Adapter Pattern

Structural pattern

Problem statement:

An "off the shelf" component offers compelling functionality that you would like to reuse, but its "view of the world" is not compatible with the philosophy and architecture of the system currently being developed.

Pattern Idea:

The adapter pattern convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn’t otherwise because of incompatible interfaces.

The adapter design pattern Use for problems like:

How can a class be reused that does not have an interface that a client requires?

How can classes that have incompatible interfaces work together?

How can an alternative interface be provided for a class?

Adapter design pattern **Advantages:**

* Helps achieve reusability and flexibility.
* Client class is not complicated by having to use a different interface and can use

Drawbacks:

* All requests are forwarded, so there is a slight increase in the overhead.
* Sometimes many adaptations are required along an adapter chain to reach the type which is required.

Code Implementation:

**Before**

**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);

}

}

}

}

**After**

**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: " + 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);

}

}

}