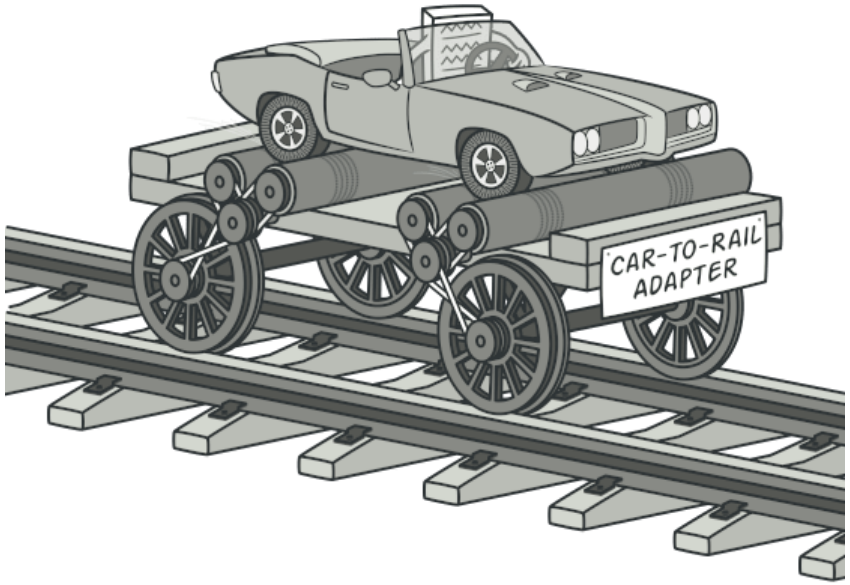
Adaptor Pattern

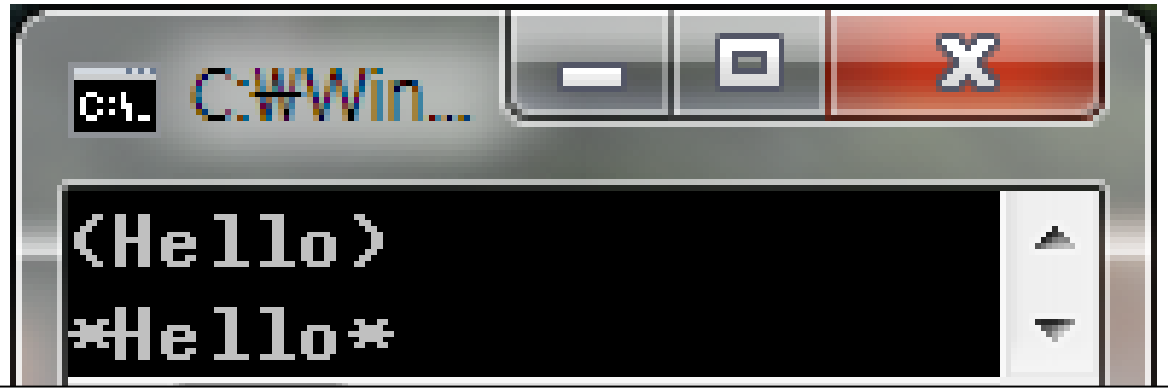
- A role of a bridge between two incompatible interface
- structural pattern : combining the capability of two independent interfaces.
- class adaptor and object adaptor

What is an adaptor?

https://refactoring.guru/design-patterns/adapter

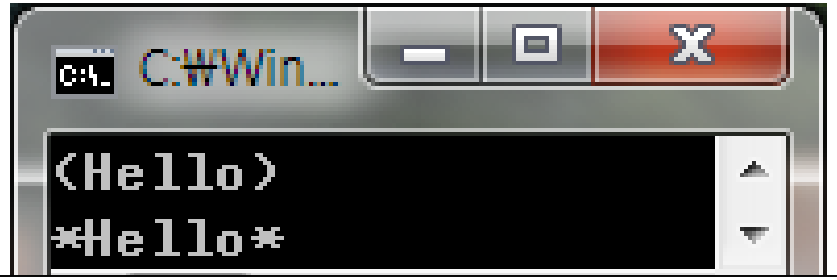


Example 1

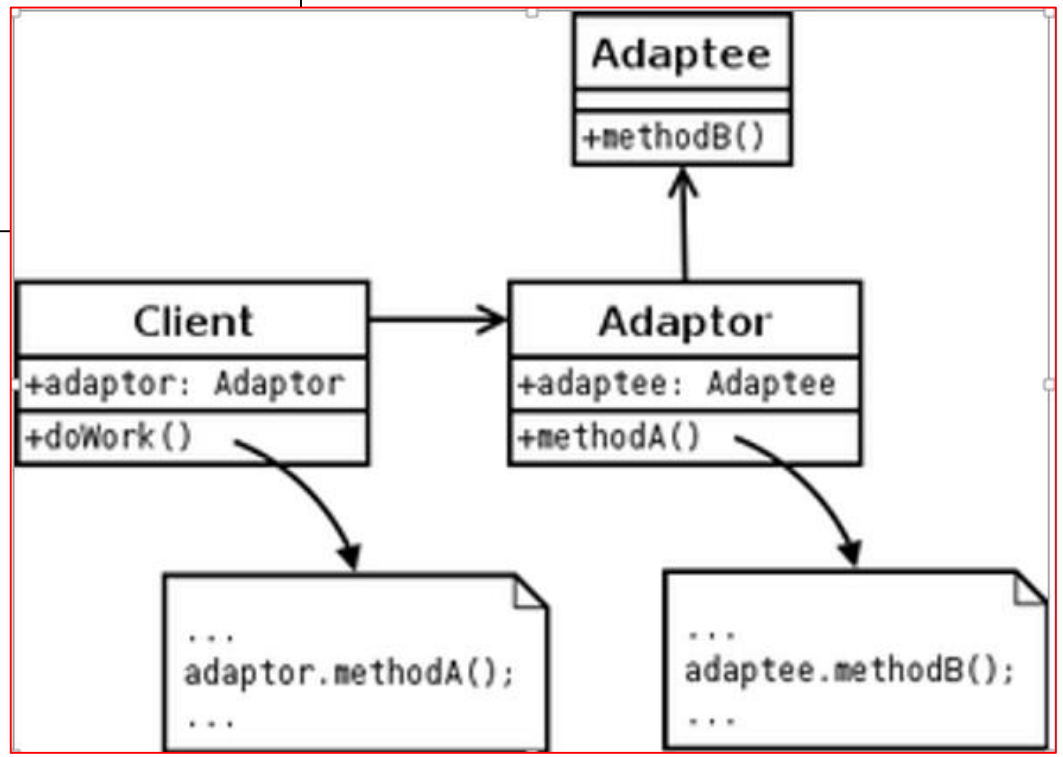



```
public class Main {
    public static void main(String[] args) {
        Print p = new PrintBanner("Hello");
        p.printWeak();
        p.printStrong();
    }
}
```

Suppose the client want to use her/his own methods, printWeak and printStrong..




```
public class Main {
    public static void main(String[] args) {
        Print p = new PrintBanner("Hello");
        p.printWeak();
        p.printStrong();
    }
}
```

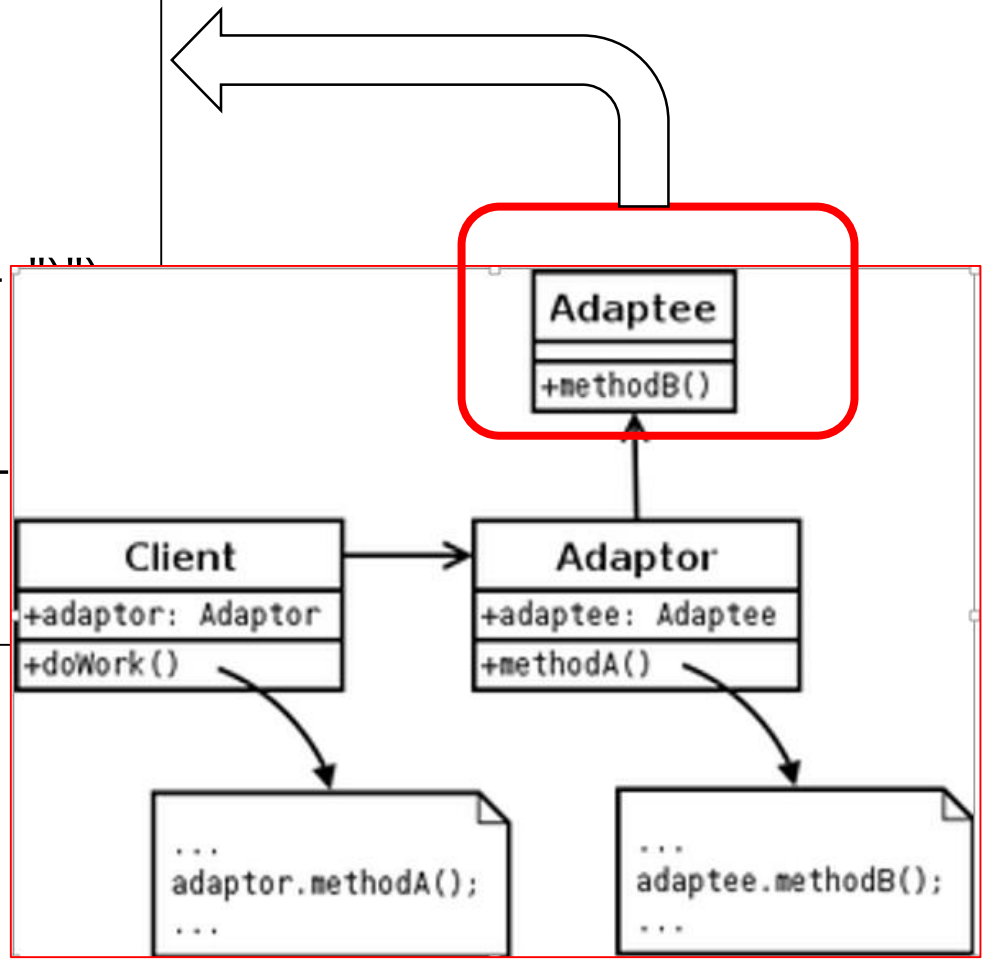


```java
public interface Print {
    public abstract void printWeak();
    public abstract void printStrong();
}
```

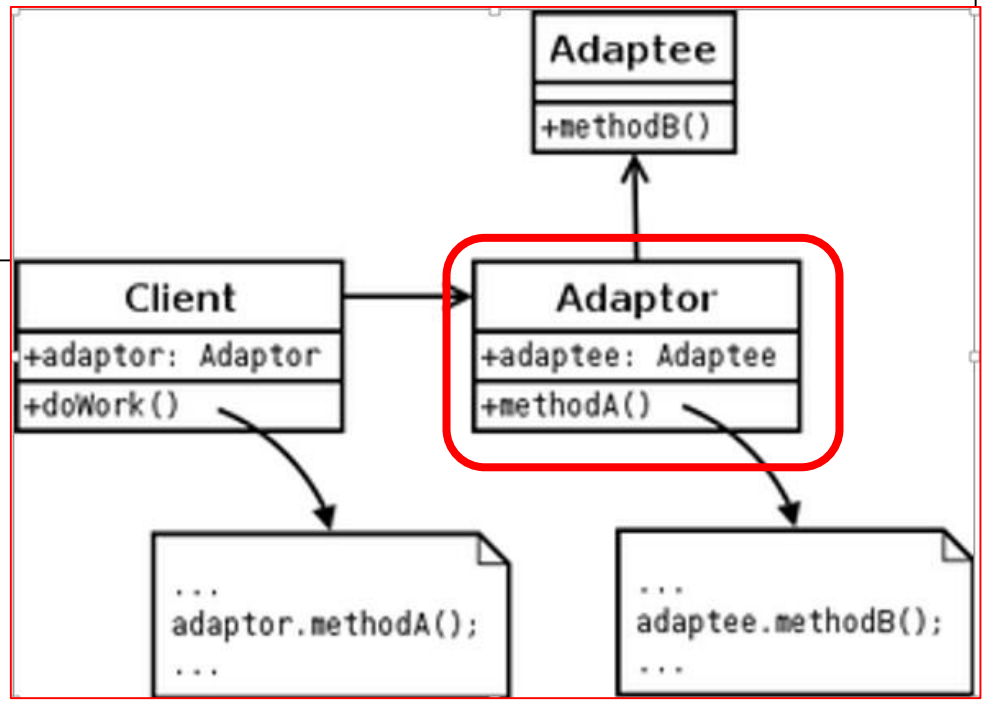
```java
public class Banner {
    private String string;
    public Banner(String string) {
        this.string = string;
    }
    public void showWithParen() {
        System.out.println("(" + string + ")");
    }
    public void showWithAster() {
        System.out.println("*" + string + "*");
    }
}
```

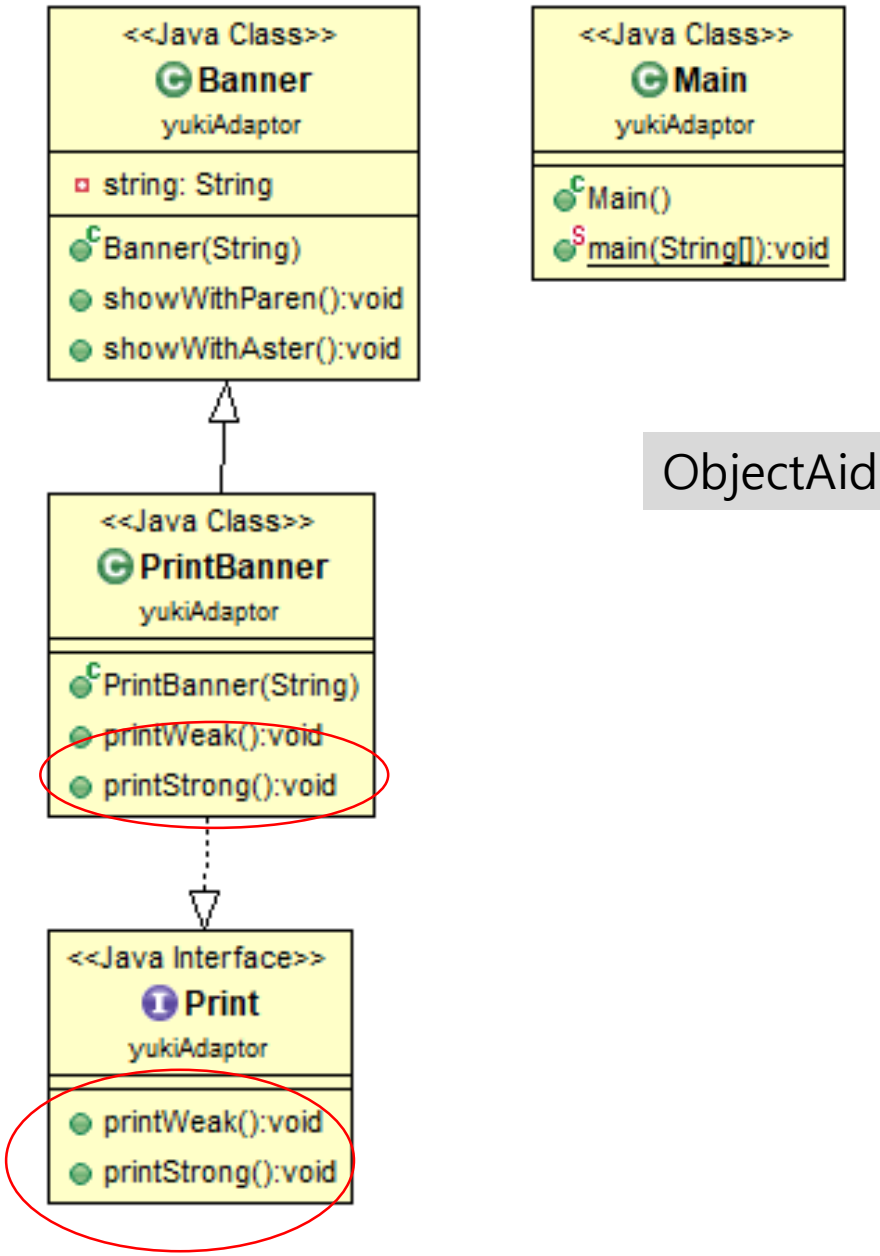
```
public interface Print {
    public abstract void printWeak();
    public abstract void printStrong();
}
```

```
public class Banner {
    private String string;
    public Banner(String string) {
        this.string = string;
    }
    public void showWithParen() {
        System.out.println("(" + string +
    }
    public void showWithAster() {
        System.out.println("*" + string +
    }
}
```



```
public class PrintBanner extends Banner implements Print {
    public PrintBanner(String string) {
        super(string);
    }
    public void printWeak() {
        showWithParen();
    }
    public void printStrong() {
        showWithAster();
    }
}
```





ObjectAid Not Available Anymore

Example 2

```
class LegacyLine {
    public void draw(int x1, int y1, int x2, int y2) {
        System.out.println("line from (" + x1 + ',' + y1 + ") to
            (" + x2 + ',' + y2 + ')');
    }
}

class LegacyRectangle {
    public void draw(int x, int y, int w, int h) {
        System.out.println("rectangle at (" + x + ',' + y + ")
            with width " + w + " and height " + h);
    }
}
```

```java
public class AdapterDemo {
    public static void main(String[] args) {
        Object[] shapes = {
            new LegacyLine(), new LegacyRectangle()
        };
        // A begin and end point from a graphical editor
        int x1 = 10, y1 = 20;
        int x2 = 30, y2 = 60;

        for (Object obj : shapes) {
            if (LegacyLine.class.isInstance(obj)) {
                LegacyLine.class.cast(obj).draw(x1, y1, x2, y2);
            } else if (LegacyRectangle.class.isInstance(obj)) {
              LegacyRectangle.class.cast(obj).draw(Math.min(x1, x2),
             Math.min(y1, y2), Math.abs(x2 - x1), Math.abs(y2 - y1));
            }
        }
    }
}
```

```
line from (10,20) to (30,60)
rectangle at (10,20) with width 20 and height 40
```

Example 3

```
interface Shape
{
  void draw(int x1, int y1, int x2, int y2);
}
class Line implements Shape //the role of adaptor
{
    private LegacyLine adaptee = new LegacyLine();
    public void draw(int x1, int y1, int x2, int y2)
    {        adaptee.draw(x1, y1, x2, y2);    }
}
class Rectangle implements Shape //the role of adaptor
{
    private LegacyRectangle adaptee = new LegacyRectangle();
    public void draw(int x1, int y1, int x2, int y2)
    {        adaptee.draw(Math.min(x1, x2), Math.min(y1, y2),
             Math.abs(x2 - x1), Math.abs(y2 - y1));    }
}
```

```
public class AdapterDemo2
{
    public static void main(String[] args)
    {
        Shape[] shapes =
        {
            new Line(), new Rectangle()
        };
        // A begin and end point from a graphical editor
        int x1 = 10, y1 = 20;
        int x2 = 30, y2 = 60;
        for (Shape shape : shapes)
          shape.draw(x1, y1, x2, y2);
    }
}
```

Example 4

```
class Main
{
    public static void main(String args[])
    {
        Sparrow sparrow = new Sparrow();
        ToyDuck toyDuck = new PlasticToyDuck();
        ToyDuck birdAdapter = new BirdAdapter(sparrow);
        System.out.println("Sparrow...");
        sparrow.fly();
        sparrow.makeSound();
        System.out.println("ToyDuck...");
        toyDuck.squeak();
        System.out.println("BirdAdapter...");
        birdAdapter.squeak();
    }
}
```

```
Sparrow...
Flying
Chirp Chirp
ToyDuck...
Squeak
BirdAdapter...
Chirp Chirp
```

```java
class Sparrow implements Bird
{
    public void fly()
    {
        System.out.println("Flying");
    }
    public void makeSound()
    {
        System.out.println("Chirp Chirp");
    }
}
```

```java
interface Bird
{
    public void fly();
    public void makeSound();
}
```

```
class PlasticToyDuck implements ToyDuck
{
    public void squeak()
    {
        System.out.println("Squeak");
    }
}
```

```
interface ToyDuck
{
    public void squeak();
}
```

```
class BirdAdapter implements ToyDuck
{
    Bird bird;
    public BirdAdapter(Bird bird)
    {
        this.bird = bird;
    }
    public void squeak()
    {
        bird.makeSound();
    }
}
```

```
interface ToyDuck
{
    public void squeak();
}
```

Difference between PrintBanner and Shape

- Try to find the **structural difference** between PrintBanner example and Shape example.
- There are two kinds of adaptor pattern, a **class** adaptor and an **object** adaptor.