**Adapter Pattern**

Match interfaces of different classes

**Problem**

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**

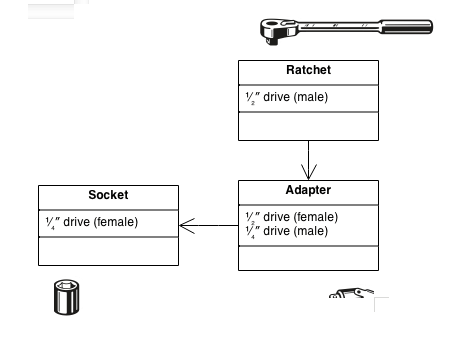
* Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.
* Reuse has always been painful and elusive. One reason has been the tribulation of designing something new, while reusing something old. There is always something not quite right between the old and the new. It may be physical dimensions or misalignment. It may be timing or synchronization. It may be unfortunate assumptions or competing standards.

**Example**



The Adapter pattern allows otherwise incompatible classes to work together by converting the interface of one class into an interface expected by the clients. Socket wrenches provide an example of the Adapter. A socket attaches to a ratchet, provided that the size of the drive is the same. Typical drive sizes in the United States are 1/2" and 1/4". Obviously, a 1/2" drive ratchet will not fit into a 1/4" drive socket unless an adapter is used. A 1/2" to 1/4" adapter has a 1/2" female connection to fit on the 1/2" drive ratchet, and a 1/4" male connection to fit in the 1/4" drive socket.

**Structure**



.

**Usage**

* When we need to reuse old components or 3rd part libraries
* When we need to Wrap an existing class with a new interface.
* Impedance match an old component to a new system

**Pattern implementation (Java)**

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package Adapter;

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);

}

}

/\*\*

\*

\* @author pcc

\*/

public class Adapter {

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);

}

}

}

Drawbacks

* If not Sometimes many adaptations are required along an adapter chain to reach the type which is required.