



[Home](#) / [Design Patterns](#) / [Structural patterns](#) / [Adapter](#)

Adapter in *Java*: *Before and after*

[← Back to Adapter description](#)

Before

Because the interface between Line and Rectangle objects is incompatible, the user has to recover the type of each shape and manually supply the correct arguments.

```
class Line {
    public void draw(int x1, int y1, int x2, int y2) {
        System.out.println("Line from point A(" + x1 + ";" + y1 + "), to point B(" + x2 + ";" + y2 + ")");
    }
}

class Rectangle {
    public void draw(int x, int y, int width, int height) {
        System.out.println("Rectangle with coordinate left-down point (" + x + ";" + y + "), width: " + width + ", height: " + height);
    }
}

public class AdapterDemo {
    public static void main(String[] args) {
        Object[] shapes = {new Line(), new Rectangle()};
        int x1 = 10, y1 = 20;
        int x2 = 30, y2 = 60;
        int width = 40, height = 40;
        for (Object shape : shapes) {
            if (shape.getClass().getSimpleName().equals("Line")) {
                ((Line)shape).draw(x1, y1, x2, y2);
            } else if (shape.getClass().getSimpleName().equals("Rectangle")) {
                ((Rectangle)shape).draw(x2, y2, width, height);
            }
        }
    }
}
```

Output

```
Line from point A(10;20), to point B(30;60)
Rectangle with coordinate left-down point (30;60), width: 40, height: 40
```

After

The Adapter's "extra level of indirection" takes care of mapping a user-friendly common interface to legacy-specific peculiar interfaces.

```
interface Shape {
    void draw(int x, int y, int z, int j);
}

class Line {
    public void draw(int x1, int y1, int x2, int y2) {
        System.out.println("Line from point A(" + x1 + ";" + y1 + "), to point B(" + x2 + ";" + y2 +
    }
}

class Rectangle {
    public void draw(int x, int y, int width, int height) {
        System.out.println("Rectangle with coordinate left-down point (" + x + ";" + y + "), width:
        + ", height: " + height);
    }
}

class LineAdapter implements Shape {
    private Line adaptee;

    public LineAdapter(Line line) {
        this.adaptee = line;
    }

    @Override
    public void draw(int x1, int y1, int x2, int y2) {
        adaptee.draw(x1, y1, x2, y2);
    }
}

class RectangleAdapter implements Shape {
    private Rectangle adaptee;

    public RectangleAdapter(Rectangle rectangle) {
        this.adaptee = rectangle;
    }

    @Override
    public void draw(int x1, int y1, int x2, int y2) {
        int x = Math.min(x1, x2);
        int y = Math.min(y1, y2);
        int width = Math.abs(x2 - x1);
        int height = Math.abs(y2 - y1);
        adaptee.draw(x, y, width, height);
    }
}

public class AdapterDemo {
```

```

public static void main(String[] args) {
    Shape[] shapes = {new RectangleAdapter(new Rectangle()),
                      new LineAdapter(new Line())};

    int x1 = 10, y1 = 20;
    int x2 = 30, y2 = 60;
    for (Shape shape : shapes) {
        shape.draw(x1, y1, x2, y2);
    }
}

```

Output

Rectangle with coordinate left-down point (10;20), width: 20, height: 40
 Line from point A(10;20), to point B(30;60)

Support our free website and own the eBook!

- 22 design patterns and 8 principles explained in depth
- 406 well-structured, easy to read, jargon-free pages
- 228 clear and helpful illustrations and diagrams
- An archive with code examples in 4 languages
- All devices supported: EPUB/MOBI/PDF formats

 [Learn more...](#)



Code examples

Java	Adapter in Java: Before and after	Adapter in Java
C++	Adapter in C++	Adapter in C++: External Polymorphism

PHP	Adapter in PHP
Delphi	Adapter in Delphi
Python	Adapter in Python

▲ More info, diagrams and examples of the [Adapter design pattern](#) you can find on our page.

[Design Patterns](#)

[AntiPatterns](#)

[Refactoring](#)

[UML](#)

[My account](#)

[Forum](#)

[Contact us](#)

[About us](#)

© 2007-2019 SourceMaking.com
All rights reserved.

[Terms / Privacy policy](#)