Adapter Pattern

Michal Moravik, SD20w2

Name: Adapter Pattern

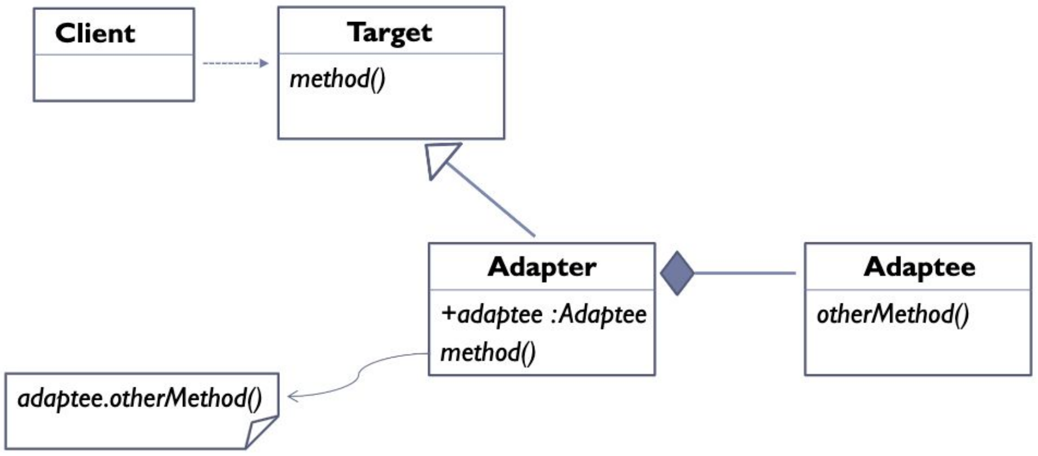
Category: Structural

Intent: Adapter allows objects with different interfaces to collaborate

Motivation:
We can end up in a situation when one component, for example, a legacy component has an incompatible type with the component we develop currently.

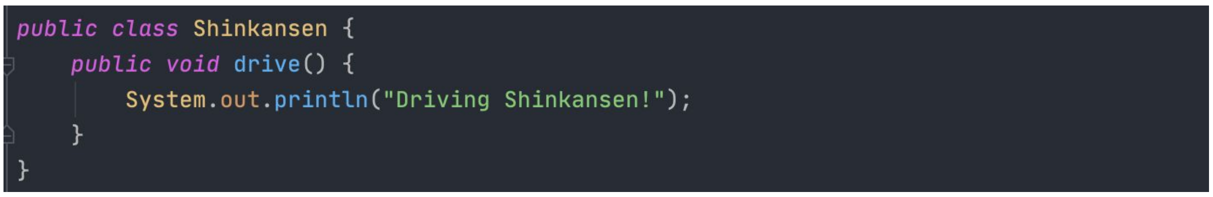
The adapter serves as a middleman. It gets an interface compatible with one of the existing objects. After that, we can call methods of that object. Then the adapter passes the request to the second object but in a format which the second object expects.

UML:

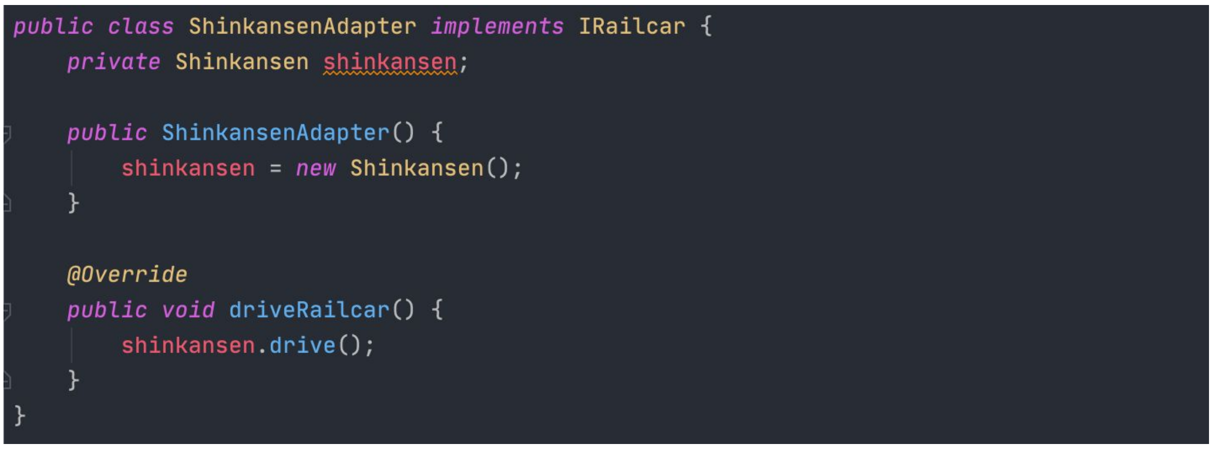


Implementation:

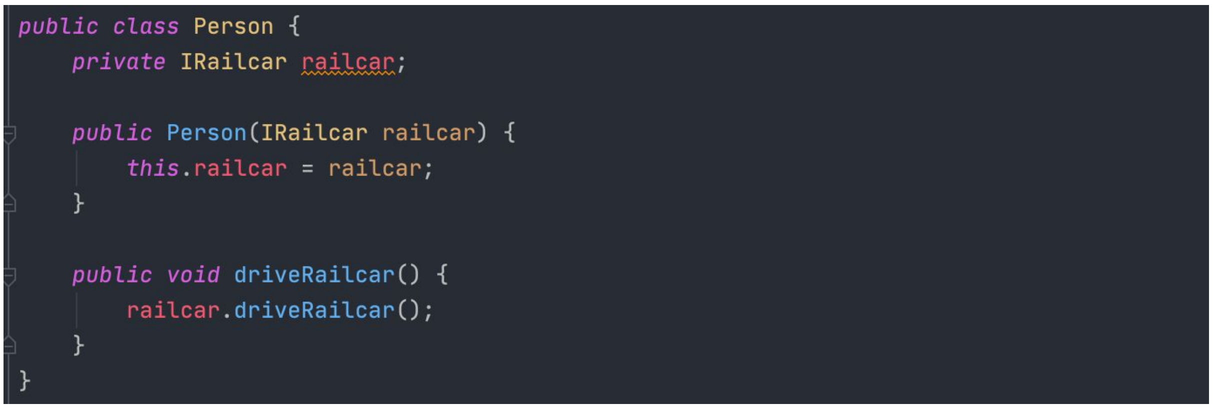
```
public interface IRailcar {
    void driveRailcar();
}
```



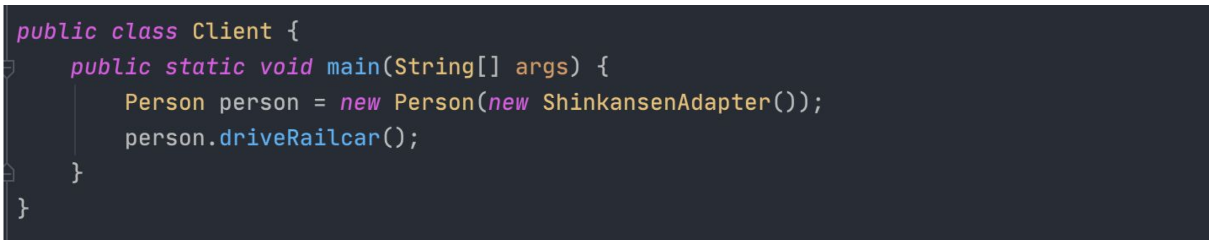
```java
public class Shinkansen {
    public void drive() {
        System.out.println("Driving Shinkansen!");
    }
}
```



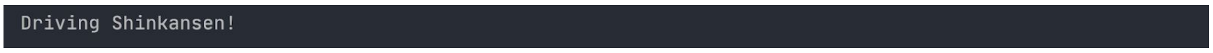
```java
public class ShinkansenAdapter implements IRailcar {
    private Shinkansen shinkansen;

    public ShinkansenAdapter() {
        shinkansen = new Shinkansen();
    }

    @Override
    public void driveRailcar() {
        shinkansen.drive();
    }
}
```



```java
public class Person {
    private IRailcar railcar;

    public Person(IRailcar railcar) {
        this.railcar = railcar;
    }

    public void driveRailcar() {
        railcar.driveRailcar();
    }
}
```



```java
public class Client {
    public static void main(String[] args) {
        Person person = new Person(new ShinkansenAdapter());
        person.driveRailcar();
    }
}
```



```
Driving Shinkansen!
```

There is a story. A person was trained to know how to ride a normal railcar. So the person calls method to drive a railcar. The colleague of the person showed him the other day how he is able to drive a shinkansen. But during their journey, the colleague just pass out. The person needs to take his seat and drive the shinkansen. But he does not know how to drive it. So we need to build for him an adapter. After using the adapter, the person was able to apply skills he has from driving a normal railcar and successfully managed to drive the shinkansen to the next stop. (PS: his colleague survived).

Consequences:
Pros:

- Allow collaboration between otherwise incompatible system
- The older components become reusable

Cons:

- After introducing a set of new interfaces and classes, the complexity increases a bit

Known uses:

- When we want to implement an older component but its interface is incompatible with the new system
- Java core libraries use Adapter

Related patterns:

- Facade
- Decorator
- Proxy
- Bridge