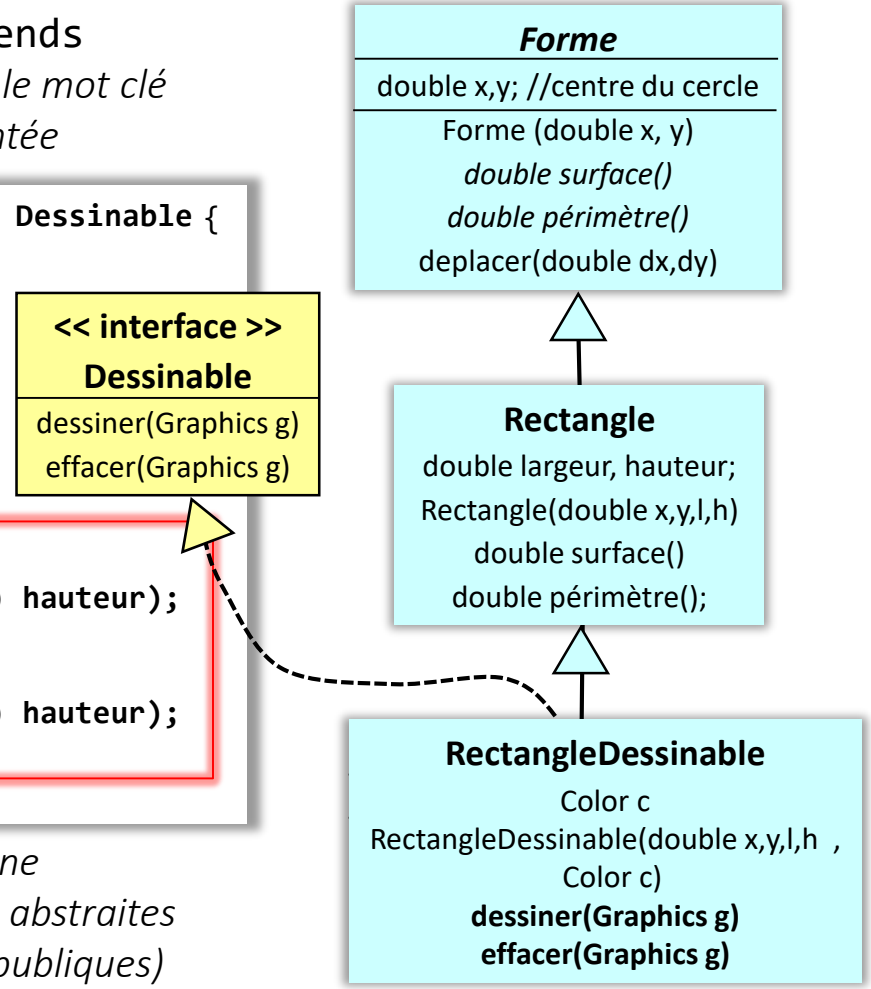


Interfaces

"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);
    }
}
```



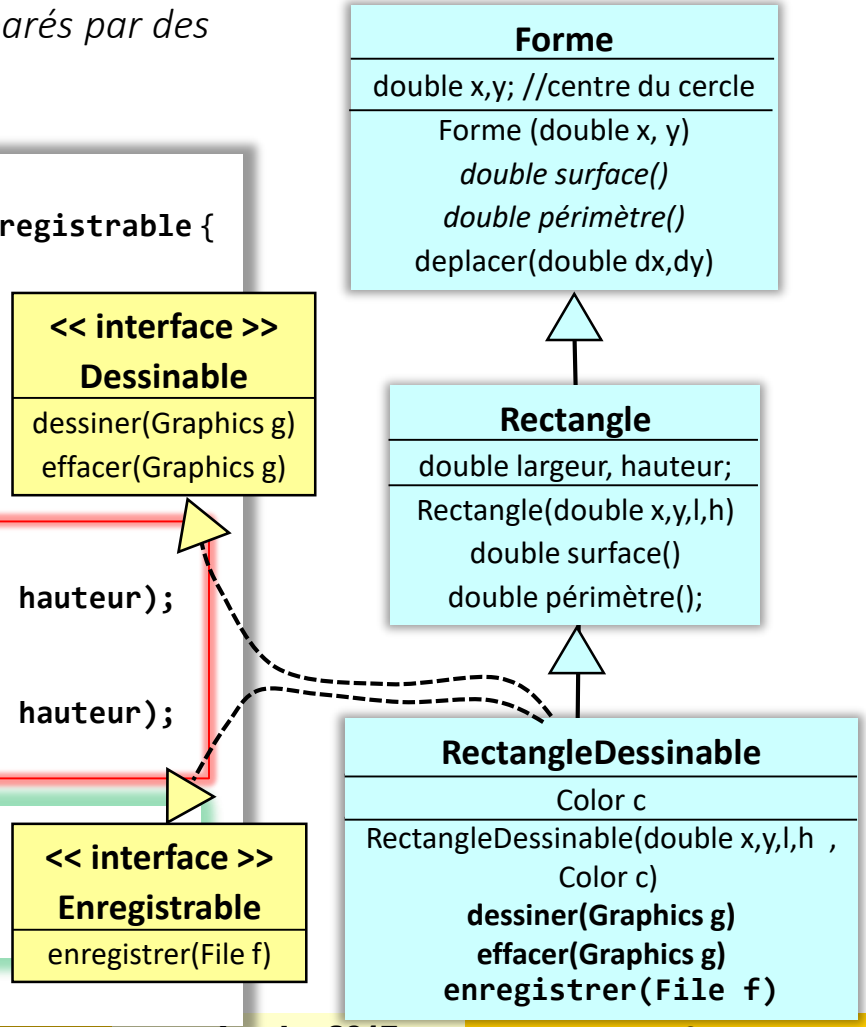
  - *si la classe est une classe concrète* ***elle doit*** *fournir une implémentation (un corps) à chacune des méthodes abstraites définies dans l'interface (qui doivent être déclarées publiques)*

Interfaces

"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) {
       ...
    }
}
```

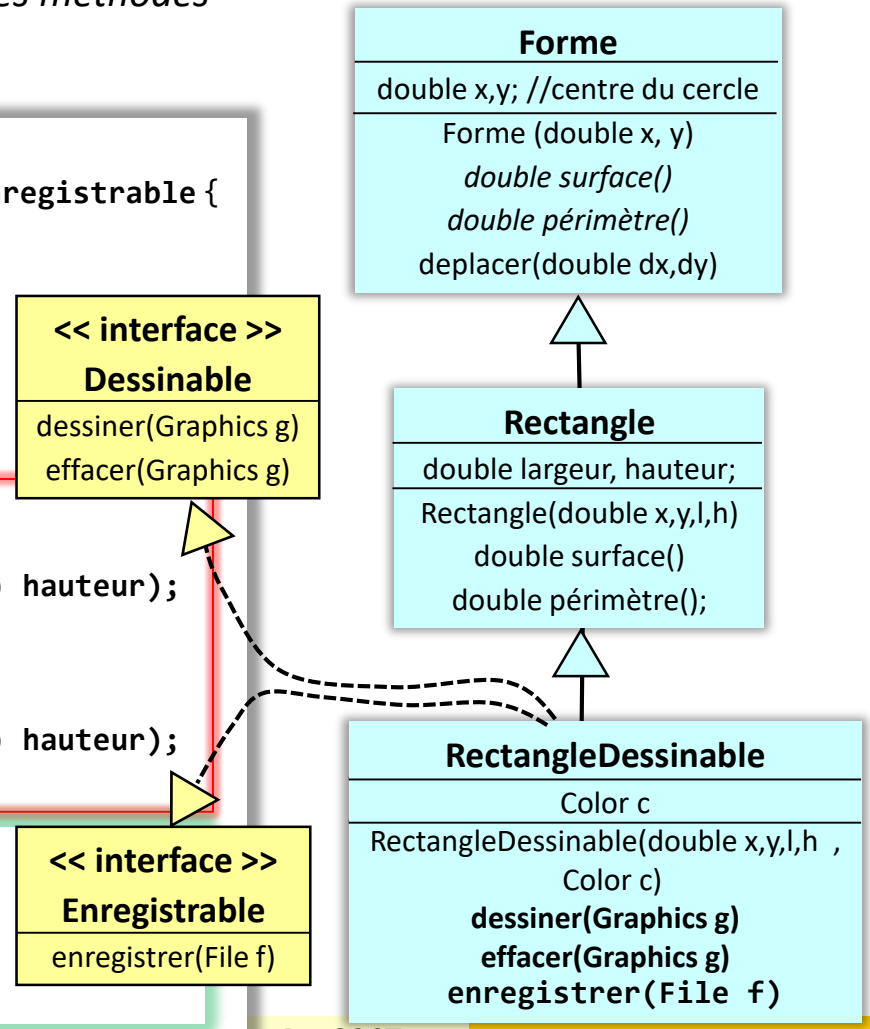


Interfaces

"réalisation" d'une interface

- *pour éviter des redéfinitions de méthodes penser à mettre des directives `@Override` lorsque implémentation des méthodes d'une interface*

```
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;
    }
    @Override
    public void dessiner(Graphics g){
      g.drawRect((int) x, (int) y, (int) largeur, (int) hauteur);
    }
    @Override
    public void effacer(Graphics g){
      g.clearRect((int) x, (int) y, (int)largeur, (int) hauteur);
    }

    @Override
    public void enregistrer(File f) {
       ...
    }

}
```



Interfaces

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

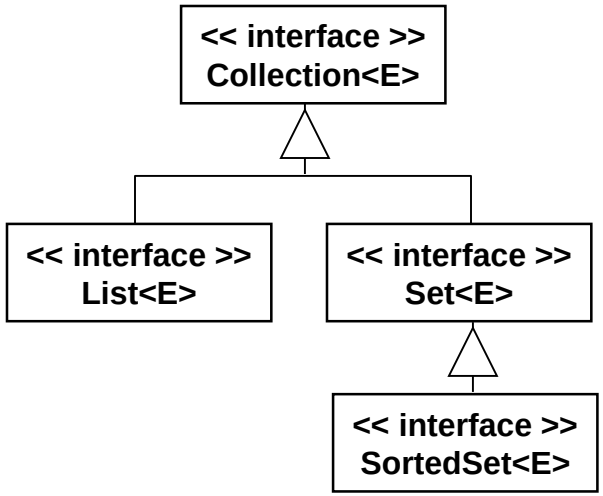
règles du polymorphisme s'appliquent de la même manière que pour les classes :

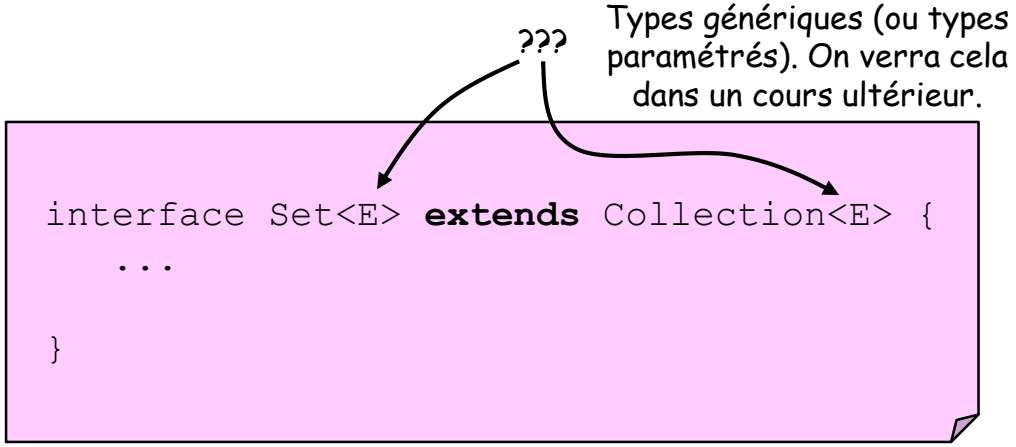
- vérification statique du code
- liaison dynamique

Interfaces

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*





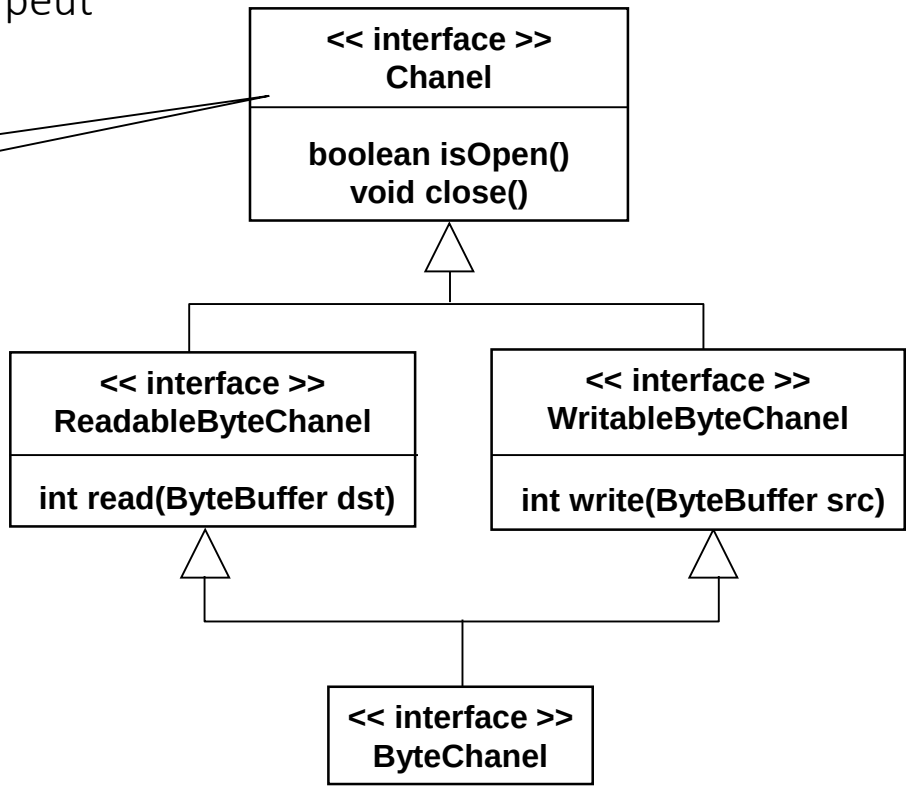
- Une classe qui implémente une interface doit implémenter toutes les méthodes abstraites définies dans l'interface et dans les interfaces dont elle hérite.

Interfaces

héritage d'interfaces

- A la différence des classes une interface peut étendre plus d'une interface à la fois

représente une connexion ouverte vers une entité telle qu'un dipositif hardware, un fichier, une "socket" réseau, ou tout composant logiciel capable de réaliser une ou plusieurs opérations d'entrée/sortie.




```
package java.nio;
interface ByteChannel extends ReadableByteChanel, WriteableByteChanel {
}
```

Interfaces

interêt

- Les interfaces permettent de s ’affranchir d ’éventuelles contraintes d’héritage.
  - *Lorsqu’on examine une classe implémentant une ou plusieurs interfaces, on est sûr que le code d’implémentation est dans le corps de la classe. Excellente localisation du code (défaut de l’héritage multiple, sauf si on hérite de classes purement abstraites).*

- Permet une **grande évolutivité** du modèle objet

Interfaces

interêt

« Smarter Java development » *Michael Cymerman* , javaworld août 99.

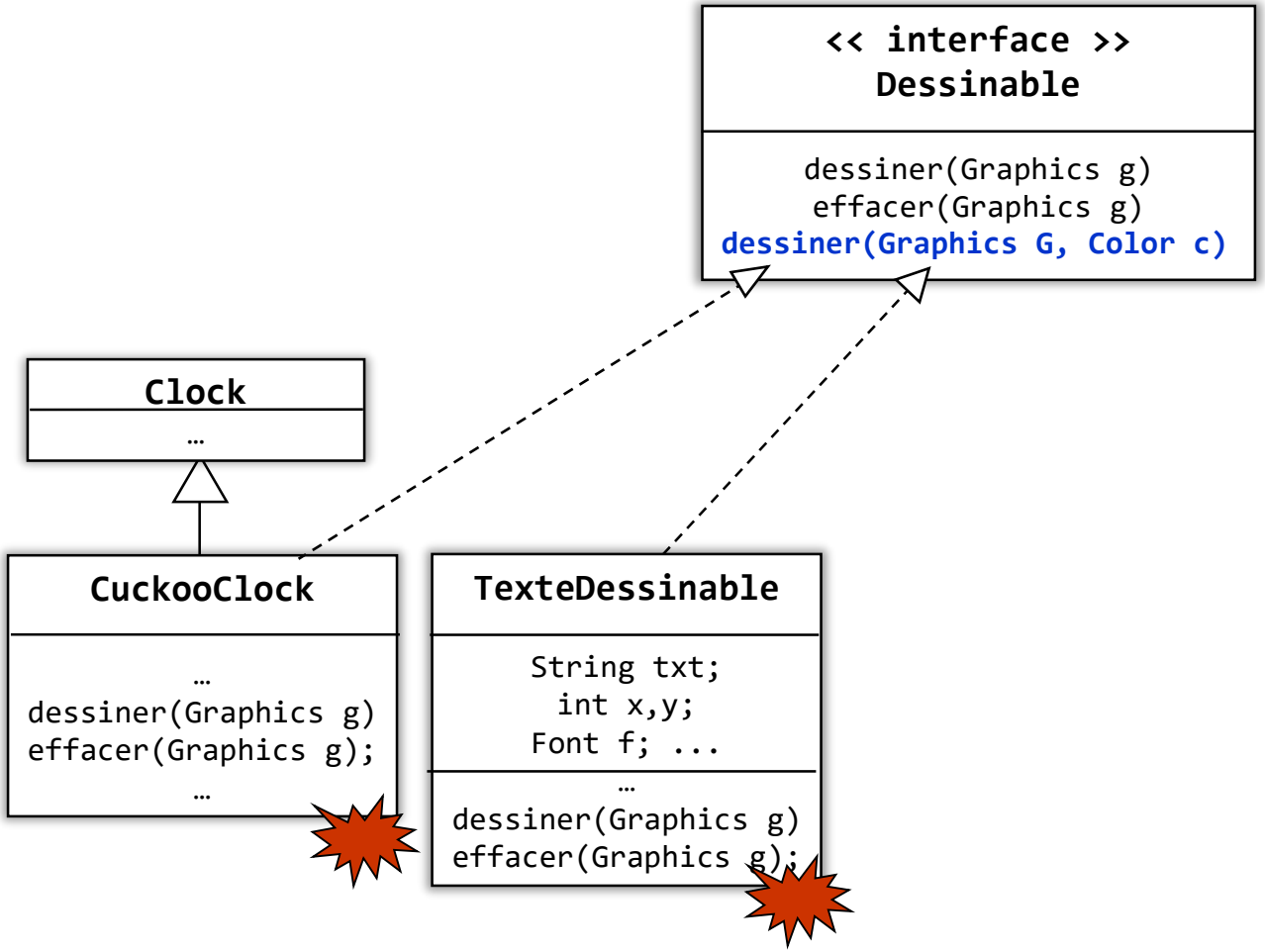
http://www.javaworld.com

- By incorporating interfaces into your next project, you will notice benefits throughout the lifecycle of your development effort. The technique of coding to interfaces rather than objects will improve the efficiency of the development team by:

  - *Allowing the development team to quickly establish the interactions among the necessary objects, without forcing the early definition of the supporting objects*
  - *Enabling developers to concentrate on their development tasks with the knowledge that integration has already been taken into account*
  - *Providing flexibility so that new implementations of the interfaces can be added into the existing system without major code modification*
  - *Enforcing the contracts agreed upon by members of the development team to ensure that all objects are interacting as designed*

Evolution des interfaces

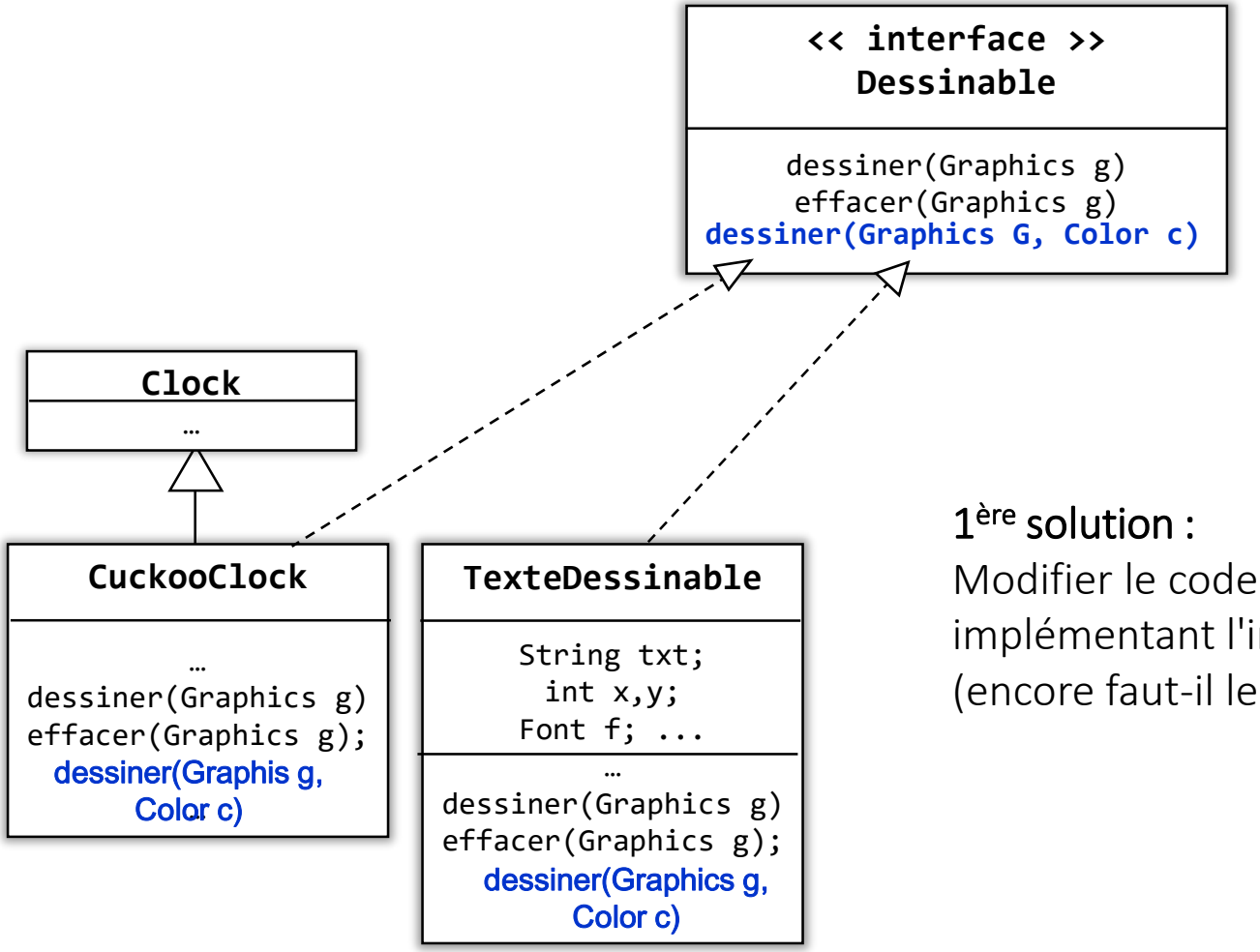
- Quand on définit des interfaces il faut être prudent : tout ajout ultérieur "brise" le code des classes qui implémentent l'interface.



Ces classes n'implémentent plus l'interface

Evolution des interfaces

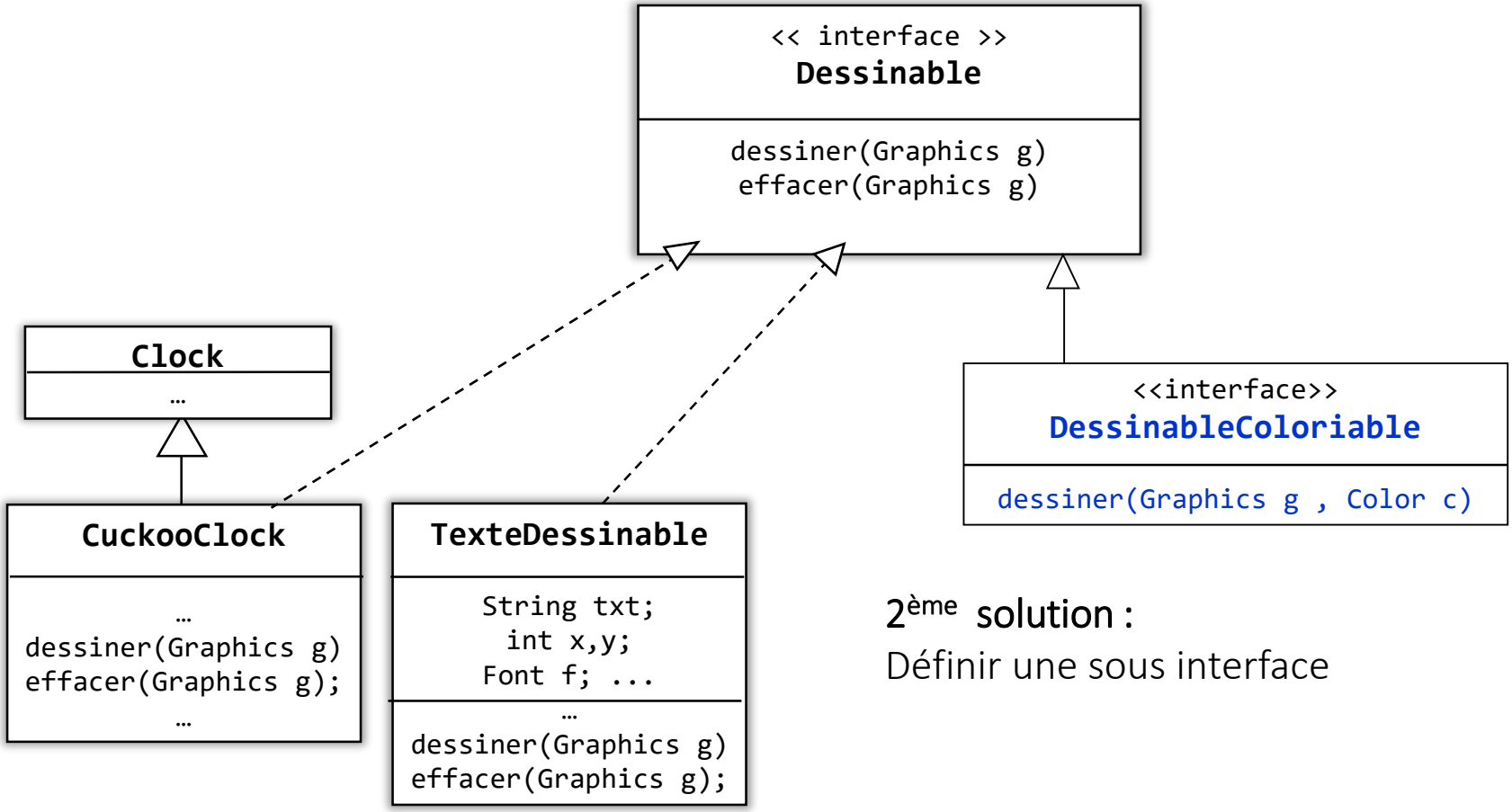
- Quand on définit des interfaces il faut être prudent : tout ajout ultérieur "brise" le code des classes qui implémentent l'interface.



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

Evolution des interfaces

- Quand on définit des interfaces il faut être prudent : tout ajout ultérieur "brise" le code des classes qui implémentent l'interface.

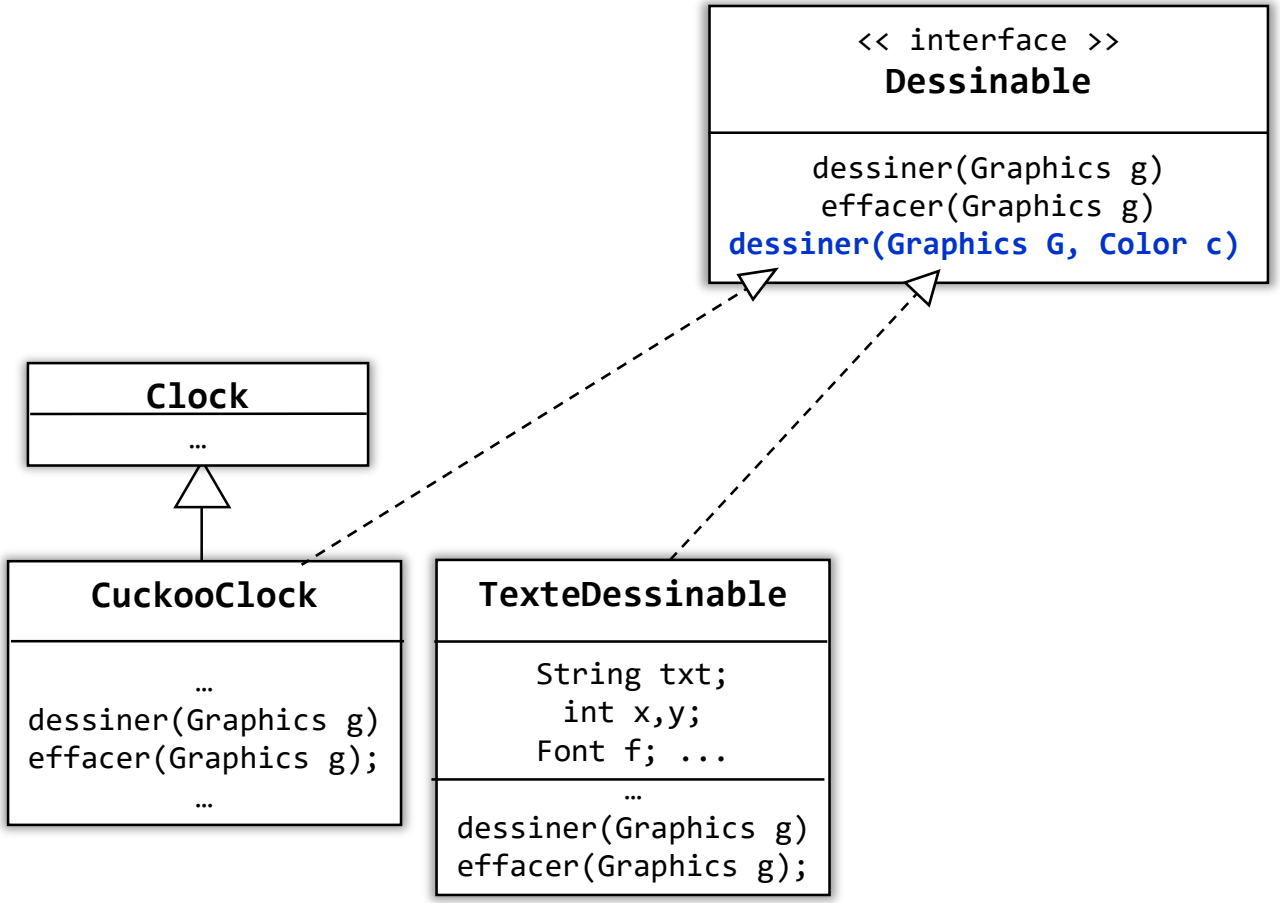


2ème solution :

Définir une sous interface

Evolution des interfaces

- Quand on définit des interfaces il faut être prudent : tout ajout ultérieur "brise" le code des classes qui implémentent l'interface.



3ème solution : Définir une méthode par défaut

Java 8: quoi de neuf dans les interfaces ? *

- 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*

* titre inspiré du titre de l'article ***Java 8 : du neuf dans les interfaces !*** du blog d'Olivier Croisier
http://thecodersbreakfast.net/index.php?post/2014/01/20/Java8-du-neuf-dans-les-interfaces

Interfaces Java 8 : méthodes par défaut

- déclaration d'une méthode par défaut
  - *fournir un corps à la méthode*
  - *qualifier la méthode avec le mot clé* **`default`**

```
public interface Foo {
   public default void foo() {
     System.out.println("Default implementation of foo()");
   }
}
```

- les classes filles sont libérées de fournir une implémentation d'une méthode **`default`**, en cas d'absence d'implémentation spécifique c'est la méthode par défaut qui est invoquée

```
public interface Itf {

    /** Pas d'implémentation - comme en Java 7
        et antérieur */
    public void foo();

    public default void bar() {
        System.out.println("Itf -> bar() [default]");
    }

    public default void baz() {
        System.out.println("Itf -> baz() [default]");
    }
}
```

```
public class Cls implements Itf {

    @Override
    public void foo() {
        System.out.println("Cls -> foo()");
    }

    @Override
    public void bar() {
        System.out.println("Cls -> bar()");
    }
}
```

```
Cls cls = new Cls();
cls.foo();  → Cls -> foo()
cls.bar();  → Cls -> bar()
cls.baz();  → Itf -> baz() [default]
```

Interfaces Java 8 : méthodes par défaut

- mais qu'en est-il de l'héritage en diamant ?

```
public interface InterfaceA {
    public default void foo() {
        System.out.println("A -> foo()");
    }
}
```

```
public interface InterfaceB {
    public default void foo() {
        System.out.println("B -> foo()");
    }
}
```

```
public class Cls implements InterfaceA, InterfaceB {
}
```

Erreur de compilation
"class Test inherits unrelated defaults for foo() from types InterfaceA and InterfaceB"

Pour résoudre le conflit, une seule solution : implémenter la méthode au niveau de la classe elle-même, car l'implémentation de la classe est toujours prioritaire.

```
Cls cls = new Cls();
cls.foo();
```

```
public class Cls implements InterfaceA, InterfaceB {
 public void foo() {
        System.out.println("Test -> foo()");
    }
}
```

Interfaces Java 8 : méthodes par défaut

- mais qu'en est-il de l'héritage en diamant ?


```
public interface InterfaceA {
    public default void foo() {
        System.out.println("A -> foo()");
    }
}
```

```
public interface InterfaceB {
    public default void foo() {
        System.out.println("B -> foo()");
    }
}
```

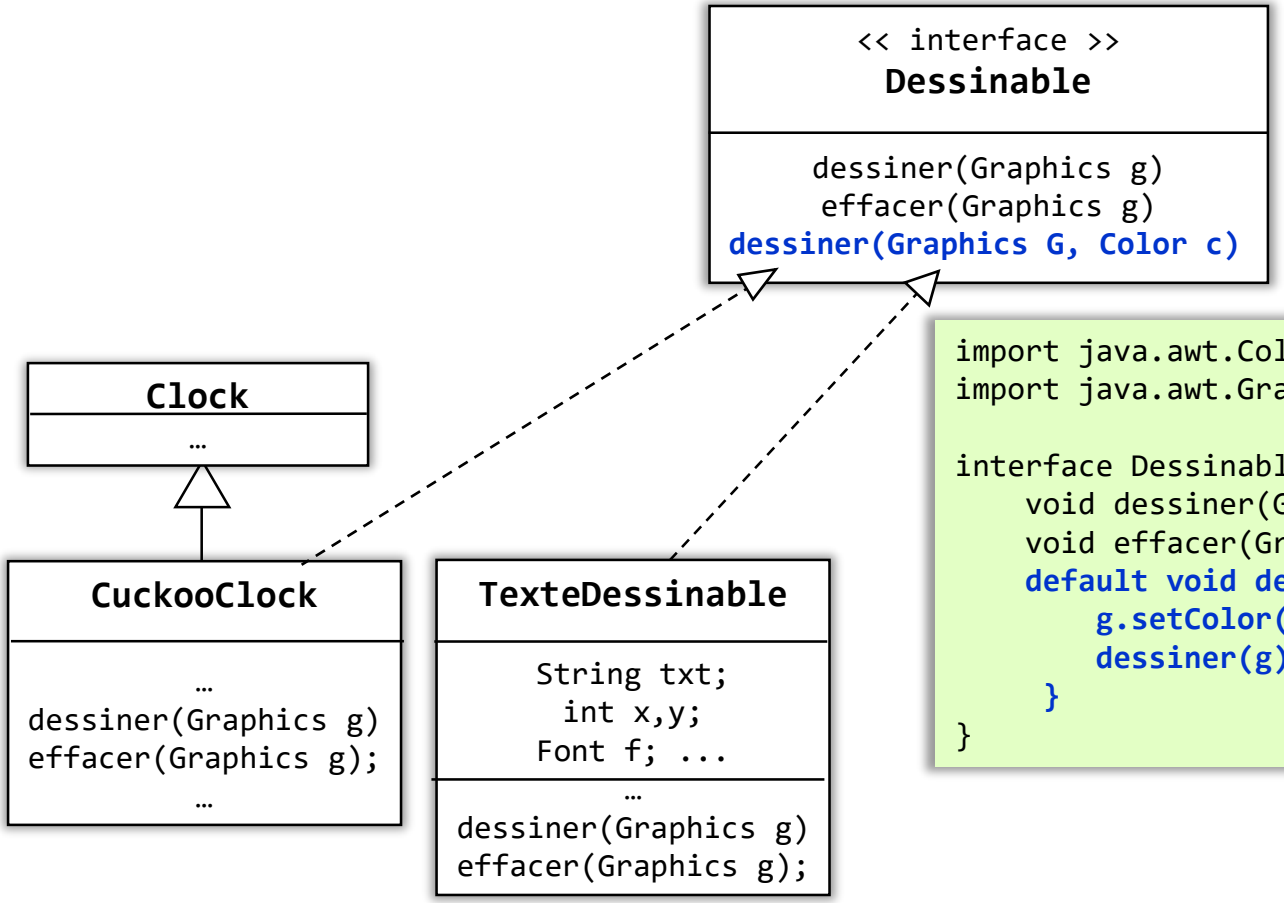
```
public class Cls implements InterfaceA, InterfaceB {
 public void foo() {
        InterfaceB.super.foo();
    }
}
```

Possibilité d'accéder sélectivement aux implémentations par défaut :
nomInterface.**super**.*méthode*

```
Cls cls = new Cls();
cls.foo();
```

Evolution des interfaces

- Quand on définit des interfaces il faut être prudent : tout ajout ultérieur "brise" le code des classes qui implémentent l'interface.



3ème solution : Définir une méthode par défaut

```
import java.awt.Color;
import java.awt.Graphics;

interface Dessinable {
    void dessiner(Graphics g);
    void effacer(Graphics g);
    default void dessiner(Graphics g, color c) {
        g.setColor();
        dessiner(g);
     }
}
```