Wzorce strukturalne

- Opisują struktury powiązanych ze sobą obiektów

Wzorce strukturalne (ang *Structural patterns*)

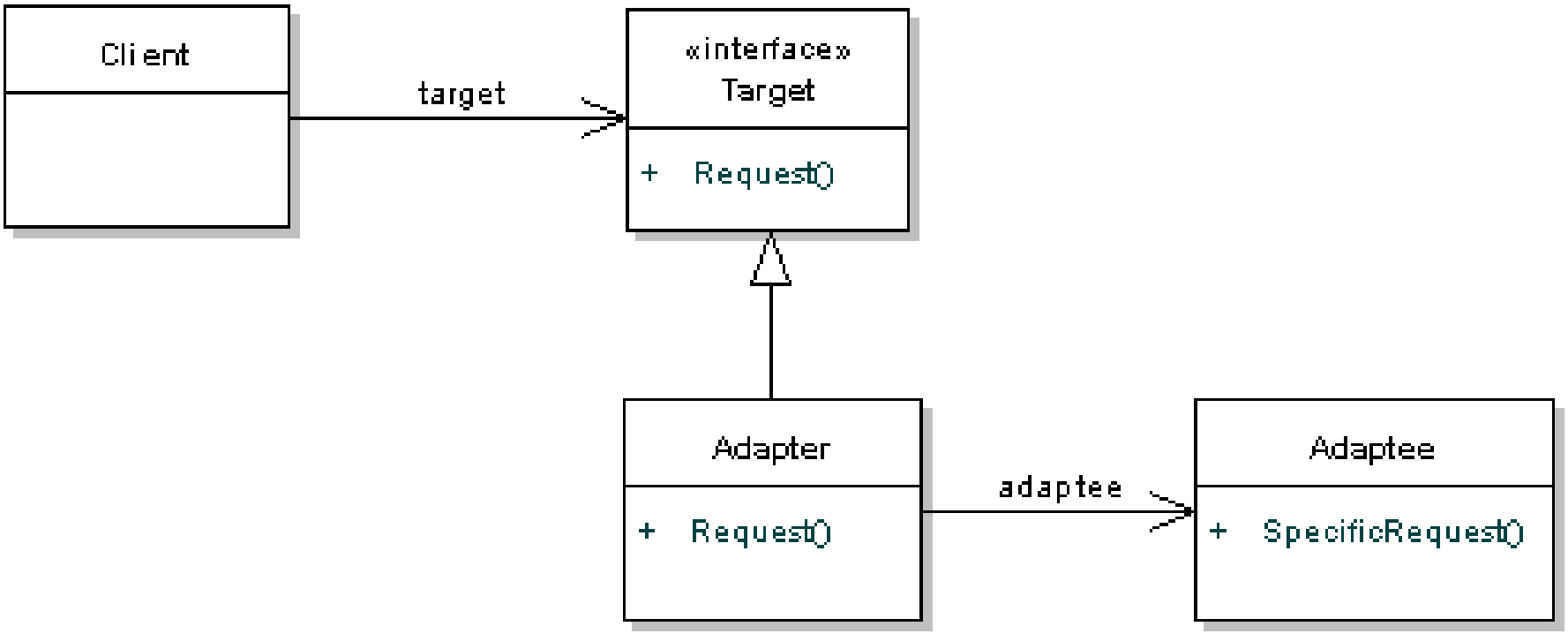
- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy

Adapter

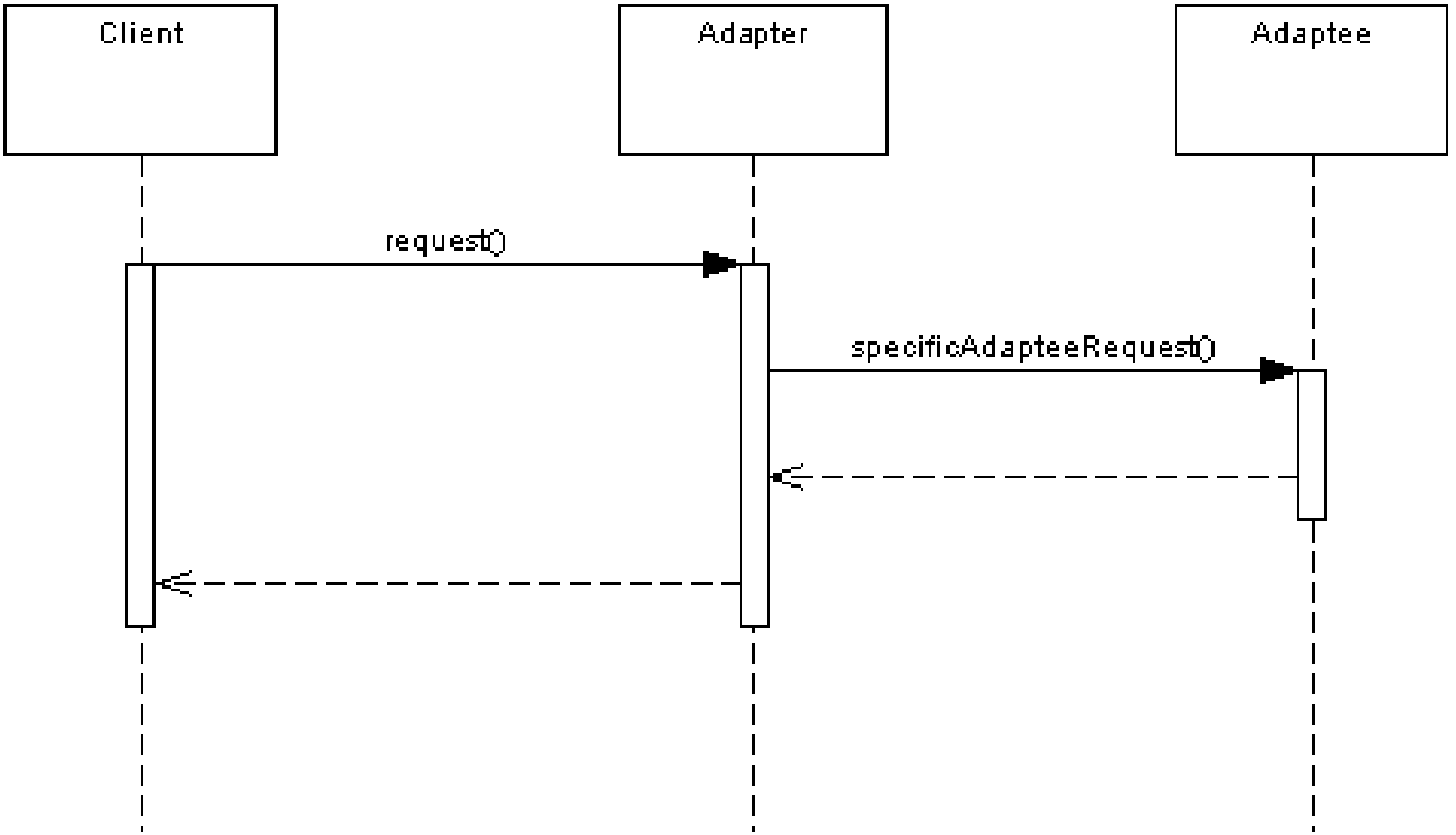
Adapter

- służy do przystosowania interfejsów obiektowych, tak aby możliwa była współpraca obiektów o niezgodnych interfejsach
- szczególnie przydaje się przypadku wykorzystania gotowych bibliotek o interfejsach niezgodnych ze stosowanymi w aplikacji

Adapter


Client
target
«interface»
Target
+ Request()
Adapter
+ Request()
adaptee
Adaptee
+ SpecificRequest()

Adapter



Adapter

```
public class GermanToUKPlugConnectorAdapterTest {
  @Test
  public void sholudUkElectricalSocketProvideElectricityToGermanPlug() {
    GermanPlugConnector plugConnector = new GermanPlugConnectorImpl();
    UKElectricalSocket electricalSocket = new UKElectricalSocket();
    UKPlugConnector ukAdapter = new GermanToUKPlugConnectorAdapter(plugConnector);
    electricalSocket.plugIn(ukAdapter);
  }
}
```

Zastosowanie

- w przypadku, gdy wykorzystanie istniejącej klasy jest niemożliwe ze względu na jej niekompatybilny interfejs
- przydatny z *legacy* kodem
- tworzenie klasy, która będzie współpracowała z innymi klasami o nieokreślonych interfejsach

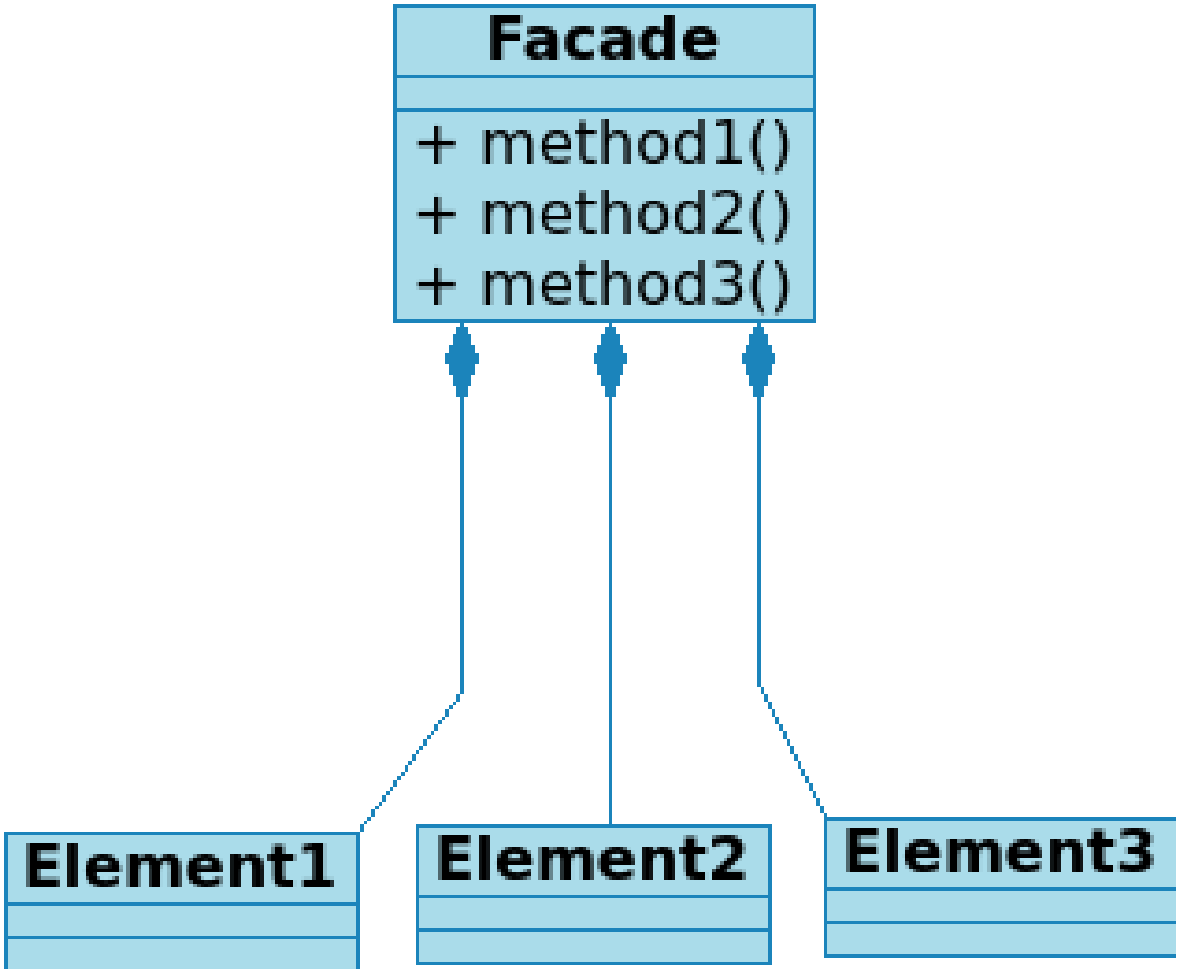
Adapter w JDK

- java.util.Arrays#asList()
- java.util.Collections#list()
- java.util.Collections#enumeration()
- java.io.InputStreamReader(InputStream) (returns a Reader)
- java.io.OutputStreamWriter(OutputStream) (returns a Writer)
- javax.xml.bind.annotation.adapters.XmlAdapter#marshal() and #unmarshal()

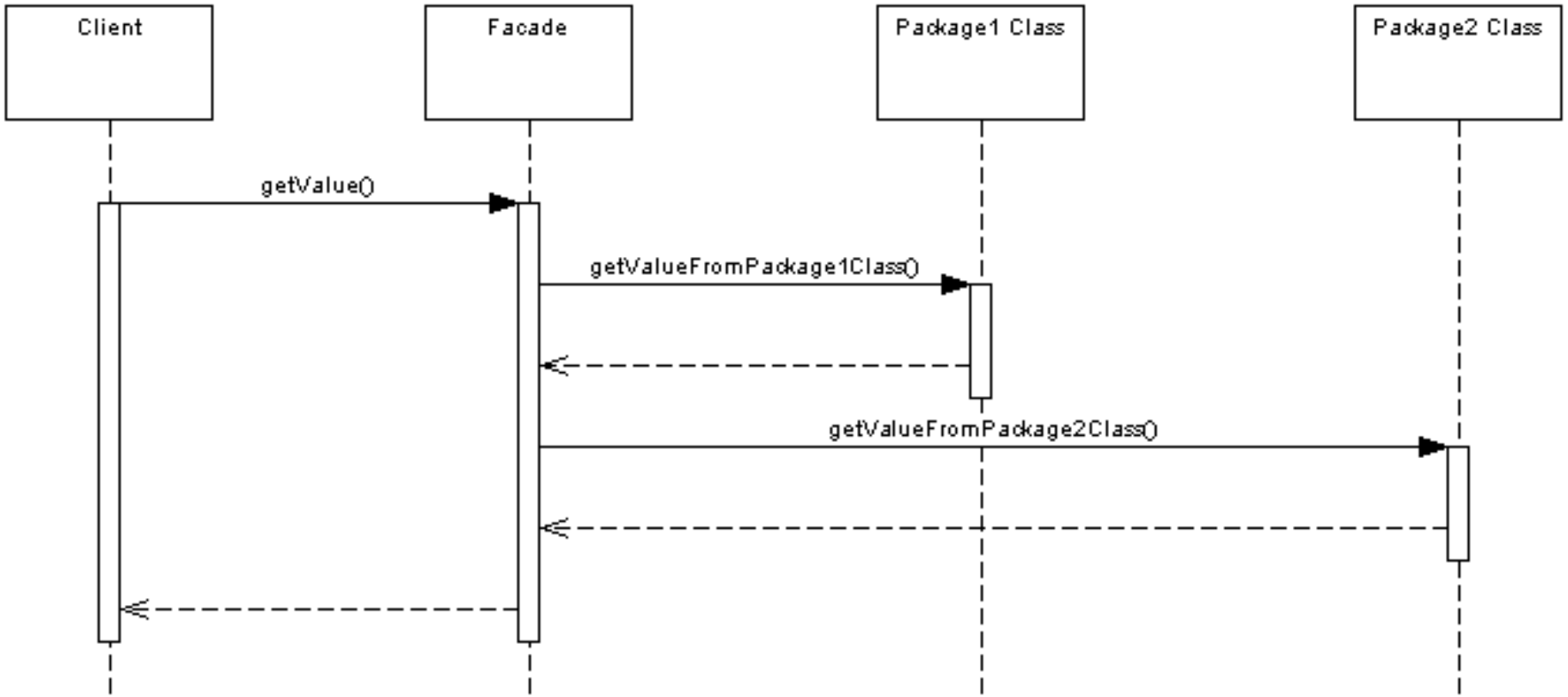
Facade

- służy do ujednolicenia dostępu do złożonego systemu poprzez udostępnienie uproszczonego i uporządkowanego interfejsu programistycznego.
- Fasada zwykle implementowana jest w bardzo prosty sposób – w postaci jednej klasy powiązanej z klasami reprezentującymi system, do którego klient chce uzyskać dostęp.

Facade


Facade
+ method1()
+ method2()
+ method3()
Element1
Element2
Element3

Facade


Client
Facade
Package1 Class
Package2 Class
getValue()
getValueFromPackage1Class()
getValueFromPackage2Class()

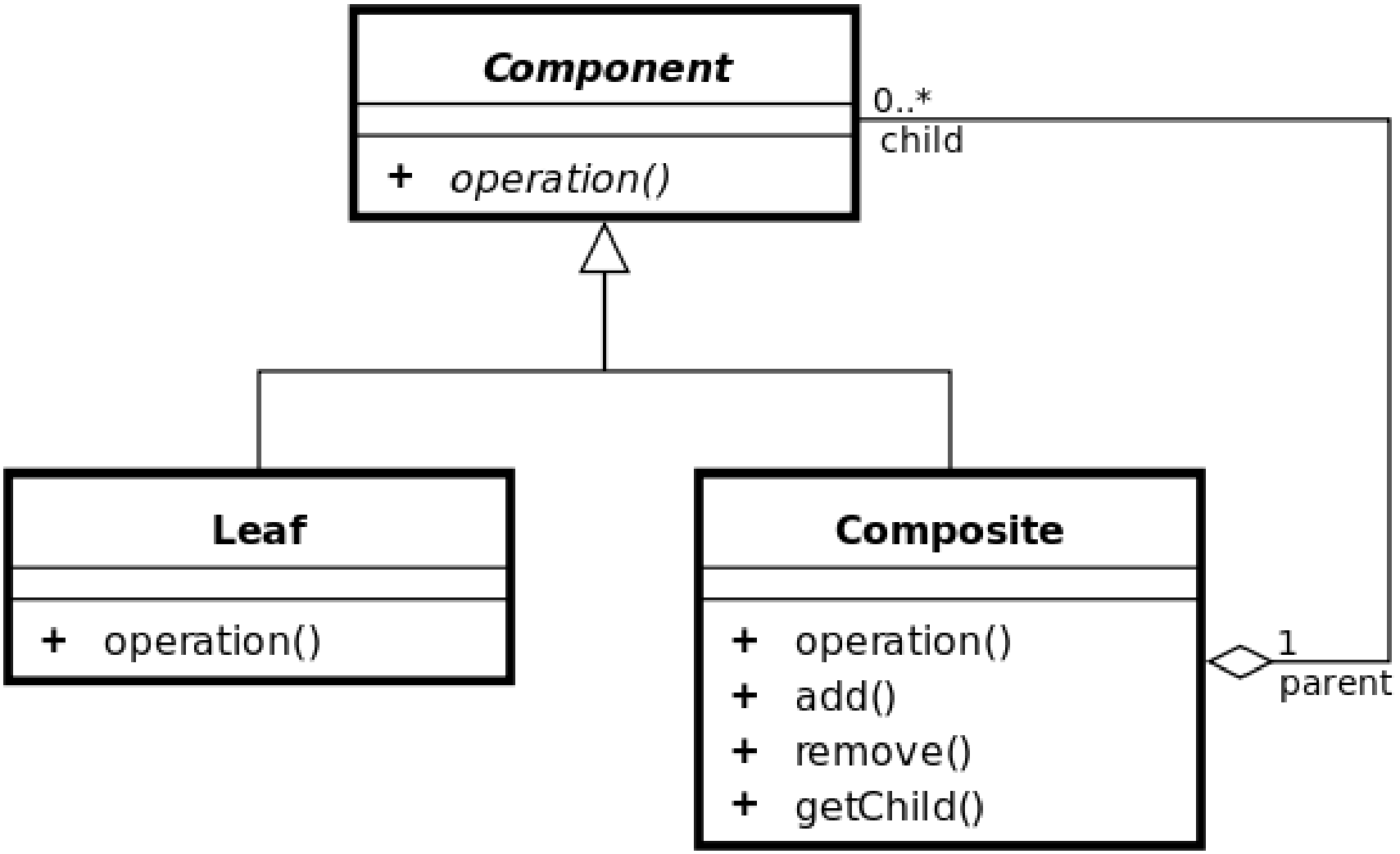
Composite

- umożliwia on tworzenie struktur drzewiastych, gdzie podstawową jednostką jest liść *ang.* ***Leaf*** a rozszerzoną kompozyt *ang.* ***Composite***

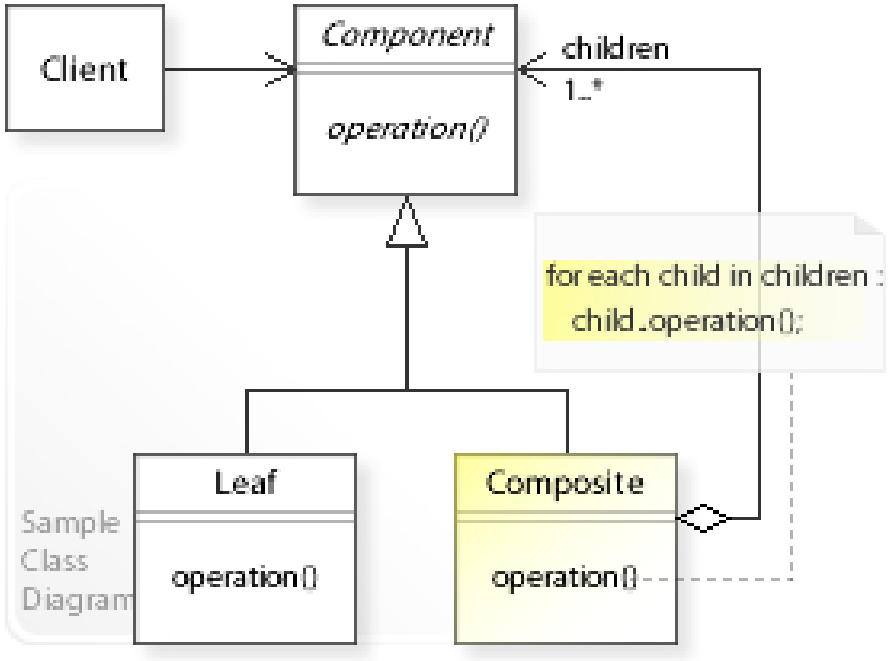
Composite

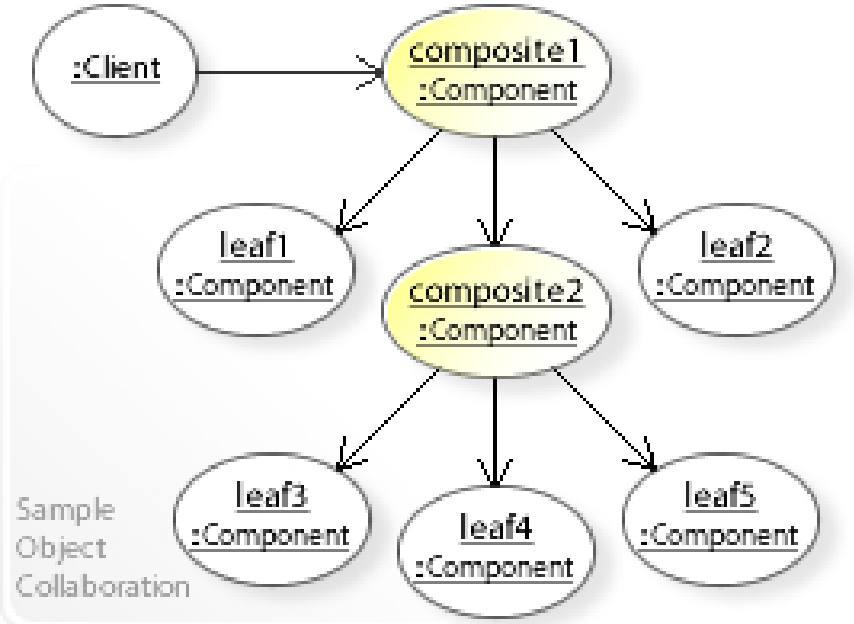
- Wzorzec składa się z:
  - Composite (Kompozyt) – grupa obiektów, które składają się z liści , implementuje interfejs Composite
  - Leaf (Liść) – pojedynczy obiekt który nie ma potomków, implementuje interfejs Composite
  - Komponent – interfejs który określa zachowania dla danego obiektu lub grupy obiektów
  - Klient – operuje na danych zawartych we wzorcu

Composite – diagram klas



Composite – diagram obiektów



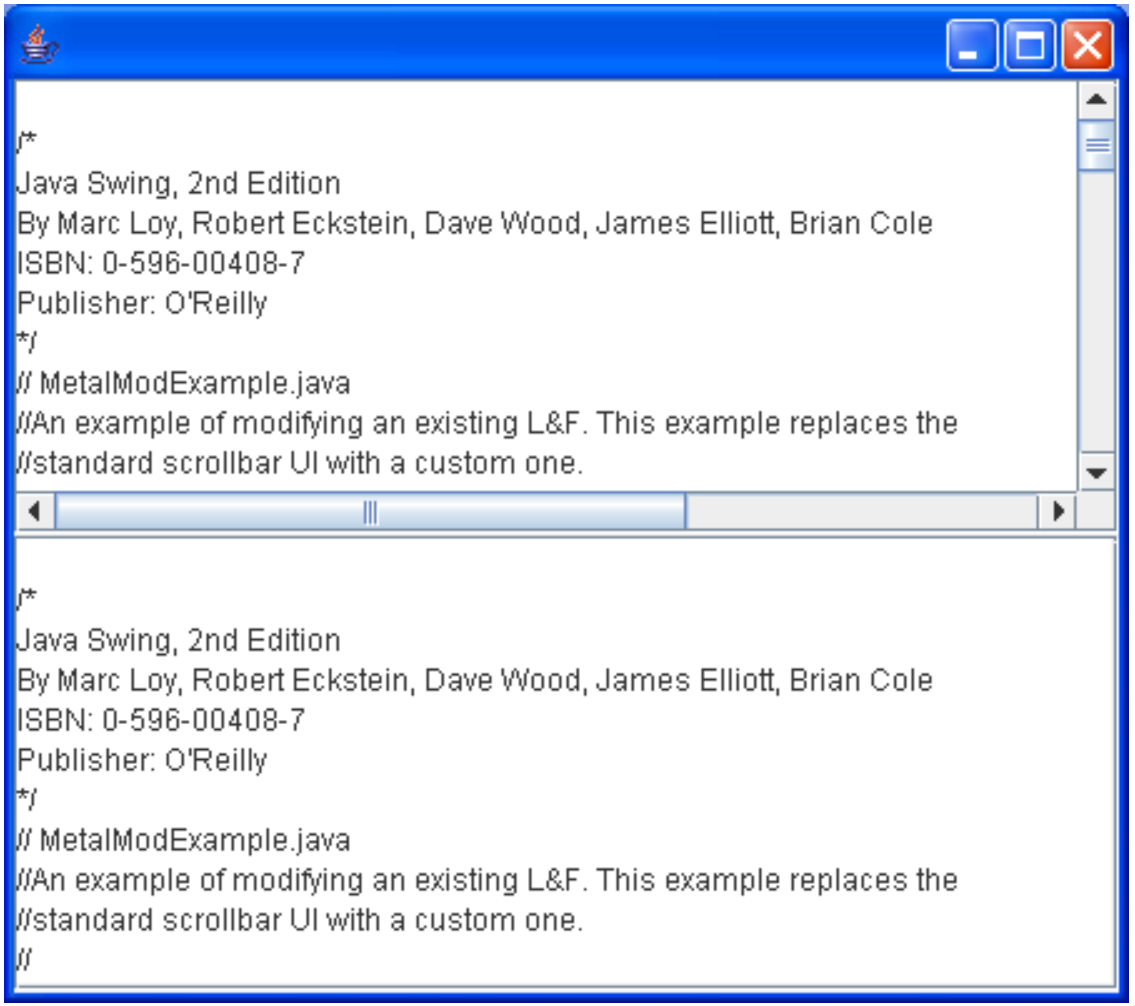


Composite – przykład kodu

- java.awt.Container#add(Component) (praktycznie wszykie klasy w Swing reprezentujące komponenty graficzne)
- Dostępny w teście *CompositeTest*

Decorator - przykład

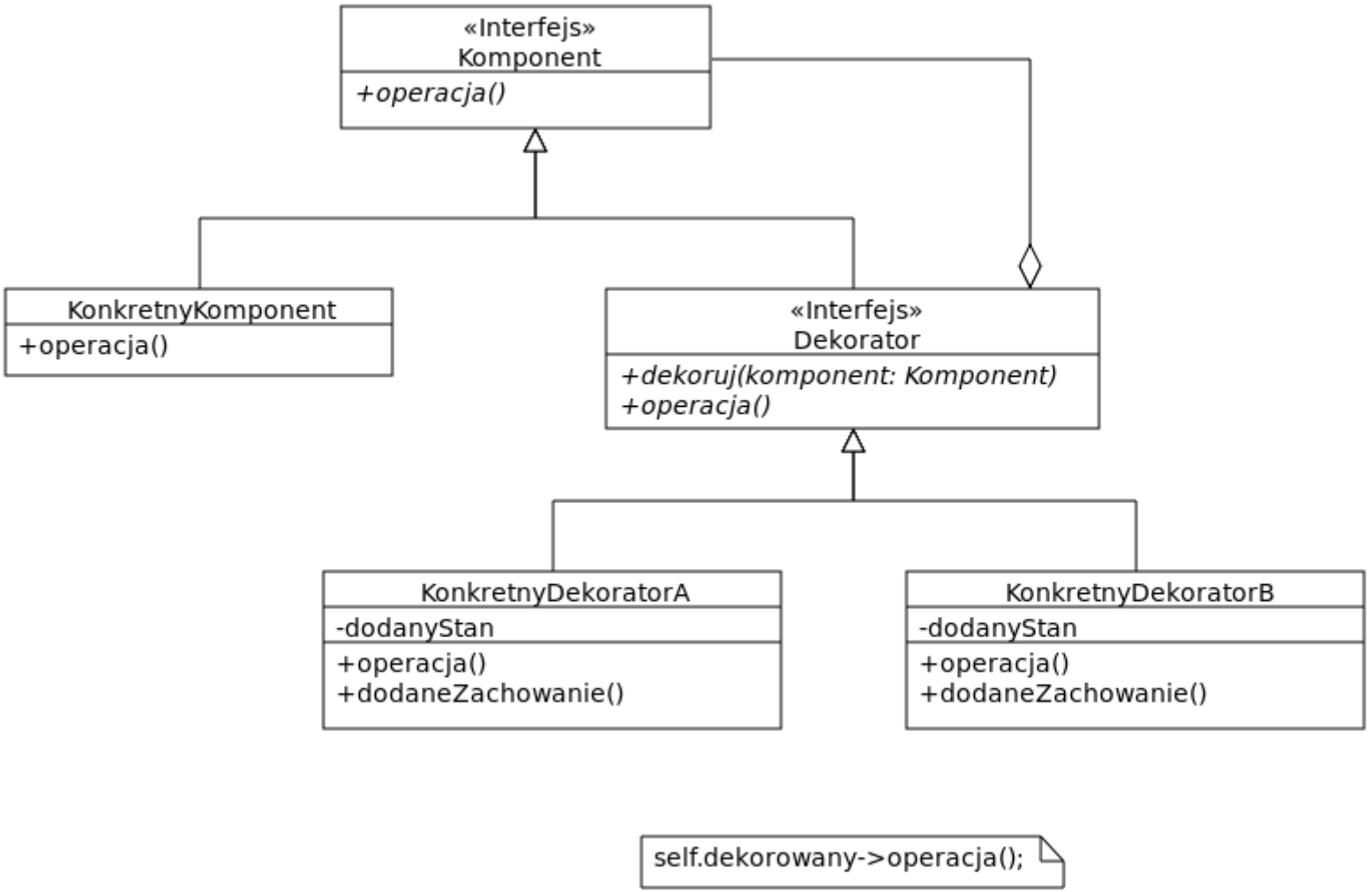
- Window i Scroll



Decorator

- Wzorzec projektowy Dekorator pozwala na dynamiczne przydzielanie danemu obiektowi nowych zachowań.
- Dekoratory dają elastyczność podobną do tej, jaką daje dziedziczenie, oferując jednak w zamian znacznie rozszerzoną funkcjonalność

Decorator – diagram klas



Decorator – konsekwencje stosowania

- Zapewnia większą elastyczność niż statyczne dziedziczenie
- Pozwala uniknąć tworzenia przeładowanych funkcjami klas na wysokich poziomach hierarchii
- Dekorator i powiązany z nim komponent nie są identyczne
- Powstawanie wielu małych obiektów

Decorator - przykład

```
@Test
public void bikeDecoratorTest() {
  System.out.println("Wzorzec projektowy Dekorator");
  Bike cityBike = new CityBike();
  System.out.println("Waga roweru miejskiego bez akcesoriów: " + String.format("%.4g",
cityBike.getWeight()));
  cityBike = new Carrier(cityBike);
  cityBike = new RearFender(cityBike);
  cityBike = new FrontFender(cityBike);
  cityBike = new BicycleBell(cityBike);
  System.out.println("Waga roweru miejskiego z akcesoriami: " + String.format("%.4g",
cityBike.getWeight()));
  Bike roadBike = new RoadBike();
  System.out.println("Waga roweru szosowego bez akcesoriów: " + String.format("%.4g",
roadBike.getWeight()));
  roadBike = new BottleCage(roadBike);
  System.out.println("Waga roweru szosowego z akcesoriami: " + String.format("%.4g",
roadBike.getWeight()));
}
```

Decorator przykłady

- FileReader i BuffredReader z JDK
- BikeDecoratorTest z ćwiczeń
- WindowDecoratorTest z ćwiczeń