



Design Pattern Adapter

PAREILLEUX KILLIAN
DIGNAC DORIAN

BOUYSSOU GATIEN
LE SAUX LOGAN

Plan

- I - Présentation
 - Design Pattern
 - Adapter
- II – Exemple
- Conclusion

Présentation

Design Pattern

Une solution générale et reproductible

Conception de logiciels

Modèle → pas de code

Transposable

A large, dark blue puzzle piece is the central focus, featuring the word "Solution" in a light blue, sans-serif font. To the left of this piece, a stylized, light blue figure of a person in a suit is shown holding the piece. The background is a solid dark blue, and the overall theme is design and problem-solving.

Solution



Problématique

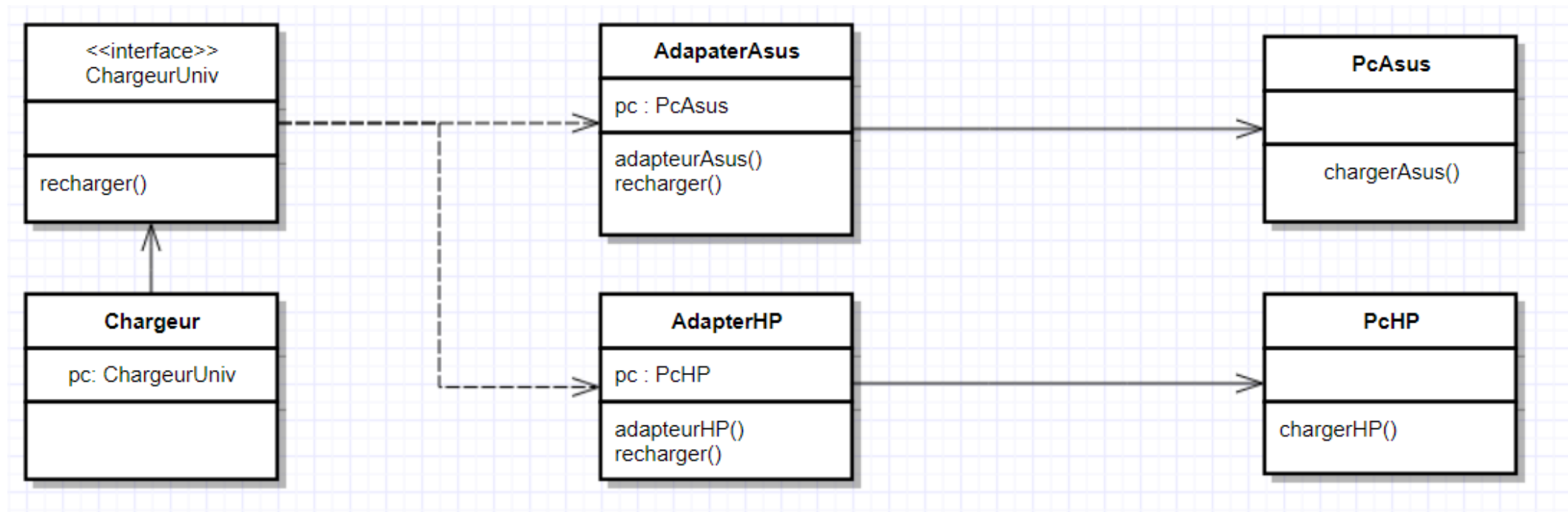
Faire fonctionner des objets incompatibles ensemble

Entreprise de production d'alimentations en tout genre

Nouveau marché : chargeurs PC

Chaque marque a son type de chargeur, comment faire ?

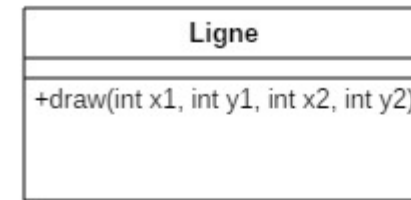
Représentation UML



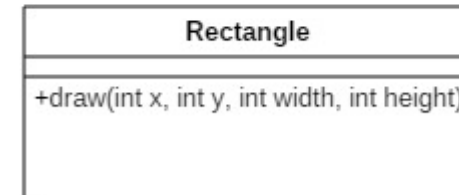
Exemple

II - Exemple

```
2 public class Ligne {  
3     public void draw(int x1, int y1, int x2, int y2) {  
4         System.out.println("Ligne du Point A(" +  
5             x1 + ";" + y1 + "), au point B(" +  
6             x2 + ";" + y2 + ")");  
7     }  
8 }  
9  
10
```

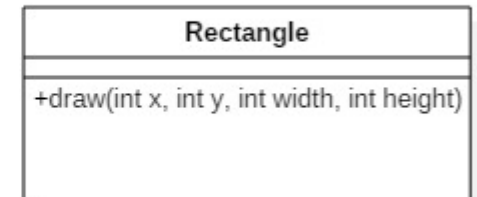
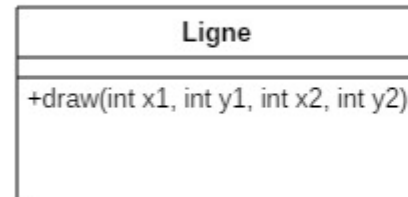
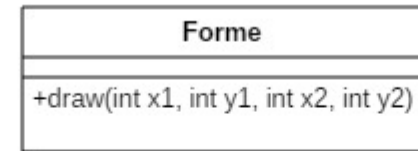


```
1  
2 public class Rectangle {  
3     public void draw(int x, int y, int width, int height) {  
4         System.out.println("Rectangle avec pour coordonnées (" +  
5             x + ";" + y + "), largeur: " + width +  
6             ", hauteur: " + height);  
7     }  
8 }  
9  
10
```



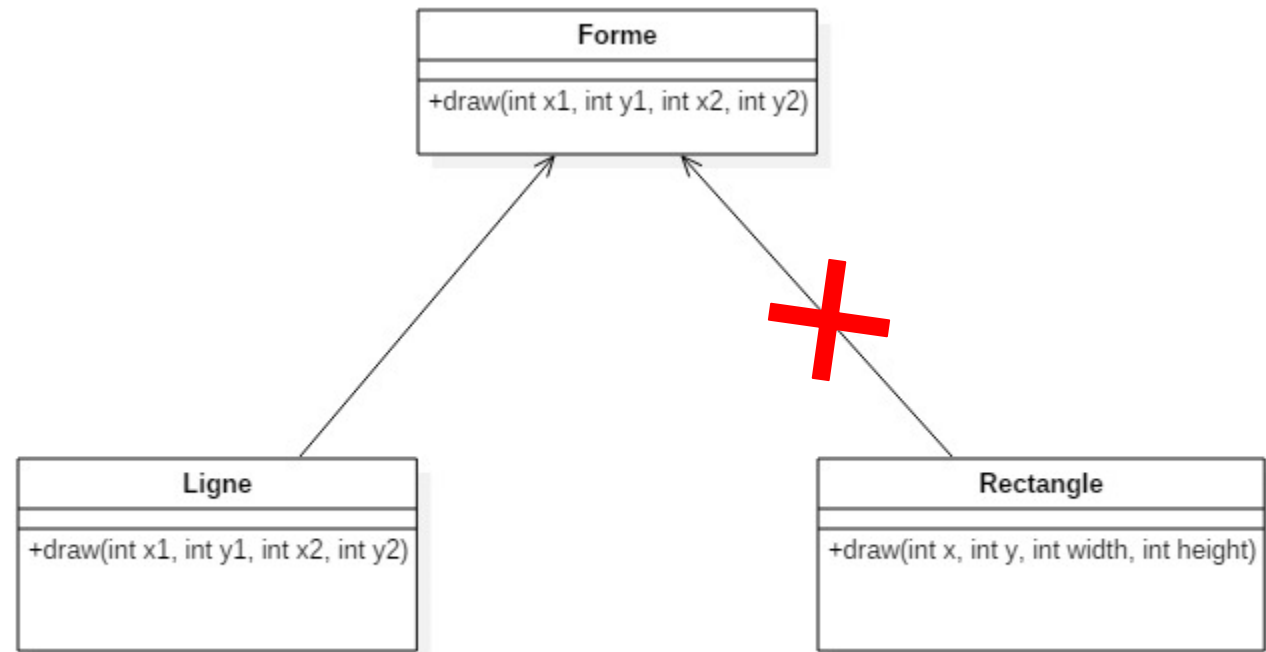
II - Exemple

```
1  
2 public interface Forme {  
3     void draw(int x, int y, int z, int j);  
4 }  
5
```

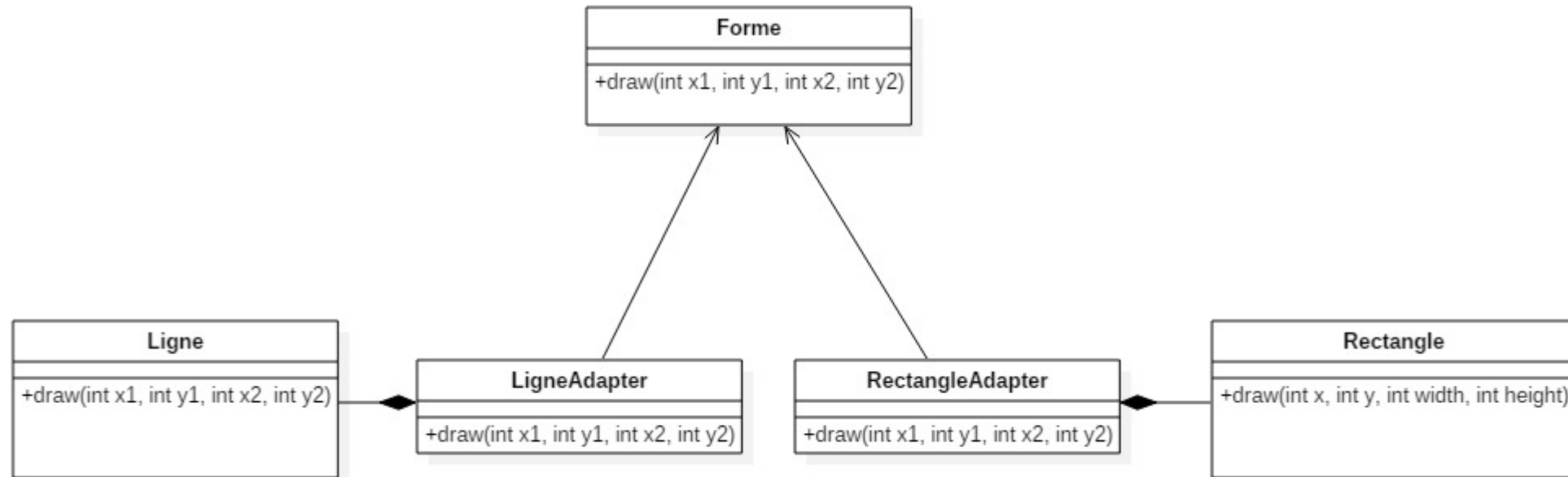


II - Exemple

```
1  
2 public interface Forme {  
3     void draw(int x, int y, int z, int j);  
4 }  
5
```



II - Exemple



II - Exemple

```
2 public class RectangleAdapter implements Forme{
3
4     private Rectangle adapte;
5
6     public RectangleAdapter(Rectangle rectangle) {
7         this.adapte = rectangle;
8     }
9
10    @Override
11    public void draw(int x1, int y1, int x2, int y2) {
12        int x = Math.min(x1, x2);
13        int y = Math.min(y1, y2);
14        int width = Math.abs(x2 - x1);
15        int height = Math.abs(y2 - y1);
16        adapte.draw(x, y, width, height);
17    }
18 }
19
```

```
1
2 public class LigneAdapter implements Forme{
3
4     private Ligne adapte;
5
6     public LigneAdapter(Ligne line) {
7         this.adapte = line;
8     }
9
10    @Override
11    public void draw(int x1, int y1, int x2, int y2) {
12        adapte.draw(x1, y1, x2, y2);
13    }
14 }
15
```

Conclusion

Faire communiquer deux classes sans les modifier

Faire collaborer de classes alors que leurs interfaces sont incompatibles

Quand l'utiliser ?



Sources

<http://badger.developpez.com/tutoriels/dotnet/patterns/adaptateur/>

[https://fr.wikipedia.org/wiki/Adaptateur \(patron de conception\)](https://fr.wikipedia.org/wiki/Adaptateur_(patron_de_conception))

https://sourcemaking.com/design_patterns/adapter