

# Laborbericht

# Programmierparadigmen

von Jonas Kuhlo

[REDACTED]

[REDACTED] Reutlingen

[REDACTED]

Immatrikulationsnummer: [REDACTED]

im Modul PRG24 zum Thema Klassen und Operationen

fertiggestellt am 07.10.2021

## Inhaltsverzeichnis

1.	Einleitung .....	1
2.	Erstellung eines Klassendiagramms .....	1
2.1.	Suche nach Klassen .....	2
2.2.	Beziehungen, Attribute und Methoden .....	3
3.	Ableitung eines Datenbankmodells.....	7
4.	Exemplarische Codierung .....	9
4.1.	Codierung der Datenbank.....	9
4.2.	Codierung der Klassen und Methoden .....	12
5.	Schlusswort .....	15
	References .....	16
	Anlagen .....	17

## 1. Einleitung

Im Rahmen des Moduls Programmierparadigmen habe ich mich mit unterschiedlichen Ansätzen der Programmierung befasst. Ein solcher Ansatz ist die objektorientierte Programmierung, welchen ich in dieser Arbeit, anhand eines praktischen Beispiels, vertiefen möchte. Neben der Objektorientierung werden wir aber auch auf den Entwurf von Datenbanken eingehen.

Als praktisches Beispiel für diesen Bericht soll ein Onlineportal dienen, welches vom Verfasser immer wieder zu Bestellung von elektronischen Zigaretten und entsprechendem Zubehör genutzt wird und über die URL <https://www.intaste.de/> erreichbar ist.

## 2. Erstellung eines Klassendiagramms

Zu Beginn dieser Arbeit steht als die Überlegung, wie ein System aussehen kann, welches die Funktionalität des Webshops der Firma InTaste abbildet. Hierzu möchte ich als erstes klären, welchen Teil des Systems wir hier überhaupt betrachten. Anschließend werden wir uns Gedanken zu dessen Gestaltung machen.

Wenn wir den Webshop in unserm Browser betrachten, sehen wir eine Darstellung, die als HTML- und CSS-Code generiert und an uns übermittelt wird. Der Browser lässt diesen dann hübsch aussehen, wir könnten uns aber auch den eigentlichen Code anzeigen lassen. Auf der anderen Seite des Systems muss es jedoch eine Repräsentation der Daten geben, die von diesem System verwaltet werden, welches diese z.B. in Form von Objekten unterschiedlicher Klassen darstellt und den Zugriff auf die Datenbank des Systems organisiert. Dazwischen gibt es vermutlich eine weitere Instanz, die Anfragen und Eingaben aus der Weboberfläche entgegennimmt und an die entsprechenden Komponenten im Datenmodell weitergibt. Ein passendes Design-Pattern hierzu könnte z.B. das MVP-Entwurfsmuster sein. [1]

Wir haben also eine Ansicht, eine Steuerungseinheit und ein Datenmodell mit Datenbank. Im Rahmen dieser Arbeit werden wir den Bereich der Webansicht außen vor lassen und das Datenmodell betrachten. Hierbei nehmen wir teilweise die Ansicht der Steuereinheit ein und werden am Ende auch aus dieser Perspektive in der Lage sein, bestimmte Anwendungsfälle im Datenmodell durchzugehen.

## 2.1. Suche nach Klassen

Wir überlegen nun also, welche Objekte unser Datenmodell repräsentieren soll. Beim Blick auf die Webansicht fallen mir hierzu als ersten die angebotenen Waren, nämlich die Produkte, ins Auge. Hier finden sich unterschiedliche Arten von Produkten, die wir im Rahmen dieser Arbeit nicht alle einbeziehen können. Beschränken möchte ich mich daher auf die Verdampfer, die Verdampferköpfe sowie die Akkuträger. Hinzukommen sollen noch die Startersets. Hier jedoch nur solche, die sich aus genau einem Verdampfer, einem Verdampferkopf und einem Akkuträger zusammensetzen.

Aus eigener Erfahrung weiß ich zudem, dass für das Aufgeben einer Bestellung im Shop ein Benutzerkonto nötig ist. Somit liegt es nahe, dass es auch eine Klasse Benutzer gibt. Vermutlich gibt es neben den Kunden, also Benutzern, die Einkäufe tätigen, auch noch einen Zugang für Mitarbeiter, um z.B. neue Produkte hinzuzufügen oder alte zu entfernen. Auch hier könnte es verschiedene Benutzergruppen geben. Für diese Arbeit möchte ich sie aber als Administratoren zusammenfassen.

Neben diesen offensichtlichen Klassen lässt sich aber vermuten, dass es noch weitere Klassen gibt. Als Kunde können z.B. Adressen und Bankverbindungen hinterlegt werden. Auch diese könnten eigene Klassen darstellen. Auch die Bestellungen (sowohl die aktuelle als auch vergangenen) könnten eine eigene Klasse darstellen.

Um den Bogen zu den bestellten Produkten zu schlagen, scheint es zudem hilfreich, wenn die einzelnen Posten der Bestellung ebenfalls als eigene Objekte dargestellt werden können. Es kommt also noch eine Klasse für Bestellposten hinzu.

Die bisher angesprochenen Klassen sind noch weitestgehend naheliegend. Im Laufe der Arbeit an diesem System schien es mir jedoch sinnvoll noch eine weitere Klasse zu ergänzen, welche ich mit Warenhaus benannt habe, da sie zur Laufzeit die Referenzen zu allen Produkten (bzw. Produkt-Subklassen) zentral verwaltet.

Die Überlegungen hinter der Klasse Warenhaus greifen den bisherigen Überlegungen ein wenig vor und ergeben sich aus der Betrachtung der Beziehung zwischen Bestellposten und den Produkten und deren Repräsentation in der Datenbank.

Jeder Bestellposten bezieht sich auf ein Produkt. Zur Laufzeit könnte er also eine Referenz zu einer Instanz des entsprechenden Produktes hinterlegen. Dies würde aber zu sehr vielen Instanzen führen. So würden 10 Benutzer mit 10 Bestellungen mit je 10 Bestellposten schon

1000 (teilweise identische) Instanzen von Produkten verursachen. Hinzu kommt, dass die Informationen, die in den instanzierten Objekten hinterlegt sind, von den Informationen in der Datenbank abweichen können, wenn es zwischenzeitlich eine Änderung gab. Es müsste also eine Aktualisierung erfolgen.

Eine Lösung wäre eine Instanziierung bei Bedarf. Hierfür müssen jedoch die Daten aus der Datenbank gelesen werden. Die Repräsentation der Produkte in der Datenbank wurde (von mir) allerdings in der Form von einzelnen Tabellen mit gemeinsamem Primärschlüssel gewählt. Es muss somit beim Instanziieren der Daten auf zwei Tabellen der Datenbank zugegriffen werden, wobei die zweite nicht bekannt ist. In unserem Beispiel haben wir 4 mögliche Tabellen für die Produkt-Subklassen (in der Realität wären es mehr!). Jede Instanziierung benötigt also 5 Datenbankaufrufe. 10 Benutzer, die sich die Informationen zu ihren 10 aktuellen Bestellposten anschauen wollen, würden also 400 Datenbankaufrufe verursachen.

Insgesamt schien mir dies eher ineffizient zu sein, daher wurde die Hilfsklasse Warenhaus hinzugefügt. Diese verwaltet zentral für jedes Produkt eine Instanz und aktualisiert diese regelmäßig und organisiert diese als Hashmap, um schnelle Zugriffe über die Produkt-ID zu ermöglichen. Benutzer erhalten eine Referenz zur zentralen Instanz des Warenhauses und geben diese weiter, so dass vom Bestellposten aus den benötigten Informationen zum jeweiligen Produkt (mittels der Produkt-ID) bei Bedarf abgefragt werden können.

Konkret erfolgt eine Aktualisierung (maximal) alle 60 Sekunden. Hierfür sind 5 Datenbankzugriffe nötig, so dass pro Stunde 300 Datenbankzugriffe nötig werden, um bei Bedarf immer Informationen zu erhalten, die nicht älter als eine Minute sind. Mehrere Instanziierungen desselben Produktes werden komplett vermieden. Diese Zahlen bilden hierbei jedoch eine Konstante: auch wenn 100 Nutzer gleichzeitig aktiv wären, blieben sie unverändert.

## 2.2. Beziehungen, Attribute und Methoden

Nachdem im ersten Schritt die Klassen gefunden wurden, können wir uns im nächsten Schritt mit den Beziehungen der Klassen zueinander beschäftigen und abschließend einen kurzen Blick auf die Attribute und Methoden werfen.

Wenn wir überlegen, in welchen Beziehungen die Klassen (und somit die instanzierten Objekte) zueinanderstehen, fallen als erstes zwei Strukturen auf, die sich gut für eine Vererbung eignen. So haben alle Benutzer (Kunden und Administratoren) sowie alle Produkte

(Verdampfer, Verdampferköpfe, Akkuträger und Starter-Sets) Gemeinsamkeiten, die sich jeweils in einer Super-Klasse (Elternklasse) verallgemeinern lassen.

Die Klasse Starter-Sets besteht aus anderen Produkten. Daher lässt sich diese Beziehung als Komposition beschreiben, denn wenn ein Bestandteil nicht mehr vorhanden wäre, wäre dieses Set nicht mehr möglich. Das gleiche Verhältnis findet sich auch zwischen Bestellungen und den jeweiligen Bestellposten.

Die Klassen Bestellung, Adresse und Bankverbindung hingegen sind zwar eng mit dem jeweiligen Kunden verknüpft, allerdings ist es nach meiner Auffassung hier angebracht eher von einer Aggregation zu sprechen. Die genannten Daten sollten ggf. auch ohne den zugeordneten Kunden verbleiben, da sie noch an anderen Stellen benötigt werden (z.B. für Auswertungen oder zum Bearbeiten einer offenen Bestellung) auch wenn das verknüpfte Kundenkonto gelöscht wurde.

Auch Objekte vom Typ Warenhaus sind unabhängig von der Existenz anderer Objekte. Es hat aber Produkte hinterlegt und wird vom Bestellposten aus angefragt, wenn dieser Infos benötigt. Somit ist auch hier eine Aggregation im Klassendiagramm eingezeichnet.

Als letzte Beziehung im Schaubild sei hier noch die nicht klassifizierte Beziehung zwischen Verdampfern und Verdampferköpfen erwähnt, die Passungen beschreibt. Diese werden wir später im Datenbankentwurf wieder treffen.

Im fertigen Klassendiagramm sind zudem noch die Kardinalitäten zu den Beziehungen hinterlegt. Diese geben z.B. an, dass eine Bankverbindung immer einem spezifischen Kunden zugeordnet ist, ein Kunde aber beliebig viele Bankverbindungen hinterlegen kann. Diese spiegeln sich ebenfalls stark im Datenbankentwurf wieder, wenn wir überlegen, an welchen Stellen Fremdschlüssel gesetzt werden.

Bezüglich der in den einzelnen Klassen hinterlegten Attributen gibt es keine großen Überraschungen. Die meisten können mehr oder weniger direkt der Webansicht entnommen werden, wobei an vielen Stellen vereinfacht wurde, um eine gewisse Überschaubarkeit für den Rahmen dieser Arbeit zu wahren. Auch die Datentypen ergeben sich überwiegend direkt aus der Betrachtung. An einigen Stellen wäre es ggf. noch interessant gewesen, über Enumeration die sinnvollen Werte für bestimmte Felder zu fixieren. Dies wäre also ein möglicher Schritt zur Weiterentwicklung des Systems.

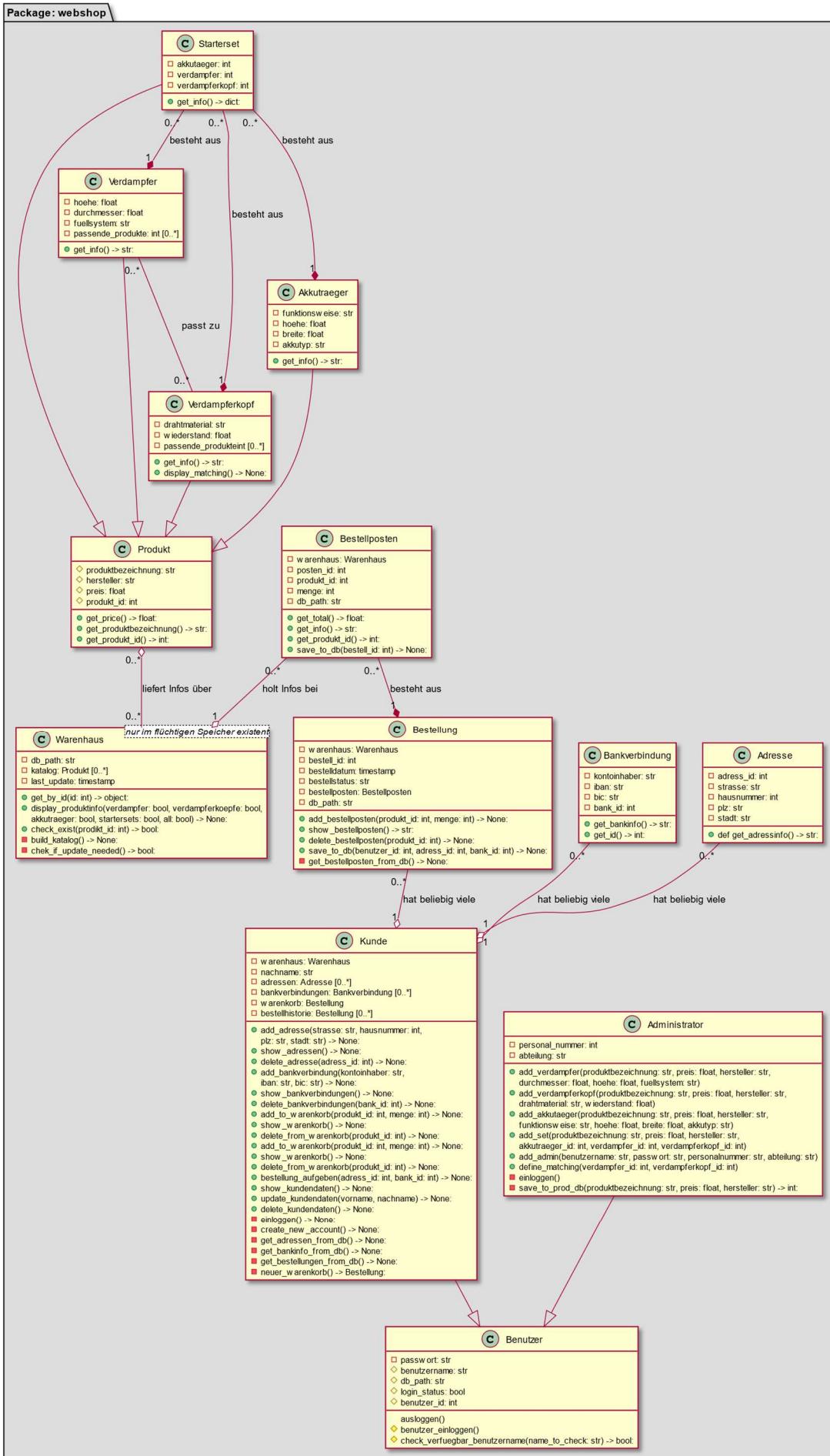
Bei den Methoden finden sich zunächst viele der nötigen Funktionalitäten, um aus dem Model heraus die Datenbank anzusprechen. In Summe sind alle vier klassischen Operationen für persistente Speicher (Create, Read, Update, Delete) vertreten. Wir erstellen Einträge (z.B.

beim Anlegen neuer Produkte), wir lesen Daten ein und instanziieren damit Objekte. Beim Kunden wurde eine Funktion zur Aktualisierung der Kundendaten und auch das Löschen von Datensätzen implementiert.

Neben den Datenbankoperationen finden sich noch Getter-Methoden, die es erlauben, private Werte oder andere zusammengestellte Informationspakete von den einzelnen Objekten zu erhalten. Dies ermöglicht es, dass die konkreten Attribute überwiegend privat sind und somit vor (versehentlichen) Manipulationen geschützt bleiben.

Als letzte Gruppe von Methoden gibt es noch jene, die interne Werte manipulieren (verändern). Hier ist vor allem das Einloggen und Ausloggen der Benutzer zu nennen, welches das private Attribut Login-Status verändert.

Nachdem wir nun, zugegebenermaßen im Eilschritt, durch den Entwurf gegangen sind, findet sich das fertige Schaubild des Klassendiagramms auf der folgenden Seite. Zu beachten sei hier noch, dass die Darstellung nicht zu 100% den Vorgaben der UML entspricht, da z.B. Super-Klassen teilweise nicht oberhalb der Sub-Klassen dargestellt sind. Dafür findet sich eine codierte Version des Schaubildes als .puml-Datei in den Projektdateien und auch eine beliebig vergrößerbare Vektorgrafik ist dort zu finden.



### 3. Ableitung eines Datenbankmodells

Wenden wir uns nun also dem Entwurf unserer Datenbank zu. Als einführender Hinweis sei gesagt, dass für die exemplarische Codierung eine SQLite-Datenbank zum Einsatz kommen soll, was sich auch in einigen Ausführungen und insbesondere in der abschließenden Darstellung unseres Datenbankentwurfes niederschlagen wird. Als praktischen Einstieg wenden wir uns als erstes den Vererbungsbeziehungen zu.

Beim Entwurf unserer Datenbank ist zu beachten, dass die Vererbung kein Konzept ist, das sich direkt in eine relationale Datenbank übertragen lässt. Scott W. Ambler beschreibt Wege, um Vererbung in einer Datenbank abzubilden, und diskutiert auch die Vor- und Nachteile. Die für diese Arbeit relevanten Ansätze lassen sich zusammenfassen als:

1. Alle Daten der Super- und Sub-Klassen werden in einer gemeinsamen Tabelle abgebildet.
2. Jede Subklasse erhält eine eigene Tabelle, in der auch die Daten der Superklasse abgebildet werden.
3. Jede Super- und Subklasse erhält eine eigene Tabelle, wobei die Subklassen den Primärschlüssel der Superklasse sowohl als eigenen Primärschlüssel als auch als Fremdschlüssel zur Verknüpfung mit der Superklasse nutzen.

[2, 2.1 - 2.3]

Für die Klassen Administrator und Kunde bietet es sich an, alle Datensätze in einer zentralen Tabelle aller Benutzer zu speichern, da die Subklassen eine überschaubare Anzahl von zusätzlichen Attributen haben und so die Anzahl der Leerstellen in der Tabelle überschaubar bleibt. Für das spätere Einlesen der Daten werden in der Tabelle zwei Spalten ergänzt, die anzeigen, ob ein Benutzer Kunde ist oder Administrator (IS\_ADMIN und IS\_KUNDE).

Zu beachten ist hier, dass in SQLite-Datenbank kein eigener Datentyp für Wahrheitswerte zur Verfügung steht. Daher werden diese als Integer Felder definiert. Für den Wert FALSCH wird dort eine 0 und für den Wert WAHR eine 1 gespeichert. Beim Auslesen wird jeder Wert, der nicht 0 ist, als WAHR gelesen.

Anders als bei den Benutzern werden bei den Produkten in den Subklassen relativ viele Attribute geführt und wir betrachten hier auch nur eine stark verringerte Anzahl von Produktarten. Die Lösung, alle in eine gemeinsame Tabelle zu schreiben, würde daher zu relativ vielen Leerstellen führen und somit die Datenbank unnötig vergrößern.

Daher stellt sich die Frage, ob es hier praktikabler ist, eine Tabelle für jede Subklasse zu erstellen oder die Tabellen aufzuteilen und über einen gemeinsamen Schlüssel zu verknüpfen.

Keine der beiden Optionen scheint auf den ersten Blick ideal zu sein. Um zu einer bestmöglichen Lösung zu kommen, möchte ich betrachten, wie die späteren Zugriffe auf die Daten aussehen und was hier günstiger ist.

Gibt es für jede Subklasse eine Tabelle, in der alle Daten (inklusive der Daten, die in der Superklasse gekapselt sind) hinterlegt sind, wäre ein Aufruf der Produkte einer Kategorie relativ einfach, da diese ja in einer fixen Tabelle zu finden sind. Aufwändig scheint mir jedoch der Zugriff für die Klasse Bestellposten. Wie im Kapitel 2.1 bereits diskutiert, habe ich die aus meiner Sicht effizienteste Methode der Aufgliederung in Tabellen für Super- und Sub-Klassen gewählt und bin dem Zugriffsproblem mit einem Hilfskonstrukt begegnet.

Nachdem nun eine Lösung für die Abbildung der Vererbungsstrukturen in der Datenbank gefunden ist, können wir uns den weiten Klassen zuwenden. Die Klassen für Bankverbindung, Adresse, Bestellung und Bestellposten erhalten jeweils eine eigene Tabelle. Interessant ist die Frage der Fremdschlüssel. Für diese Klassen stellt sich die Frage, wo jeweils ein Fremdschlüssel gespeichert werden soll. Reicht es, wenn z.B. die ADRESS\_ID beim BENUTZER gespeichert wird oder sollte auch in der Tabelle ADRESSE die zugeordnete BENUTZER\_ID hinterlegt sein? Möglich wäre es grundsätzlich beides zu tun, was aber vermutlich redundant wäre. Im Hinblick auf die Zugriffe reicht es vollkommen aus, wenn in der Tabelle ADRESSE die BENUTZER\_ID hinterlegt ist. So kann vom Benutzer aus eine Suche nach allen zugeordneten Adressen stattfinden und umgekehrt.

Auch bei den Subklassen der Superklasse Produkt sollten wir noch einen Blick auf die Setzung der Fremdschlüssel werfen. Hier haben wir zunächst die Klasse Starterset, die für jeden Bestandteil einen Fremdschlüssel zu den Produkten benötigt, aus denen es sich zusammensetzt.

Spannend wird es bei den Klassen Verdampfer und Verdampferkopf. Hier kann jeder Verdampfer mehrere passende Verdampferköpfe haben, ein Verdampferkopf kann aber auch zu verschiedenen Verdampfern passen. Wir haben also eine "Many-to-Many"- Beziehung, für die wir eine eigene Tabelle benötigen.

Da sich die Beziehung mit dem Ausdruck „passt zu“ beschreiben lässt, können wir auch die entsprechende Tabelle PASST\_ZU nennen. In der Tabelle selbst benötigen wir lediglich zwei Spalten, um die Passungen anhand der jeweiligen Primärschlüssel (PRODUKT\_ID) abzubilden. Zudem benötigen wir einen Primärschlüssel, den wir in diesem Fall PASSUNG\_ID nennen.

Da wir auch hier nur im Eilschritt durch die Überlegungen gehen konnten, findet sich zur besseren Übersicht ein Schaubild zu Datenbankentwurf in den Anlagen. Auch eine Darstellung der Tabellen in der Datenbank ist dort zu finden.

## 4. Exemplarische Codierung

Nachdem wir nun ein Klassendiagramm und einen Entwurf für die Datenbank haben, wenden wir uns der exemplarischen Codierung zu. Hierfür müssen wir zunächst eine passende Programmiersprache wählen. Für diese Arbeit soll eine Codierung in Python erfolgen, eine Sprache, die sich in Umfragen immer wieder großer Beliebtheit erfreut.

Vielen ist Python zwar als Skriptsprache bekannt, es bietet jedoch auch die Voraussetzungen für eine objektorientierte Programmierung, wie wir gleich sehen werden. Den Umstand, dass uns Python nicht immer zwingt alles objektorientiert zu programmieren, werden wir uns aber gleich zu Beginn beim Erstellen der Datenbank zu Nutze machen.

### 4.1. Codierung der Datenbank

Bevor wir mit der Programmierung unserer Klassen und Methoden beginnen, möchte ich hier einen kurzen Blick auf die Erstellung der Datenbank werfen. Vorteilhaft wäre es, wenn diese bereits vorhanden ist, bevor wir unsere Anwendung das erste Mal aufrufen oder Methoden programmieren, die einen Zugriff auf die Datenbank implementieren sollen. Da die Erstellung der Datenbank ein einmaliger Prozess ist, der nur beim ersten Start der Anwendung erfolgen muss, werden wir hierfür ein separates Skript verwenden, also einen Ablauf, den wir aufrufen können, wenn noch keine entsprechende Datenbank vorhanden ist. Dieses Skript enthält die Definitionen der einzelnen Tabellen und sorgt dafür, dass diese als leere Tabellen erstellt werden. Einige der Befehle, die zur Erstellung der Datenbank verwendet werden, möchte ich hier exemplarisch betrachten. Die Gesamtheit des Codes findet sich im Anhang.

Um die Kommunikation zwischen unserer Anwendung und der Datenbank zu realisieren, können wir für diese exemplarische Codierung eine lokale Datenbank verwenden. Hierfür bietet Python uns ein Paket, welches direkt in der Standartbibliothek integriert ist, um mit SQLite-Datenbanken zu arbeiten. Eine ausführliche Dokumentation hierzu findet sich in der entsprechenden Dokumentation. [3]

Beginnen wir also mit unserer ersten Tabelle, der Tabelle für die Benutzer. Hier ergibt sich aus unserem Entwurf folgender SQLite-Befehl:

CREATE	TABLE	"BENUTZER"			(
"BENUTZER_ID"	INTEGER	NOT	NULL	UNIQUE,	
"BENUTZERNAME"	TEXT	NOT	NULL	UNIQUE,	
"PASSWORT"	TEXT		NOT		NULL,
"IS_ADMIN"	INTEGER		NOT		NULL,
"IS_KUNDE"	INTEGER		NOT		NULL,
"PERSONALNUMMER"					TEXT,
"ABTEILUNG"					TEXT,
"VORNAME"			TEXT		,
"NACHNAME"					TEXT,
PRIMARY KEY("BENUTZER_ID" AUTOINCREMENT));					

Wir sprechen also die Datenbank an und sagen ihr zunächst, dass wir eine neue Tabelle erstellen wollen (CREATE TABLE). Anschließend führen wir die Spalten auf, die in dieser Tabelle enthalten sein sollen und definieren den Datentyp sowie weitere Eigenschaften für die darin enthaltenen Werte. Für uns ist hier vor allem die Eigenschaft NOT NULL und UNIQUE interessant. Am Ende der Definition geben wir zudem noch an, welches Feld unseren Primärschlüssel enthält. Hierbei fällt die Angabe AUTOINCREMENT auf.

NOT NULL als Eigenschaft zeigt an, dass in dieser Spalte für jeden Datensatz (also in jeder Zeile) ein Wert eingetragen sein muss. Bei einem Integer-Feld kann dies durchaus auch der Wert 0 sein.

Die Eigenschaft UNIQUE legt fest, dass der Wert dieser Spalte für jeden Datensatz einzigartig sein muss. Wir können also keine zwei Datensätze mit der gleichen BENUTZER\_ID abspeichern und es darf auch keine zwei Benutzer mit dem gleichen BENUTZERNAME geben.

Würden wir versuchen, einen Datensatz zu speichern, der diesen Regeln nicht entspricht, würde unsere Datenbank dies verweigern und eine entsprechende Fehlermeldung zurückgeben.

Die Angabe AUTOINCREMENT bei der Definition des Primärschlüssels weist uns auf ein interessantes Problem hin. Wenn wir neue Datensätze in eine Tabelle schreiben, sollen diese stets einen einzigartigen Primärschlüssel erhalten. Dies umzusetzen ist aber aus Sicht der Anwendung recht kompliziert, denn wir müssten ja zunächst schauen, welche Schlüssel bereits vergeben sind. Hier hilft uns die Angabe AUTOINCREMENT, welche die Vergabe für uns übernimmt. Wenn wir künftig einen neuen Datensatz in die Tabelle schreiben, können wir auf

die Angabe des Primärschlüssels verzichten und die Datenbank wird automatisch den nächsten freien Schlüssel vergeben, indem sie vom letzten vergebenen Schlüssel um eins hoch zählt.

Wir haben nun also gesehen, wie sich die einzelnen Spalten und der Primärschlüssel festgelegt werden. Um auch auf die Definition der Fremdschlüssel eingehen zu können, betrachten wir noch kurz den Code zu Erstellung der Tabelle Adressen.

CREATE	TABLE	"ADRESSEN"
("ADRESS_ID" INTEGER	NOT	NULL
"BENUTZER_ID"INTEGER	NOT	NULL,
"STRASSE" TEXT	NOT	NULL,
"HAUSNUMMER" INTEGER	NOT	NULL,
"PLZ" TEXT	NOT	NULL,
"STADT" TEXT	NOT	NULL,
PRIMARY KEY("ADRESS_ID")		AUTOINCREMENT),
FOREIGN KEY("BENUTZER_ID")		REFERENCES
BENUTZER(BENUTZER_ID) ON DELETE CASCADE);		

Hier legen wir in der letzten Zeile fest, dass die Spalte BENUTZER\_ID, ein Fremdschlüssel ist, welcher sich auf das Feld BENUTZER\_ID aus der Tabelle BENUTZER bezieht. Zudem legen wir mit der Angabe ON DELETE CASCADE fest, dass dieser Datensatz gelöscht werden soll, wenn der korrespondierende Datensatz in der Tabelle BENUTZER gelöscht wird, und können so direkt im Datenbankentwurf die Beziehung als Komposition definieren.

Am Beispiel der Postleitzahlen sehen wir, dass an einigen Stellen auch für Werte, die eigentlich in Zahlen ausgedrückt werden, der Datentyp Text verwendet wird. Dies ist notwendig, um sicherzustellen, dass auch eine führende 0 beim Speichern nicht verloren geht.

Bevor wir den Abschnitt zur Erstellung der Datenbank abschließen, möchte ich noch kurz auf die Produkt-Subklassen eingehen. Wie im Entwurf diskutiert, haben diese einen Primärschlüssel, der gleichzeitig ein Fremdschlüssel ist. Dementsprechend muss dort auf die Funktion von AUTOINCREMENT verzichtet werden. Wie wir die Angabe des korrekten Schlüssels realisieren, werden wir bei der Implementierung der entsprechenden Methode nochmals betrachten.

## 4.2.Codierung der Klassen und Methoden

Nachdem die Datenbank vorerst als Sammlung leerer Tabellen steht, wenden wir uns als nächstes der Codierung unseres objektorientierten Datenmodells zu. Hierzu werden wir betrachten, wie wir eine Klasse und die Vererbung definieren. Anschließend werden wir exemplarisch einen Blick auf einige der Methoden werfen. Da wir nun in den konkreten Quellcode einsteigen, sei an dieser Stelle noch gesagt, dass wir hier nur exemplarische Auszüge betrachten werden. Der gesammelte Quellcode findet sich in den Anlagen und kann zudem via GitHub unter <https://github.com/jmk74871/Laborbericht-PRG24> heruntergeladen werden.

Zu Beginn der Codierung müssen wir die einzelnen Klassen und Attribute, entsprechend unserem Klassendiagramm anlegen, was wir mit dem «class» Statement und dem folgenden Klassennamen tun können. Wenn die Klasse eine Superklasse hat, von der sie Attribute und Funktionen erbt, muss diese in den folgenden Klammern angegeben werden. Der Aufruf wird mit einem Doppelpunkt beendet.

```
10  class Benutzer:
11      def __init__(self, benutzer_name: str, password: str, warenhaus: Warenhaus) -> None:
12          super().__init__(benutzer_name, password)
13          self.__warenhaus = warenhaus
14          self.__vorname = None
15          self.__nachname = None
16          self.__adressen = []
17          self.__bankverbindungen = []
18          self.__warenkorb = self.__neuer_warenkorb()
19          self.__bestellhistorie = []
20
21          self.__einloggen()
22
23      if self._login_status:
24          self.__get_adressen_from_db()
25          self.__get_bankinfo_from_db()
26          self.__get_bestellungen_from_db()
```

Abbildung 1: class\_benutzer\_init.png

Um dieser Klasse nun Attribute zuzuordnen, legen wir als erstes einen Konstruktor an, der beim Instanziieren entsprechende Werte entgegennimmt und verarbeitet. In Python geschieht dies über die `__init__`-Methode. In den runden Klammern legen wir nun fest, welche Parameter beim Instanziieren eines Objektes vom Typ Benutzer übergeben werden müssen und welchem Datentyp diese entsprechen sollen. Der Parameter «`self`» ist hierbei die Referenz auf das Objekt selbst (und muss entsprechend nicht übergeben werden).

Ist eine Vererbung gegeben, kann zudem der Konstruktor der Superklasse aufgerufen werden, (`<<super().__init__()`) dem jene Parameter übergeben werden, die im Konstruktor der Superklasse zu verarbeiten sind.

Die im Konstruktor übergebenen Werte können wir nun in entsprechende Variablen, die diesem Objekt zugeordnet sind, ablegen. Dass die Variable dem Objekt zugeordnet ist, erkennen wir durch das vorangestellte `«self»`. Auch weitere Parameter, die z.B. statisch immer gegeben sein sollen, können wir an dieser Stelle hinterlegen, um später auf diese zuzugreifen. Hier legen wir zum Beispiel fest, dass die Parameter `«self.__vorname»` und `«self.__nachname»` zunächst leer bleiben. Auch Aufrufe zu Methoden sind an dieser Stelle möglich, wie wir am Code in Zeile 18 sehen können.

Beim Betrachten der Namen der Variablen sehen wir die vorangestellten Unterstriche. Ein doppelter Unterstrich markiert hier "private", ein einfacher "geschützte" Werte und Methoden. Da Python über kein unumgängliches System der Privatheit von Werten und Methoden verfügt, ist diese Definition aber nur begrenzt möglich. So ist bei "geschützten" Werten trotzdem ein Zugriff möglich, die meisten Entwicklungsumgebungen geben aber eine entsprechende Warnung (z.B. `«Access to a protected member of a class»`) aus.

Bei privaten Werten und Methoden erfolgt zur Laufzeit eine Umbenennung. So wird unser Parameter `self.__passwort` zur Laufzeit in `self._Benutzer__passwort` geändert. Es ist also theoretisch auch hier weiterhin ein Zugriff möglich, es wird aber anderen Personen, die am Quelltext arbeiten, signalisiert, dass dieser nicht vorgesehen ist. Weiteres kann der Python-Dokumentation unter Punkt 9.6 entnommen werden. [4]

Als nächstes wollen wir einen Blick auf das Anlegen von Methoden werfen. Hierzu nehmen wir wieder ein passendes Codebeispiel zu Hand.

```

10  class Kunde(Benutzer):
11      def __init__(self, benutzer_name: str, passwort: str, warenhaus: Warenhaus) -> None:
12          ...
13
14      # Öffentliche Schnittstellen:
15
16      def add_adresse(self, strasse: str, hausnummer: int, plz: str, stadt: str):
17          if self._login_status:
18              conn = sqlite3.connect(self._db_path)
19              cursor = conn.cursor()
20
21              # add to ADDRESS-DB
22              cursor.execute(
23                  f"INSERT INTO ADRESSEN (BENUTZER_ID, STRASSE, HAUSNUMMER, PLZ, STADT)"
24                  f"VALUES(:benutzer_id, :strasse, :hausnummer, :plz, :stadt);",
25                  {'benutzer_id': self._benutzer_id, 'strasse': strasse, 'hausnummer': hausnummer,
26                   'plz': plz, 'stadt': stadt})
27              conn.commit()
28
29              print(f'Adresse in der {strasse} {hausnummer} in {stadt} erfolgreich angelegt.')
30
31          else:
32              self._get_adressen_from_db()
33
34      def __get_adressen_from_db(self):
35          ...
36
37      def __einloggen(self):
38          ...

```

Abbildung 2: class\_benutzer\_methode.png

Wie auch beim Konstruktor, der im Prinzip selbst eine Methode ist, die immer beim Instanziieren eines Objekts ausgeführt wird, legen wir Methoden über das Signalwort «def» (für define) an und benennen die Parameter, welche übergeben werden sollen, in den runden Klammern. Auch hier haben wir die Möglichkeit festzulegen, welchen Datentyp ein jeder Parameter haben soll, und wir können einen Rückgabewert für die Methode hinterlegen, wie z.B. «() -> None» (entspricht «void» in Java). Anders als z.B. Java zwingt uns Python aber nicht dazu dies zu tun.

Variablen, welche im Methodenaufruf übergeben werden, können wir innerhalb der Methode verwenden. Im Beispiel sehen wir die Methode zur Speicherung einer neuen Adresse in der Datenbank. Hierzu greifen wir auf bereits erwähntes Modul sqlite3, um auf die Datenbank zuzugreifen und fügen die Variablen über Platzhalter in das entsprechende SQLite-Statement ein. Da wir in unserem Exempel keine Weboberfläche haben, die eine entsprechende Rückmeldung empfangen könnte, wurde hier auf einen Rückgabewert verzichtet. Es erfolgt aber eine entsprechende Ausgabe über die Konsole über die Funktion «print()» in Zeile 43.

An anderen Stellen haben wir jedoch entsprechende Rückgabewerte gegeben. So gibt beispielsweise die private Methode «self.\_neuer\_warenkorb()», das wir im Konstruktor gesehen haben, ein Objekt der Typs Bestellung zurück. Andere Methoden greifen direkt auf die privaten Attribute, um dort Werte zu hinterlegen. So hinterlegt z.B. die Methode «self.\_get\_adressen\_from\_db()» Objekte vom Typ Adresse in der Liste für das Attribut «self.\_adressen»

Bevor wir diesen Schnelldurchlauf zur Codierung von Klassen und Methoden abschließen, müssen wir noch ein letztes Thema betrachten, und zwar das Organisieren von Quelltext in verschiedene Dateien und Pakete. In Anlehnung an das Vorgehen in Java wurde der Quellcode für jede Klasse in einer eigenen Datei gespeichert, die nach der jeweiligen Klasse benannt ist (z.B. class\_kunde.py). Alle Dateien bilden gemeinsam ein Paket mit dem Namen webshop und liegen daher in einem entsprechend benannten Ordner. Für das Ausführen des Codes müssen also die entsprechenden Dateien geladen werden. Um nicht jede Klasse einzeln laden zu müssen, wurden entsprechende Angaben in der Datei \_\_init\_\_.py gemacht, die ebenfalls im Paketordner hinterlegt ist. In den einzelnen Dateien erfolgt jeweils ein Import für die jeweils benötigten Funktionalitäten aus den anderen Klassen oder weiterer Python-Module. Die so entstehenden Mehrfachimporte werden von Python-Interpreter zur Laufzeit abgefangen.

Das Main-Skript, welches auch Code für einen Testdurchlauf des Systems enthält, und das Skript zum Initialisieren der Datenbank liegen außerhalb des Pakets.

## 5. Schlusswort

Um ein zuverlässiges System zu erhalten, wäre noch mehr systematisches Testen und weitere Iterationen über Entwurf und Umsetzung nötig. Die Grundzüge lassen sich aber erkennen. So schreibt auch S. Kleuker in seinem Grundkurs Software-Engineering mit UML:

*Abhängig von den Erfahrungen des Analytikers werden die Anforderungen schrittweise in das Klassendiagramm eingebaut. Die UML unterstützt dabei den Ansatz, zunächst Informationen unpräzise zu spezifizieren und diese dann in folgenden Iterationen zu konkretisieren. [5]*

Die hier dargelegte Rekonstruktion kann also nur ein grober Entwurf sein, der keinesfalls Anspruch auf Vollständigkeit hat und noch weitere Iterationen benötigen würde, um einen tatsächlich verwertbaren Systementwurf zu ergeben.

Ideen hierzu wären bereits vorhanden: so ist das Speichern von Werten bisher zentralisiert in den Benutzerklassen geregelt. Eine „kaskadierende“ Speicherung erfolgt nur bei Bestellungen. Dies wäre ein potenzieller Ansatzpunkt zur Weiterentwicklung des Systems, der insbesondere die Erweiterbarkeit verbessern würde. Ein weiterer Ansatzpunkt wäre die Einführung von Enumerationen z.B. für den Bestellstatus, um hier zuverlässig für eindeutige Werte zu sorgen.

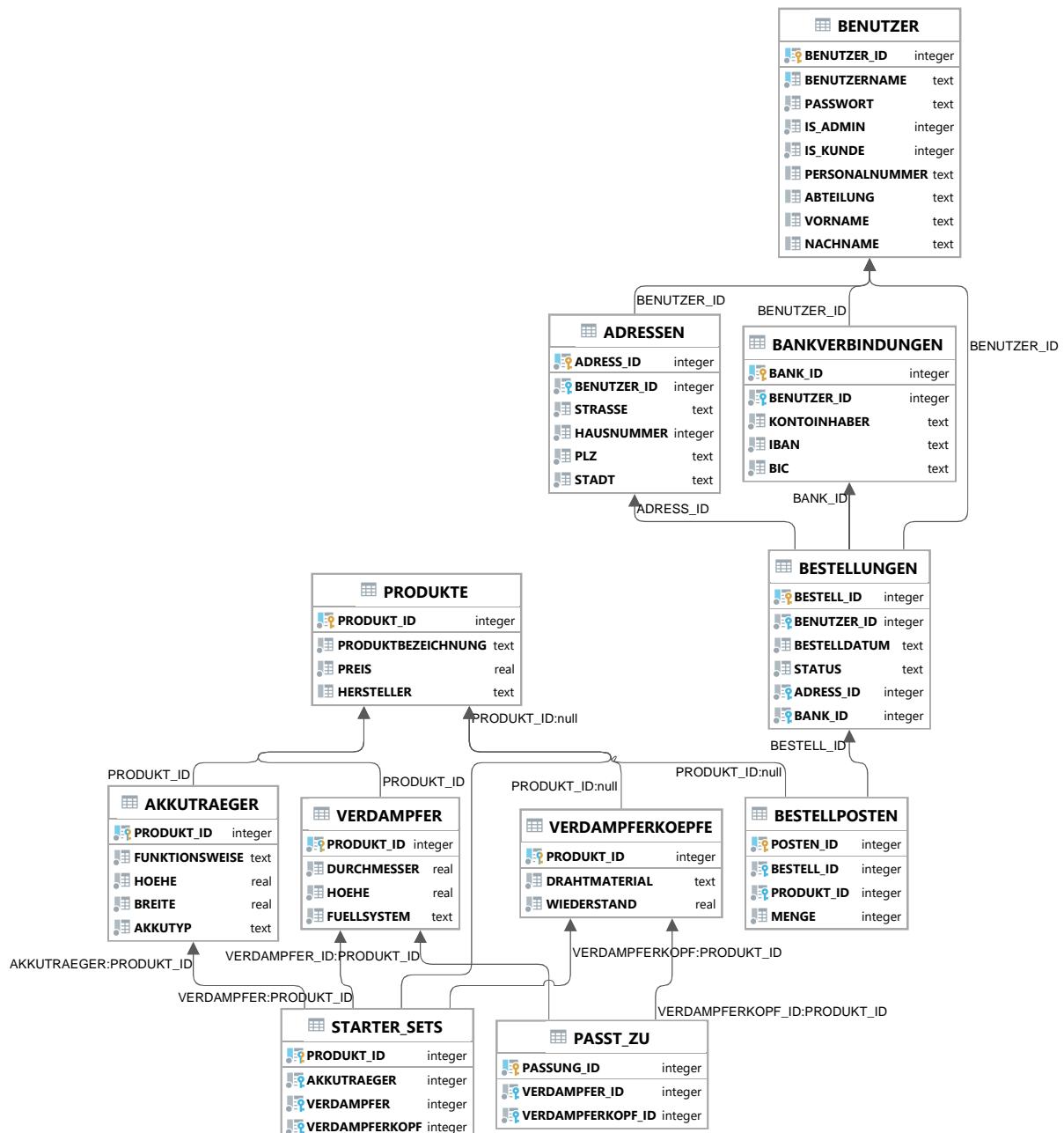
Für den Rahmen dieser Arbeit möchte ich hier jedoch den Schlusspunkt setzen.

## References

- [1] Wikipedia, *Model–view–presenter*. [Online]. Available: <https://en.wikipedia.org/w/index.php?title=Model–view–presenter&oldid=1019879801> (accessed: Oct. 4 2021).
- [2] Scott W. Ambler, *Mapping Objects to Relational Databases: O/R Mapping In Detail*. [Online]. Available: <http://www.agiledata.org/essays/mappingObjects.html#MappingInheritance> (accessed: Sep. 25 2021).
- [3] *sqlite3 — DB-API 2.0 interface for SQLite databases — Python 3.9.7 documentation*. [Online]. Available: <https://docs.python.org/3/library/sqlite3.html> (accessed: Sep. 24 2021).
- [4] *9. Classes — Python 3.9.7 documentation*. [Online]. Available: <https://docs.python.org/3/tutorial/classes.html#a-first-look-at-classes> (accessed: Sep. 24 2021).
- [5] S. Kleuker, *Grundkurs Software-Engineering mit UML: Der pragmatische Weg zu erfolgreichen Softwareprojekten*, 3rd ed. Wiesbaden: Springer Vieweg, 2013. [Online]. Available: <http://swbplus.bsz-bw.de/bsz392250349cov.htm>
- [6] H. Ernst, J. Schmidt, and G. Beneken, *Grundkurs Informatik*. Wiesbaden: Springer Fachmedien Wiesbaden, 2020.
- [7] J. Goll, *Methoden und Architekturen der Softwaretechnik*, 1st ed. Wiesbaden: Vieweg + Teubner, 2011.
- [8] B. Rumpe, *Modellierung mit UML*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.
- [9] E. Schicker, *Datenbanken und SQL*. Wiesbaden: Springer Fachmedien Wiesbaden, 2014.
- [10] C. Wagenknecht, *Programmierparadigmen*. Wiesbaden: Springer Fachmedien Wiesbaden, 2016.
- [11] R. Steyer, *Programmierung in Python*. Wiesbaden: Springer Fachmedien Wiesbaden, 2018.
- [12] H. J. van Randen, C. Bercker, and J. Fiendl, *Einführung in UML*. Wiesbaden: Springer Fachmedien Wiesbaden, 2016.

## Anlagen

## A. Schaubild zum Datenbankentwurf



## B. Darstellung der einzelnen Datenbanktabellen

TABELLE BENUTZER

BENUTZER_ID	BENUTZERNAME	PASSWORT	IS_ADMIN	IS_KUNDE	PERSONALNUMMERN	ABTEILUNG	VORNAME	NACHNAME
1	admin	admin	1	0	SHOPADMIN	IT		
2	PeterV	pass234	0	1			Peter	Vogel
3	MaraM	56741	0	1			Mara	Musterfrau

TABELLE VERDAMPFERKOEPEF

PRODUKT_ID	DRAHTMATERIAL	WIEDERSTAND
5	Stainless Steel	0,5
6	Stainless Steel Mesh	0,33

TABELLE STARTERSET

PRODUKT_ID	AKKUTRAEGER	VERDAMPFER	VERDAMPFERKOPF
7	4	1	6

TABELLE BESTELLPOSTEN

POSTEN_ID	BESTELL_ID	PRODUKT_ID	MENGE
1	1	1	1
2		4	1
3		5	2
4	1	6	2
5	2	7	1
6	2	6	5



TABELLE PASST ZU

PASSUNG_ID	VERDAMPFER_ID	VERDAMPFERKOPF_ID
1	2	5
2	2	6

TABELLE AKKUTAEGER

PRODUKT_ID	FUNKTIONSWEISE	HOEHE	BREITE	AKKUTYP
3	geregelt	5,4	3,2	2 x 18650er Standard Akku
4	geregelt	4,8	3,1	18650er Standard Akku

TABELLE BANKVERBINDUNGEN

BANK_ID	BENUTZER_ID	KONTODRIBER	IBAN	BIC
1	2	Peter Vogel	DE42XXX7394	STVU7DRE
2	3	Mara Musterfrau	DE42XXX9327	STVU7DRE

TABELLE ADRESSEN

ADRESS_ID	BENUTZER_ID	STRASSE	HAUSNUMM	PLZ	STADT
1	2	Stuttgarter Str.	3	74700	Stuttgart
2	3	Reutlinger Str.	15	74700	Stuttgart

TABELLE BESTELLUNGEN

BESTELL_ID	BENUTZER_ID	BESTELLDATUM	STATUS	ADDRESS_ID	BANK_ID
1	2	2021-10-05 23:37:42,385	in Bearbeitung	1	1
2	3	2021-10-05 23:42:28,596	in Bearbeitung	2	2

TABELLE VERDAMPFER

PRODUKT_ID	DURCHMESSER	HOEHE	FUELLSYSTEM
1	2,8	2,74	top fill
2	2,8	2,74	top fill



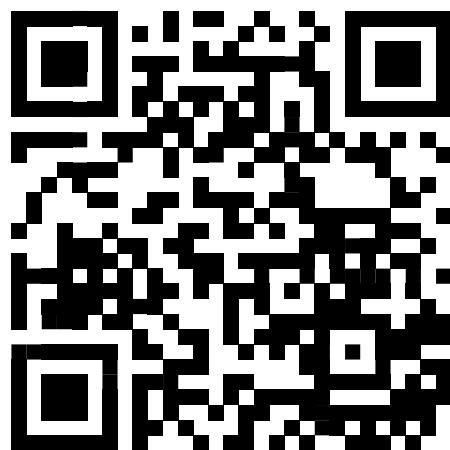
TABELLE PRODUKTE

PRODUKT_ID	PRODUKTBEZEICHNUNG	PREIS	HERSTELLER
1	SuperVape24	8,99	VaperG
2	Nebelmaschine V3	9,99	B-Vape
3	Powerbank24	15,99	aspire
4	Glint II	9,99	aspire
5	B-Vape V3	0,43	B-Vape
6	B-Vape V3-Mesh	0,6	B-Vape
7	Easy Starter Set	12,99	VaperG + aspire

	FREMDSCHLÜSSEL
	PRIMÄRSCHLÜSSEL
PS + FS	

## C. Quellcode

Der Quellcode zur Arbeit ist auch unter <https://github.com/jmk74871/Laborbericht-PRG24> verfügbar.



```
1 from . import class_administrator
2 # from . import class_adresse
3 # from . import class_akkutaeger
4 # from . import class_bankverbindung
5 # from . import class_benutzer
6 # from . import class_bestellposten
7 # from . import class_bestellung
8 from . import class_kunde
9 # from . import class_produkt
10 # from . import class_starterset
11 # from . import class_verdampfer
12 # from . import class_verdampferkopf
13 from. import class_warenhaus
14
```

```
1 import sqlite3
2
3 from webshop.class_adresse import Adresse
4 from webshop.class_bankverbindung import Bankverbindung
5 from webshop.class_benutzer import Benutzer
6 from webshop.class_bestellung import Bestellung
7 from webshop.class_warenhaus import Warenhaus
8
9
10 class Kunde(Benutzer):
11     def __init__(self, benutzer_name: str, passwort: str,
12                  warenhaus: Warenhaus) -> None:
13         super().__init__(benutzer_name, passwort)
14         self.__warenhaus = warenhaus
15         self.__vorname = None
16         self.__nachname = None
17         self.__adressen = []
18         self.__bankverbindungen = []
19         self.__warenkorb = self.__neuer_warenkorb()
20         self.__bestellhistorie = []
21
22         self.__einloggen()
23
24         if self._login_status:
25             self.__get_adressen_from_db()
26             self.__get_bankinfo_from_db()
27             self.__get_bestellungen_from_db()
28
29         # Öffentliche Schnittstellen:
30
31     def add_adresse(self, strasse: str, hausnummer: int,
32                    plz: str, stadt: str) -> None:
33         if self._login_status:
34             conn = sqlite3.connect(self._db_path)
35             cursor = conn.cursor()
36
37             # add to ADDRESS-DB
38             cursor.execute(
39                 f"INSERT INTO ADRESSEN (BENUTZER_ID,
40                 STRASSE, HAUSNUMMER, PLZ, STADT)"
41                 f"VALUES(:benutzer_id, :strasse, :
42                 hausnummer, :plz, :stadt);",
43                 {'benutzer_id': self._benutzer_id, 'strasse': strasse, 'hausnummer': hausnummer,
```

```

40                 'plz': plz, 'stadt': stadt})
41             conn.commit()
42
43             print(f'Adresse in der {strasse} {hausnummer}
44             in {stadt} erfolgreich angelegt.')
45
46             self.__get_adressen_from_db()
47         else:
48             self.__einloggen()
49
50     def show_adressen(self) -> None:
51         if self._login_status:
52             displaystring = ''
53             for adresse in self.__adressen:
54                 displaystring += adresse.get_adressinfo()
55             print(displaystring)
56         else:
57             self.__einloggen()
58
59     def delete_adresse(self, adress_id: int) -> None:
60         if self._login_status:
61             conn = sqlite3.connect(self._db_path)
62             cursor = conn.cursor()
63             cursor.execute('DELETE from ADRESSEN WHERE
64 ADDRESS_ID = :adress_id',
65                           {'adress_id': adress_id})
66             conn.commit()
67             conn.close()
68         else:
69             self.__einloggen()
70
71     def add_bankverbindung(self, kontoinhaber: str, iban:
72 str, bic: str) -> None:
73         if self._login_status:
74             conn = sqlite3.connect(self._db_path)
75             cursor = conn.cursor()
76
77             # add to ADRESS-DB
78             cursor.execute(
79                 f"INSERT INTO BANKVERBINDUNGEN (
80                 BENUTZER_ID, KONTOINHABER, IBAN, BIC)"
81                 f"VALUES(:benutzer_id, :kontoinhaber, :
82 iban, :bic);",
83                 {'benutzer_id': self._benutzer_id, '

```

```
78 kontoinhaber': kontoinhaber, 'iban': iban, 'bic': bic})
79             conn.commit()
80
81             print(f'Bankverbindung mit der IBAN endend
82 auf {iban[-4:]} erfolgreich angelegt.')
83             self.__get_bankinfo_from_db()
84         else:
85             self.__einloggen()
86
87     def show_bankverbindungen(self) -> None:
88         if self._login_status:
89             displaystring = ''
90             for bankverbindung in self.__bankverbindungen
91                 :
92                 displaystring += bankverbindung.
93                 get_bankinfo()
94             print(displaystring)
95         else:
96             self.__einloggen()
97
98     def delete_bankverbindungen(self, bank_id: int) ->
99     None:
100         if self._login_status:
101             conn = sqlite3.connect(self._db_path)
102             cursor = conn.cursor()
103             cursor.execute('DELETE from BANKVERBINDUNGEN
104 WHERE BANK_ID = :bank_id AND BENUTZER_ID = :benutzer_id',
105                           {'bank_id': bank_id, '
106                           'benutzer_id': self._benutzer_id})
107             conn.commit()
108             conn.close()
109         else:
110             self.__einloggen()
111
112     def add_to_warenkorb(self, produkt_id: int, menge:
113     int) -> None:
114         self.__warenkorb.__add_bestellposten(produkt_id=
115         produkt_id, menge=menge)
116
117     def show_warenkorb(self) -> None:
118         print(self.__warenkorb.__show_bestellposten())
119
120     def delete_from_warenkorb(self, produkt_id: int) ->
```

```
113     None:
114         self._warenkorb._delete_bestellposten(produkt_id
115 =produkt_id)
116     def bestellung_aufgeben(self, adress_id: int, bank_id
117 : int) -> None:
118         if self._login_status:
119             self._warenkorb._save_to_db(self.
120 _benutzer_id, adress_id, bank_id)
121         self._warenkorb = self._neuer_warenkorb()
122     else:
123         self._einloggen()
124
125     def show_kundendaten(self) -> None:
126         print(f'\nIhre Daten lauten: \nVorname: {self.
127 __vorname}\nNachname:{self.__nachname}'
128             f'\nIhr Benutzername lautet: {self.
129 _benutzername}')
130
131     def update_kundendaten(self, vorname=None, nachname=
132 None) -> None:
133         if self._login_status:
134             if vorname is None:
135                 vorname = self.__vorname
136             if nachname is None:
137                 nachname = self.__nachname
138             conn = sqlite3.connect(self._db_path)
139             cursor = conn.cursor()
140             cursor.execute(
141                 'UPDATE BENUTZER SET VORNAME = :vorname,
142                 NACHNAME = :nachname WHERE BENUTZER_ID = :benutzer_id;',
143                 {'vorname': vorname, 'nachname': nachname
144 , 'benutzer_id': self._benutzer_id})
145             conn.commit()
146
147             conn.row_factory = sqlite3.Row
148             cursor = conn.cursor()
149
150             cursor.execute("SELECT * from BENUTZER WHERE
151 BENUTZER_ID = :benutzer_id",
152                         {"benutzer_id": self.
153 _benutzer_id})
```

```

147             res = cursor.fetchall()
148             conn.close()
149
150         if len(res) == 1 and bool(res[0]['IS_KUNDE']):
151             ds = res[0]
152             self.__vorname = ds['VORNAME']
153             self.__nachname = ds['NACHNAME']
154         else:
155             self.__einloggen()
156
157     def delete_kundendaten(self) -> None:
158         if self._login_status:
159             conn = sqlite3.connect(self._db_path)
160             cursor = conn.cursor()
161             cursor.execute('DELETE from BENUTZER WHERE
BENUTZER_ID = :benutzer_id',
162                           {'benutzer_id': self.
_benutzer_id})
163             conn.commit()
164             conn.close()
165
166         self._login_status = False
167         print(f'\nIhre Kundenkonto mit dem
Benutzernamen {self._benutzername} wurde erfolgreich
gelöscht.\n')
168     else:
169         self.__einloggen()
170
171     # Interne Methoden:
172
173     def __einloggen(self) -> None:
174         if self._benutzer_einloggen():
175             conn = sqlite3.connect(self._db_path)
176             conn.row_factory = sqlite3.Row
177             cursor = conn.cursor()
178
179             cursor.execute("SELECT * from BENUTZER WHERE
BENUTZER_ID = :benutzer_id",
180                           {"benutzer_id": self.
_benutzer_id})
181
182             res = cursor.fetchall()
183

```

```

184             if len(res) == 1 and bool(res[0]['IS_KUNDE'])
185                 ]):
186                     ds = res[0]
187                     self._vorname = ds['VORNAME']
188                     self._nachname = ds['NACHNAME']
189                     self._login_status = True
190                     print(f'\nWillkommen {self._benutzername}\nIhre Anmeldung war erfolgreich!\n')
191             else:
192                     print('Login fehlgeschlagen! Bitte
193 stellen Sie sicher, dass Sie das richtige Anmeldeportal
194 verwenden.')
195             else:
196                     self._create_new_account()
197
198     def __create_new_account(self) -> None:
199
200         new_account = input('Möchten Sie ein neues
201 Kundenkonto anlegen? (ja/nein)')
202         if new_account == 'ja':
203             vorname = input('Geben Sie ihren Vornamen an
204 :')
205             nachname = input('Geben Sie ihren Nachnamen
206 an:')
207
208             self._benutzername = None
209             while self._benutzername is None:
210                 check_name = input('Bitte wählen Sie
211 einen Benutzernamen aus.')
212                 if self._check_verfuegbar_benutzername(
213                     check_name):
214                     self._benutzername = check_name
215                 else:
216                     print('Benutzername ist bereits
217 vergeben.')
218
219             passwort = input('Geben Sie ein Passwort an:'
220 )
221
222             conn = sqlite3.connect(self._db_path)
223             cursor = conn.cursor()
224
225             # add to BENUTZER-TABLE:
226             cursor.execute("INSERT INTO BENUTZER (

```

```
216 BENUTZERNAME, PASSWORT, IS_ADMIN, IS_KUNDE, VORNAME,  
NACHNAME) "  
217                                     "VALUES(:benutzername, :  
passwort, :is_admin, :is_kunde, :vorname, :nachname)",  
218                                     {'benutzername': self.  
_benutzername, 'password': password, 'is_admin': False,  
219                                     'is_kunde': True,  
220                                     'vorname': vorname, 'nachname  
': nachname})  
221                                     print(f'{vorname} {nachname} wurde als Kunde  
mit dem Benutzernamen {self._benutzername} angelegt.')222  
223                                     conn.commit()  
224                                     conn.close()  
225  
226                                     self.__einloggen()  
227  
228     def __get_adressen_from_db(self) -> None:  
229  
230         self.__adressen = []  
231  
232         conn = sqlite3.connect(self._db_path)  
233         conn.row_factory = sqlite3.Row  
234         cursor = conn.cursor()  
235  
236         cursor.execute("SELECT * from ADRESSEN WHERE  
BENUTZER_ID = :benutzer_id;", {'benutzer_id': self.  
_benutzer_id, })  
237  
238         res = cursor.fetchall()  
239         conn.close()  
240  
241         for ds in res:  
242             adresse = Adresse(ds['ADDRESS_ID'], ds['  
STRASSE'], ds['HAUSNUMMER'], ds['PLZ'], ds['STADT'])  
243             self.__adressen.append(adresse)  
244  
245     def __get_bankinfo_from_db(self) -> None:  
246         self.__bankverbindungen = []  
247  
248         conn = sqlite3.connect(self._db_path)  
249         conn.row_factory = sqlite3.Row  
250         cursor = conn.cursor()
```

```

252         cursor.execute("SELECT * from BANKVERBINDUNGEN
253             WHERE BENUTZER_ID = :benutzer_id;",
254             {'benutzer_id': self._benutzer_id
255             , })
256
257
258     for ds in res:
259         bankverbindung = Bankverbindung(ds['BANK_ID'
260             ], ds['KONTOINHABER'], ds['IBAN'], ds['BIC'])
261         self.__bankverbindungen.append(bankverbindung
262             )
263
264
265     def __get_bestellungen_from_db(self) -> None:
266         self.__bestellhistorie = []
267
268         conn = sqlite3.connect(self._db_path)
269         conn.row_factory = sqlite3.Row
270         cursor = conn.cursor()
271
272         cursor.execute("SELECT * from BESTELLUNGEN WHERE
273             BENUTZER_ID = :benutzer_id;",
274             {'benutzer_id': self._benutzer_id
275             })
276
277         res = cursor.fetchall()
278         conn.close()
279
280         for ds in res:
281             bestellung = Bestellung(db_path=self._db_path
282             , bestell_id=ds['BESTELL_ID'],
283                 bestelldatum=ds['
284                 BESTELLDATUM'], bestellstatus=ds['STATUS'],
285                 warenhaus=self.
286                 __warenhaus)
287             self.__bestellhistorie.append(bestellung)
288
289
290     def __neuer_warenkorb(self) -> Bestellung:
291         return Bestellung(db_path=self._db_path,
292             warenhaus=self.__warenhaus)
293

```

```
1 import os
2
3 class Adresse():
4
5     def __init__(self, adress_id: int, strasse: str,
6      hausnummer: int, plz: str, stadt: str):
7         self.__adress_id = int(adress_id)
8         self.__strasse = str(strasse)
9         self.__hausnummer = int(hausnummer)
10        self.__plz = str(plz)
11        self.__stadt = str(stadt)
12
13    def get_adressinfo(self) -> str:
14        return f'\n{self.__strasse} {self.__hausnummer}\n{self.__plz} {self.__stadt}\nID: {self.__adress_id}\n'
15
16
17
18
19
```

```
1 import sqlite3
2 from webshop.class_administrator import Administrator
3
4
5 class Produkt():
6
7     def __init__(self, produktbezeichnung: str, preis:
8         float, hersteller: str, produkt_id: int):
9             self._produktbezeichnung = str(produktbezeichnung)
10            self._hersteller = str(hersteller)
11            self._preis = float(preis)
12            if produkt_id is not None:
13                self._produkt_id = int(produkt_id)
14            else:
15                self._produkt_id = None
16
17    def get_price(self) -> float:
18        return self._preis
19
20    def get_produktbezeichnung(self) -> str:
21        return self._produktbezeichnung
22
23    def get_produkt_id(self) -> int:
24        return self._produkt_id
```

```
1 import sqlite3
2
3
4 class Benutzer():
5     def __init__(self, benutzername: str, passwort: str):
6
7         self._benutzername = str(benutzername)
8         self._passwort = str(passwort)
9
10        self._db_path = 'test_db.db'
11        self._login_status = False
12        self._benutzer_id = None
13
14    def ausloggen(self) -> None:
15        self._login_status = False
16
17    def _benutzer_einloggen(self) -> bool:
18        conn = sqlite3.connect(self._db_path)
19        conn.row_factory = sqlite3.Row
20        cursor = conn.cursor()
21
22        cursor.execute("SELECT * from BENUTZER WHERE
BENUTZERNAME = :name AND PASSWORT = :pw",
23                       {"name": self._benutzername, "pw": self._passwort})
24
25        res = cursor.fetchall()
26
27        if len(res) == 1:
28            ds = res[0]
29            self._benutzer_id = ds['BENUTZER_ID']
30            return True
31        else:
32            print('Login fehlgeschlagen! Bitte lege ein
passendes Benutzerkonto an.')
33            return False
34
35    def _check_verfuegbar_benutzername(self, name_to_check
: str) -> bool:
36        conn = sqlite3.connect(self._db_path)
37        conn.row_factory = sqlite3.Row
38        cursor = conn.cursor()
39
40        cursor.execute("SELECT * from BENUTZER WHERE
```

```
40 BENUTZERNAME = :name", {"name": name_to_check})
41         res = cursor.fetchall()
42
43     if len(res) == 0:
44         conn.close()
45         return True
46     else:
47         conn.close()
48         return False
49
```

```

1 import sqlite3
2 from datetime import datetime
3 from webshop.class_verdampfer import Verdampfer
4 from webshop.class_akkutaege import Akkutraege
5 from webshop.class_verdampferkopf import Verdampferkopf
6 from webshop.class_starterset import Starterset
7
8
9 class Warenhaus():
10
11     def __init__(self):
12         self.__db_path = 'test_db.db'
13         self.__katalog = dict()
14
15         self.__last_update = None
16
17         if self.__last_update is None:
18             self.__build_katalog()
19
20     # öffentliche Schnittstellen
21
22     def get_by_id(self, id: int) -> object:
23         if self.__chek_if_update_needed():
24             try:
25                 return self.__katalog[id]
26             except KeyError:
27                 print('Item not found in catalog.')
28
29         def display_produktinfo(self, verdampfer: bool =False
30 , verdampferkoepfe: bool =False, akkutraege: bool =False,
31                         startersets: bool =False, all
32 : bool =True) -> None:
33
34             if self.__chek_if_update_needed():
35                 if all:
36                     for product in self.__katalog.values():
37                         if isinstance(product, Starterset):
38                             info = product.get_info()
39                             print(info['head'])
40                             for id in info['components']:
41                                 print(self.get_by_id(id).
42                                     get_info())
43                                     print(info['tail'])
44                 else:

```

```
42                     print(product.get_info())
43             else:
44                 for product in self.__katalog.values():
45                     if verdampfer and isinstance(product,
46                         Verdampfer):
47                         print(product.get_info())
48                     elif verdampferkoepfe and isinstance(
49                         product, Verdampferkopf):
50                         print(product.get_info())
51                     elif akkutraeger and isinstance(
52                         product, Akkutraeger):
53                         print(product.get_info())
54                     elif startersets and isinstance(
55                         product, Starterset):
56                         info = product.get_info()
57                         print(info['head'])
58                         for id in info['components']:
59                             print(self.get_by_id(id).
60                               get_info())
61                         print(info['tail'])
62
63     # interne Methoden
64
65     def __build_katalog(self) -> None:
66
67         conn = sqlite3.connect(self.__db_path)
68         conn.row_factory = sqlite3.Row
69         cursor = conn.cursor()
70
71         cursor.execute("SELECT * from PRODUKTE;")
72         produkte = cursor.fetchall()
73
74         cursor.execute("SELECT * from VERDAMPFER;")
75         verdampfer = cursor.fetchall()
76
77         cursor.execute("SELECT * from VERDAMPFERKOEPFE;")
78         verdampferkoepfe = cursor.fetchall()
79
80         cursor.execute("SELECT * from AKKUTRAEGER;")
81         akkutraeger = cursor.fetchall()
82
83         cursor.execute("SELECT * from STARTER_SETS;")
84         startersets = cursor.fetchall()
```

```

81         cursor.execute("SELECT * FROM PASST_ZU;")
82         passungen = cursor.fetchall()
83
84         conn.close()
85
86         p_ds = dict()
87         for ds in produkte:
88             p_id = ds['PRODUKT_ID']
89             p_ds[p_id] = {'produktbezeichnung': ds['
PRODUKTBEZEICHNUNG'], 'preis': ds['PREIS'],
90                           'hersteller': ds['Hersteller']}}
91
92         for ds in verdampfer:
93             id = ds['PRODUKT_ID']
94             matches = [p['VERDAMPFERKOPF_ID'] for p in
passungen if p['VERDAMPFER_ID'] == id]
95             prod = Verdampfer(produkt_id=id,
produktbezeichnung=p_ds[id]['produktbezeichnung'], preis=
p_ds[id]['preis'],
96                           hersteller=p_ds[id]['
hersteller'], durchmesser=ds['DURCHMESSER'],
97                           hoehe=ds['HOEHE'],
fuellsystem=ds['FUELLSYSTEM'], passende_produkte=matches)
98             self.__katalog[id] = prod
99
100        for ds in verdampferkoepfe:
101            id = ds['PRODUKT_ID']
102            matches = [p['VERDAMPFER_ID'] for p in
passungen if p['VERDAMPFERKOPF_ID'] == id]
103            prod = Verdampferkopf(produkt_id=id,
produktbezeichnung=p_ds[id]['produktbezeichnung'],
104                           preis=p_ds[id]['preis'],
105                           hersteller=p_ds[id]['
hersteller'], drahtmaterial=ds['DRAHTMATERIAL'],
106                           wiederstand=ds['
WIEDERSTAND'], passende_produkte=matches)
107            self.__katalog[id] = prod
108
109        for ds in akkutaeger:
110            id = ds['PRODUKT_ID']
111            prod = Akkutraeger(produkt_id=id,
produktbezeichnung=p_ds[id]['produktbezeichnung'],
112                           preis=p_ds[id]['preis'],

```



```
1 import sqlite3
2
3 from webshop.class_produkt import Produkt
4
5 class Akkutraeger(Produkt):
6
7     def __init__(self, produktbezeichnung: str, preis:
8         float, hersteller: str, funktionsweise: str, hoehe: float
9         , breite: float, akkutyp: str, produkt_id=None):
10        super().__init__(produkt_id=produkt_id,
11                        produktbezeichnung=produktbezeichnung, preis=preis,
12                        hersteller=hersteller)
13        self.__funktionsweise = str(funktionsweise)
14        self.__hoehe = float(hoehe)
15        self.__breite = float(breite)
16        self.__akkutyp = str(akkutyp)
17
18    def get_info(self) -> str:
19        return f'Akkuträger {self._produktbezeichnung} mit
20              einer Breite von {self.__breite} mm und Höhe ' \
21                  f'von {self.__hoehe}mm.\n  Preis:{self.
22 _preis} /  Produkt-ID: {self._produkt_id}'
```

```

1 from datetime import datetime
2 import sqlite3
3 from webshop.class_bestellposten import Bestellposten
4 from webshop.class_warenhaus import Warenhaus
5
6
7 class Bestellung():
8
9     def __init__(self, db_path: str, warenhaus: Warenhaus
10      , bestell_id=None, bestelldatum=None, bestellstatus='offen
11      '):
12         # todo: add enum for bestellstatus, also needs a
13         # change in the DB (Text to Integer)
14         self.__warenhaus = warenhaus
15         self.__bestell_id = bestell_id
16         self.__bestelldatum = bestelldatum
17         self.__bestellstatus = bestellstatus
18         self.__bestellposten = []
19         self.__db_path = db_path
20
21         if self.__bestell_id is not None:
22             self.__get_bestellposten_from_db()
23
24         # öffentliche Schnittstellen - sollen über andere
25         # Klassen angesprochen werden
26
27     def _add_bestellposten(self, produkt_id: int, menge:
28         int) -> None:
29         if self.__bestellstatus == 'offen' and self.
30         __warenhaus._check_exist(produkt_id):
31             posten = Bestellposten(produkt_id=produkt_id,
32             menge=menge, warenhaus=self.__warenhaus,
33                                         db_path=self.__db_path)
34             self.__bestellposten.append(posten)
35         else:
36             print(f'Das Produkt mit der ID {produkt_id}
37             konnte nicht im Produktkatalog gefunden werden.')
38
39     def _show_bestellposten(self) -> str:
40         returnstring = ''
41         returnstring += '\nIhr aktueller Warenkorb enthält
42         :'
43         gesamtpreis = 0
44         for bestellposten in self.__bestellposten:

```

```

36             returnstring += bestellposten.get_info()
37             gesamtpreis += bestellposten.get_total()
38
39             returnstring += f'\n\nDer Gesamtpreis aller
Produkte im aktuellen Warenkorbes beträgt: {gesamtpreis:.2f}€'
40
41         return returnstring
42
43     def _delete_bestellposten(self, produkt_id: int) ->
None:
44         self.__bestellposten = [posten for posten in self.
__bestellposten if posten.get_produkt_id() != produkt_id]
45
46     def _save_to_db(self, benutzer_id: int, adress_id: int
, bank_id: int) -> None:
47         self.__bestelldatum = datetime.now()
48
49         conn = sqlite3.connect(self.__db_path)
50         cursor = conn.cursor()
51
52         # add to BESTELLUNGEN-DB
53         cursor.execute(
54             f"INSERT INTO BESTELLUNGEN (BENUTZER_ID,
BESTELLDATUM, STATUS, ADRESS_ID, BANK_ID) "
55             f"VALUES(:benutzer_id, :bestelldatum, :status
, :adress_id, :bank_id);",
56             {'benutzer_id': benutzer_id, 'bestelldatum':
self.__bestelldatum, 'status': 'in Bearbeitung',
57             'adress_id': adress_id, 'bank_id': bank_id})
58         conn.commit()
59
60         self.__bestell_id = cursor.lastrowid
61
62         conn.close()
63
64         self.__bestellstatus = 'in Bearbeitung'
65
66         for posten in self.__bestellposten:
67             posten._save_to_db(bestell_id=self.
__bestell_id)
68
69         date = '%d.%m.%Y'
70         time = '%H:%M'

```

```
71         print(f'Ihre Bestellung wurde am {self.  
__bestelldatum.strftime(date)} um {self.__bestelldatum.  
strftime(time)} aufgegeben.')
72
73     # interne Methoden
74
75     def __get_bestellposten_from_db(self) -> None:
76         self.__bestellposten = []
77
78         conn = sqlite3.connect(self.__db_path)
79         conn.row_factory = sqlite3.Row
80         cursor = conn.cursor()
81
82         cursor.execute("SELECT * from BESTELLPOSTEN WHERE  
BESTELL_ID = :bestell_id;",
83                         {'bestell_id': self.__bestell_id})
84
85         res = cursor.fetchall()
86         conn.close()
87
88         for ds in res:
89             bestellposten = Bestellposten(posten_id=ds['  
POSTEN_ID'], produkt_id=ds['PRODUKT_ID'], menge=ds['MENGE  
'],
90                                         warenhaus=self.  
__warenhaus, db_path=self.__db_path)
91             self.__bestellposten.append(bestellposten)
92
```

```
1 from webshop.class_produkt import Produkt
2
3
4 class Starterset(Produkt):
5
6     def __init__(self, produkt_id: int, produktbezeichnung
7      : str, preis: float, hersteller: str, akkutraeger: int,
8                  verdampfer: int, verdampferkopf: int):
9         super().__init__(produkt_id=produkt_id,
10                      produktbezeichnung=produktbezeichnung, preis=preis,
11                      hersteller=hersteller)
12         self.__akkutaeger = akkutraeger
13         self.__verdampfer = verdampfer
14         self.__verdampferkopf = verdampferkopf
15
16     def get_info(self) -> dict:
17         return {'head': f'\nDas Set {self.
18                     _produktbezeichnung} enthält:',
19                 'components': [self.__akkutaeger, self.
20                               __verdampfer, self.__verdampferkopf],
21                 'tail': f'Setpreis: {self._preis}€ / Set
22 -Produkt-ID: {self._produkt_id}\n'}
```

```
1 from webshop.class_produkt import Produkt
2
3
4 class Verdampfer(Produkt):
5
6     def __init__(self, produktbezeichnung: str, preis:
7         float, hersteller: str, durchmesser: float, hoehe: float,
8             fuellsystem: str, passende_produkte: list
9             , produkt_id: int):
10            super().__init__(produktbezeichnung, preis,
11                hersteller, produkt_id)
12            self.__hoehe = float(hoehe)
13            self.__durchmesser = float(durchmesser)
14            self.__fuellsystem = str(fuellsystem)
15            self.__passende_produkte = list(passende_produkte)
16
17        def get_info(self) -> str:
18            return f'Verdampfer {self._produktbezeichnung} mit
19            einem Durchmesser von {self.__durchmesser}mm und Höhe ' \
20                  f'von {self.__hoehe}mm.\n  Preis:{self.
21 _preis} /  Produkt-ID: {self._produkt_id}'
```

```

1 from webshop.class_benutzer import Benutzer
2 import sqlite3
3
4
5 class Administrator(Benutzer):
6     def __init__(self, benutzer_name: str, passwort: str):
7         super().__init__(benutzer_name, passwort)
8         self.__personal_nummer = None
9         self.__abteilung = None
10        self.__einloggen()
11
12    def add_verdampfer(self, produktbezeichnung: str,
13                        preis: float, hersteller: str, durchmesser: float, hoehe:
14                        float,
15                        fuellsystem: str) -> None:
16
17        # write to product table and get produkt_id:
18        produkt_id = self.__save_to_prod_db(
19            produktbezeichnung, preis, hersteller)
20
21        if produkt_id > 0:
22            conn = sqlite3.connect(self._db_path)
23            cursor = conn.cursor()
24
25            # add to VERDAMPFER-DB
26            cursor.execute(
27                f"INSERT INTO VERDAMPFER (PRODUKT_ID,
28 DURCHMESSER, HOEHE, FUELLSYSTEM)"
29                f"VALUES(:produkt_id, :durchmesser, :hoehe
30 , :fuellsystem);",
31                {'produkt_id': produkt_id, 'durchmesser':
32 durchmesser, 'hoehe': hoehe, 'fuellsystem': fuellsystem
33 })
34            conn.commit()
35
36            print(f'{produktbezeichnung} saved to db with
37 id: {produkt_id}')
38        else:
39            print('Insertion to db failed!')
40
41    def add_verdampferkopf(self, produktbezeichnung: str,
42                          preis: float, hersteller: str, drahtmaterial: str,
43                          wiederstand: float) -> None:
44
45        # write to product table and get produkt_id:

```

```
36     produkt_id = self.__save_to_prod_db(
37         produktbezeichnung, preis, hersteller)
38
39     if produkt_id > 0:
40         conn = sqlite3.connect(self._db_path)
41         cursor = conn.cursor()
42
43         # add to VERDAMPFERKOEPFE-DB
44         cursor.execute(
45             f"INSERT INTO VERDAMPFERKOEPFE(PRODUKT_ID
46 , DRAHTMATERIAL, WIEDERSTAND) "
47             f"VALUES(:produkt_id, :drahtmaterial, :
48 wiederstand);",
49             {'produkt_id': produkt_id, 'drahtmaterial':
50 : drahtmaterial,
51             'wiederstand': wiederstand})
52         conn.commit()
53
54         print(f'{produktbezeichnung} saved to db with
55 id: {produkt_id}')
56     else:
57         print('Insertion to db failed!')
58
59     def add_akkutaeger(self, produktbezeichnung: str,
60                         preis: float, hersteller: str, funktionsweise: str, hoehe:
61                         float,
62                             breite: float, akkutyp: str) ->
63     None:
64         # write to product table and get produkt_id:
65         produkt_id = self.__save_to_prod_db(
66             produktbezeichnung, preis, hersteller)
67
68         if produkt_id > 0:
69             conn = sqlite3.connect(self._db_path)
70             cursor = conn.cursor()
71
72             # add to AKKUTRAEGER-DB
73             cursor.execute(
74                 f"INSERT INTO AKKUTRAEGER(PRODUKT_ID,
75 FUNKTIONSWEISE, HOEHE, BREITE, AKKUTYP) "
76                 f"VALUES(:produkt_id, :funktionsweise, :
77 hoehe, :breite, :akkutyp);",
78                 {'produkt_id': produkt_id, 'funktionsweise':
79 ': funktionsweise, 'hoehe': hoehe,
```

```

68                 'breite': breite, 'akkutyp': akkutyp})
69             conn.commit()
70
71             print(f'{produktbezeichnung} saved to db')
72         with id: {produkt_id}')
73     else:
74         print('Insertion to db failed!')
75
76     def add_set(self, produktbezeichnung: str, preis:
77     float, hersteller: str, akkutraeger_id: int,
78     verdampfer_id: int,
79                 verdampferkopf_id: int) -> None:
80         # write to product table and get produkt_id:
81         produkt_id = self.__save_to_prod_db(
82             produktbezeichnung, preis, hersteller)
83
84         if produkt_id > 0:
85             conn = sqlite3.connect(self._db_path)
86             cursor = conn.cursor()
87
88             # add to AKKUTRAEGER-DB
89             cursor.execute(f"INSERT INTO STARTER_SETS(
PRODUKT_ID, AKKUTRAEGER, VERDAMPFER, VERDAMPFERKOPF) "
90                             f"VALUES(:produkt_id, :
akkutaeger_id, :verdampfer_id, :verdampferkopf_id);",
91                             {'produkt_id': produkt_id, '
akkutaeger_id': akkutraeger_id, 'verdampfer_id':
verdampfer_id,
92                             'verdampferkopf_id':
verdampferkopf_id})
93             conn.commit()
94
95             print(f'{produktbezeichnung} saved to db')
96         with id: {produkt_id}')
97     else:
98         print('Insertion to db failed!')
99
100    def add_admin(self, benutzername: str, passwort: str,
101                  personalnummer: str, abteilung: str) -> None:
102
103        conn = sqlite3.connect(self._db_path)
104        cursor = conn.cursor()
105
106        # add to BENUTZER-TABLE:

```

```

101         cursor.execute("INSERT INTO BENUTZER (
102             BENUTZERNAME, PASSWORT, IS_ADMIN, IS_KUNDE,
103             PERSONALNUMMER, ABTEILUNG) "
104                 "VALUES(:benutzername, :password
105 , :is_admin, :is_kunde, :personalnummer, :abteilung)",
106                 {'benutzername': benutzername,
107                  'passwort': passwort, 'is_admin': True, 'is_kunde': False,
108                  'personal_number': personalnummer
109 , 'abteilung': abteilung})
110             print(f'{benutzername} wurde als administrator
111 angelegt.')
112             conn.commit()
113             conn.close()
114
115     def define_matching(self, verdampfer_id: int,
116     verdampferkopf_id: int):
117         conn = sqlite3.connect(self._db_path)
118         cursor = conn.cursor()
119
120         # add to PASST_ZU-TABLE:
121         cursor.execute("INSERT INTO PASST_ZU (
122             VERDAMPFER_ID, VERDAMPFERKOPF_ID) "
123                 "VALUES(:verdampfer_id, :
124             verdampferkopf_id)",
125                 {'verdampfer_id': verdampfer_id, 'verdampferkopf_id': verdampferkopf_id})
126         conn.commit()
127         conn.close()
128
129     def __einloggen(self):
130         if self._benutzer_einloggen():
131             conn = sqlite3.connect(self._db_path)
132             conn.row_factory = sqlite3.Row
133             cursor = conn.cursor()
134
135             cursor.execute("SELECT * from BENUTZER WHERE
136             BENUTZER_ID = :benutzer_id",
137                 {"benutzer_id": self.
138             _benutzer_id})
139
140             res = cursor.fetchall()
141
142             if len(res) == 1 and bool(res[0]['IS_ADMIN'
143 ]):

```

```
132             ds = res[0]
133             self.__personal_nummer = ds['
134             PERSONALNUMMER']
135             self.__abteilung = ds['ABTEILUNG']
136             self._login_status = True
137         else:
138             print('Login fehlgeschlagen! Bitte
139             benutze das Anmeldeportal für Kunden!')
140
141     def __save_to_prod_db(self, produktbezeichnung: str,
142     preis: float, hersteller: str) -> int:
143
144         try:
145             conn = sqlite3.connect(self._db_path)
146             cursor = conn.cursor()
147
148             # add to PRODUKT-DB
149             cursor.execute(
150                 f"INSERT INTO PRODUKTE(PRODUKTBEZEICHNUNG
151                 ,PREIS, HERSTELLER)"
152                 f"VALUES(:produktbezeichnung,:preis,:
153                 hersteller);",
154                 {'produktbezeichnung': produktbezeichnung
155                 , 'preis': preis, 'hersteller': hersteller})
156             conn.commit()
157             # get id
158             produkt_id = int(cursor.lastrowid)
159             conn.close()
160
161             return produkt_id
162
163     except Exception as e:
164         print(e)
165         return 0
```

```
1 from webshop.class_warenhaus import Warenhaus
2 import sqlite3
3
4
5 class Bestellposten():
6
7     def __init__(self, produkt_id: int, menge: int,
8      warenhaus: Warenhaus, db_path: str, posten_id=None):
9         self.__warenhaus = warenhaus
10        self.__posten_id = posten_id
11        self.__produkt_id = produkt_id
12        self.__menge = menge
13        self.__db_path = db_path
14
15    def get_total(self) -> float:
16        return float(self.__warenhaus.get_by_id(self.
17          __produkt_id).get_price() * self.__menge)
18
19    def get_info(self) -> str:
20        prod = self.__warenhaus.get_by_id(self.
21          __produkt_id)
22        return f"\n{self.__menge} mal {prod.
23          get_produktbezeichnung()} (ID: {prod.get_produkt_id()}) "
24        \
25        f"mit einem Stückpreis von {prod.get_price
26        ():.2f}\n Gesamtpreis: {self.get_total():.2f}"
27
28    def get_produkt_id(self) -> int:
29        return self.__produkt_id
30
31    def _save_to_db(self, bestell_id: int) -> None:
32        conn = sqlite3.connect(self.__db_path)
33        cursor = conn.cursor()
34
35        # add to BESTELLPOSTEN-DB
36        cursor.execute("INSERT INTO BESTELLPOSTEN (
37            BESTELL_ID, PRODUKT_ID, MENGE) "
38            "VALUES(:bestell_id, :produkt_id, :
39            menge);",
40            {'bestell_id': bestell_id, 'produkt_id': self.__produkt_id, 'menge': self.__menge})
41        conn.commit()
42        pass
43
44
```

```
1 class Bankverbindung():
2
3     def __init__(self, bank_id, kontoinhaber, iban, bic):
4         self.__kontoinhaber = str(kontoinhaber)
5         self.__iban = str(iban)
6         self.__bic = str(bic)
7         self.__bank_id = int(bank_id)
8
9     def get_bankinfo(self) -> str:
10        return f'\nKontoinhaber: {self.__kontoinhaber}\n
11        Iban: {self.__iban} \nBIC: {self.__bic}' \
12        f'\nID: {self.__bank_id}\n'
13
14     def get_id(self) -> int:
15        return self.__bank_id
16
17
```

```
1 from webshop.class_produkt import Produkt
2
3
4 class Verdampferkopf(Produkt):
5
6     def __init__(self, produkt_id: int, produktbezeichnung
7      : str, preis: float, hersteller: str, drahtmaterial: str,
8                  wiederstand: float, passende_produkte:
9                  list):
10            super().__init__(produkt_id=produkt_id,
11                            produktbezeichnung=produktbezeichnung, preis=preis,
12                            hersteller=hersteller)
13            self.__drahtmaterial = str(drahtmaterial)
14            self.__wiederstand = float(wiederstand)
15            self.__passende_produkte = passende_produkte
16
17        def get_info(self) -> str:
18            return f'Verdampferkopf {self._produktbezeichnung}'
19            mit {self.__drahtmaterial}-Draht und einem Widerstand '
20            \
21            f'von {self.__wiederstand} Ohm. \n Preis:
22            {self._preis} / Produkt-ID: {self._produkt_id}'
23
24        def display_matching(self) -> None:
25            print('Passende Verdampfer zu diesem
26            Verdampferkopf sind:\n')
27
28        pass
29
```

```
1 import os.path
2 import webshop
3 from setup_script import run_setup
4
5
6 def test_admin_creating_products():
7     user = webshop.class_administrator.Administrator(
8         'admin', 'admin')
9
10    user.add_verdampfer('SuperVape24', 8.99, 'VaperG', 2.8
11    , 2.74, 'top fill')
12    user.add_verdampfer('Nebelmaschine V3', 9.99, 'B-Vape'
13    , 2.8, 2.74, 'top fill')
14
15    user.add_akkutaeger('Powerbank24', 15.99, 'aspire', 'geregelt', 5.4, 3.2, '2 x 18650er Standard Akku')
16    user.add_akkutaeger('Glint II', 9.99, 'aspire', 'geregelt', 4.8, 3.1, '18650er Standard Akku')
17
18    user.define_matching(verdampfer_id=2,
19        verdampferkopf_id=5)
20    user.define_matching(verdampfer_id=2,
21        verdampferkopf_id=6)
22
23
24 def test_user_placing_order():
25     warenhaus = webshop.class_warenhaus.Warenhaus()
26     warenhaus.display_produktinfo()
27     user = webshop.class_kunde.Kunde('PeterV', 'pass234',
28         warenhaus)
29
30     user.add_to_warenkorb(1, 1)
31     user.add_to_warenkorb(2, 1)
32
33     user.add_to_warenkorb(4, 1)
```

```
34     user.add_to_warenkorb(5, 2)
35     user.add_to_warenkorb(6, 2)
36     user.show_warenkorb()
37
38     user.delete_from_warenkorb(2)
39     user.show_warenkorb()
40
41     user.add_adresse('Stuttgarter Str.', 3, '74700', 'Stuttgart')
42     user.show_adressen()
43     user.add_bankverbindung('Peter Vogel', 'DE42XXX7394', 'STVU7DRE')
44     user.show_bankverbindungen()
45
46     user.bestellung_aufgeben(adress_id=1, bank_id=1)
47
48
49 def test_user_changing_details_and_deleting_account():
50     warenhaus = webshop.class_warenhaus.Warenhaus()
51     user = webshop.class_kunde.Kunde('PeterV', 'pass234', warenhaus)
52
53     user.show_kundendaten()
54     user.update_kundendaten(nachname='Vogel-Frei')
55     user.show_kundendaten()
56
57     user.delete_kundendaten()
58
59
60 def main():
61     if not os.path.exists('test_db.db'):
62         run_setup()
63     test_admin_creating_products()
64     test_user_placing_order()
65     test_user_changing_details_and_deleting_account()
66
67
68 if __name__ == "__main__":
69     main()
70
```

```
1 import sqlite3
2
3
4 def setup_database():
5     conn = sqlite3.connect('test_db.db')
6     print('DB created successfully!')
7
8     cursor = conn.cursor()
9
10    cursor.execute("PRAGMA foreign_keys=on")
11
12    cursor.execute('''CREATE TABLE "BENUTZER" (
13        "BENUTZER_ID"    INTEGER NOT NULL UNIQUE,
14        "BENUTZERNAME"   TEXT NOT NULL UNIQUE,
15        "PASSWORT"      TEXT NOT NULL,
16        "IS_ADMIN"       INTEGER NOT NULL,
17        "IS_KUNDE"       INTEGER NOT NULL,
18        "PERSONALNUMMER" TEXT,
19        "ABTEILUNG"     TEXT,
20        "VORNAME"        TEXT ,
21        "NACHNAME"       TEXT,
22        PRIMARY KEY("BENUTZER_ID" AUTOINCREMENT));
23        ''')
24    print('Table BENUTZER created successfully!')
25
26    # cursor.execute('''CREATE TABLE "ADMINISTRATOREN"
27    #     (ID INTEGER NOT NULL UNIQUE,
28    #      BENUTZER_ID INT NOT NULL,
29    #      PERSONALNUMMER TEXT NOT NULL,
30    #      ABTEILUNG TEXT,
31    #      PRIMARY KEY("ID" AUTOINCREMENT),
32    #      FOREIGN KEY(BENUTZER_ID) REFERENCES BENUTZER(ID
33    ) ON DELETE CASCADE);
34    #        ''')
35    # print('Table ADMINISTRATOR created successfully!')
36
37    # cursor.execute('''CREATE TABLE "KUNDEN"
38    #     ("ID" INTEGER NOT NULL UNIQUE,
39    #      "BENUTZER_ID" INTEGER NOT NULL,
40    #      "VORNAME" TEXT NOT NULL,
41    #      "NACHNAME" TEXT NOT NULL,
42    #      "ADRESSE" INTEGER,
43    #      "BANKVERBINDUNG" INTEGER,
44    #      PRIMARY KEY("ID" AUTOINCREMENT),
```

```

44      #          FOREIGN KEY("ADRESSE") REFERENCES ADRESSEN(
45      #          FOREIGN KEY("BANKVERBINDUNG") REFERENCES
46      #          FOREIGN KEY("BENUTZER_ID") REFERENCES
47      #          BENUTZER(ID) ON DELETE CASCADE);
48      #          '')
49
50      cursor.execute('''CREATE TABLE "ADRESSEN"
51          ("ADRESS_ID"INTEGER NOT NULL UNIQUE,
52          "BENUTZER_ID"    INTEGER NOT NULL,
53          "STRASSE"      TEXT NOT NULL,
54          "HAUSNUMMER"   INTEGER NOT NULL,
55          "PLZ"          TEXT NOT NULL,
56          "STADT"        TEXT NOT NULL,
57          PRIMARY KEY("ADRESS_ID" AUTOINCREMENT),
58          FOREIGN KEY("BENUTZER_ID") REFERENCES BENUTZER(
59          BENUTZER_ID) ON DELETE CASCADE);
60          ''')
61      print('Table ADRESSEN created successfully!')
62
62      cursor.execute('''CREATE TABLE "BANKVERBINDUNGEN"
63          ("BANK_ID"  INTEGER NOT NULL UNIQUE,
64          "BENUTZER_ID"  INTEGER NOT NULL,
65          "KONTOKHABER"  TEXT NOT NULL,
66          "IBAN"  TEXT NOT NULL,
67          "BIC"  TEXT NOT NULL,
68          PRIMARY KEY("BANK_ID" AUTOINCREMENT),
69          FOREIGN KEY("BENUTZER_ID") REFERENCES BENUTZER(
70          BENUTZER_ID) ON DELETE CASCADE);
71          ''')
71      print('Table BANKVERBINDUNGEN created successfully!')
72
73      cursor.execute('''CREATE TABLE "BESTELLUNGEN" (
74          "BESTELL_ID"INTEGER NOT NULL UNIQUE,
75          "BENUTZER_ID"    INTEGER NOT NULL,
76          "BESTELLDATUM"  TEXT NOT NULL,
77          "STATUS"TEXT NOT NULL,
78          "ADRESS_ID"    INTEGER NOT NULL,
79          "BANK_ID"        INTEGER NOT NULL,
80          FOREIGN KEY("BENUTZER_ID") REFERENCES BENUTZER(
81          BENUTZER_ID),
81          FOREIGN KEY("ADRESS_ID") REFERENCES ADRESSEN(

```

```
81 ADRESS_ID),
82     FOREIGN KEY("BANK_ID") REFERENCES
83     BANKVERBINDUNGEN(BANK_ID),
84     PRIMARY KEY(BESTELL_ID AUTOINCREMENT));
85     '''
86     print('Table BESTELLUNGEN created successfully!')
87
88     cursor.execute('''CREATE TABLE "BESTELLPOSTEN" (
89         "POSTEN_ID" INTEGER NOT NULL UNIQUE,
90         "BESTELL_ID"INTEGER NOT NULL,
91         "PRODUKT_ID"INTEGER NOT NULL,
92         "MENGE" INTEGER NOT NULL,
93         PRIMARY KEY("POSTEN_ID" AUTOINCREMENT),
94         FOREIGN KEY("BESTELL_ID") REFERENCES BESTELLUNGEN
95         (BESTELL_ID) ON DELETE CASCADE,
96         FOREIGN KEY("PRODUKT_ID") REFERENCES PRODUKTE(ID
97     ));
98     '''
99     print('Table BESTELLPOSTEN created successfully!')
100
101    cursor.execute('''CREATE TABLE "PRODUKTE" (
102        "PRODUKT_ID"INTEGER NOT NULL UNIQUE,
103        "PRODUKTBEZEICHNUNG"TEXT NOT NULL,
104        "PREIS" REAL NOT NULL,
105        "HERSTELLER"TEXT,
106        PRIMARY KEY("PRODUKT_ID" AUTOINCREMENT));
107        '''
108        print('Table PRODUKTE created successfully!')
109
110        cursor.execute('''CREATE TABLE "AKKUTRAEGER" (
111            "PRODUKT_ID"INTEGER NOT NULL UNIQUE,
112            "FUNKTIONSWEISE"TEXT NOT NULL,
113            "HOEHE" REAL NOT NULL,
114            "BREITE"REAL NOT NULL,
115            "AKKUTYP" TEXT NOT NULL,
116            PRIMARY KEY("PRODUKT_ID"),
117            FOREIGN KEY("PRODUKT_ID") REFERENCES "PRODUKTE"(
118                "PRODUKT_ID") ON DELETE CASCADE);
119            '''
120            print('Table AKKUTRAEGER created successfully!')
121
122            cursor.execute('''CREATE TABLE "VERDAMPFER" (
123                "PRODUKT_ID"INTEGER NOT NULL UNIQUE,
124                "DURCHMESSER"    REAL NOT NULL,
```

```

121          "HOEHE" REAL NOT NULL,
122          "FUELLSYSTEM" TEXT NOT NULL,
123          PRIMARY KEY("PRODUKT_ID"),
124          FOREIGN KEY("PRODUKT_ID") REFERENCES "PRODUKTE"("
125          PRODUKT_ID") ON DELETE CASCADE);
126      ''')
127      print('Table VERDAMPFER created successfully!')
128
129      cursor.execute('''CREATE TABLE "VERDAMPFERKOEPFE" (
130          "PRODUKT_ID"INTEGER NOT NULL UNIQUE,
131          "DRAHTMATERIAL" TEXT NOT NULL,
132          "WIEDERSTAND" REAL NOT NULL,
133          PRIMARY KEY("PRODUKT_ID"),
134          FOREIGN KEY("PRODUKT_ID") REFERENCES "PRODUKTE"("
135          ID") ON DELETE CASCADE);
136      ''')
137      print('Table VERDAMPFERKOEPFE created successfully!')
138
139      cursor.execute('''CREATE TABLE "PASST_ZU" (
140          "PASSUNG_ID"INTEGER NOT NULL UNIQUE,
141          "VERDAMPFER_ID" INTEGER NOT NULL,
142          "VERDAMPFERKOPF_ID" INTEGER NOT NULL,
143          FOREIGN KEY("VERDAMPFER_ID") REFERENCES
144          VERDAMPFER(PRODUKT_ID) ON DELETE CASCADE ,
145          FOREIGN KEY("VERDAMPFERKOPF_ID") REFERENCES
146          VERDAMPFERKOEPFE(PRODUKT_ID) ON DELETE CASCADE,
147          PRIMARY KEY("PASSUNG_ID" AUTOINCREMENT));
148      ''')
149      print('Table PASST_ZU created successfully!')
150
151      cursor.execute('''CREATE TABLE "STARTER_SETS" (
152          "PRODUKT_ID"INTEGER NOT NULL UNIQUE,
153          "AKKUTRAEGER" INTEGER NOT NULL,
154          "VERDAMPFER"INTEGER NOT NULL,
155          "VERDAMPFERKOPF"INTEGER NOT NULL,
156          PRIMARY KEY("PRODUKT_ID"),
157          FOREIGN KEY("VERDAMPFER") REFERENCES VERDAMPFER(
158          PRODUKT_ID) ON DELETE CASCADE,
159          FOREIGN KEY("AKKUTRAEGER") REFERENCES AKKUTRAEGER(
160          PRODUKT_ID) ON DELETE CASCADE,
161          FOREIGN KEY("VERDAMPFERKOPF") REFERENCES
162          VERDAMPFERKOEPFE(PRODUKT_ID) ON DELETE CASCADE,
163          FOREIGN KEY("PRODUKT_ID") REFERENCES PRODUKTE(ID
164          ) ON DELETE CASCADE);

```

```
157     ''')
158     print('Table STARTER_SETS created successfully!')
159     conn.commit()
160     conn.close()
161     print('Connection to DB closed')
162
163 def add_admin():
164
165     print('creating initial admin user')
166
167     conn = sqlite3.connect('test_db.db')
168     cursor = conn.cursor()
169
170     cursor.execute(f"INSERT INTO BENUTZER(BENUTZERNAME,
171 PASSWORT,IS_ADMIN,IS_KUNDE,PERSONALNUMMER,ABTEILUNG)
172 VALUES('admin','admin',True, False,'SHOPADMIN','IT');")
173     conn.commit()
174
175     # cursor = conn.execute(f"SELECT ID,BENUTZERNAME from
176     # BENUTZER")
177     # for row in cursor:
178     #     if 'admin' == row[1]:
179     #         benutzer_id = row[0]
180     #     #
181     # conn.execute(
182     #     f"INSERT INTO ADMINISTRATOREN(BENUTZER_ID,
183 PERSONALNUMMER,ABTEILUNG) VALUES({benutzer_id}, 'SHOPADMIN
184 ', 'IT');");
185     # conn.commit()
186
187     conn.close()
188     print('admin created successfully')
189
190 def run_setup():
191     setup_database()
192     add_admin()
```

## D. Konsolenausgabe der Beispielanwendungen im Main-Skript

```
1 [REDACTED] /Laborgericht-PRG24/  
main.py  
2 DB created successfully!  
3 Table BENUTZER created successfully!  
4 Table ADRESSEN created successfully!  
5 Table BANKVERBINDUNGEN created successfully!  
6 Table BESTELLUNGEN created successfully!  
7 Table BESTELLPOSTEN created successfully!  
8 Table PRODUKTE created successfully!  
9 Table AKKUTRAEGER created successfully!  
10 Table VERDAMPFER created successfully!  
11 Table VERDAMPFERKOEPFE created successfully!  
12 Table PASST_ZU created successfully!  
13 Table STARTER_SETS created successfully!  
14 Connection to DB closed  
15 creating initial admin user  
16 admin created successfully  
17  
18 Process finished with exit code 0  
19
```

```
1 [REDACTED]\npython.exe [REDACTED] /  
main.py  
2 SuperVape24 saved to db with id: 1  
3 Nebelmaschine V3 saved to db with id: 2  
4 Powerbank24 saved to db with id: 3  
5 Glint II saved to db with id: 4  
6 B-Vape V3 saved to db with id: 5  
7 B-Vape V3-Mesh saved to db with id: 6  
8 Easy Starter Set saved to db with id: 7  
9  
10 Process finished with exit code 0  
11
```

1 [REDACTED]\python.exe [REDACTED]  
main.py  
2 Katalog updated!  
3 Verdampfer SuperVape24 mit einem Durchmesser von 2.8mm und  
Höhe von 2.74mm.  
4 Preis:8.99 / Produkt-ID: 1  
5 Verdampfer Nebelmaschine V3 mit einem Durchmesser von 2.  
8mm und Höhe von 2.74mm.  
6 Preis:9.99 / Produkt-ID: 2  
7 Verdampferkopf B-Vape V3 mit Stainles Steel-Draht und  
einem Widerstand von 0.5 Ohm.  
8 Preis:0.43 / Produkt-ID: 5  
9 Verdampferkopf B-Vape V3-Mesh mit Stainles Steel Mesh-  
Draht und einem Widerstand von 0.33 Ohm.  
10 Preis:0.6 / Produkt-ID: 6  
11 Akkuträger Powerbank24 mit einer Breite von 3.2 mm und  
Höhe von 5.4mm.  
12 Preis:15.99 / Produkt-ID: 3  
13 Akkuträger Glint II mit einer Breite von 3.1 mm und Höhe  
von 4.8mm.  
14 Preis:9.99 / Produkt-ID: 4  
15  
16 Das Set Easy Starter Set enthält:  
17 Akkuträger Glint II mit einer Breite von 3.1 mm und Höhe  
von 4.8mm.  
18 Preis:9.99 / Produkt-ID: 4  
19 Verdampfer SuperVape24 mit einem Durchmesser von 2.8mm und  
Höhe von 2.74mm.  
20 Preis:8.99 / Produkt-ID: 1  
21 Verdampferkopf B-Vape V3-Mesh mit Stainles Steel Mesh-  
Draht und einem Widerstand von 0.33 Ohm.  
22 Preis:0.6 / Produkt-ID: 6  
23 Setpreis: 12.99€ / Set-Produkt-ID: 7  
24  
25 Login fehlgeschlagen! Bitte lege ein passendes  
Benutzerkonto an.  
26 Möchten Sie ein neues Kundenkonto anlegen? (ja/nein)ja  
27 Geben Sie ihren Vornamen an:Peter  
28 Geben Sie ihren Nachnamen an:Vogel  
29 Bitte wählen Sie einen Benutzernamen aus.PeterV  
30 Geben Sie ein Passwort an:pass234  
31 Peter Vogel wurde als Kunde mit dem Benutzernamen PeterV  
angelegt.  
32

33 Willkommen PeterV Ihre Anmeldung war erfolgreich!  
34  
35  
36 Ihr aktueller Warenkorb enthält:  
37 1 mal SuperVape24 (ID: 1) mit einem Stückpreis von 8.99€  
38 Gesamtpreis: 8.99€  
39 1 mal Nebelmaschine V3 (ID: 2) mit einem Stückpreis von 9.  
99€  
40 Gesamtpreis: 9.99€  
41 1 mal Glint II (ID: 4) mit einem Stückpreis von 9.99€  
42 Gesamtpreis: 9.99€  
43 2 mal B-Vape V3 (ID: 5) mit einem Stückpreis von 0.43€  
44 Gesamtpreis: 0.86€  
45 2 mal B-Vape V3-Mesh (ID: 6) mit einem Stückpreis von 0.  
60€  
46 Gesamtpreis: 1.20€  
47  
48 Der Gesamtpreis aller Produkte im aktuellen Warenkorbes  
beträgt: 31.03€  
49  
50 Ihr aktueller Warenkorb enthält:  
51 1 mal SuperVape24 (ID: 1) mit einem Stückpreis von 8.99€  
52 Gesamtpreis: 8.99€  
53 1 mal Glint II (ID: 4) mit einem Stückpreis von 9.99€  
54 Gesamtpreis: 9.99€  
55 2 mal B-Vape V3 (ID: 5) mit einem Stückpreis von 0.43€  
56 Gesamtpreis: 0.86€  
57 2 mal B-Vape V3-Mesh (ID: 6) mit einem Stückpreis von 0.  
60€  
58 Gesamtpreis: 1.20€  
59  
60 Der Gesamtpreis aller Produkte im aktuellen Warenkorbes  
beträgt: 21.04€  
61 Adresse in der Stuttgarter Str. 3 in Stuttgart erfolgreich  
angelegt.  
62  
63 Stuttgarter Str. 3  
64 74700 Stuttgart  
65 ID: 1  
66  
67 Bankverbindung mit der IBAN endend auf 7394 erfolgreich  
angelegt.  
68  
69 Kontoinhaber: Peter Vogel  
70 Iban: DE42XXX7394

71 BIC: STVU7DRE  
72 ID: 1  
73  
74 Ihre Bestellung wurde am 05.10.2021 um 23:16 aufgegeben.  
75  
76 Process finished with exit code 0  
77

```
1 [REDACTED]\npython.exe [REDACTED] /  
main.py  
2 Katalog updated!  
3  
4 Willkommen PeterV Ihre Anmeldung war erfolgreich!  
5  
6  
7 Ihre Daten lauten:  
8 Vorname: Peter  
9 Nachname: Vogel  
10 Ihr Benutzername lautet: PeterV  
11  
12 Ihre Daten lauten:  
13 Vorname: Peter  
14 Nachname: Vogel-Frei  
15 Ihr Benutzername lautet: PeterV  
16  
17 Ihre Kundenkonto mit dem Benutzernamen PeterV wurde  
erfolgreich gelöscht.  
18  
19  
20 Process finished with exit code 0  
21
```

## E. Eidesstattliche Erklärung

„Ich versichere, dass ich das beiliegende Assignment selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie alle wörtlich oder sinngemäß übernommenen Stellen in der Arbeit gekennzeichnet habe.“

Reutlingen, den 07.10.2021

Jonas Kuhlo