JAVA

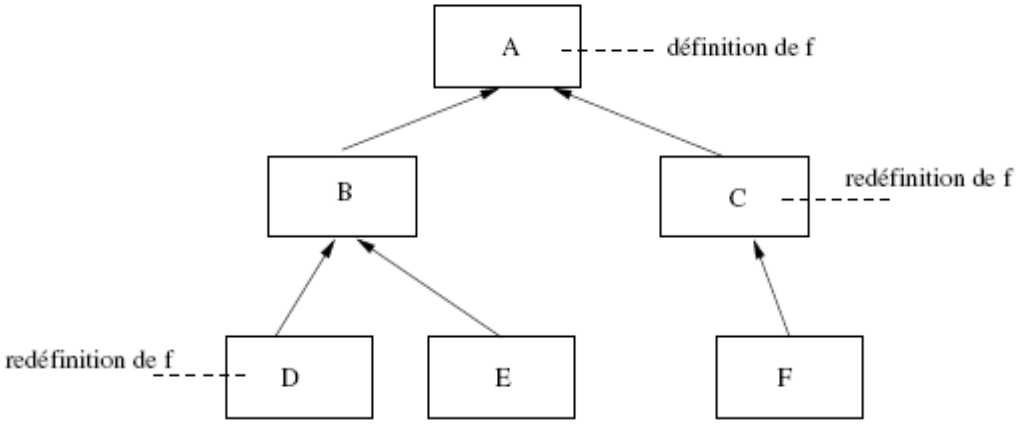
Surdéfinition, redéfinition et héritage

La Redéfinition

La *redéfinition* consiste à re-écrire une méthode définit dans la super-classe et à changer son comportement. Le nombre et le type des arguments ainsi que la valeur de retour doivent être exactement les mêmes.

```
public class Point
{
  private int x,y;
  ...
  public void affiche()
  {
    System.out.println("Je suis un point de coordonnées "+x+" "+y);
  }
}

public class PointCouleur extends Point
{
  private byte couleur;
  ...
  public void affiche()
  {
    super.affiche();
    System.out.println("   de couleur "+couleur);
  }
}
```



Voici la liste des méthodes `f` qui seront appelées en fonction du type de l'objet considéré :

- pour `A`, la méthodes `f` de `A`,
- pour `B`, la méthodes `f` de `A`,
- pour `C`, la méthodes `f` de `C`,
- pour `D`, la méthodes `f` de `D`,
- pour `E`, la méthodes `f` de `A`,
- pour `F`, la méthodes `f` de `C`.

Attention. Les droits d'accès des méthodes redéfinies ne doivent pas être diminués : une méthode publique dans la super-classe ne peut pas être redéfinie privée dans la classe dérivée, l'inverse est cependant possible.

Le Surcharge

La *surcharge* ou *surdéfinition* consiste à modifier le prototype d'une méthode existante, en changeant le nombre d'arguments et/ou le type des arguments. Nous avons déjà vu cette notion auparavant, dans le cadre de l'héritage, la recherche d'une méthode acceptable se fait en "remontant" les dérivations successives.

Polymorphisme

Le *polymorphisme* permet de manipuler des objets sans connaître tout à fait leur type. C'est un principe extrêmement puissant en programmation orientée objet, qui complète l'héritage.

```
public class Point
{
  private int x,y;

 public Point(int x,int y){ ..

 public Point(){ ... }

 public void affiche(){ ... }

 ...
}
```

```
public class PointCouleur extends Point
{
  private byte couleur;

  public PointCouleur(int x,int y,byte couleur){ ... }

  public PointCouleur(){ ... }

  public void affiche(){ ... }

  ...
}
```

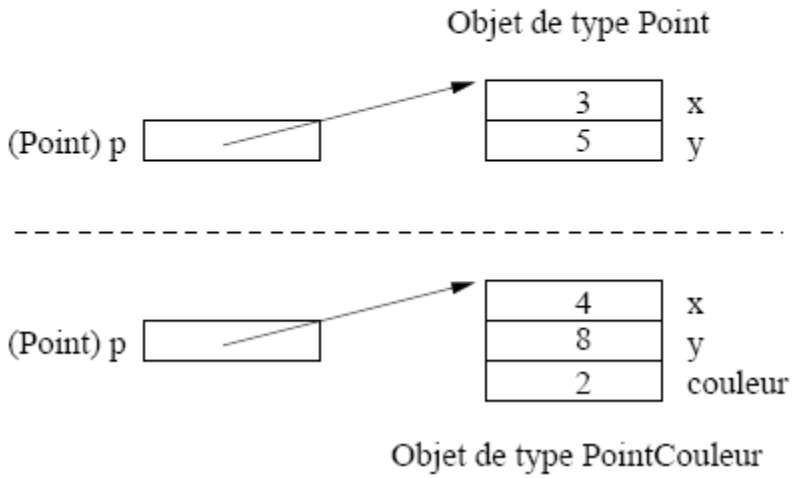
Polymorphisme

Tout élément de type `PointCouleur` est également de type `Point`, il est possible d'utiliser cette caractéristique commune.

```
Point p;
p = new Point(3,5);

...

p = new PointCouleur(4,8,(byte) 2);
```



Polymorphisme

Autre exemple.

```
Point p = new Point(3,5);
p.affiche(); //methode affiche de la classe Point

p = new PointCouleur(4,8,(byte)2);
p.affiche(); //méthode affiche de la classe PointCouleur
```

Le choix de la méthode à appliquer se fait automatiquement en fonction du type effectif de l'objet et non pas en fonction du type de la variable qui référence l'objet. Ce choix porte le nom de *liaison dynamique* ou *ligature dynamique*.

Tester