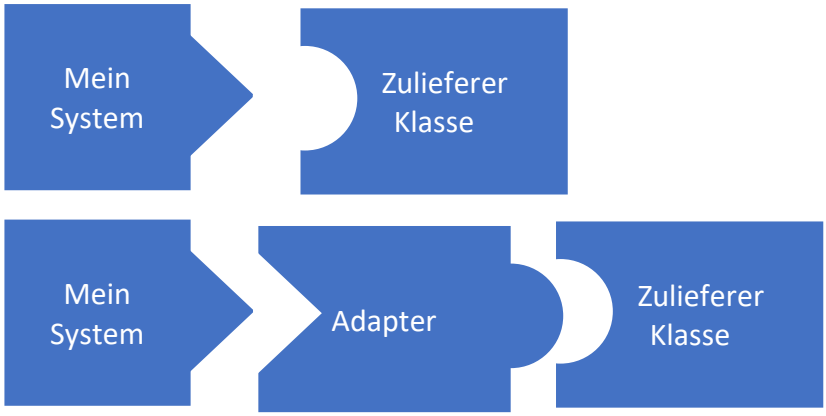
Adapter Design Pattern

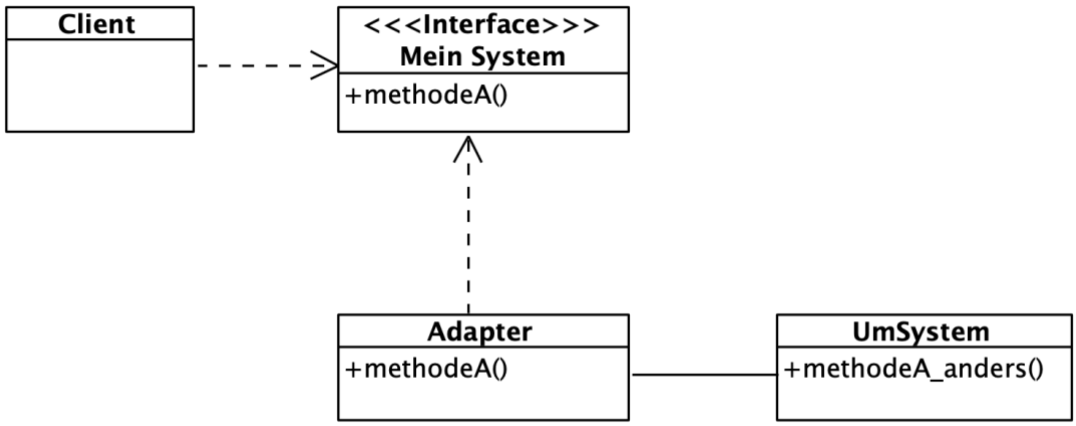
Kurzbeschreibung, Zweck

Ein Adapter wird dann eingesetzt, wenn ein oder mehrere fremde Systeme auch Um Systeme genannt, auf mein Programm zugreifen müssen.



Der Adapter ist ein Vermittler, welcher Anfragen von meinem System erhält und diese in Anfragen umwandelt, welche die Fremden Klassen verstehen.

UML-Diagramm



Mein System kann durch den Adapter nun die Funktion methodeA_anders() ausführen, indem er die Funktion methodeA() auf dem Adapter ausführt. Dieser wandelt die Anfrage des Clients so um, dass sie kompatibel mit dem Um System ist.

Ausführliche Beschreibung

Problem Stellung

Wir haben ein System, welches gewisse Methoden hat. Nun kommt jedoch ein externes System zum Einsatz welches wir in unser bestehendes System einbauen wollen. Leider sind die Methoden dieser beiden Systeme nicht miteinander kompatibel und da man die Systeme an sich nicht verändern will, schreibt man eine so genannte Adapter Klasse.

Lösung

Die Adapter Klasse wird in unser System integriert und kann von unserem bestehenden System aufgerufen werden. Der Aufruf aus dem bestehenden System wird aufgenommen, und so umgewandelt, dass er mit dem neuen System kompatibel ist. Ein riesen Vorteil dabei ist, dass man die Systeme an sich nicht verändern muss sondern dass man nur die Adapter Klasse verändert. Ein weiterer Vorteil ist die Wiederverwendbarkeit von vorhandenen Software Lösungen.

Beispiel Code (Java)

Klasse BMW (Client)

```
package adapter;

public class BMW implements DeutschesAuto{

	double kilometerstand = 0;

	@Override
	public void fahrenkilometer(double anzkilometer) {

		kilometerstand += anzkilometer;
		System.out.println("Ich bin " + anzkilometer + " Kilometer gefahren.");
		System.out.println("Mein Kilometerstand ist jetzt " + anzkilometer);
	}

}
```

Klasse Ford (Zulieferer Klasse)

```
package adapter;

public class Ford {
	double meilenstand = 0;
	public void fahrenmeilen(double anzmeilen)
	{
		meilenstand += anzmeilen;
		System.out.println("Ich bin " + anzmeilen + " Meilen gefahren.");
		System.out.println("Mein Meilenstand ist jetzt " + meilenstand);
	}
}
```

Klasse DeutschesAuto (Interface Client)

```
package adapter;

public interface DeutschesAuto {
	public void fahrenkilometer(double anzkilometer);
}
```

Klasse DeutscherFord (Adapter)

```
package adapter;

public class DeutscherFord implements DeutschesAuto{

	Ford ford = new Ford();
	double kilometerstand = 0;
	@Override
	public void fahrenkilometer(double anzkilometer) {
		ford.fahrenmeilen(anzkilometer * 0.621371);

	}

}
```