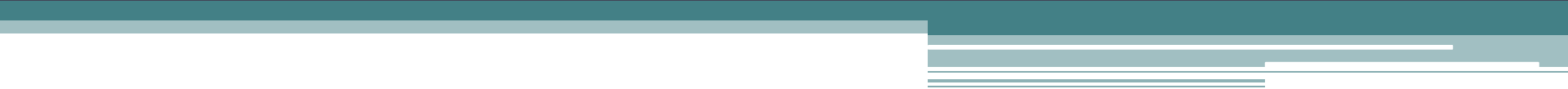


Chapitre 4:

Classes Abstraites, Interfaces

Classes imbriquées

A decorative graphic consisting of several horizontal lines of varying lengths and colors (teal, light blue, and white) extending from the left edge of the slide towards the right, positioned below the chapter title.

Classes Abstraites

Intérêts

- **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

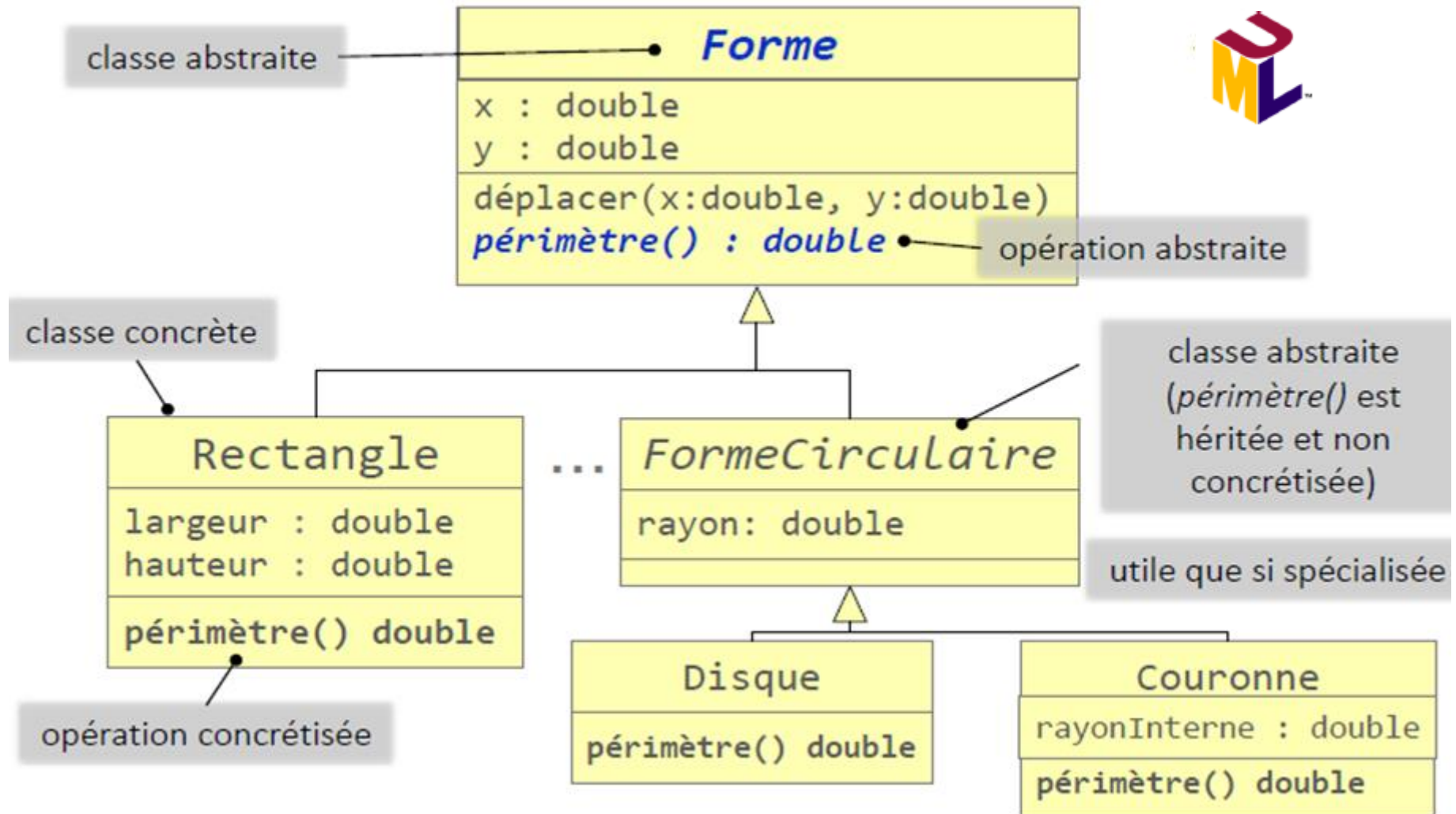
Classes Abstraites

Intérêts

- Une classe abstraite est une description d'objets 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.
 - elle doit implémenter toutes les méthodes abstraites
- 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 souvent l'utilisation de classes abstraites.

Classes Abstraites

UML



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);  
}
```

Dessinable.java

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



opérations abstraites

«interface»
Dessinable

dessiner(g : Graphics)
effacer(g: Graphics)

interface



Possibilité
d'implémenta-
tion par
défaut avec



Toutes les méthodes
sont abstraites

Elles sont implicitement
publiques

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

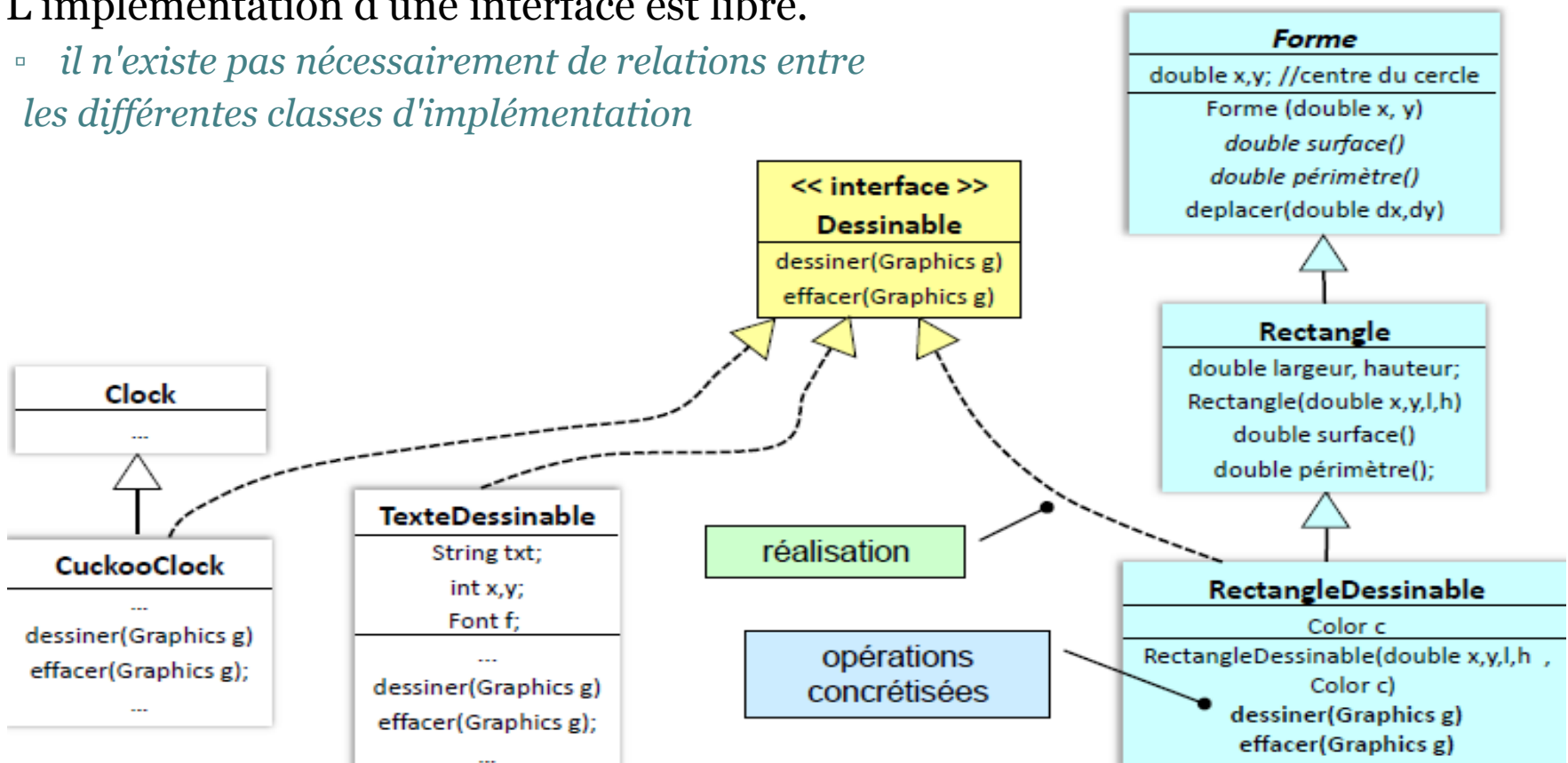
```
import java.awt.Graphics;  
public interface Dessinable {  
    public static final int MAX_WIDTH = 1024;  
    int MAX_HEIGHT = 768;  
  
    public void dessiner(Graphics g);  
    void effacer(Graphics g);  
}
```

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);  
    }  
}
```

<< interface >>

Dessinable

dessiner(Graphics g)

effacer(Graphics g)

Forme

double x,y; //centre du cercle

Forme (double x, y)

double surface()

double périmètre()

deplacer(double dx,dy)

Rectangle

double largeur, hauteur;

Rectangle(double x,y,l,h)

double surface()

double périmètre();

RectangleDessinable

Color c

RectangleDessinable(double x,y,l,h ,
Color c)

dessiner(Graphics g)

effacer(Graphics g)

- 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

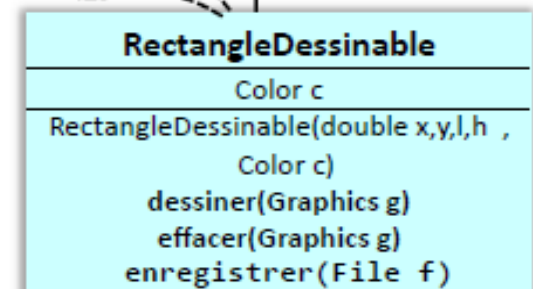
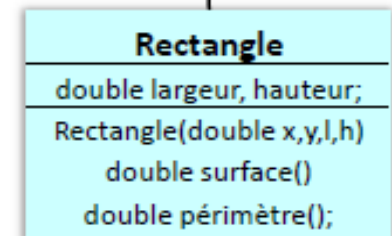
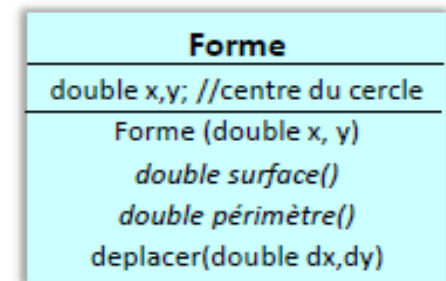
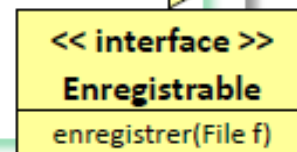
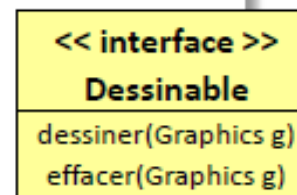
- Une classe JAVA peut implémenter simultanément plusieurs interfaces
- la liste des noms des interfaces à implémenter séparés par des virgules doit suivre le mot clé `implements`

```
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

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.*
- règles du polymorphisme s'appliquent de la même manière que pour les classes :
 - vérification statique du code
 - liaison dynamique

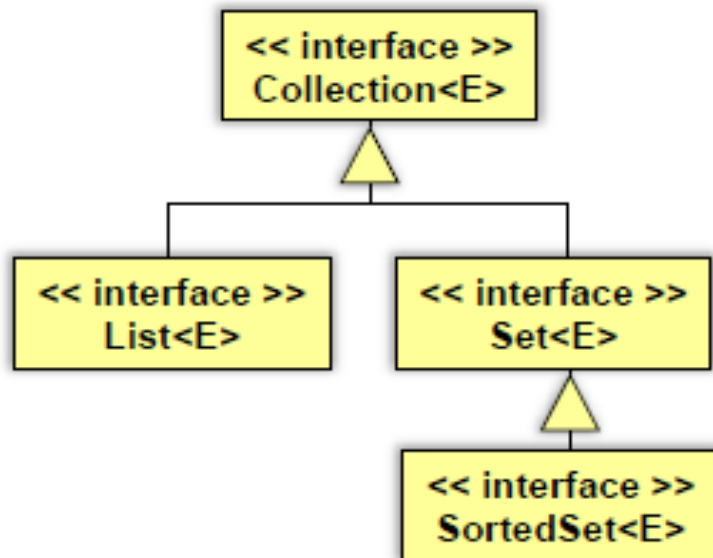
```
public class ZoneDeDessin {  
    private nbFigures;  
    private Dessinable[] figures;  
    ...  
    public void ajouter(Dessinable d){  
        ...  
    }  
    public void supprimer(Dessinable o){  
        ...  
    }  
  
    public void dessiner() {  
        for (int i = 0; i < nbFigures; i++)  
            figures[i].dessiner(g);  
    }  
}
```

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

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.

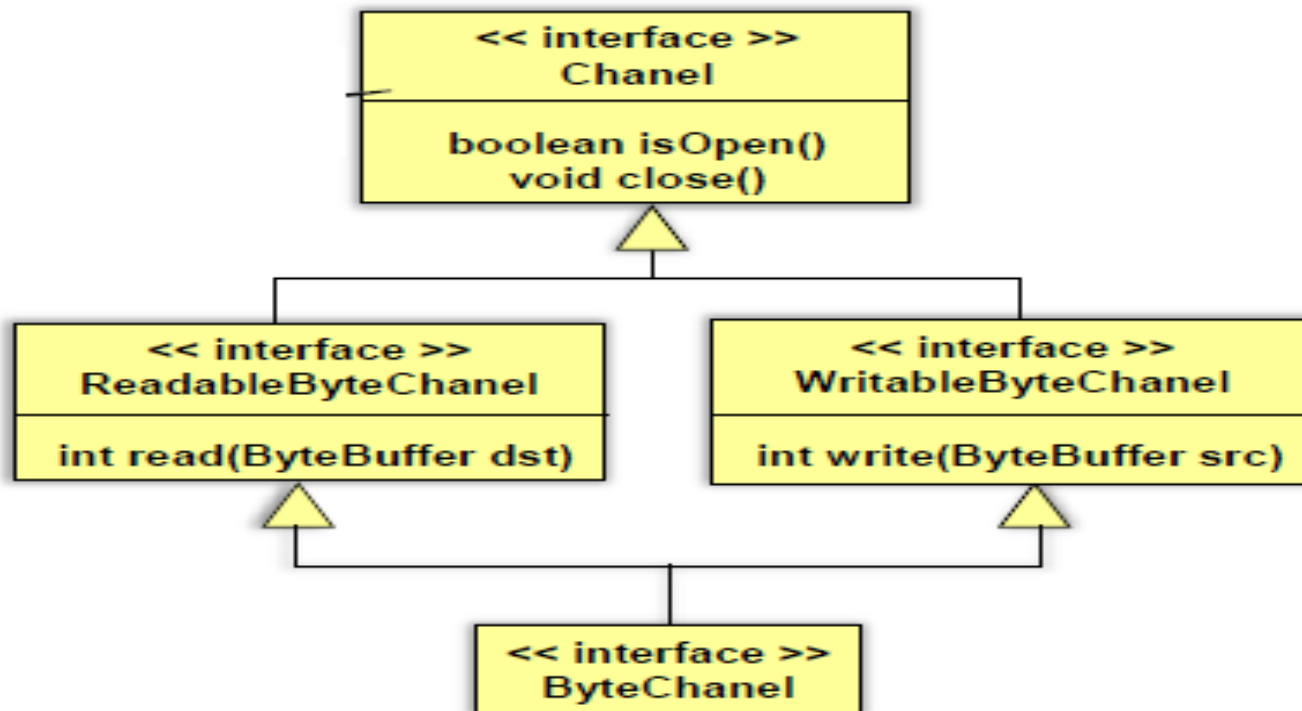


```
interface Set<E> extends Collection<E> {  
    ...  
}
```

Interfaces

héritage d'interface

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



```
package java.nio;  
interface ByteChannel extends ReadableByteChannel, WritableByteChannel {  
}
```

Interfaces

Intérêt

- 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

Choix entre classe et interface : principe

Une interface peut servir à faire du polymorphisme comme l'héritage, alors comment choisir entre classe et interface ?

1. **Choix dicté par l'existant** : L'héritage n'est plus possible, la classe hérite déjà d'une autre classe. Il ne reste plus que celui l'interface.
2. **Choix à la conception** : On étudie la relation entre A et B ?
 - ▶ Un objet de classe B **"EST UN"** A
⇒ Héritage : B **extends** A.
 - ▶ Un objet de classe B **"EST CAPABLE DE FAIRE"** A
⇒ Interface : B **implements** A.

Interfaces

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

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

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

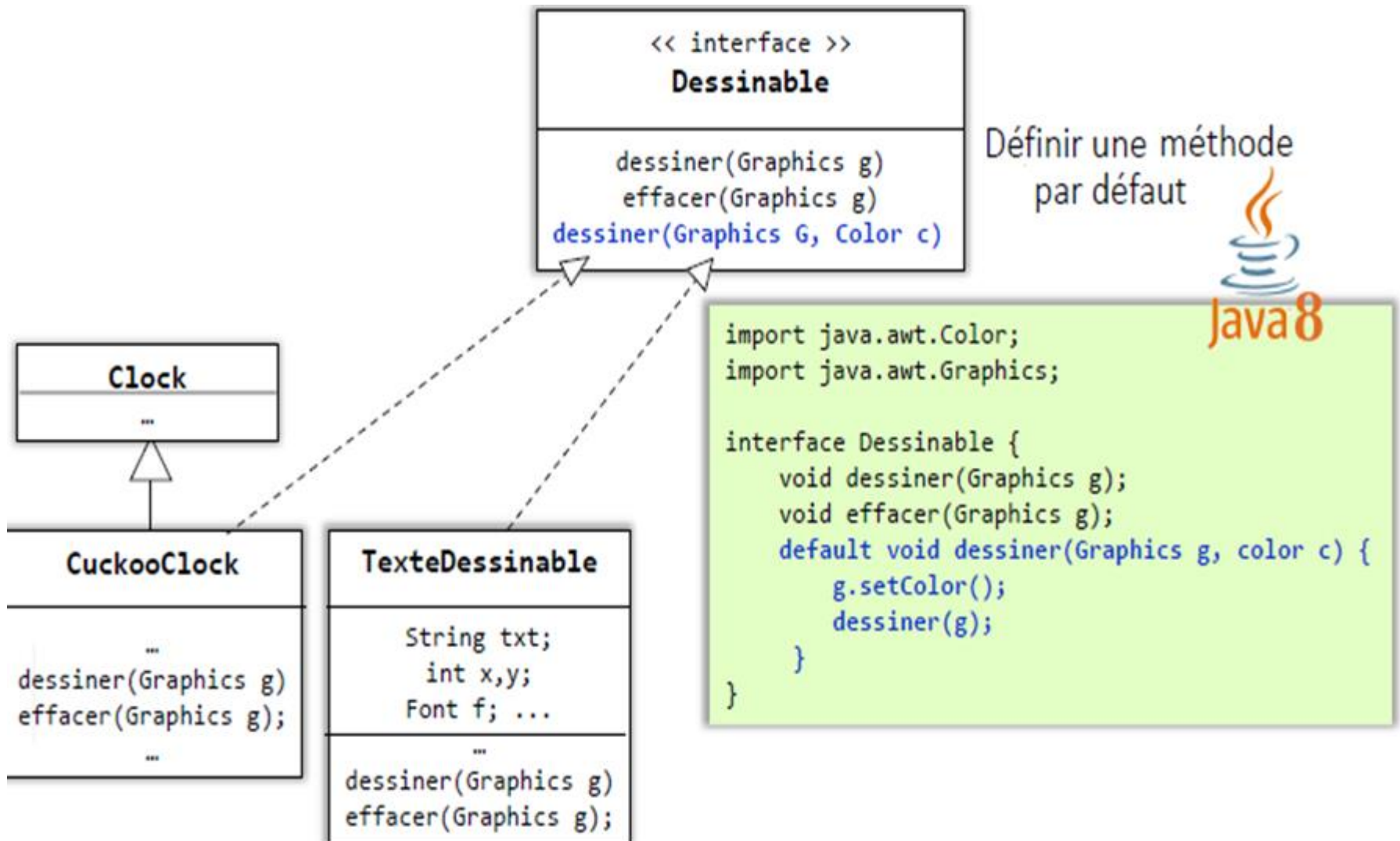
```
public class Cls implements InterfaceA, InterfaceB {  
    public void foo() {  
        InterfaceB.super.foo();  
    }  
}
```

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

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

Interfaces

Java 8: méthodes par défaut



Classes imbriquées

Types de classes imbriquées

- Classes internes statiques

```
public class A {  
    public static class B {  
    }  
}
```

Classe interne statique de classe

- Classes internes non statiques (= classes *internes*)

- Standards

- Locales

- Locales

- Anonymes

```
public class A {  
    public class B {  
    }  
}
```

Inner-class de classe

```
public class A {  
    public void m() {  
        class B {  
        }  
    }  
}
```

Classe interne de méthode

```
public class A {  
    public void m() {  
        new Object() {  
            ...  
        };  
    }  
}
```

Classe anonyme de méthode

Classes imbriquées

Classes imbriquées statiques

- Déclaration
 - Dans une classe

```
Visibilité static class NomClasse {  
    ...  
}
```
- Visibilité : public, protégée, package, privée
 - Utilisation : `ClasseEnglobante.ClasseImbriquée`
 - Particularité :
 - accès aux variables de classe de la classe englobantes, mêmes privées.
 - n'a pas besoin d'un objet de la classe englobante pour exister.
- Intérêts
 - structuration plus fine des classes
- La machine virtuelle ne fait pas la différence entre une classe classique et une classe interne

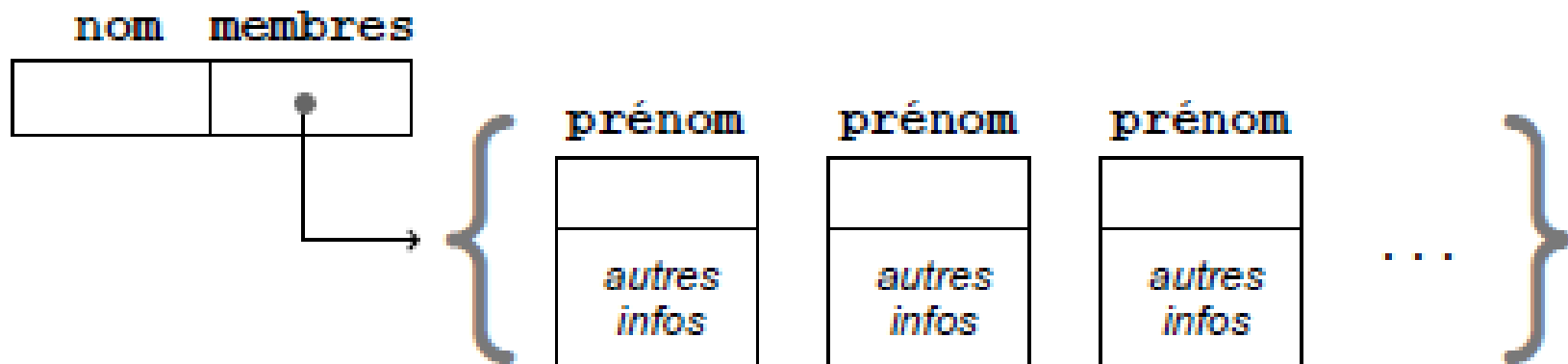
Classes imbriquées

Classes internes

- Déclaration
 - Dans une classe
 - *Visibilité* class NomClasse { ... }
- Visibilité : public, protégée, package, privée
- Utilisation
 - Création : InstanceClasseEnglobante.new ClasseInterne(...)
 - Type : ClasseEnglobante.ClasseInterne
- Particularité
 - accès aux variables d'instance de la classe englobantes, mêmes privées
 - Accès à l'objet englobant associé : ClasseEnglobante.this
- Intérêts
 - structuration plus fine des classes
 - Implantation de la notion de "classe amie"

Classes imbriquées

Classes internes



Classes imbriquées

Classes internes locales

- Déclaration
 - Dans un bloc de code (constructeur, méthode, ...)
- Visibilité : hors de propos
- Utilisation
 - Type visible uniquement dans le bloc de code englobant
 - Si instance retournée par la méthode, utiliser un super-type (superclasse, interface implantée) comme type de retour
- Particularité
 - accès aux variables d'instance de la classe englobantes, mêmes privées
 - Accès à l'objet englobant associé : `ClasseEnglobante.this`
 - Accès aux paramètres et variables locales du bloc englobant s'ils sont "final"
- Intérêt
 - Retourner des instances de classes différentes suivant les cas
 - Retourner des instances paramétrées par l'appel à la méthode englobante

Classes imbriquées

Classes internes anonymes

- **Déclaration**
 - Après un new, dans une expression requérant une instance de classe
 - New SuperType(...) {...} où SuperType peut être une classe ou une interface.
 - Dans le cas où l'on souhaite passer par un constructeur du super-type autre que le constructeur sans argument, mettre les arguments requis entre les parenthèses
- **Visibilité : hors de propos**
- **Utilisation**
 - Lors de la création, là où la classe est déclarée
- **Particularité**
 - Sans utiliser la réflexivité, une seule instance possible
 - Implantation d'une interface au maximum.
 - Implantation de zéro interface si super-classe != Object
- **Intérêt**
 - Alléger le code en évitant la multiplication de classes "visibles"