Joshua Isaacson

Homework 9: *Adapter Design Pattern*

Description

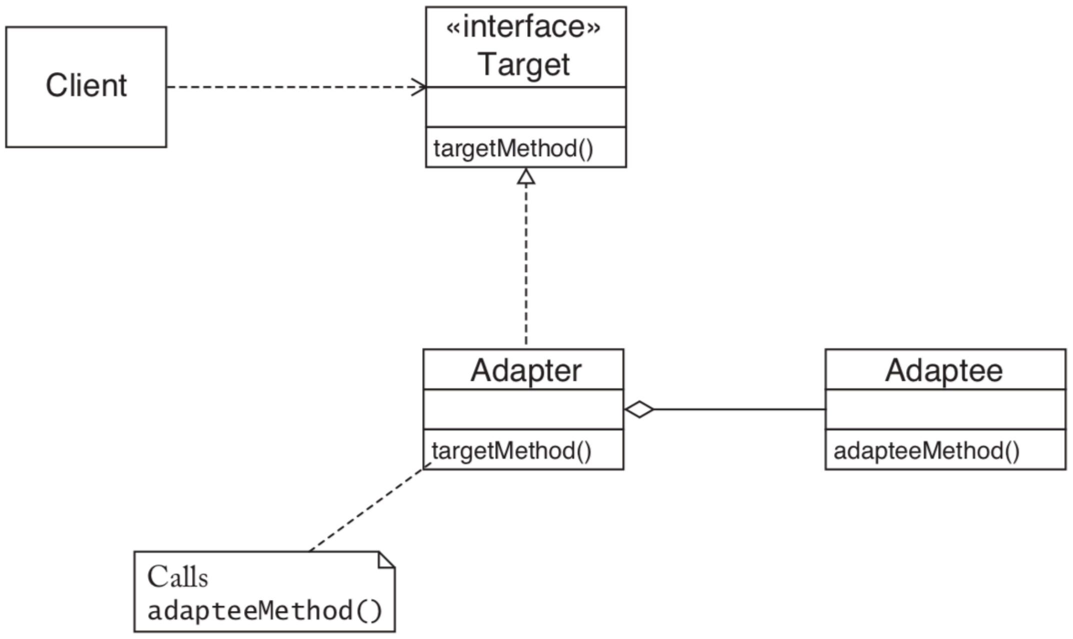
1. The adapter design pattern lets you change the interface of one class to fit the interface of what is desired. This solves the problem of incompatible interfaces. You would want to use the Adapter pattern when you have a class that has the proper functionality, but doesn't quite fit with the architecture of your system.

2. It is a great pattern because it lets the user reuse existing code rather than have to spend time implementing it from scratch just to make changes to something small and also allows interaction without changes to the original code.

3. The limitations of the adapter design pattern are that a lot of code is duplicated between classes and sometimes many adaptions are necessary along an adapter chain to get the required type. Overusing this pattern can result in bloated codebases and can make refactoring more complicated.

Solution

The adapter class implement the target interface. The adapter class holds a reference to the adaptee. It translates target methods to adaptee methods. The client wraps the adaptee into an adapter class object.



Example

Here is an example of the adapter design pattern. There is a ratchet with a particular drive size and a socket that has a drive size that is half the size of the ratchet's drive. The adapter class takes the ratchet object's drive and adjusts it to fit that of the socket object's drive.

Ratchet

```
public class Ratchet {
        private double driveRatchetSize;
        public Ratchet(double drive) { this.driveRatchetSize = drive; }
        public double getDrive() { return driveRatchetSize; }
        public void setDrive(double drive) {
                this.driveRatchetSize = drive;
        }
}
```

Socket

```
public class Socket {
        private double driveSocketSize;
        public Socket(double drive) { this.driveSocketSize = drive; }
        public double getDrive() { return driveSocketSize; }
}
```

Adapter

```
public class Adapter {
        public static void makeFit(Ratchet ratchet, Socket socket) {
                double adaptedDrive = ratchet.getDrive() / 2;
                ratchet.setDrive(adaptedDrive);
        }
}
```

Client

```
public class Client {
        public static void main(String[] args) {
                Ratchet ratchet = new Ratchet(5);
                System.out.println("Racket drive size: " +
                                        ratchet.getDrive());

                Socket socket = new Socket(2.5);
                System.out.println("Socket drive size: " +
                                        socket.getDrive());

                Adapter.makeFit(ratchet, socket);
                System.out.println("Racket has an adapter of size: " +
                                        ratchet.getDrive());
        }
}
```