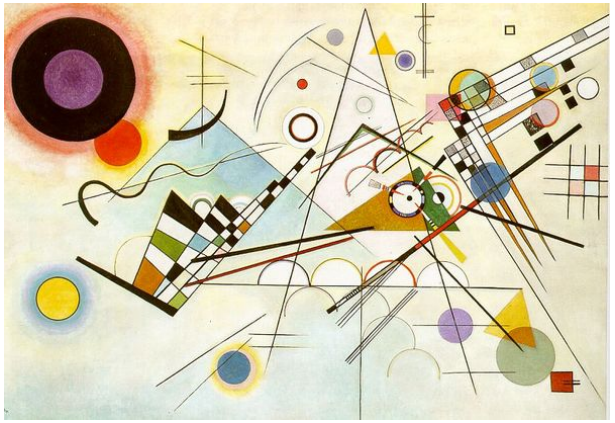
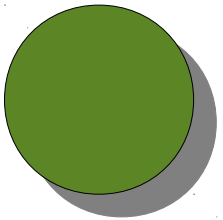
Classes abstraites et Interfaces



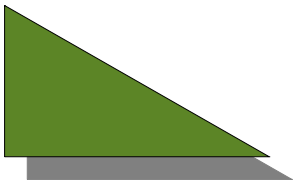
Les formes géométriques

- 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) ; on peut vouloir calculer son périmètre, sa surface



Attributs :
double x,y; //centre du cercle
double r; // rayon
Méthodes :
déplacer(double dx, double dy)
double surface()
double périmètre()

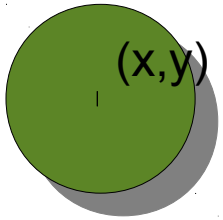
Attributs :
double l,L;
double x,y; // coin
Méthodes :
déplacer(double dx, double dy)
double surface()
double périmètre()

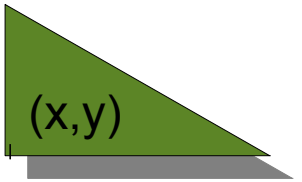


Attributs :
double x,y;
double x1,y1;
double x2,y2;
Méthodes :
déplacer(double dx, double dy)
double surface()
double périmètre()

→ Factoriser le code ?

Formes abstraites






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

}
```

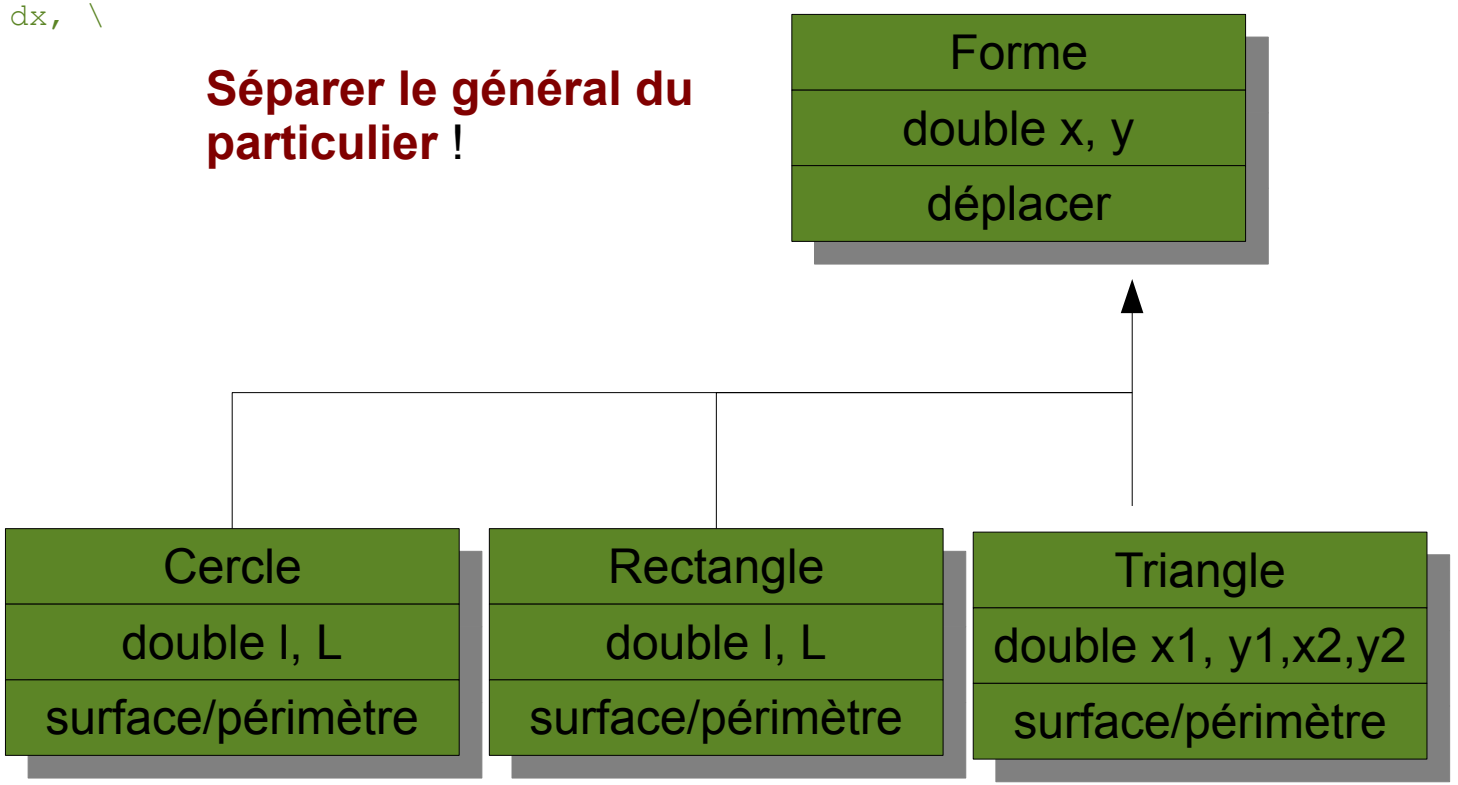
```
class Cercle extends Forme {
protected double r;

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

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

}
```

Séparer le général du particulier !



Liste de formes

```
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++)
    tabFrome[i].deplace(dx,dy);
}
public double périmetreTotal(){
    double pt = 0.0;
    for (int i=0; i < NbFormes++; i++)
        pt += tabForme[i].perimetre();
    return pt;
}

}
```

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

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

```
public double perimetre() {
    return  -1;
}
```

????

Classe abstraite

La solution est de définir une classe abstraite : qui permet de définir des concepts incomplets qui devront être implémentés dans les sous-classes

```
public abstract class Forme {

protected double x,y;

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

public abstract double perimetre() ;

public abstract double surface();
}
```

Classe abstraite

Méthodes abstraites

Classe abstraites

- **Classe abstraite est non *instanciable*, c'est-à-dire qu'elle n'admet pas d'instance directe.**
  - Impossible de faire **new** ClasseAbstraite(…);
- **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).**

- 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 même souvent souhaitable !!!**

Classe abstraites

- Une classe abstraite est une description d'objets **destinée à être héritée** par des classes plus spécialisées.

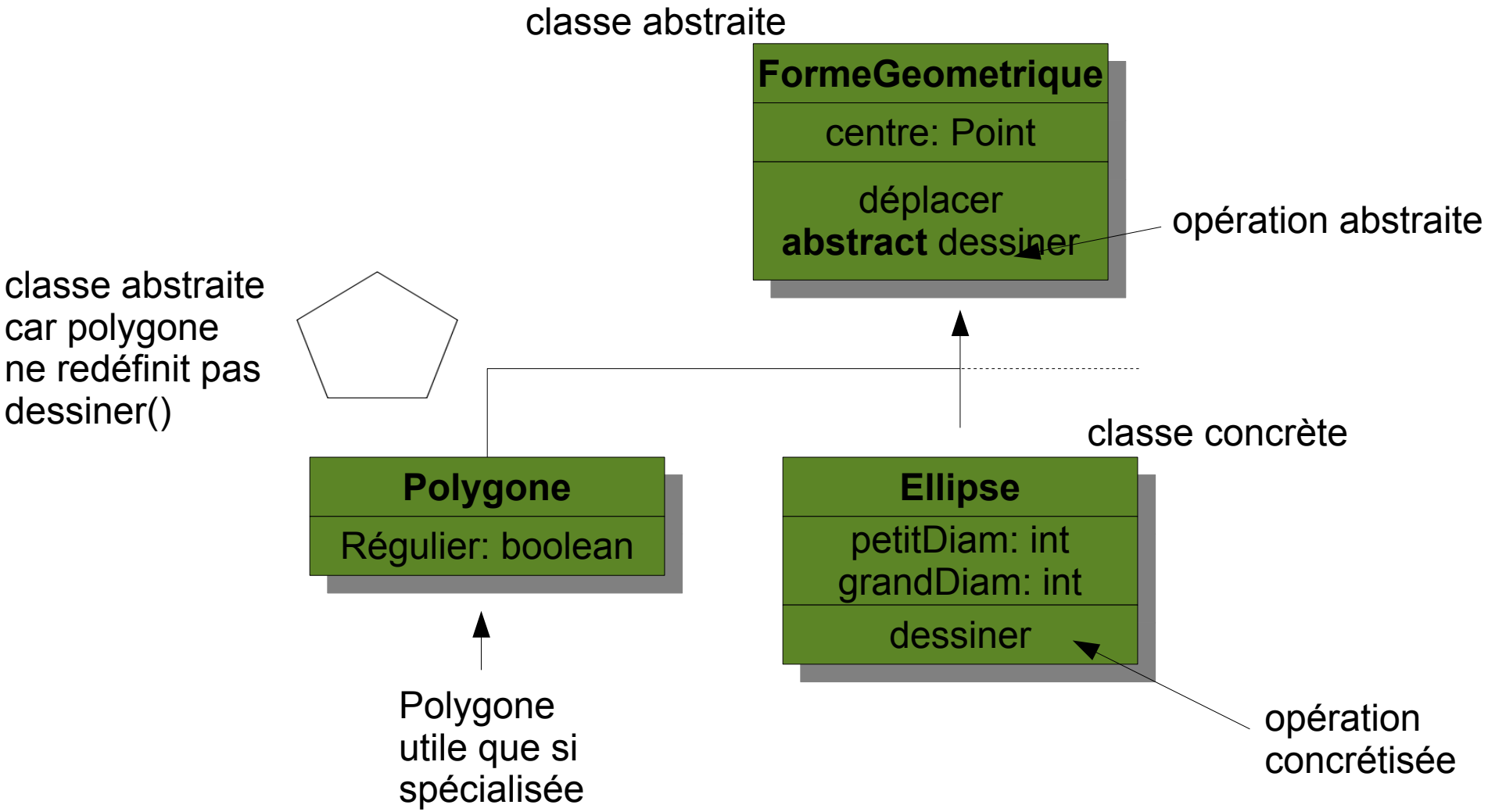
- Pour être utile, une classe abstraite doit **admettre des classes descendantes** concrètes.

- Toute classe concrète sous-classe d'une classe abstraite doit **“concrétiser” toutes les opérations abstraites** de cette dernière.

- Une classe abstraite permet de regrouper certaines caractéristiques communes à ses sous-classes et définit un comportement minimal commun.

- **La factorisation de propriétés communes à un ensemble de classes (par généralisation) nécessite le plus souvent l'utilisation des classes abstraites.**

Retour sur les FormeGéométrique

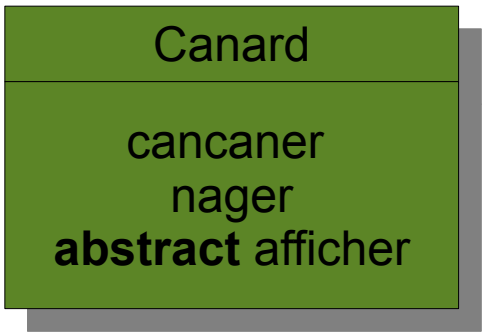


Des classes abstraites aux Interfaces

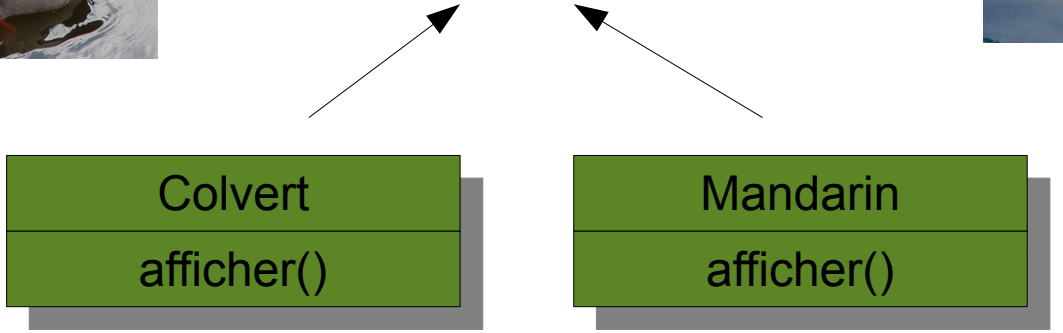
Application SuperCanard

- Notre société à rencontrer un énorme succès avec l'application **SuperCanard**, un simulateur de mare aux canards. Le jeu affiche toutes sortes de canards qui nagent et qui émettent des sons.

Tous les canards cancanent et nagent. La super classe gère le code pour tous.



Tous les canards ont un aspect différent, donc la méthode afficher est abstraite

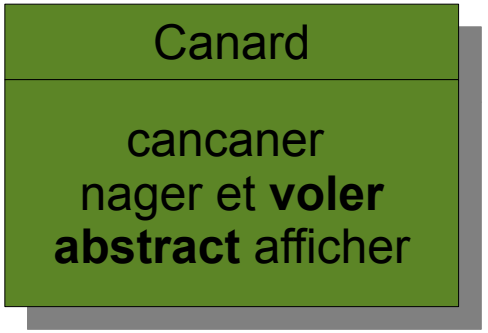


Chaque sous-type à la charge de savoir s'afficher → rien de 9 sous le soleil

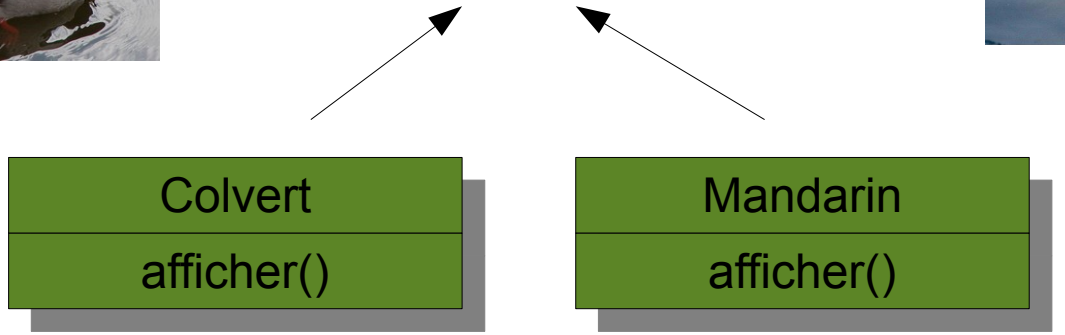
Application SuperCanard

- Nos dirigeants ont décidé que voler était ce qu'il fallait pour mettre à terre tous nos concurrents. Comment faire ? Ajouter la comportement voler à la classe mère.

Tous les canards cancanent et nagent. La super classe gère le code pour tous.



Tous les canards ont un aspect différent, donc la méthode afficher est abstraite



Chaque sous-type à la charge de savoir s'afficher → rien de 9 sous le soleil

Application SuperCanard

- La concurrence vous talonne et SimuCoinCoin, application concurrente gère maintenant les leurres … Il faut réagir !!! vous décidez d'ajouter des leurres et des canards en plastique.

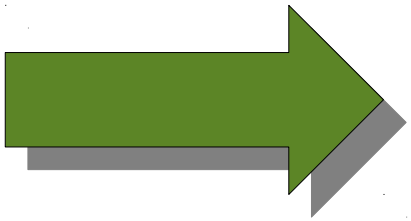
Ok mais ni les canards en plastique, ni les leurres ne volent !!???

Notion d'interface

Les interfaces permettent de décrire des comportement identiques pour un ensemble de classes.

Par exemple :

- Tous les appareils électriques peuvent être allumés ou éteint
- Tous les objets comestibles peuvent être mangés
- Tous les **vrais** canards volent...



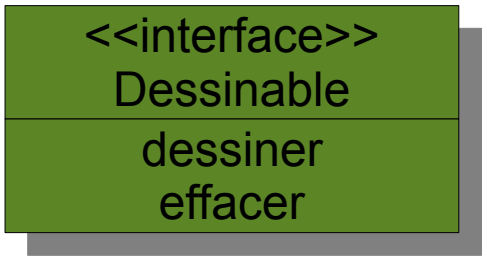
Pour définir qu'une certaine catégorie de classes doit implémenter un comportement, c.-à-d. un ensemble de méthodes, on peut regrouper les déclarations de ces méthodes dans une interface.

Notion d'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 abstraite sans attribut et dont toutes les opérations sont abstraites.

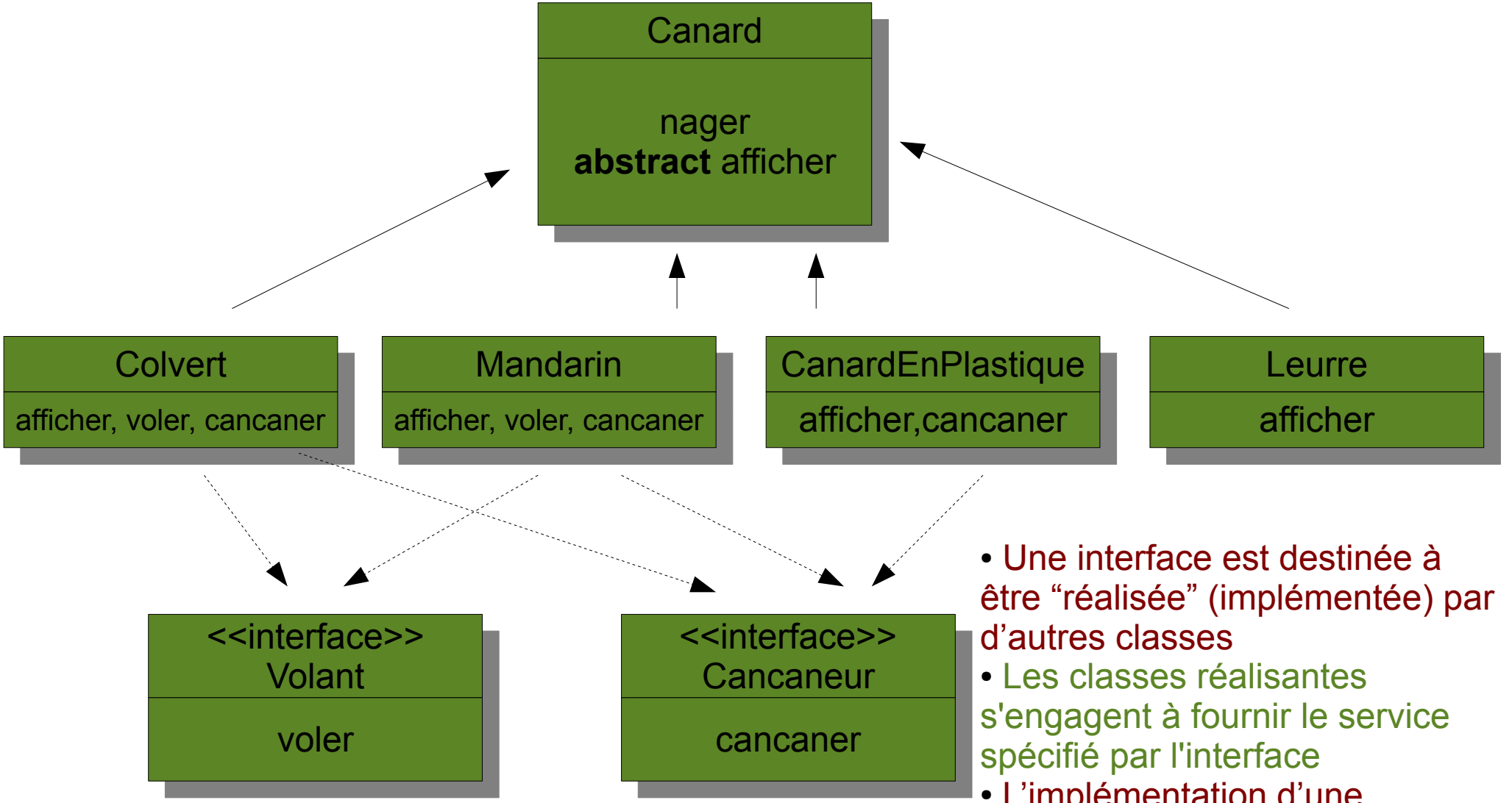
```
import java.awt.*;
public interface Dessinable {
public void dessiner(Graphics g);
void effacer(Graphics g);
}
```

→ dans le fichier Dessinable.java



• Toutes les méthodes sont abstraites
• Elles sont implicitement publiques
• Possibilité de définir des attributs à condition qu'il s'agisse d'attributs de type primitif
• Ces attributs sont implicitement déclarés comme **static final**

Application SuperCanard



- Une interface est destinée à être “réalisée” (implémentée) par d’autres classes
- Les classes réalisantes s'engagent à fournir le service spécifié par l'interface
- L’implémentation d’une interface est libre.

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 {

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

```
class RectangleDessinable extends Rectangle implements
Dessinable,Comparable {

  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 int compareTo(Object o) {
    ...
  }
}
```

Dessinable

Comparable

Une interface est un type

- Une interface peut être utilisée comme un type
  - À 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 Fenetre {
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);
}
}
```

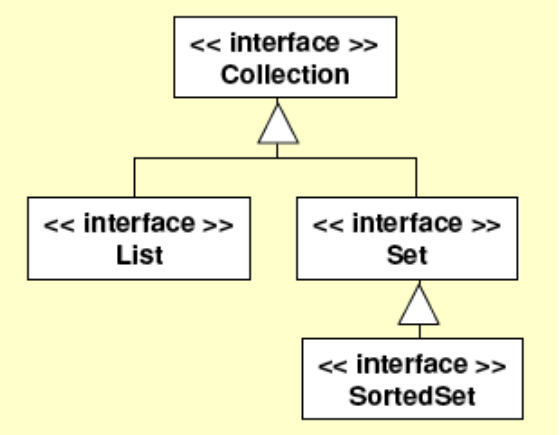
```
Dessinable d;
..
d = new RectangleDessinable(…);
...
d.dessiner(g);
d.surface();
```

Les interfaces focalisent certaines fonctionnalités uniquement ; ici, le dessin.

Les 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 "superinterface"
  - peut définir de nouvelles constantes et méthodes abstraites

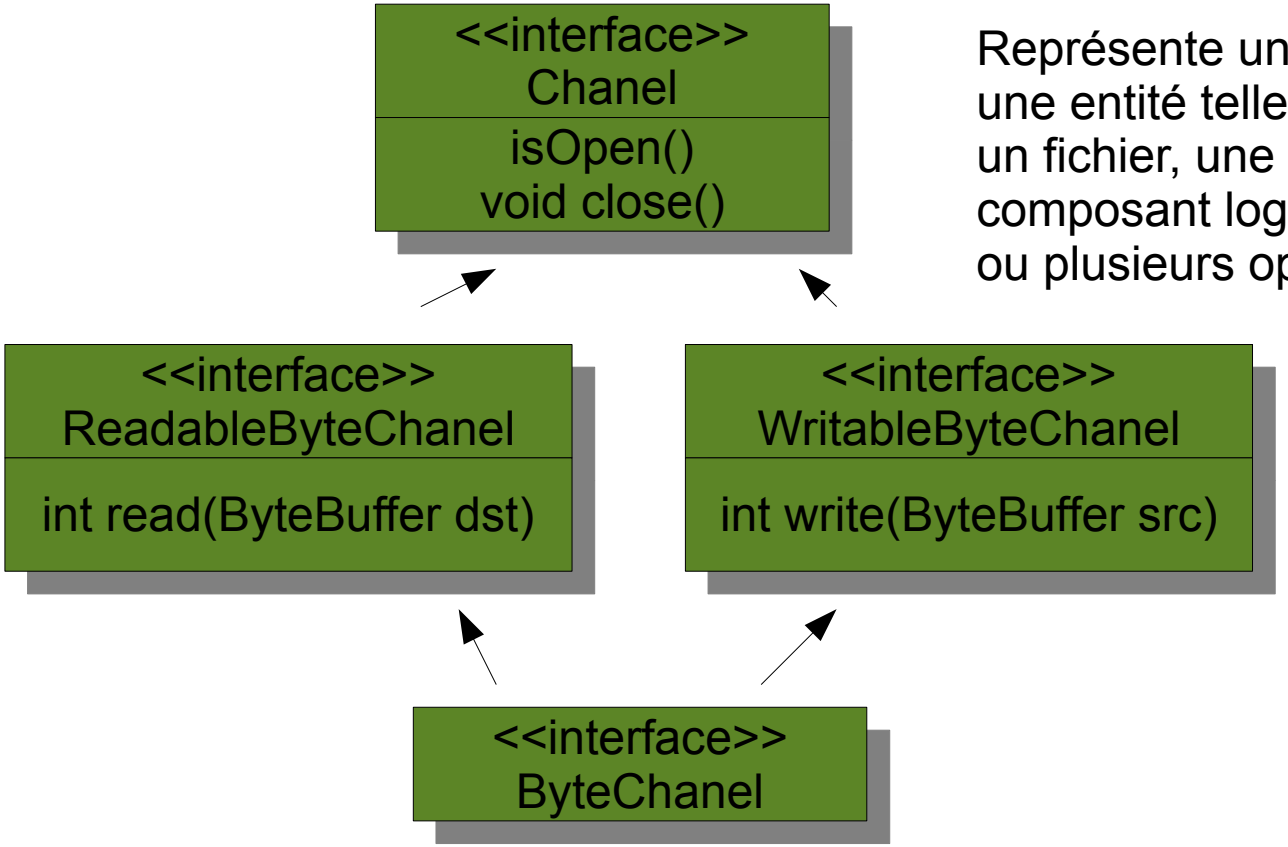



```
interface Set extends Collection
{
...
}
```

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

Héritage d'interface

- **À 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 dispositif 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** { }