

Abstrakte Klassen und Interfaces I

Lernziele

- Sie setzen abstrakte Klassen zur Lösung von Problemstellungen im Umfang von einigen Klassen gezielt und korrekt ein.
- Sie fällen Entscheide zur Aufteilung der Funktionalität auf einzelne Klassen bewusst und können diese begründen.
- Sie können ein gegebenes Interface in ein Klassendesign integrieren und passend implementieren.

Tipp: Nutzen Sie das Konzept der Packages, um die unterschiedlichen „Versionen“ der in diesem Praktikum entwickelten Klassen im gleichen Projekt speichern zu können. Erzeugen Sie pro Aufgabe ein Package und speichern Sie die Lösung für diese Aufgabe jeweils im zugehörigen Package.

Aufgabe 1

Forken Sie für diese Aufgabe das Projekt https://github.engineering.zhaw.ch/prog1-kurs/10_Praktikum-1_Kaffee. Nutzen Sie Eclipse um die eigene Projektkopie auf Ihren Computer zu holen und zu bearbeiten. Das Projekt beinhaltet die folgenden zwei Klassen, um Kaffee und Tee zu kochen (javadoc Kommentare nicht gezeigt):

```
public class Kaffee {
    public void bereiteZu() {
        kocheWasser();
        braueFilterKaffee();
        giesseInTasse();
        fuegeZuckerUndMilchHinzu();
    }
    private void kocheWasser() {
        // Implementieren Sie z.B. eine Ausgabe
    }
    // Weitere Methoden
}

public class Tee {
    public void bereiteZu() {
        kocheWasser();
        taucheTeebeutel();
        giesseInTasse();
        fuegeZitroneHinzu();
    }
    void kocheWasser() {
        // Implementieren Sie z.B. eine Ausgabe
    }
    // Weitere Methoden
}
```

Vervollständigen Sie die beiden Klassen, damit sie funktionieren. Die Methoden können Sie sehr simpel halten. Eine einfache Bildschirmausgabe reicht aus. So könnte die kocheWasser Methode z.B. "Koche Wasser." ausgeben.

Aufgabe 2

Unter der Annahme, dass Sie die Klassen Kaffee und Tee nicht verändern dürfen: Ist es möglich, Code zu schreiben, der sowohl mit Kaffee als auch mit Tee umgehen kann? Überlegen Sie sich das an folgendem konkreten Beispiel:

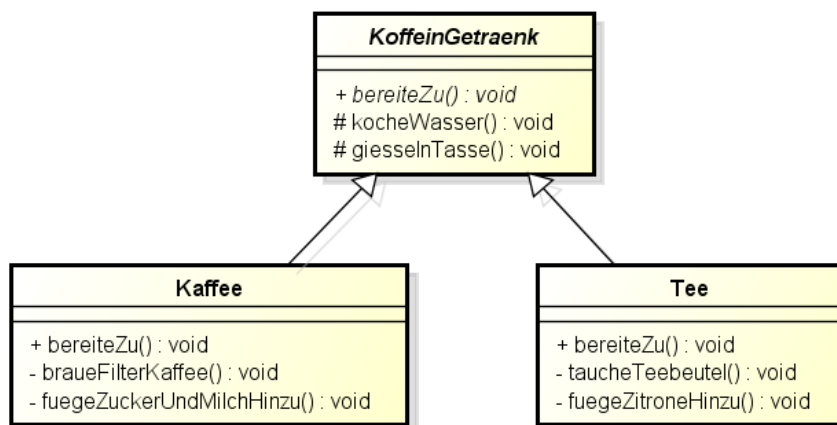
- Eine Methode soll sowohl ein an sie übergebenes Objekt vom Typ Kaffee, als auch vom Typ Tee ohne Fallunterscheidung (Verzweigungen) zubereiten können.

Ja, mit einer abstrakten Klasse. Diese erstellt selber

kein Objekt von sich selbst.

Aufgabe 3

Kaffee und Tee haben offensichtliche Gemeinsamkeiten. Diese Gemeinsamkeiten können Sie in einer Superklasse KoffeinGetraenk modellieren. Ihr neuer Ansatz könnte z.B. wie folgt aussehen:



Die Superklasse KoffeinGetraenk soll dabei abstract sein, da wir keine Objekte dieses Typs zulassen möchten. Verwenden Sie in dieser Teilaufgabe aber noch keine abstrakten Methoden und implementieren Sie die Methode `bereiteZu` in der Superklasse „leer“.

- Nutzen Sie nun die Gelegenheit, diesen Ansatz näher zu diskutieren. Was sind die Vorteile dieses neuen Klassenentwurfs, auch unter Berücksichtigung von Aufgabe 2?

Keine Code-duplizierung.

Struktur

Man kann die Methoden aber trotzdem auch noch überschreiben.

- Schreiben Sie nun die Superklasse und modifizieren Sie die beiden Subklassen wo nötig.

- Testen Sie die neuen Implementationen, indem Sie eine Klasse Getraenkezubereiter schreiben, die eine Liste mit einigen KoffeinGetraenk Objekten (Kaffee und Tee) erzeugt. Diese Liste wird anschliessend einer Methode übergeben, welche die Getränke zubereitet. Die Klasse Getraenkezubereiter soll eine main Methode besitzen, damit der Getraenkezubereiter als eigenständige Anwendung gestartet werden kann.

Aufgabe 4

Wir wollen nun einen neuen Ansatz untersuchen. In Aufgabe 3 haben Sie die offensichtlichen Gemeinsamkeiten „zentralisiert“. Aber wir können noch weiter gehen. Studieren Sie den folgenden Vorschlag:

```
public abstract class KoffeinGetraenk {  
  
    public final void bereiteZu() {  
        kocheWasser();  
        braue();  
        giesseInTasse();  
        fuegeZutatenHinzu();  
    }  
    // Weiterer Code  
}
```

- Die Methode bereiteZu wurde als final deklariert. Was bedeutet dies und wieso ist das in diesem Fall sinnvoll?

Weil sie in allen Subklassen das gleiche liefert, sollte sie
das auch bei allen Subklassen tun.

- Wie ist mit den Methoden braue und fuegeZutatenHinzu in KoffeinGetraenk zu verfahren?

Diese müssen noch spezialisiert werden. Für Kaffee kaffe
braueFilterKaffee und für Tee Taucheteebeutel. Diese Spezial-
isierungen können in der Klasse Tee resp. Kaffee gemacht werden.

- Was haben Sie mit diesem Design im Vergleich zu Aufgabe 3 dazugewonnen?

Jedes Getränk kann diesem Schema folgen und kann detailliert-
ere Prozesse ausweiten. Dadurch kann man die Getränke auch gleich
besser vergleichen oder weiterverarbeiten.

Ausserdem hat man jetzt alle Methoden in KoffeinGetränk.

- Realisieren Sie nun die Klassen `KoffeinGetraenk`, `Kaffee` und `Tee` entsprechend diesen Überlegungen und testen Sie das Ganze wieder, indem Sie einen passenden `Getraenkezubereiter` wie in Aufgabe 3 schreiben.

Aufgabe 5

Kaffee und Tee sollen nun zusätzlich auch als ein Objekt vom Typ `Trinkbar` behandelt werden können. Die einzige Methode von `Trinkbar`-Objekten ist die Methode `trinke()`, die nur den Text: „Ich trinke einen <Klassenname>“ ausgibt. Verwenden Sie für den Klassennamen die Methode `getClass()`, welche jedes Java Objekt besitzt. Diese Methode gibt ein Objekt vom Typ `Class` zurück, auf welchem Sie `.getSimpleName()` aufrufen können, um den Klassennamen des Objektes zu erhalten. Ergänzen Sie Ihren Code um die geforderte Funktionalität und erweitern Sie zusätzlich die Klasse `Getraenkezubereiter`, damit ein Getränk nach der Zubereitung auch noch getrunken wird.

Aufgabe 6

Verwenden Sie das `Trinkbar`-Interface aus Aufgabe 5 für ein weiteres Produkt, welches zwar trinkbar aber kein Getränk ist. Überlegen Sie sich, welche Verarbeitungsschritte an eine Superklasse delegiert werden können, falls in Zukunft mehrere solche Produkte benötigt werden. Implementieren Sie diese nach ähnlichem Schema wie in Aufgabe 5. Die Produkte sollen ebenfalls zubereitet und getrunken werden. Sie können zum Beispiel Kraftstoffe, welche auf Erdöl basieren, wählen.