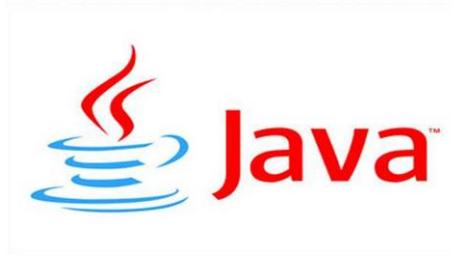


Programmation Orientée Objet

Application avec Java



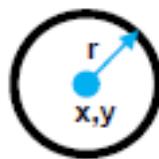
Chapitre 5

Classe Abstraite et Interface

Classe abstraite

Exemple introductif

- 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

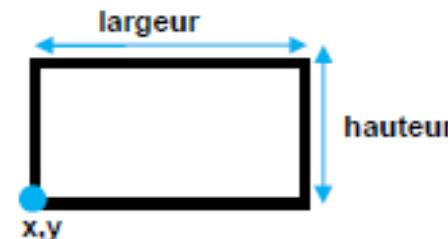


Attributs :

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

Méthodes :

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

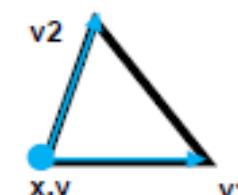


Attributs :

```
double x,y; //coin inférieur gauche
double largeur, hauteur;
```

Méthodes :

```
deplacer(double dx, double dy)
double surface()
double périmètre();
```



Attributs :

```
double x,y; //1 des sommets
double x1,y1; // v1
double x2,y2; // v2
```

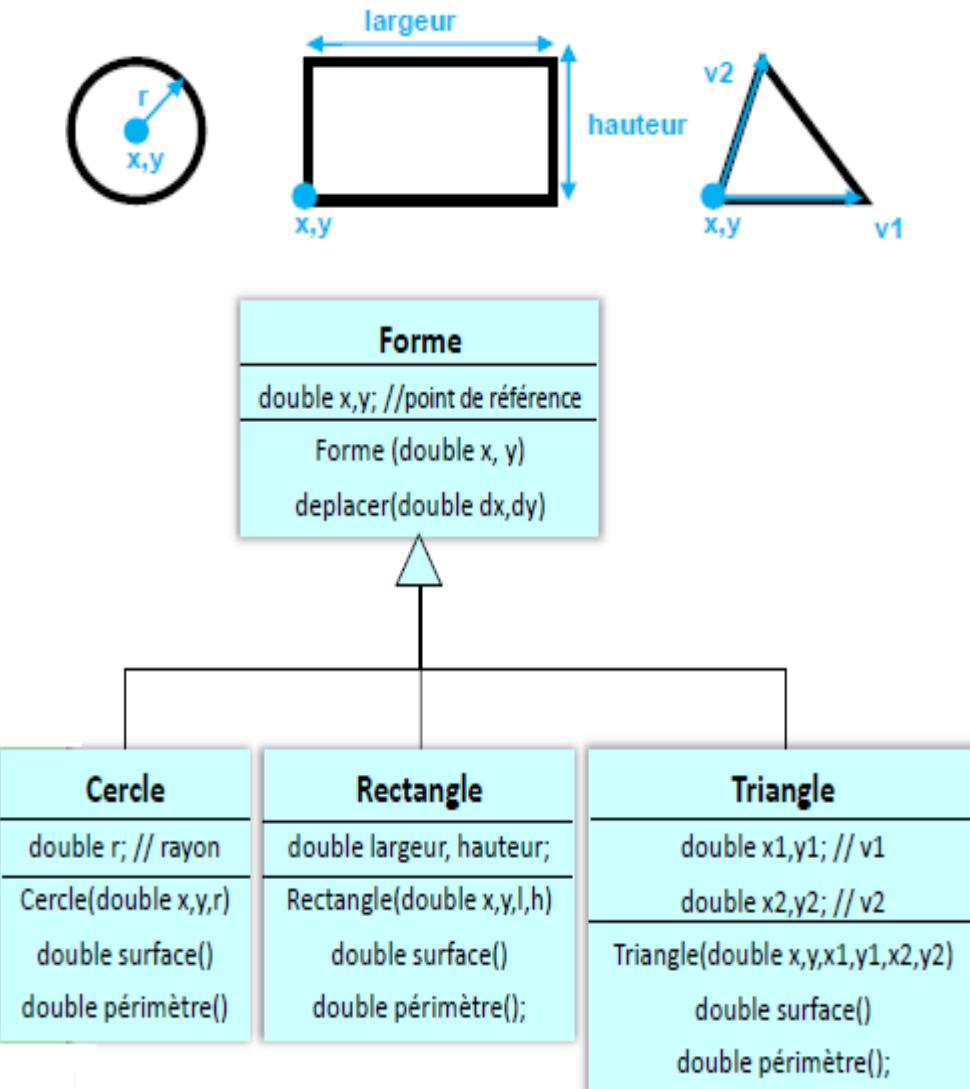
Méthodes :

```
deplacer(double dx, double dy)
double surface()
double périmètre();
```

Factoriser le code ?

Classe abstraite

Exemple introductif



```

public 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 class Cercle extends Forme {
    protected double r;

    public Cercle(double x, double y,
                 double r) {
        super(x,y);
        this.r = r;
    }

    public double surface(){
        return Math.PI * r * r;
    }

    protected double périmètre(){
        return 2 * Math.PI * r;
    }
}
  
```

The code shows the implementation of the **Forme** class and its subclasses. The **Forme** class has a protected attribute **x,y** and a constructor that initializes **x** and **y**. It also has a **deplacer(dx,dy)** method that updates **x** and **y**. The **Cercle** class extends **Forme** and adds its own attribute **r** (radius). It overrides the **surface()** method to calculate the area of the circle using πr^2 . It also implements the **périmètre()** method to calculate the circumference using $2\pi r$.

Classe abstraite

Exemple introductif

```

public class ListeDeFormes {

    public static final int NB_MAX = 30;
    private Forme[] tabFormes = new Forme[NB_MAX];
    private int nbFormes = 0;

    public void ajouter(Forme f) {
        if (nbFormes < NB_MAX)
            tabFormes[nbFormes++] = f
    }

    public void toutDeplacer(double dx,double dy) {
        for (int i=0; i < nbFormes; i++)
            tabFormes[i].deplacer(dx,dy);
    }

    public double périmètreTotal() {
        double perimTotal = 0.0;
        for (int i=0; i < nbFormes++; i++)
            perimTotal += tabFormes[i].périmètre();
        return perimTotal;
    }
}

```

On veut pouvoir gérer des listes de formes

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

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

Définir une méthode **périmètre** dans Forme ?

```

public double périmètre() {
    return 0.0; // ou -1. ???
}

```

Une solution propre et élégante : les **classes abstraites**

Classe abstraite

Exemple introductif

Classe
abstraite

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

Méthodes
abstraites

la classe doit être déclarée comme étant explicitement abstraite

on spécifie qu'un objet de type Forme aura une méthode périmètre et une méthode surface

par contre on ne sait pas comment cela sera implémenté → méthodes sans corps : ; au lieu de { ... }

Classe abstraite

Exemple introductif

- Utilité:
 - définir des concepts incomplets qui devront être implémentés dans les sous classes
 - factoriser le code
 - *les opérations abstraites sont particulièrement utiles pour mettre en oeuvre le polymorphisme.*
 - *l'utilisation du nom d'une classe abstraite comme type pour une (des) référence(s) est toujours possible (et souvent souhaitable !!!)*

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

Classe abstraite

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

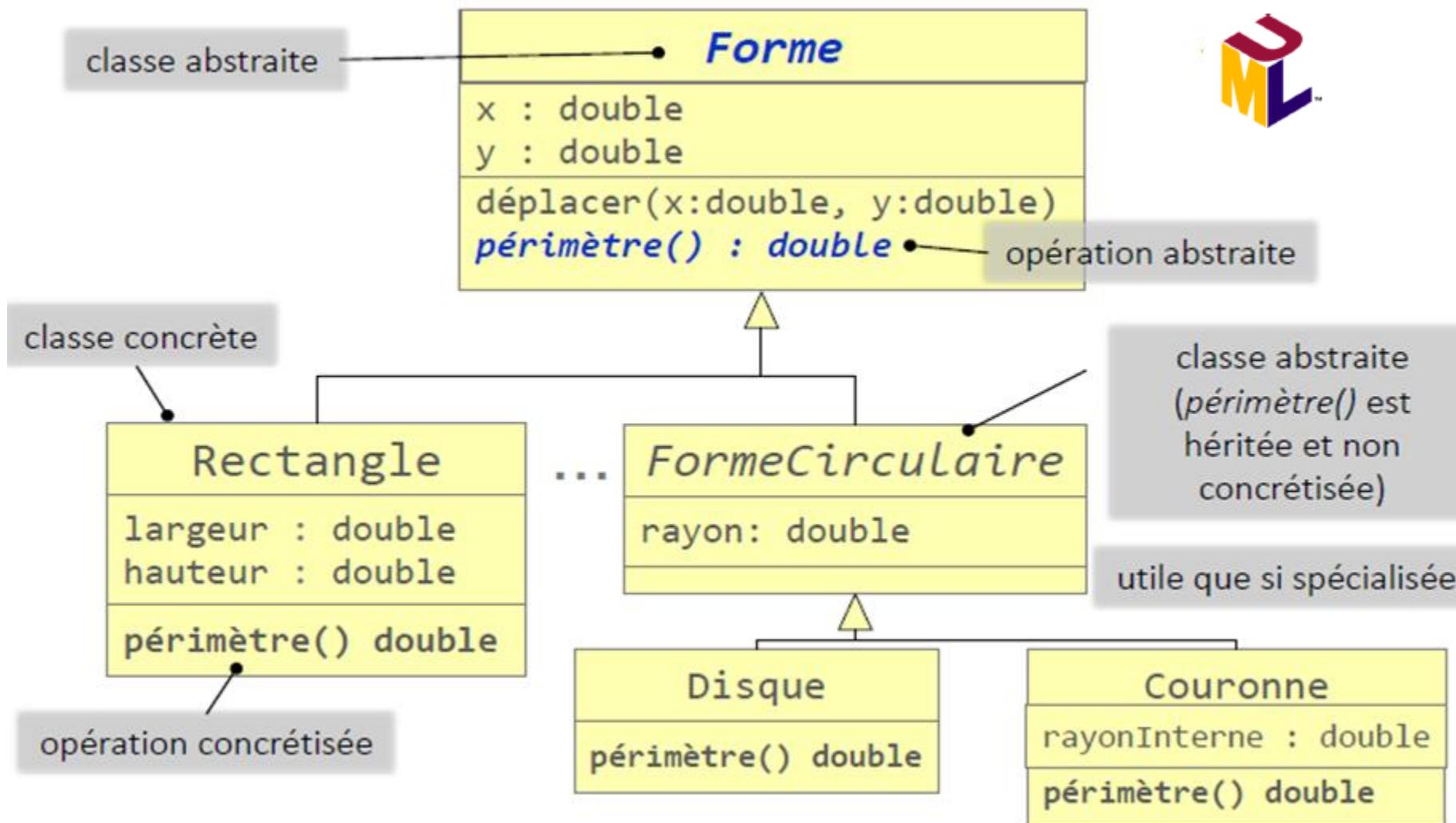
Classe abstraite

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.

Classe abstraite

UML



Classe abstraite

Polymorphisme

```
public class ListeDeFormes {  
  
    public static final int NB_MAX = 30;  
    private Forme[] tabFormes = new Forme[NB_MAX];  
    private int nbFormes = 0;  
  
    public void ajouter(Forme f) {  
        if (nbFormes < NB_MAX)  
            tabFormes[nbFormes++] = f  
    }  
  
    public void toutDeplacer(double dx,double dy) {  
        for (int i=0; i < nbFormes; i++)  
            tabFormes[i].deplacer(dx,dy);  
    }  
  
    public double périmètreTotal() {  
        double perimTotal = 0.0;  
        for (int i=0; i < nbFormes++; i++)  
            perimTotal += tabFormes[i].périmètre();  
        return perimTotal;  
    }  
}
```



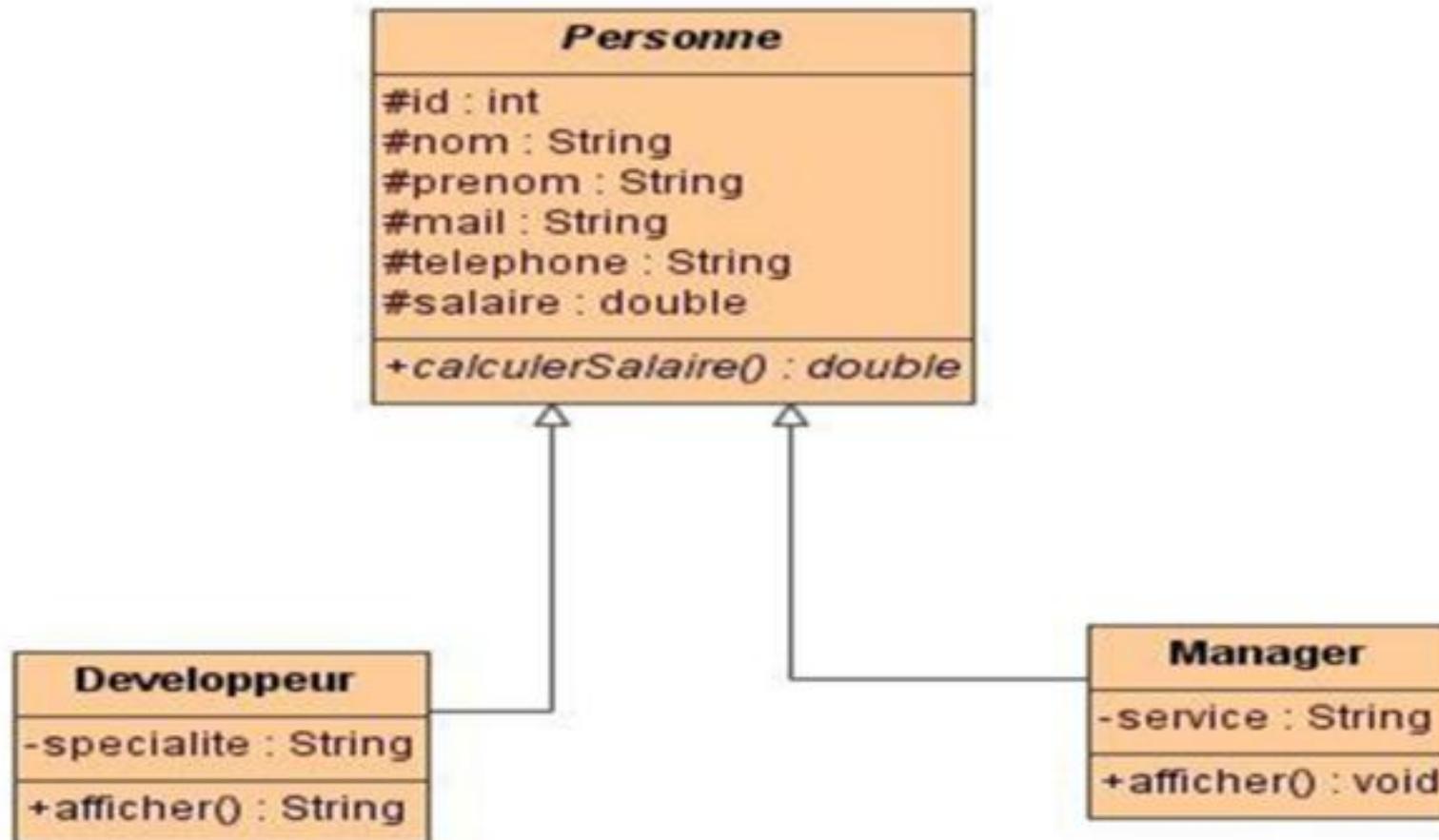
```
public abstract class Forme {  
    protected double x,y;  
  
    public Forme(double x, double y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    public void deplacer(double dx,  
                        double dy) {  
        x += dx; y += dy;  
    }  
  
    public abstract double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

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

Classe abstraite

Exercice

Une société souhaite développer un module pour la gestion des utilisateurs de son service, ci-dessous le diagramme de classe établi :



Classe abstraite

Exercice

1. Créer la classe abstraite « Personne ».
2. Créer les classes « Developpeur » et « Manager ».
3. Redéfinir la méthode `calculerSalaire()`. Sachant que :
 - Le développeur aura une augmentation de 20% par rapport à son salaire normal.
 - Le manager aura une augmentation de 35% par rapport à son salaire normal.
4. Créer deux développeurs et deux managers.
5. Afficher les informations des objets créés.

Sous la forme :

 - Le salaire du manager ELMAMOUN Mohamed est : 30 000 dh, son service : Informatique
 - Le salaire du développeur SLAMI Omar est : 10 000 dh, sa spécialité : PHP
6. Créer un objet de type Personne. Qu'est ce que vous remarquez ?

Classe abstraite

Exercice

```
package ma.projet;

public abstract class Personne {

    protected int id;
    protected String nom;
    protected String prenom;
    protected String mail;
    protected String telephone;
    protected double salaire;

    private static int comp;

    public Personne(String nom, String prenom, String mail, String
                    telephone, double salaire) {
        comp++;
        this.id = comp;
        this.nom = nom;
        this.prenom = prenom;
        this.mail = mail;
        this.telephone = telephone;
        this.salaire = salaire;
    }

    public Personne() {
        comp++;
        this.id = comp;
    }

    public abstract double calculerSalaire();

}
```

Classe abstraite

Exercice

```
package ma.projet.bean;
import ma.projet.Personne;

public class Developpeur extends Personne{

    private String specialite;

    public Developpeur(String nom, String prenom, String email,
                       String telephone, double salaire, String specialite) {
        super(nom, prenom, email, telephone, salaire);
        this.specialite = specialite;
    }

    public String afficher(){
        return ("Le salaire du developpeur "+nom+" "+prenom+" est :
               "+this.calculerSalaire()+" sa spécialité :"+specialite);
    }

    public double calculerSalaire(){
        return salaire = salaire + salaire * 0.2 ;
    }
}
```

Classe abstraite

Exercice

```
package ma.projet.bean;
import ma.projet.Personne;

public class Manager extends Personne{

    private String service;

    public Manager(String nom, String prenom, String email,
                  String telephone, double salaire, String service){
        super(nom, prenom, email, telephone, salaire);

        this.service= service;
    }

    public void afficher(){
        System.out.println("Le salaire du manager "+nom+" "+prenom+" est :
                           "+this.calculerSalaire()+" , son service : "+service);
    }

    public double calculerSalaire(){
        return salaire = salaire + salaire * 0.35 ;
    }
}
```

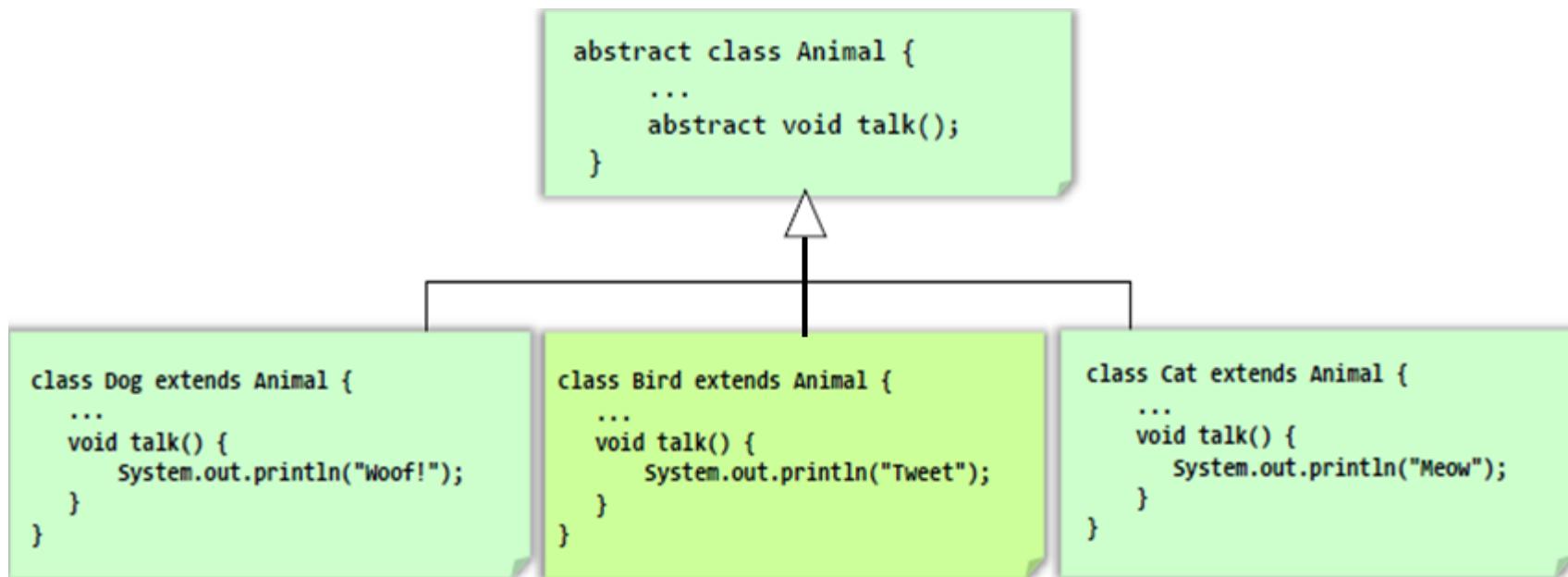
Classe abstraite

Exercice

```
public class Test {  
  
    public static void main( String[] args){  
  
        Developpeur[] developpeurs =new Developpeur[2];  
  
        developpeurs[0]= new Developpeur("Salim", "karim", "salim@gmail.cm",  
            "34567890", 8330, "PHP" );  
        developpeurs[1]= new Developpeur("Dourid", "Raouan", "dorid@gmail.cm",  
            "34567890", 1200.00, "JAVA");  
  
        Manager[] managers = new Manager[2];  
  
        managers[0]=new Manager("LACHGAR", "Mohamed", "m.lacgar@gmail.cm",  
            "34567890", 22230, "Informatique");  
        managers[1]= new Manager("laoud", "hadi", "laoud@gmail.cm", "34567890",  
            1300.00,"RH");  
  
        for (Developpeur dv : developpeurs){  
            System.out.println(dv.afficher());  
        }  
  
        for (Manager mg : managers){  
            mg.afficher();  
        }  
        // Personne per = new Personne("laoud", "hadi", "laoud@gmail.cm",  
        // "34567890", 1300.00);  
        //Une classe abstraite ne peut pas être instanciée.  
    }  
}
```

Interface

Exemple introductif



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*)

```

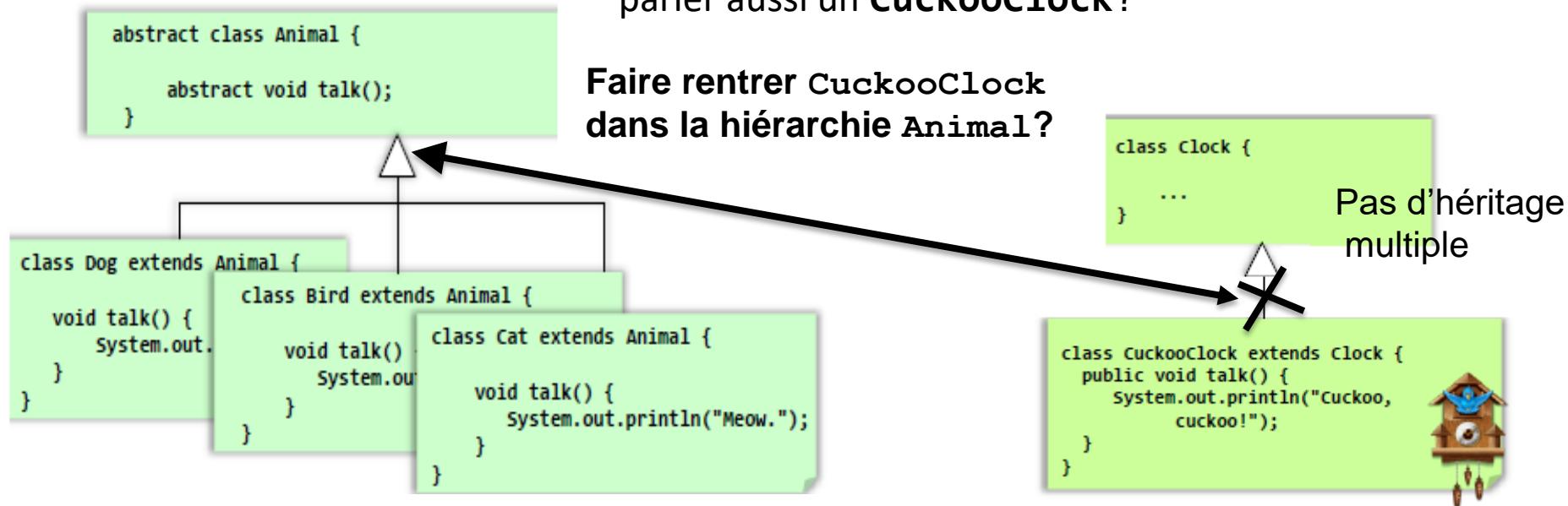
class Interrogator {
    static void makeItTalk(Animal subject) {
        subject.talk();
    }
}
  
```

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

Interface

Exemple introductif

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



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

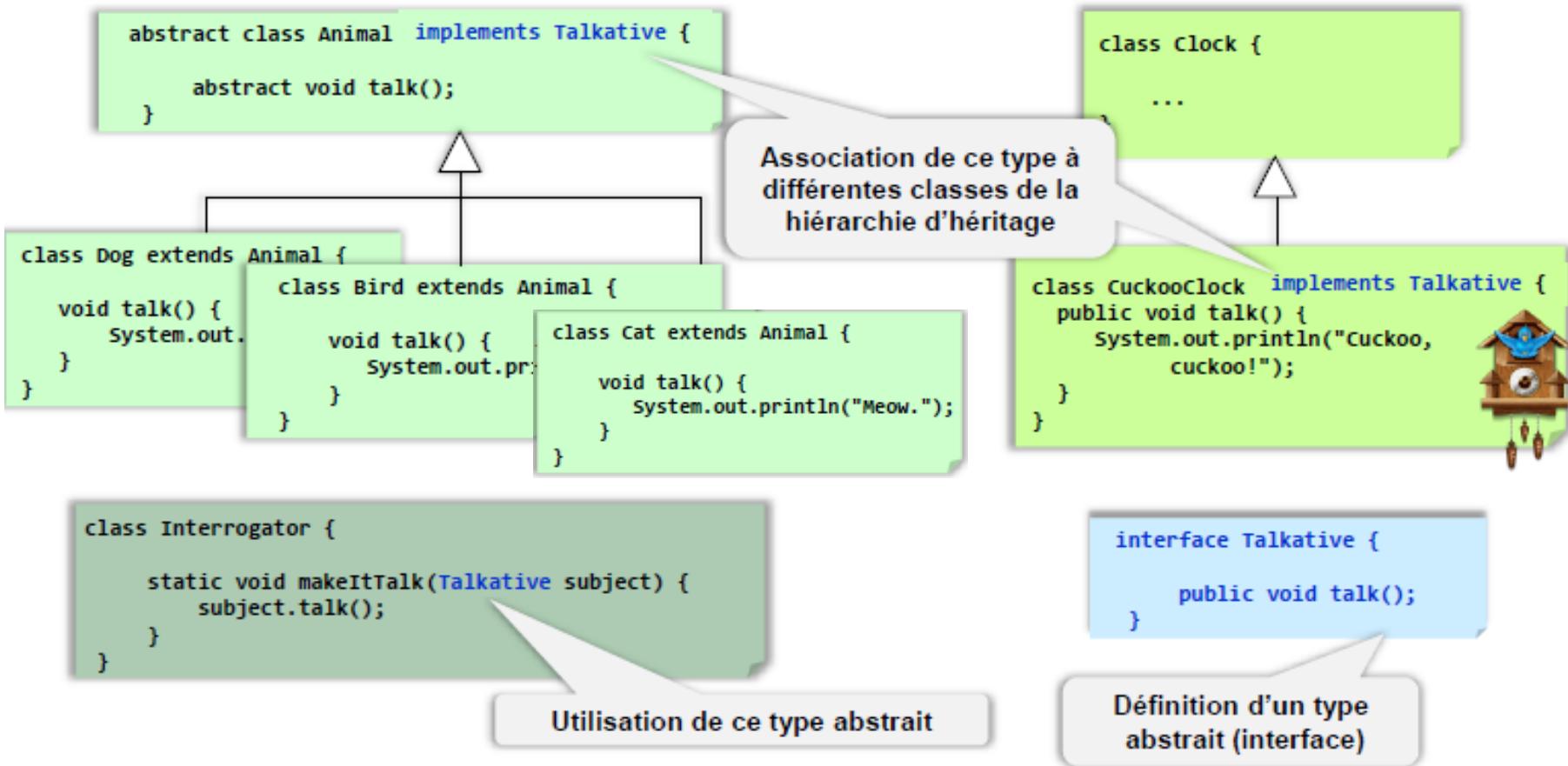
```
class CuckooClockInterrogator {
    static void makeItTalk(CuckooClock subject) {
        subject.talk();
    }
}
```

Se passer du polymorphisme ?

Interface

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



Interface

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

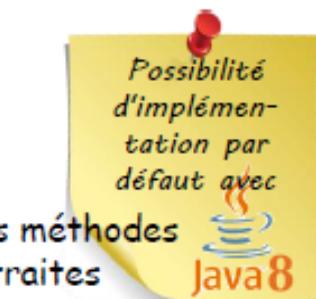
```
package m2pcci.dessin;  
import java.awt.Graphics;  
  
public interface Dessinable {  
    public void dessiner(Graphics g);  
    void effacer(Graphics g);  
}
```

Dessinable.java



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

opérations abstraites



Toutes les méthodes
sont abstraites
Elles sont implicitement
publiques



interface



Interface

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;

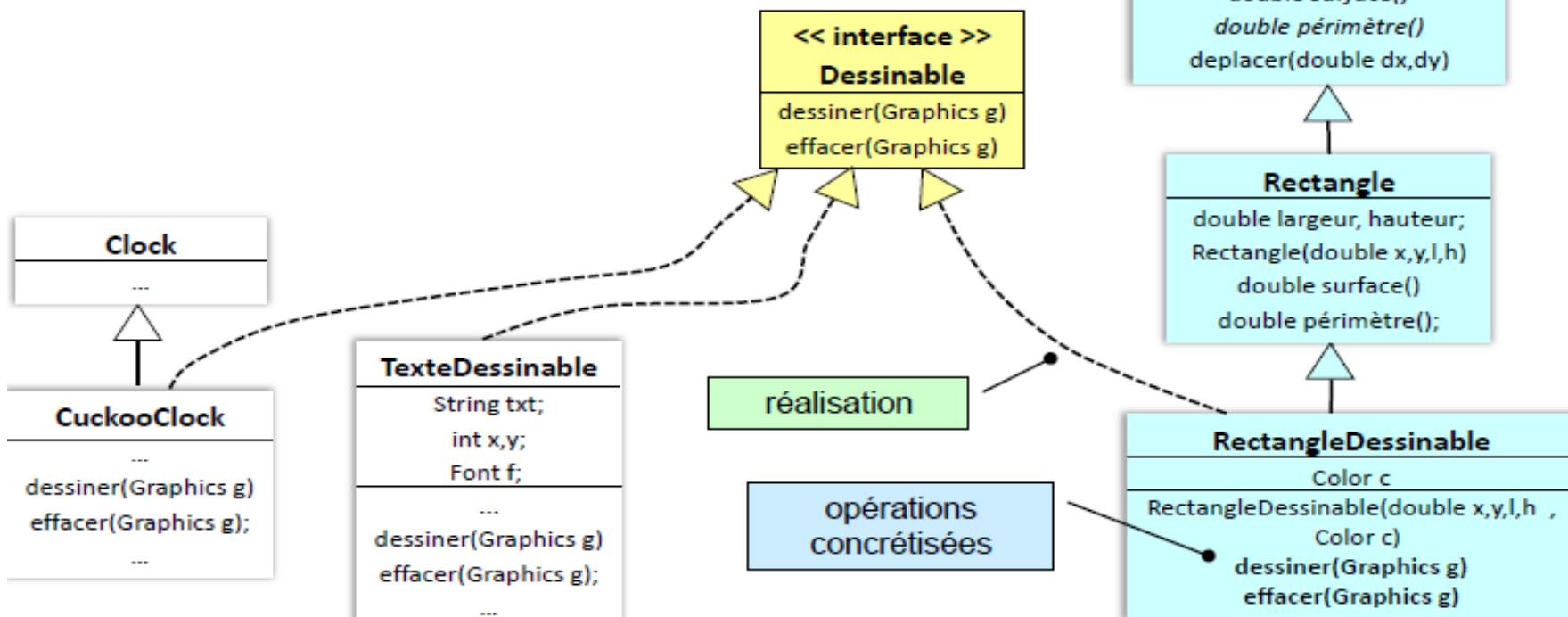
    public void dessiner(Graphics g);
    void effacer(Graphics g);
}
```

Dessinable.java

Interface

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*



Interface

Réalisation d'une interface

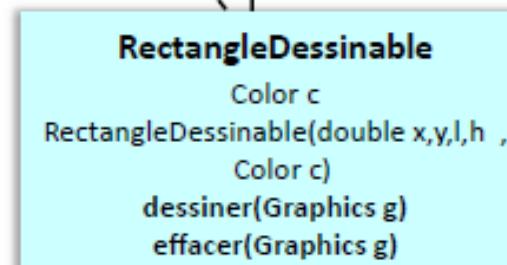
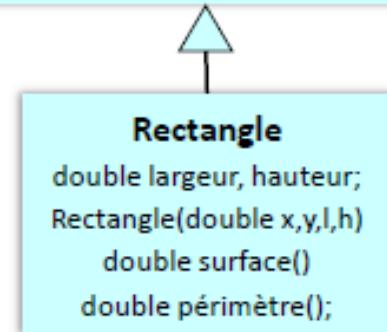
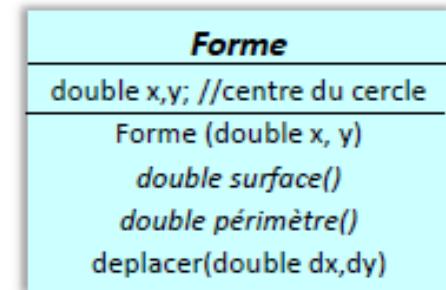
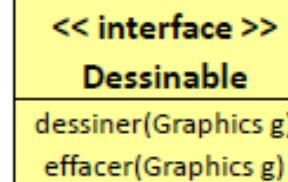
- De la même manière qu'une classe étend sa super-classe elle peut de manière optionnelle implémenter une ou plusieurs interfaces

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

```
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)*

Interface

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`

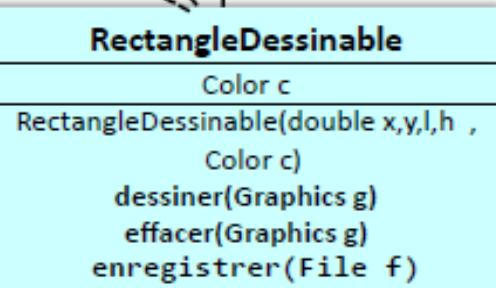
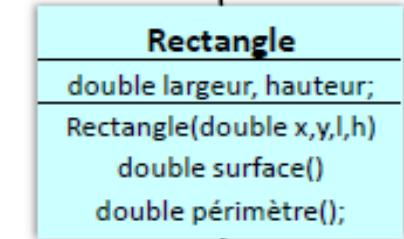
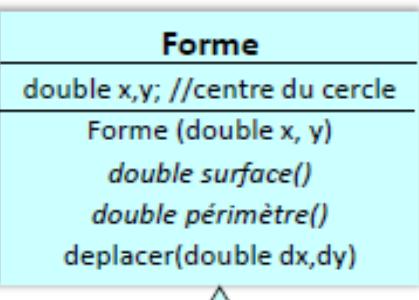
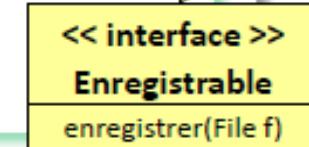
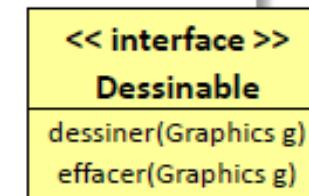
```

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) {
        ...
    }
}
  
```



Interface

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

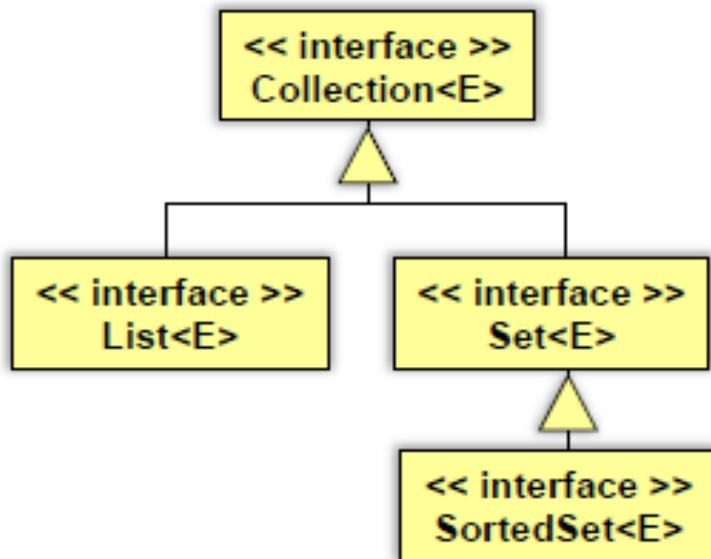
```
public class ZoneDeDessin {  
    private nbFigures;  
    private Dessinable[] figures;  
    ...  
    public void ajouter(Dessinable d){  
        ...  
    }  
    public void supprimer(Dessinable o){  
        ...  
    }  
  
    public void dessiner() {  
        for (int i = 0; i < nbFigures;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

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

Interface

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.

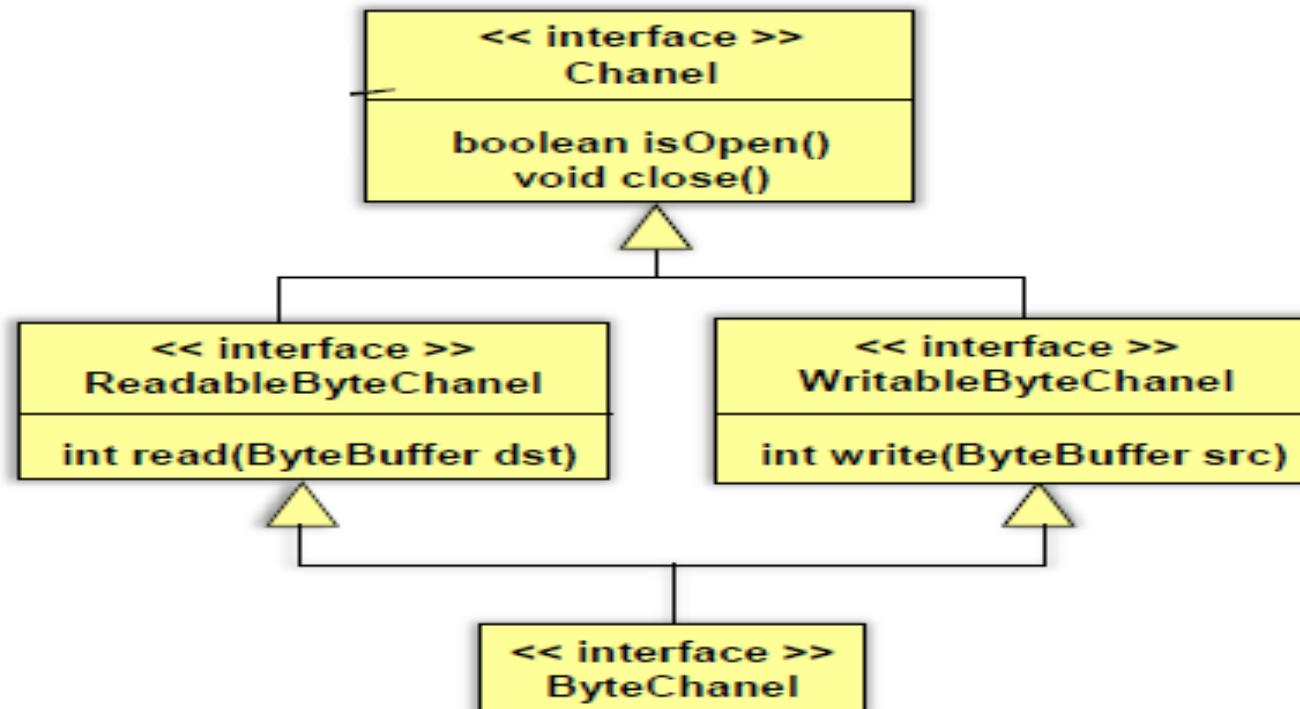


```
interface Set<E> extends Collection<E> {  
    ...  
}
```

Interface

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, WriteableByteChannel {  
}
```

Interface

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

Interface

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.

Interface

Java 8: quoi de neuf dans les interfaces ?

- Java 7
 - 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 interfaces 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>

Interface

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]
```

Interface

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 {  
     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()");  
    }  
}
```

Interface

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()");  
    }  
}
```

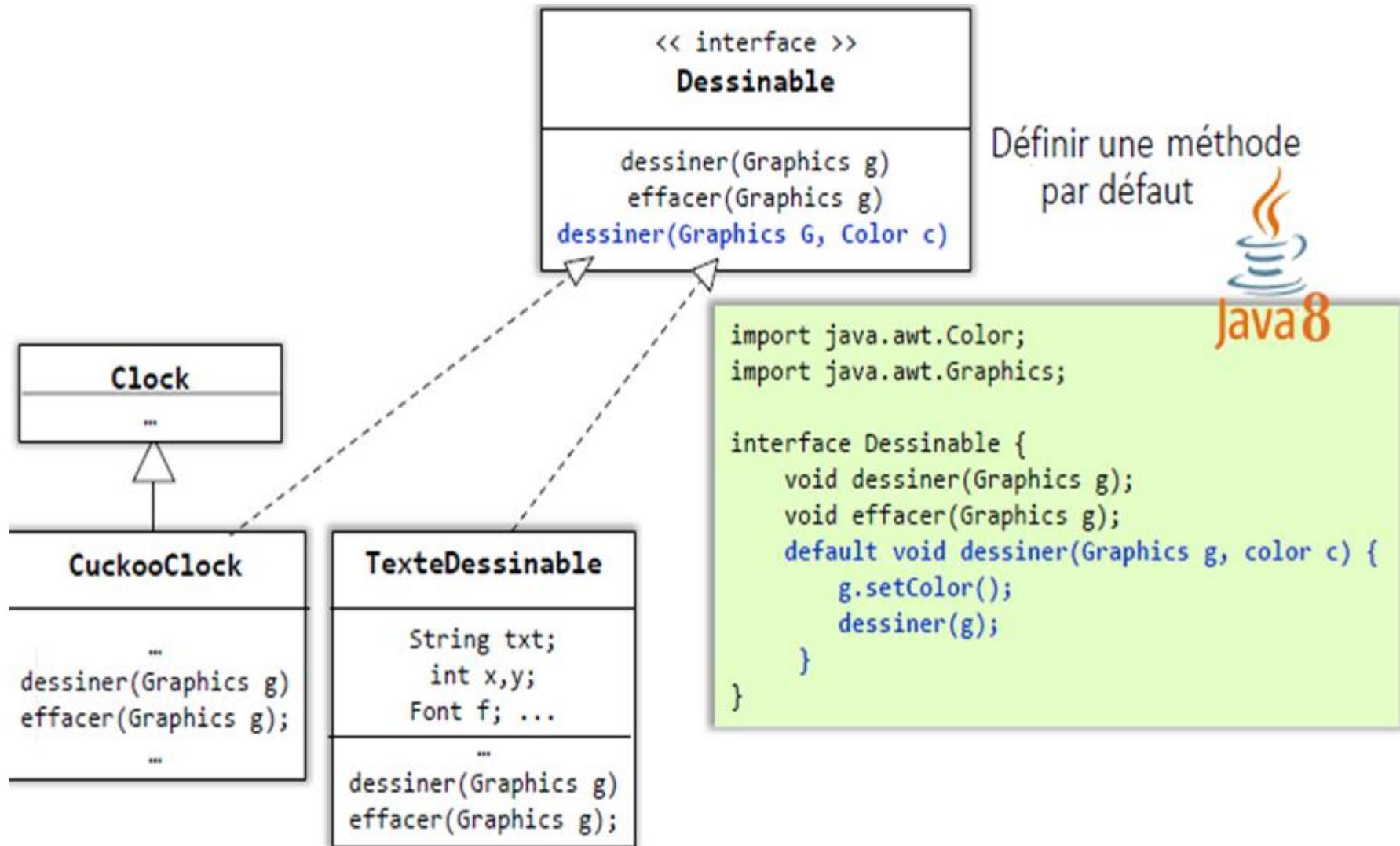
```
public class Cls implements InterfaceA, InterfaceB {  
    public void foo() {  
        InterfaceB.super.foo();  
    }  
}
```

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

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

Interface

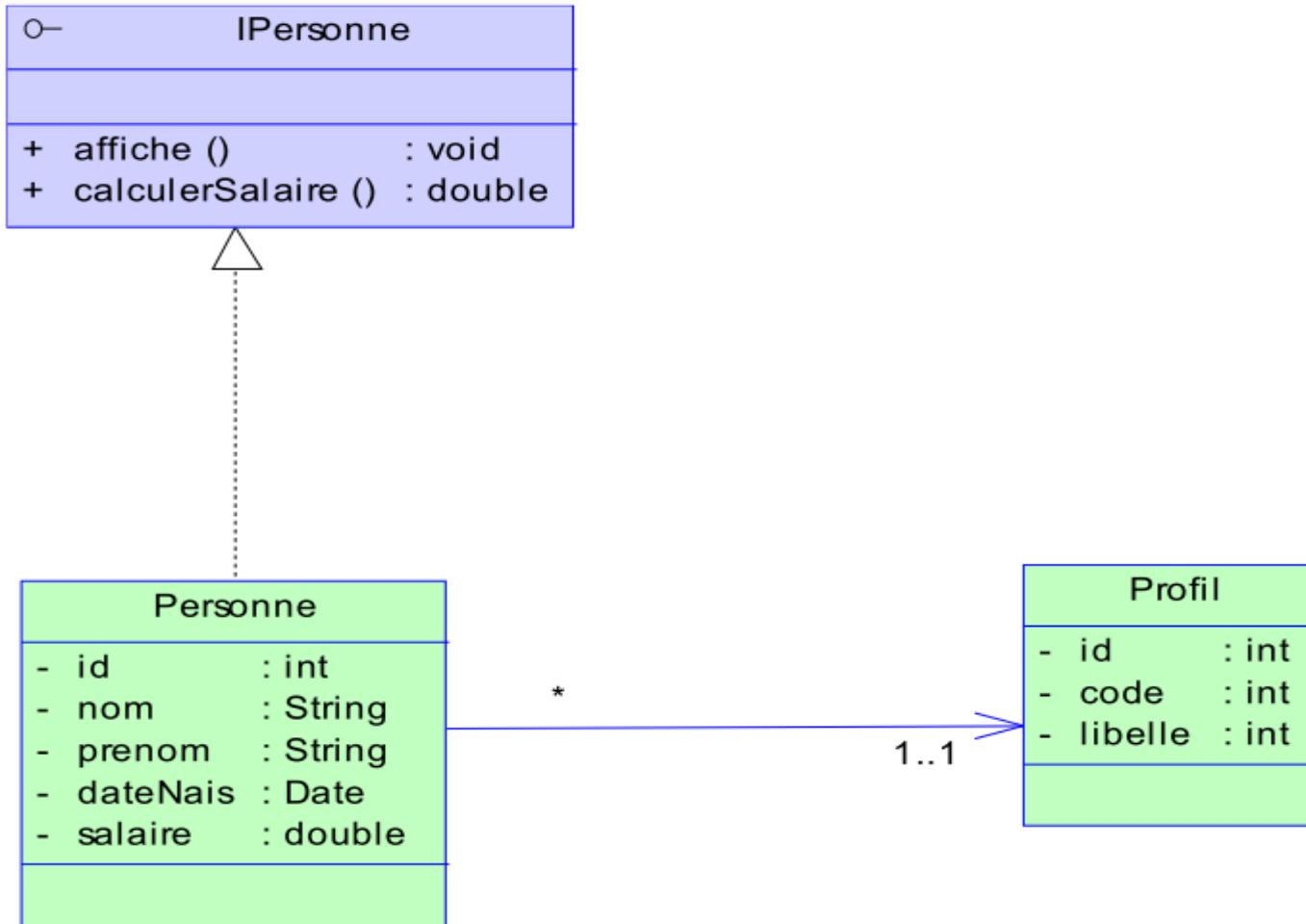
Java 8: méthodes par défaut



Interface

Exercice

Soit le diagramme de classe suivant :



Interface

Exercice

1. Créer l'interface Ipersonne.

2. Créer les classes Personne et Profil

Sans redéfinir les méthodes de l'interface, qui ce que vous remarquez ?

3. Redéfinir les méthodes affiche () et calculerSalaire () dans la classe Personne.

Sachant que :

La méthode affiche () affiche une chaîne de caractère sous la forme :

Je suis le directeur SAIMI Karim né le 02 juin 1970 mon salaire est 20 000dh

Le directeur aura une augmentation de 20% par rapport à son salaire normal,

Les autres employés auront une augmentation de 10%.

4. Ecrire un programme de test.

Interface

Exercice

```
package ma.projet.inter;

public interface IPersonne {

    public void affiche();
    public double calculerSalaire ();

}
```

Interface

Exercice

```
package ma.projet.bean;

public class Profil {
    private int id;
    private String code;
    private String libelle;
    private static int comp;

    public Profil(String code, String libelle) {
        comp++;
        this.id = comp;
        this.code = code;
        this.libelle = libelle;
    }

    public String getCode() {
        return code;
    }

    public String getLibelle() {
        return libelle;
    }
}
```

Interface

Exercice

```
package ma.projet.bean;

import java.text.DateFormat;
import java.util.Date;
import java.util.Locale;
import ma.projet.inter.IPersonne;

public class Personne implements IPersonne{

    private int id;
    private String nom;
    private String prenom;
    private Date dateNais;
    private double salaire;
    private Profil profil;
    private static int comp ;

    public Personne(String nom, String prenom, Date dateNais, Double salairee, Profil profil) {
        comp++ ;
        this.id = comp;
        this.nom = nom;
        this.prenom = prenom;
        this.dateNais = dateNais;
        this.salaire = salairee;
        this.profil = profil;
    }

    public void affiche() {
        DateFormat df = DateFormat.getDateInstance(DateFormat.LONG, Locale.FRANCE);
        String date = df.format(dateNais);
        System.out.println("Je suis le "+this.profil.getLibelle() +" "+nom+" "+prenom+ " né le "
            +date+ " mon salaire est "+calculerSalairee()+" dh");
    }

    public double calculerSalairee() {
        if(this.profil.getCode().equals("DG"))return this.salaire= this.salaire +
            0.2*this.salaire;
        if(this.profil.getCode().equals("EMP"))return this.salaire= this.salaire +
            0.1*this.salaire;
        return this.salaire;
    }
}
```

Interface

Exercice

```
package ma.projet.bean;

import java.util.Date;

public class Test {
    public static void main(String[] args) {

        Profil profiles[] = new Profil[2];

        profiles[0] = new Profil("EMP", "Employé");
        profiles[1] = new Profil("DG", "Directeur Général");

        Personne personnes[] = new Personne[3];

        personnes[0] = new Personne("ELALAOUI", "Mohamed", new
                                    Date("1986/04/04") , 9000.0, profiles[0]);
        personnes[1] = new Personne("DOURID", "Raouan", new
                                    Date("1960/03/24") , 10000.00, profiles[1] );
        personnes[2] = new Personne("ALAOUI", "Sara", new
                                    Date("1940/09/11") , 8200.00, profiles[0] );

        for (Personne per : personnes){
            per.affiche();
        }
    }
}
```