Programmieren II (Java)

4. Praktikum: Vertiefung Objektorientierung

Sommersemester 2023

Christopher Auer, Tobias Lehner

Abgabetermine

Lernziele

- Vertiefung Objektorientierung in Java
- Abstrakte Klasse, Ableitung, super-Methodenaufrufe
- Interfaces, implementieres von Interfaces
- Gleichheit und Identität

Hinweise

- Sie dürfen die Aufgaben *alleine* oder zu *zweit* bearbeiten und abgeben
- Sie müssen *4 der 5* Praktika bestehen
- *Kommentieren* Sie Ihren Code
  - Jede *Methode* (wenn nicht vorgegeben)
  - *Wichtige* Anweisungen/Code-Blöcke
  - *Nicht kommentierter* Code führt zu *Nichtbestehen*
- Bestehen Sie eine Abgabe *nicht* haben Sie einen *zweiten Versuch*, in dem Sie Ihre Abgabe *verbessern müssen*.
- *Wichtig*: Sie sind einer *Praktikumsgruppe* zugewiesen, *nur* in dieser werden Ihre Abgaben *akzeptiert*!

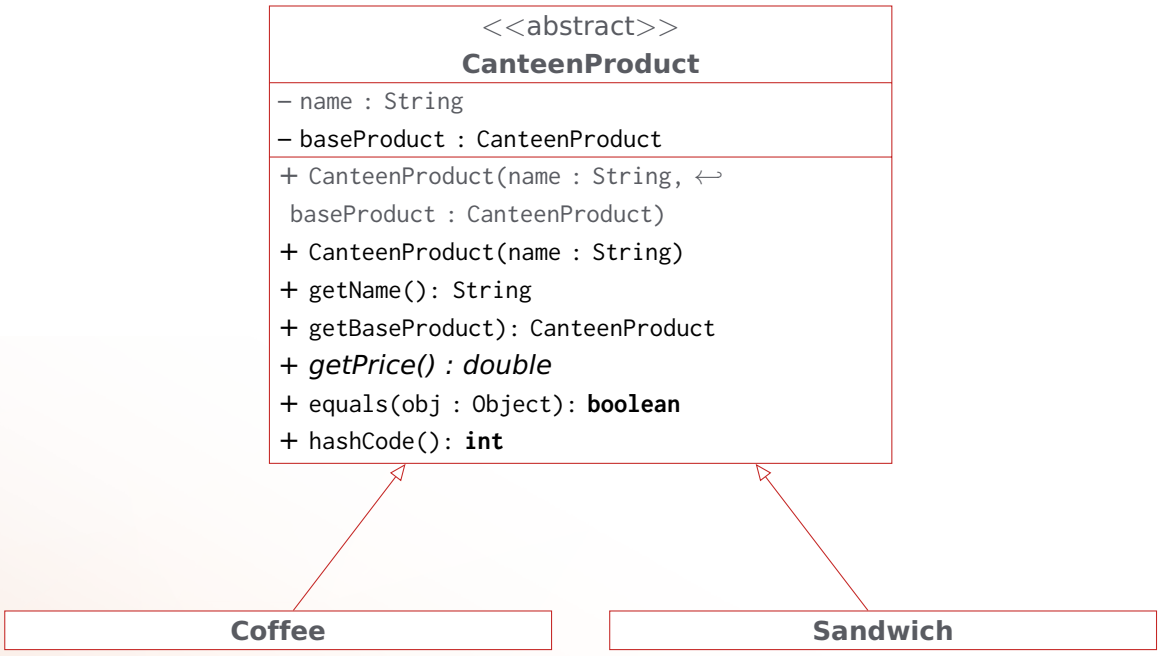
Aufgabe 1: Cafete ★ bis ★★

Die Cafete versucht mit leckeren Kaffee-Spezialitäten und besonderen Sandwiches die Studierenden von sich zu überzeugen. Damit schnell auf neue Entwicklungen reagiert werden kann, werden Sie beauftragt, eine Anwendung zu entwickeln, die bei der Produktentwicklung behilflich ist.

Implementieren Sie alle nachfolgenden Ausführungen innerhalb des Packages `canteen`*. Importieren Sie dazu das Gradle-Projekt in* `SupportMaterial/canteen` *in Ihre Entwicklungsumgebung. Enthalten ist eine Klasse* `MainCanteen`*, in der die zu implementierenden Klassen genutzt werden. Weiterhin sind auch JUnit-Testklassen enthalten. Bitte nehmen Sie keine Änderungen an den Testklassen vor. Es steht Ihnen frei, eigene Attribute und Methoden zu ergänzen, wo Sie das für sinnvoll halten.*

Abstrakte Klassen und Vererbung

Zunächst soll die abstrakte Oberklasse `CanteenProduct` implementiert werden:



- Deklarieren Sie die *unveränderliche* und *abstrakte* Klasse `CanteenProduct` mit deren Attributen.
  - `name` enthält den Namen des Produkts. Ein Name ist mindestens vier Zeichen lang.
  - Produkte können (müssen aber nicht) aufeinander aufbauen. `baseProduct` enthält das zugrundeliegende `CanteenProduct` für das aktuelle Produkt.
- Implementieren Sie die Konstruktoren zur Klasse
  - `CanteenProduct(String name, CanteenProduct baseProduct)` überprüft, ob der übergebene Name mindesten drei Zeichen enthält. Führende und endende Whitespaces werden hierfür nicht beachtet. `baseProduct` darf auch **`null`** werden.
  - `CanteenProduct(String name)` ruft den Konstruktor `CanteenProduct(String name, CanteenProduct baseProduct)` mit **`null`** als `baseProduct` auf.
- Implementieren Sie die Getter-Methoden zu den Attributen.
- Deklarieren Sie die akstrakte Methode `getPrice()`.
- Für `hashCode()` und `equals(Object obj)` sind alle Attribute relevant.

Die Klasse `Coffee`

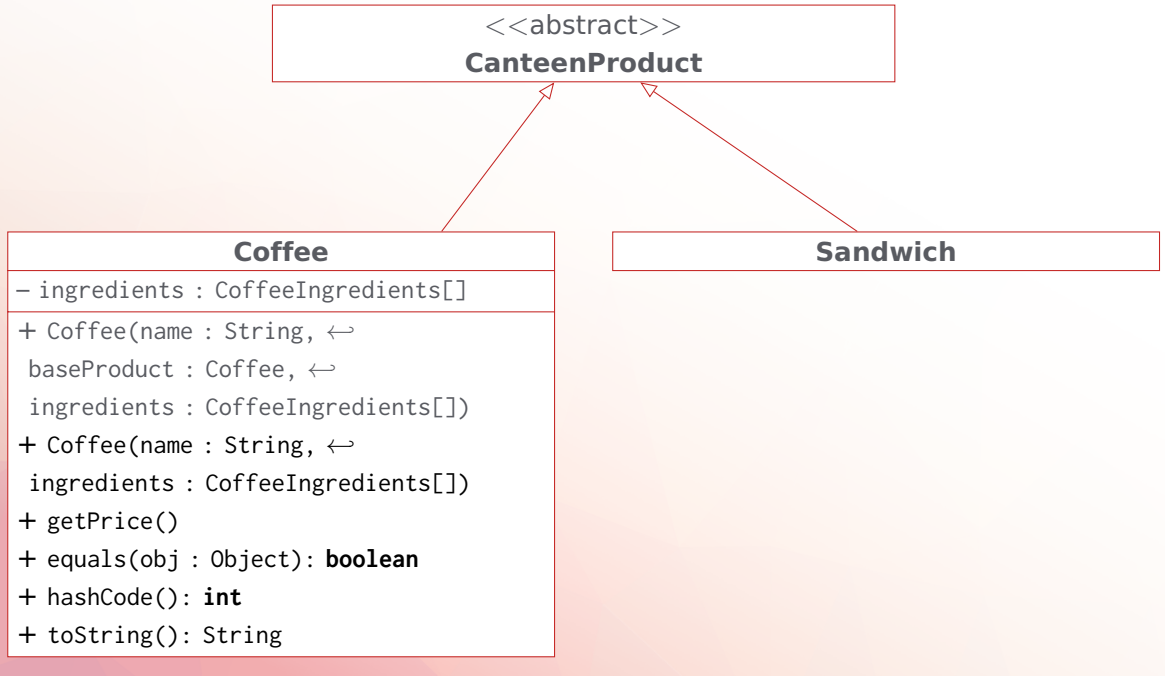
Die Klasse `Coffee` repräsentiert Kaffeeprodukte. Kaffeeprodukte bestehen aus Kaffeezutaten. Die Kaffeezutaten werden anhand der Enumeration `CoffeeIngredients` abgebildet, welche folgende Zutaten und deren Preis enthält:

Typ Enumeration	Preis
FROTHED_MILK	1.5
FROTHED_MILK_XXL	2.0
MILK	1.0
SUGAR	0.0
HAZELNUT_SYRUP	0.8
VANILLA_SYRUP	1.2
CARAMEL_SYRUP	0.8
CHOCOLATE	1.0
CHOCOLATE_POWDER	0.0
ESPRESSO_SHOT	1.2
HOT_WATER	0.0

Kaffeeprodukte können aufeinander aufbauen. Dabei ist auch die Reihenfolge der Zutaten relevant:

- Ein Espresso besteht aus einem `ESPRESSO_SHOT`.
- Ein Cappuccino verwendet das Basisprodukt Espresso und enthält weiterhin die zusätzliche Zutat `FROTHED_MILK`.
- Ein Latte Macchiato besteht aus den Zutaten `FROTHED_MILK_XXL` und `ESPRESSO_SHOT`.
- Ein Haselnuss-Macchiato verwendet das Basisprodukt Latte Machiato und enthält weiterhin die Zutat `HAZELNUT_SYRUP`.

Die Klasse `Coffee` ist entsprechend des folgendem UML-Klassendiagramms zu implementieren:



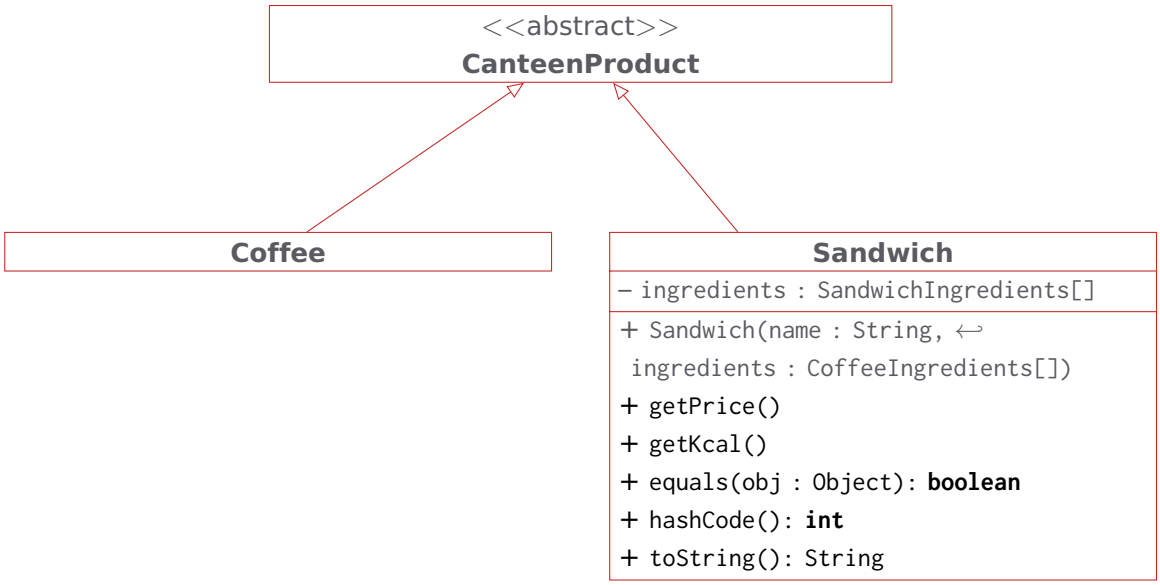
- Deklarieren Sie die Klasse `Coffee` mit deren Attribut `ingredients`. Bei `ingredients` handelt es sich um ein Array aus `CoffeeIngredients`. Die Größe des Arrays wird im Konstruktor bestimmt.
- Implementieren Sie den Konstruktor **`public`** `Coffee(String name, Coffee baseProduct, CoffeeIngredients... ingredients)`
  - Rufen Sie hierfür den passenden Konstruktor der übergeordneten Klasse auf.
  - Das Array `ingredients` wird als Vararg übergeben. Achten Sei darauf, dass ein Kaffeeprodukt zumindest eine Zutat enthalten sollte (unabhängig vom Basisprodukt).
- Implementieren Sie den Konstruktor **`public`** `Coffee(String name, CoffeeIngredients... ingredients)`. Der Konstruktor ruft den Konstruktor **`public`** `Coffee(String name, Coffee baseProduct, CoffeeIngredients... ingredients)` mit **`null`** als `baseProduct` auf.
- Überschreiben Sie die Methode `getPrice()`. Der Preis errechnet sich aus der Summe der Preise der Basisprodukte und der Summe der Preise der Kaffeezutaten des Arrays `ingredients`.
- Für `hashCode()` und `equals(Object obj)` sind alle Attribute relevant.
- Überschreiben Sie `toString()` in der Form `"$NAME$\t\t\t\t$PREIS$"`, wobei bei `$NAME$` der Name des Produkts und bei `$PRICE$` der Preis des Produkts einzufüllen ist.

Die Klasse `Sandwich`

Im Gegensatz zu Kaffeeprodukten können Sandwiches nicht aufeinander aufbauen. Sie bestehen aus Zutaten der Enumeration `SandwichIngredients`. `SandwichIngredients` enthält folgende Zutaten samt deren Preis und kcal:

Typ Enumeration	Preis	kcal
BREAD	0.5	190
WHOLE_GRAIN_BREAD	0.7	194
SALAMI	0.5	115
HAM	0.5	46
CHICKEN	0.5	55
BEEF	1.0	70
VEGAN_MEET_REPLACEMENT	1.0	117
CHEDDAR	0.5	113
MOZARELLA	0.5	35
EMMENTAL	1.0	120
CREAM_CHEESE	0.5	100
TOMATO	0.2	7
CUCUMBER	0.2	3
PAPRIKA	0.4	4
SALAD	0.2	2
KETCHUP	0.2	20
MAYONNAISE	0.2	68
SPICY_JOGHURT	0.6	15

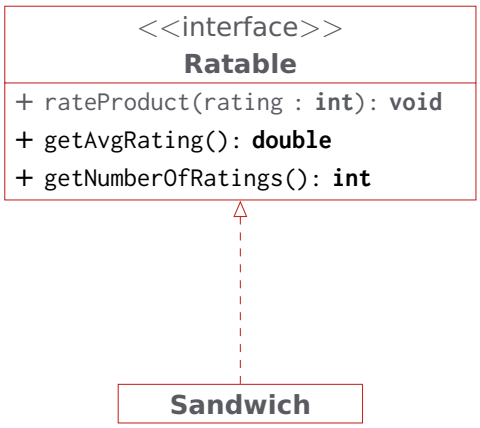
Die Klasse Sandwich ist entsprechend des folgenden UML-Klassendiagramms zu implementieren:



- Deklarieren Sie die Klasse `Sandwich` mit deren Attribut `ingredients`. Bei `ingredients` handelt es sich um ein Array aus `SandwichIngredients`. Die Größe des Arrays wird im Konstruktor bestimmt.
- Implementieren Sie den Konstruktor `public Sandwich(String name, SandwichIngredients... ingredients)`
  - Rufen Sie hierfür den passenden Konstruktor der übergeordneten Klasse auf.
  - Das Array `ingredients` wird als Vararg übergeben.
  - Sollte sich in den übergebenen Zutaten weder `BREAD` noch `WHOLE_GRAIN_BREAD` befinden muss eine selbst zu implementierende ungeprüfte Ausnahme `SandwichHasNoBreadException` geworfen werden. Implementieren Sie die Ausnahmeklasse entsprechend der Vorgaben aus der Vorlesung mit den drei Standardkonstruktoren aus `Throwable`.
  - Wenn weniger als zwei Zutaten übergeben werden, soll eine selbst zu implementierende ungeprüfte Ausnahme `SandwichHasTooFewIngredientsException` geworfen werden. Implementieren Sie die Ausnahmeklasse entsprechend der Vorgaben aus der Vorlesung mit den drei Standardkonstruktoren aus `Throwable`.
- Bei den Sandwiches funktioniert die Methode `getPrice()` grundsätzlich anders als bei den Kaffeeprodukten: Nur die teuersten vier Zutaten zählen zum Preis. Alle anderen Zutaten bleiben kostenlos.
- Die Methode `public int getKcal()` liefert die Summe der kcal aller Zutaten.
- Für `hashCode()` und `equals(Object obj)` sind alle Attribute relevant.
- Überschreiben Sie `toString()` in der Form `"$NAME$($KCAL$ kcal)\t\t\t\t$PREIS$"`, wobei bei `$NAME$` der Name des Produkts, bei `$KCAL$` die Summe der kcal aller Zutaten und bei `$PRICE$` der Preis des Produkts einzufüllen ist.

Aufgabe 2: Bewertung ★☆ bis ★★

Bei den Sandwiches sind einige sehr exotische Kombinationen möglich. Die Cafete wünscht sich deswegen eine Möglichkeit für eine Bewertung der Nutzer. Gleichzeitig soll sichergestellt werden, dass eine solche Bewertung auch für die zukünftig angedacht Klassen `Wrap` und `Pizza` möglich ist. Sie schlagen deswegen ein Interface `Ratable` vor:



Schreiben Sie das Interface und erweitern Sie die Klasse `Sandwich` dahingehend, dass sie das Interface implementiert. Ratings sollen in einem Bereich zwischen einem und fünf ganzen Punkten erfolgen. Fünf Punkte Stellen den besten Wert dar. Ein Punkt stellt den schlechtesten Wert dar. Es steht Ihnen frei `Sandwich` um beliebige Attribute und Methoden zu ergänzen.

In der ausgelieferten Klasse `MainCanteen` werden Ihre Klassen beispielhaft genutzt. Erweitern Sie die `main`-Methode der Klasse `MainCanteen` wie folgt:

- Iterieren Sie über die `List<CanteenProduct> products`.
- Geben Sie für alle Produkte, welche das Interface `Ratable` implementieren, 20 zufällige Ratings ab. Hierfür können Sie die Methode `Math.random()` verwenden.
- Schreiben Sie weiterhin eine Methode **`double`** `computeAvgRatings(List<CanteenProduct> products)`, welche den Durchschnittswert aller aller durchschnittlichen Bewertungen der Produkte berechnet.
  - Achten Sie darauf, dass Die Methode nur Produkte auswertet, für die auch ein Rating hinterlegbar ist.
  - Wählen Sie die Modifier für **`double`** `computeAvgRatings(List<CanteenProduct> products)` so, dass die Methode in der `main`-Methode aufrufbar ist.
- Rufen Sie schließlich **`double`** `computeAvgRatings(List<CanteenProduct> products)` in der `main`-Methode auf und geben den Durchschnittswert der Ratings aus.