

Aufgabe 3

In dieser Aufgabe implementieren Sie ein konzeptionelles Datenmodell für eine Firma, die Personendaten von Angestellten und Kunden verwalten möchte. Gegeben seien dazu folgende Aussagen:

- Eine *Person* hat einen *Namen* und ein *Geschlecht* (männlich oder weiblich).
 - Ein *Angestellter* ist eine *Person*, zu der zusätzlich das monatliche *Gehalt* gespeichert wird.
 - Ein *Kunde* ist eine *Person*, zu der zusätzlich eine *Kundennummer* hinterlegt wird.
- (a) Geben Sie in einer objektorientierten Programmiersprache Ihrer Wahl (geben Sie diese an) eine Implementierung des aus den obigen Aussagen resultierenden konzeptionellen Datenmodells in Form von **Klassen** und **Interfaces** an. Gehen Sie dabei wie folgt vor:

- Schreiben Sie ein Interface `Person` sowie zwei davon ererbende Interfaces `Angestellter` und `Kunde`. Die Interfaces sollen jeweils lesende Zugriffsmethoden (Getter) die entsprechenden Attribute (Name, Geschlecht, Gehalt, Kundennummer) deklarieren.

```
3 public interface Person {
4     String getName();
5
6     char getGeschlecht();
7 }

```

Code-Beispiel auf Github ansehen:
src/main/java/org/bschlangaul/examen/examen_66116/jahr_2015/fruehjahr/Person.java

```
3 public interface Angestellter extends Person {
4     int getGehalt();
5 }

```

Code-Beispiel auf Github ansehen:
src/main/java/org/bschlangaul/examen/examen_66116/jahr_2015/fruehjahr/Angestellter.java

```
3 public interface Kunde extends Person {
4     int getKundennummer();
5 }

```

Code-Beispiel auf Github ansehen:
src/main/java/org/bschlangaul/examen/examen_66116/jahr_2015/fruehjahr/Kunde.java

- Schreiben Sie eine abstrakte Klasse `PersonImpl`, die das Interface `Person` implementiert. Für jedes Attribut soll ein Objektfeld angelegt werden. Außerdem soll ein Konstruktor definiert werden, der alle Objektfelder initialisiert.

```
3 public abstract class PersonImpl implements Person {
4     protected String name;
5     protected char geschlecht;
6
7     public PersonImpl(String name, char geschlecht) {
8         this.name = name;
9         this.geschlecht = geschlecht;
10    }

```

```

11
12     public String getName() {
13         return name;
14     }
15
16     public char getGeschlecht() {
17         return geschlecht;
18     }
19 }

```

Code-Beispiel auf Github ansehen:
[src/main/java/org/bschlangaul/examen/examen_66116/jahr_2015/fruehjahr/PersonImpl.java](https://github.com/bschlangaul/examen_66116/jahr_2015/fruehjahr/PersonImpl.java)

- Schreiben Sie zwei Klassen `AngestellterImpl` und `KundeImpl`, die von `PersonImpl` erben und die jeweils dazugehörigen Interfaces implementieren. Es sollen wiederum Konstruktoren definiert werden, die alle Objektfelder initialisieren und dabei auf den Konstruktor der Basis-Klasse `PersonImpl` Bezug nehmen.

```

3     public class AngestellterImpl extends PersonImpl implements
    ↪ Angestellter {
4         protected int gehalt;
5
6         public AngestellterImpl(String name, char geschlecht, int
    ↪ gehalt) {
7             super(name, geschlecht);
8             this.gehalt = gehalt;
9         }
10
11         public int getGehalt() {
12             return gehalt;
13         }
14
15     }

```

Code-Beispiel auf Github ansehen:
[src/main/java/org/bschlangaul/examen/examen_66116/jahr_2015/fruehjahr/AngestellterImpl.java](https://github.com/bschlangaul/examen/examen_66116/jahr_2015/fruehjahr/AngestellterImpl.java)

```

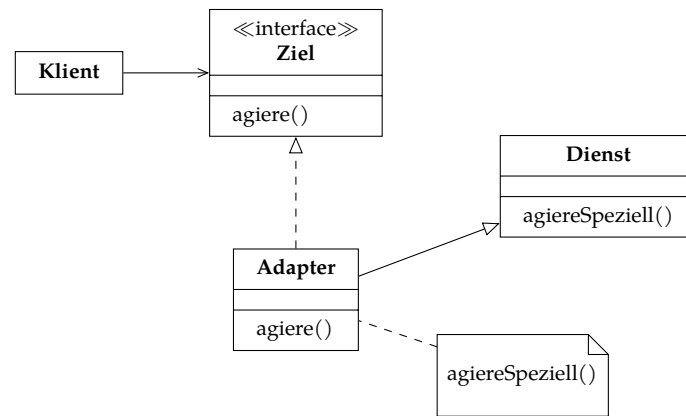
3     public class KundeImpl extends PersonImpl implements Kunde {
4         protected int kundennummer;
5
6         public KundeImpl(String name, char geschlecht, int
    ↪ kundennummer) {
7             super(name, geschlecht);
8             this.kundennummer = kundennummer;
9         }
10
11         public int getKundennummer() {
12             return kundennummer;
13         }
14     }

```

Code-Beispiel auf Github ansehen:
[src/main/java/org/bschlangaul/examen/examen_66116/jahr_2015/fruehjahr/KundeImpl.java](https://github.com/bschlangaul/examen/examen_66116/jahr_2015/fruehjahr/KundeImpl.java)

- (b) Verwenden Sie das Entwurfsmuster **Adapter**, um zu ermöglichen, dass vorhandene Angestellte die Rolle eines Kunden einnehmen können. Der Adapter soll eine zusätzliche Klasse sein, die das Kunden-Interface implementiert. Wenn möglich, sollen Methodenaufrufe an den adaptierten Angestellten delegiert werden. Möglicherweise müssen Sie neue Objektfelder einführen.

Exkurs: Entwurfsmuster „Adapter“



Ziel (Target) Das Ziel definiert die Schnittstelle, die der Klient nutzen kann.

Klient (Client) Der Klient nutzt Dienste über inkompatible Schnittstellen und greift dabei auf adaptierte Schnittstellen zurück.

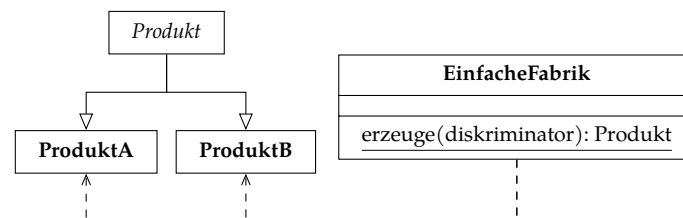
Dienst (Adaptee) Der Dienst bietet wiederzuverwendende Dienstleistungen mit fest definierter Schnittstelle an.

Adapter Der Adapter adaptiert die Schnittstelle des Dienstes auf die Schnittstelle zum Klienten.

- (c) Verwenden Sie das Entwurfsmuster **Simple Factory**, um die Erzeugung von Angestellten, Kunden, sowie Adapter-Instanzen aus Aufgabe (b) zu vereinheitlichen. Die entsprechende Erzeugungs-Methode soll neben einem Typ-Diskriminator (z. B. einem Aufzählungstypen oder mehreren booleschen Werten) alle Parameter übergeben bekommen, die für den Konstruktor irgendeiner Implementierungsklasse des Interface `Person` notwendig sind.

Hinweis: Um eine Adapter-Instanz zu erzeugen, müssen Sie möglicherweise zwei Konstruktoren aufrufen.

Exkurs: Entwurfsmuster „Einfache Fabrik“



EinfacheFabrik Eine Klasse mit einer Erzeugungsmethode, die über eine größere Bedingung verschiedene Objekt instanziiert.

Produkt Eine abstrakte Klasse, die von den konkreten Produkten geerbt wird.

KonkretesProdukt Ein konkretes Produkt, das von der einfachen



Fabrik erzeugt wird.