Erstellung einer einfachen Java-Anwendung zur Verwaltung eines Online- Warenkorbs

Assignment zum Modul:

Programmieren in Java (JAV40)

1. April 2019

Vladimir Zhelezarov

.....

Studiengang: Digital Engineering und angewandte Informatik - Bachelor of Engineering

AKAD University

Inhaltsverzeichnis

1 Einleitung

Um über Internet etwas zu verkaufen, können die Firmen oder die Verkäufer verschiedene Methoden benutzen - wie Marktplätze, Portale, Foren oder auch einen eigenen Webshop anbieten. Außer Benutzerverwaltung, Datenbankkommunikation, Sessionmanagement und Benutzeroberfläche, steht im Herzen des Shops die Warenkorb-Komponente - zuständig für die Verwaltung von Artikeln und Bestellungen während des Einkaufs.

In der vorliegenden Arbeit wird ein möglicher Aufbau der Warenkorb-Komponente vorgestellt, der die Programmiersprache Java benutzt. Um den Zusammenhang mit den anderen Komponenten des Shops und die Funktion des Warenkorbs besser zu verstehen, sind auch die nötigsten anderen Komponenten vorhanden, wenn auch mit begrenzter Funktion. Der Fokus dieses Projekts liegt auf dem Warenkorb, die Artikel und deren Verwaltung in der Zeit bevor der Benutzer die Kaufabwicklung abschließt.

Weil Java eine objektorientierte Sprache ist, sind zuerst die beteiligten Objekte mit deren Eigenschaften und Verhalten zu erkennen. Eine logische Aufteilung wären unter anderem der Warenkorb an sich, die Artikel und gemeinsame für alle Benutzer Daten wie eine Artikeldatenbank, der Shop-Bestand und die angelegten Benutzer. Bei der Analyse der Anwendung tauchen auch zusätzliche Objekte auf.

Das Modell für die so erkannten Objekte sind in Java ihre Klassen. Bei dem Aufbau der Klassen ist zu beachten, wie die Objekte untereinander Informationen austauschen und überhaupt welcher Zugang zu den eigenen Informationen notwendig ist. Einige Konzepte oder Besonderheiten bei der Funktion und dem Zusammenhang zwischen den Klassen sind geklärt, bevor diese auch endgültig aufgebaut werden.

Auch wenn hier kein kompletter Webshop aufgebaut wird, muss auch die Möglichkeit bestehen, mit dem Warenkorb zu kommunizieren und so viel wie möglich von seinen Funktionen zu testen. Ein Ansatz dafür ist über der Systemkonsole, als eine Anwendung die eigenständig läuft und über einem interaktiven Menü mit dem Benutzer interagiert.

Der Überblick über der so gebauten Anwendung ist mit einem UML-Diagramm angezeigt.

2 Grundlagen

Bevor die beteiligten Objekte erkannt werden, sind hier zuerst die benutzten Begriffe und Konzepte erklärt, wie Java als objektorientiert funktioniert und die Auswahl an Datentypen die in der Anwendung benutzt sind.

2.1 Definitionen und Konzept

Als Webshop versteht sich in dieser Arbeit eine Web-Seite, in der Produkte oder Dienste über dem Internet verkauft werden. Von dem Moment, wenn der Benutzer seinen Kauf beginnt, bis zu dem Moment, wenn er die Bestellung endgültig bestätigt, werden seine Auswahl an Artikeln, deren Anzahl und Preise in seinem Warenkorb verwaltet. Wenn die Bestätigung vorliegt oder auch wenn es zum Abbruch kommt, wird der Warenkorb geleert. Wie die weitere Behandlung der Bestellung verläuft, wird in diesem Projekt nicht betrachtet.

Durch das Speichern im Warenkorb wird die Menge an Artikeln vom Webshop-Bestand vorübergehend abgebucht und als nicht mehr verfügbar für die anderen Kunden gesehen. Bei einem Abbruch wird zurückgebucht und bei einer Bestellung wird die Menge endgültig gelöscht. Das erstmalige Anlegen von Artikeln und deren Bestand wird vom Testprogramm durchgeführt und damit die Funktionen vom Lager und Verwalter simuliert. Weitere Änderungen an den Artikeln oder dem Bestand, die nicht vom Warenkorb ausgelöst sind, sind ausgeschlossen.

Die Benutzer sind auch vom Testprogramm einmalig angelegt und unterliegen keine weiteren Änderungen.

2.2 Objektorientierte Programmierung in Java

Bei der objektorientierten Programmierung werden die Objekte - die Entitäten der realen Welt - durch Klassen vorgestellt. Jede Klasse entspricht einer Familie von Objekten, die man mit den gleichen Attributen und dem gleichen Verhalten beschreiben kann[1].

Der klare Vorteil bei diesem Vorgehen ist, dass die Software näher an der realen Welt ist und auch dass mehr Kontrolle über den Datenflüssen und Abläufen ausgeübt wird. Wie z.B. interne Informationen des Objekts, die nicht für die Kommunikation mit anderen Objekten benötigt werden, sind nicht von außerhalb der Klasse erreichbar[2].

In manchen Situationen braucht man keine Instanzen und konkrete Objekte, sondern globale Variablen oder sogar Methoden - dazu benutzt Java das Konzept von *static* - das dazugehörige Attribut oder die dazugehörige Methode gehört zu der Klasse und nicht zu dem

[1] https://docs.oracle.com/javase/tutorial/java/concepts/class.html, (Zugriff am 23.03.2019)
[2] https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html, (Zugriff am 23.03.2019)

konkreten Objekt und kann von anderen Objekten ohne Instantiierung benutzt werden[1].

2.3 Datentypen

Von den primitiven Datentypen in Java[2] benutzt die Anwendung *int* - für Zahlen, Indexen und Aufzählungen und *boolean* - um wahr/unwahr Ergebnisse auszudrücken.

Für die Arbeit mit Zeichenketten kommt die in Java eingebaute Klasse *String*[3] in Frage.

Weil der Warenkorb Operationen mit Preisen und der Währung ausführt und *double* oder *float* dafür nicht geeignet wären[4], sind die Gelder besser mit der Java Klasse *BigDecimal* aus *java.math*[5] repräsentiert.

3 Objekte und Daten

Das Abbilden der realen Welt in die Software fängt mit der Erkennung der Objekte an, die in den Prozessen teilnehmen. Daraus werden in Java die Klassen gebildet.

Die Benutzeroberfläche wird vom Mainprogramm ausgeführt. Die tatsächliche UI-Logik kann als ein separates Objekt formuliert werden. Somit ergeben sich die ersten zwei:

- *Main*: Hauptprogramm. Wird zum Testen aller anderen Klassen benutzt. Kümmert sich um das Anlegen von Benutzern, Artikeln und dem Bestand in der Datenbank. Kommuniziert direkt mit dem Benutzer mithilfe der
- *UI*: Alle Benutzeroberflächen-zentrierten Funktionalitäten.

Bei der Arbeit mit Artikeln bietet sich an drei Objekte zu unterscheiden:

- *Artikel*: jeder Artikel hat Eigenschaften wie eine Artikelnummer, einen Preis, einen Steuersatz. Diese können gelesen oder geändert werden;
- *Position*: das ist ein Hilfsobjekt, der dazu dient, einem Artikel eine Zahl (Anzahl) zuzuordnen;
- *Warenkorb*: eine Sammelstelle aller Positionen, mit den dazu gehörigen Möglichkeiten für Datenzugriff und Änderung.

In einer Anwendung der realen Welt, in der es um Speicherung, Änderung und Arbeit mit gespeicherten Daten geht, wird eine Datenbank benutzt:

[1] https://docs.oracle.com/javase/tutorial/java/javaOO/classvars.html, (Zugriff am 23.03.2019)
[2] https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html, (Zugriff am 23.03.2019)
[3] https://docs.oracle.com/javase/10/docs/api/java/lang/String.html(Zugriff am 23.03.2019)
[4] vgl. Zukowski, John (2007), Internetquelle
[5] https://docs.oracle.com/javase/10/docs/api/java/math/BigDecimal.html, (Zugriff am 23.03.2019)

- *Datenbank*: verwaltet unter anderem die Benutzer, angelegten Artikeln und den Bestand. Weil der Anwendung keine „echte“ Datenbank zur Verfügung steht, simuliert das Testprogramm ihre Funktion: Das Anlegen der Benutzer und die Artikel mit deren Verfügbarkeit (Bestand) wird vom Testprogramm ausgeführt.

Bei der Kommunikation zwischen den Objekten werden natürlich auch Ergebnisse ausgetauscht. In manchen Situationen, in den das Ergebnis ein Erfolg ist, ist der Grund dafür wenig interessant. Andersherum bei einem Fehlschlag ist es gut eine Erklärung dafür zu erhalten. Dieses Konzept lässt sich mit einem Paar von Ergebnis-Grund realisieren, was zum nächsten Objekt führt:

- *Result*: ein Ergebnis-Erklärung Paar.

Gemeinsam benutzte Werkzeuge von allen Objekten lassen sich auch als eigenes Objekt vorstellen:

- *Utils*: Sammlung aus Werkzeugen.

Die Klassen sind in der Warenkorb-Anwendung so genannt, wie die Objekte, die sie beschreiben.

4 Aufbau der Klassen

Es gibt gemeinsame Besonderheiten, die mehrere Klassen und deren Kommunikation zu einander betreffen. Wenn diese geklärt sind, können die einzelnen Klassen vorgestellt werden. Alle Klassen sind im Mainprogramm genutzt um getestet zu werden. Das Übersicht-Bild von der gesamten Anwendung erfolgt dann über einem UML-Diagramm.

4.1 Besonderheiten

- Keine öffentlichen Felder:

Keine Klasse in der Anwendung besitzt öffentliche (*public*) Felder. Der Zugriff erfolgt kontrolliert über *get-*/*set-* Methoden, oder gar nicht, wenn das Feld nur intern benötigt wird.

- Gemeinsam-benutzte Daten:

Die Klassen *Warenkorb*, *Artikel* und *UI* erwarten beim Konstruktor eine Datenbank als Parameter. So kann vermieden werden, dass die Anwendung mit globalen (statischen) Objekten arbeitet und so mehr flexibel und Modul-orientiert bleibt. Die Datenbank kann vom Mainprogramm instantiiert werden. Somit haben die Klassen gemeinsamen Zugriff auf die Benutzer, Artikeln und den Bestand.

- Artikelnummer:

Bei den Artikeln ist es zu vermeiden, dass doppelte Artikelnummern existieren. Dafür benutzt die Datenbank eine interne Variable *indexArtikel,* die bei jedem Aufruf von *newArtikel* inkrementiert wird. Wenn dann ein neuer Artikel angelegt wird, wird der Wert dieser Variable als neue Artikelnummer zugewiesen. Dadurch bleibt auch eine Artikelnummer unveränderbar, was dazu dient, Konflikte zu vermeiden.

- Position löschen:

Wenn beim Benutzer eine Position keine Artikel mehr hat (Anzahl ist Null), kann sie gelöscht werden. Bei dem Bestand aber darf die Anwendung so eine Position nicht löschen, weil der Artikel immer noch existiert, davon ist zurzeit bloß keine Menge verfügbar. Das ist auch noch ein Grund, warum die Artikelnummern erhalten werden müssen.

- Statische Methoden in *Utils*:

Utils besteht aus kleinen Werkzeugen, die die anderen Klassen häufig benutzen. Wie bei dem Konzept von *java.lang.Math*[1] müssen keine Objekte instantiiert werden.

- Netto- und Totalpreis, Steuersatz:

Jeder Preis, Steuersatz und jede Geldsumme sind intern als *BigDecimal* gespeichert, wobei der Steuersatz in der üblichen Form *XX*% beträgt. Davon nimmt die Anwendung Absicht beim Rechnen. Beim Anzeigen der Gelder-Werte sind sie in der Form *XX.XX*

- Benutzereingaben:

Auf sie sollte in der Regel nie vertraut werden. Ob aus Versehen, oder mit Absicht, könnte eine Fehleingabe zum unerwarteten Verhalten vom Programm oder sogar zum Abstürzen führen. Darum sind die Eingaben immer so gut wie möglich zu überprüfen und bei Bedarf sind entsprechende Hinweise oder Fehlermeldungen anzuzeigen. Für den schlimmsten Fall - den unvorhersehbaren Fehler - bietet Java *Exceptions*[2] an, die auch nur dort vorkommen, wenn es keine andere Möglichkeit zur Fehlerkontrolle gibt.

- Überschreiben von der *.toString* Methode:

Die Oberklasse *Object* in Java besitzt die Methode *toString*[3], um eine vom Menschen lesbare Repräsentation des Objekts darzustellen. Weil es für die Warenkorb-Anwendung eine andere, eventuell besser lesbare Variante von *toString* gibt, sind manche Methoden überschrieben.

[1] https://docs.oracle.com/javase/10/docs/api/java/lang/Math.html, (Zugriff am 23.03.2019)
[2] https://docs.oracle.com/javase/10/docs/api/java/lang/Exception.html, (Zugriff am 23.03.2019)
[3] https://docs.oracle.com/javase/10/docs/api/java/lang/Object.html#toString(), (Zugriff am 23.03.2019)

Somit wird beim Aufruf die neu-definierte Logik ausgeführt[1]. Die Klassen Warenkorb und Datenbank mit den entsprechenden Hilfsklassen verfügen alle über so eine überschriebene Methode.

4.2 Klassen

4.2.1 Datenbank

Die schon diskutierten privaten Felder sind der Artikelindex, die Liste mit existierenden Artikeln und die Liste mit dem Bestand.

Die Datenbank kann neue Artikelnummern ausgeben, neue Artikel in der Artikelliste anlegen, Artikel in der Liste nach einer Artikelnummer suchen.

Die Arbeit mit dem Bestand ist begrenzt auf:

- *setBestand* - eine Positionen-liste als Bestand übernehmen, und
- *updateBestand* - eine Position aktualisieren.

4.2.2 Artikel

Beim Aufruf vom Konstruktor, d.h. wenn neuer Artikel angelegt wird, wird eine Referenz davon in der Datenbank zugefügt. Alle Felder, bis auf die Artikelnummer haben *get-* und *set-*Methoden. Bei der Artikelnummer ist nur *get-* vorgesehen (read-only). Die interne Konstante MAX_BESCHREIBUNG begrenzt die Länge der Beschreibung.

4.2.3 Warenkorb

Die eindeutige Identifikation vom Warenkorb ist am Benutzer gebunden und ist im read-only Feld *benutzer_id* eingetragen. Eine Array-Liste hält die Referenzen zu allen Positionen. Zwei Summen - Netto und Total - beschreiben die jeweilige Summe von allen Artikeln im Warenkorb. Diese werden immer intern nachgerechnet, wenn Änderungen an den Positionen vorkommen. Andere zwei Summen haben die Information über der Anzahl von Positionen und Artikeln. Es gibt zwei Möglichkeiten den Warenkorb zu löschen: *bestelle* und *delete* und sie entsprechen jeweils einer Bestellung oder einem Abbruch mit dem Kauf. Änderungen am Warenkorb können in einer der folgenden Formen vorkommen:

- Position hinzufügen;
- Position löschen, wobei die Artikel zurückgebucht werden und

[1] https://docs.oracle.com/javase/tutorial/java/IandI/override.html, (Zugriff am 23.03.2019)

- Position ändern, um eine Position nach ihrer Artikelnummer zu suchen und die angegebene Änderung dabei vorzunehmen.

Getters für alle Summen und für die Warenkorb-Inhaber sind vorhanden.

4.2.4 UI

Die Klasse dient dem interaktiven Menü vom Mainprogramm und ist nur auf der Konsole ausgelegt. Mit den verfügbaren Methoden werden formatierte Ausgaben dem Benutzer angezeigt oder Eingaben vom letzten gefragt.

4.2.5 Hilfsklassen

Dazu zählen:

- Position, Result und Utils. Deren Funktion wurde schon betrachtet.

4.3 Mainprogramm

Das Mainprogramm muss zum Anfang die Daten in der Datenbank, die nachher als vorhanden wirken, aufschreiben. Dazu gehören die existierenden Artikeln, die Benutzer und der Bestand im Shop.

Als eingebauter Test erstellt das Mainprogramm manche Warenkörbe und füllt sie mit Artikeln. Mithilfe der UI-Methoden wird der Bestand vor und nach diesem Test ausgedruckt.

Um so viele Szenarien wie möglich zu testen, besteht dann die Option ein interaktives Menü in der Konsole zu starten, womit der Tester verschiedene Möglichkeiten ausprobieren kann. Die verfügbaren Optionen sind:

1. Warenkorb ansehen;
2. Bestand ansehen;
3. Benutzer wechseln: dabei ist die Auswahl auf die vordefinierten Benutzer eingegrenzt;
4. Neue Position anlegen: dabei wählt der Benutzer einen neuen Artikel und die Anzahl davon;
5. Position löschen: nach Eingabe der Artikelnummer;
6. Postion ändern: es wird nach der Artikelnummer und Änderung gefragt;
7. Warenkorb leeren: die Option simuliert den Abbruch;
8. Bestellen: das simuliert eine Bestätigung des Kaufs;
9. Schluss: Ende des Programms.

4.4 Klassen-Überblick

Mit den so definierten Objekten, mit Absicht auf die gezielte Funktionalität und die beschriebenen Besonderheiten, können die Klassen so modelliert werden:

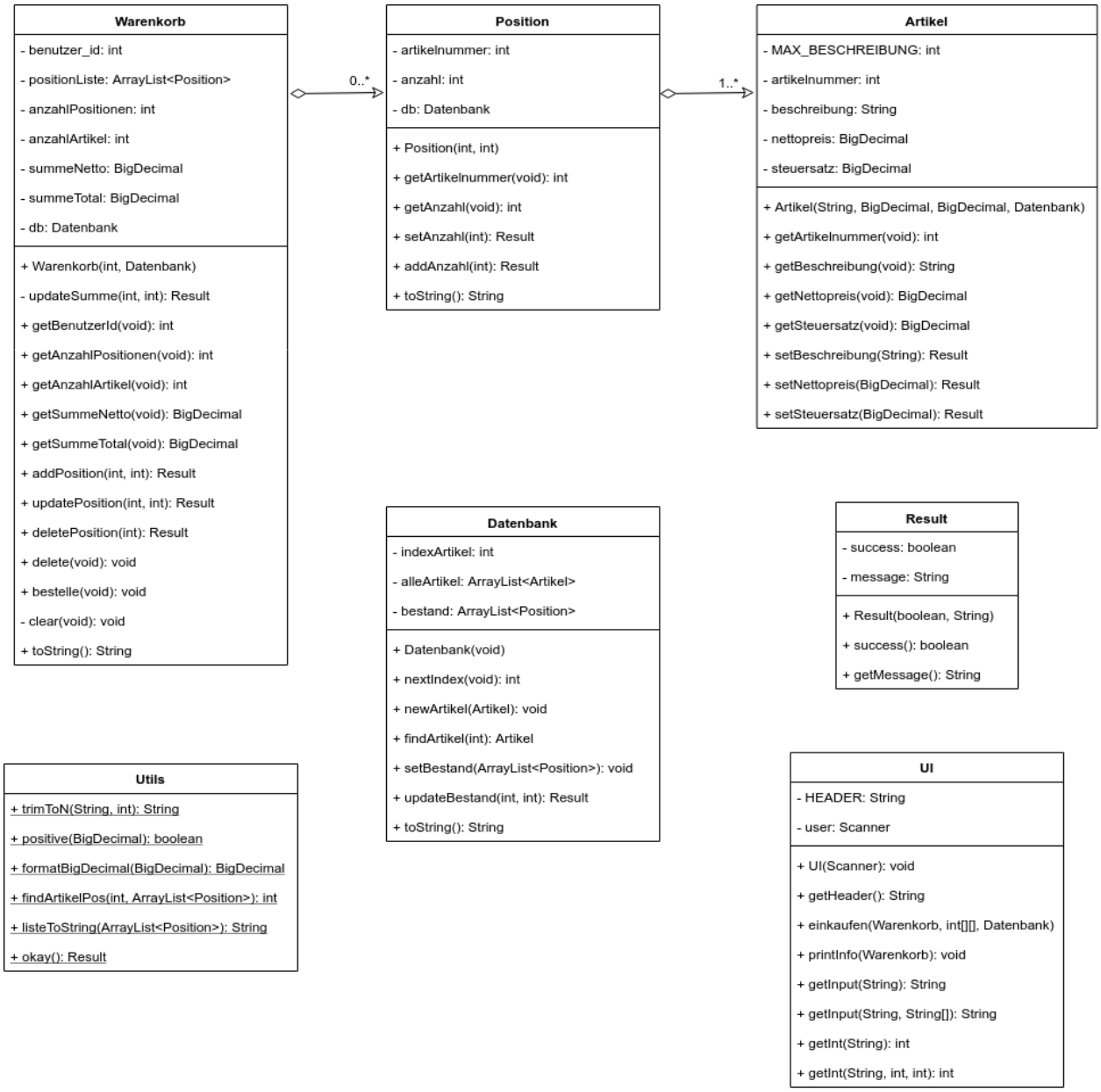

Abbildung 1: Klassendiagramm

5 Zusammenfassung

Im Internet wird immer wieder etwas verkauft, so ist kein Wunder, dass Webshops häufig zu treffen sind. Um ein Webshop aufzubauen, braucht man unter anderem die Warenkorb-Komponente. Sie unterstützt den Kunden während seiner Auswahl und vermittelt zwischen ihm und der Datenbank, bevor es zu einer Kaufabwicklung oder dem Abbruch kommt.

In der vorliegenden Arbeit wurde eine Warenkorb-Anwendung als Teil eines kleinen Webshop mit begrenzter Funktionalität vorgestellt. Dabei war der Akzent auf die Klassen *Warenkorb* und *Artikel*. Das hatte zur Folge dass die anderen Klassen, wie die Datenbank oder Benutzeroberfläche, nur als Hilfs- oder Testmodule zu *Warenkorb* und *Artikel* geplant wurden.

Die gemeinsam genutzten Daten - die Listen mit den Benutzern, Artikeln und dem Bestand sind dank der Datenbank-Klasse von allen Klassen erreichbar. Der Warenkorb und der Bestand arbeiten mit Positionen - Artikel-Anzahl Paare. Bei der Arbeit mit den Artikeln wird darauf geachtet, dass keine doppelten Artikelnummern existieren. Referenzen zu allen existierenden Artikeln liegen in der Datenbank vor.

Ein Mainprogramm testet die so aufgebauten Klassen. Dabei arbeitet *Main* als eigenständige Java Anwendung und kommuniziert mit dem Benutzer in der Konsole mithilfe der (vereinfachten) Benutzeroberflächenklasse. Um so viele Testszenarien wie möglich anzubieten, verfügt das Programm über einem interaktiven Menü, womit der Benutzer ausgewählte Funktionen der vorhandenen Klassen testen kann.

Weil eine Datenbank gar nicht zur Verfügung steht, legt das Testprogramm die Artikel, die Benutzer und den Bestand „per Hand“ an. Dabei sind eventuelle Änderungen ausgeschlossen oder schwierig. Aus der Sicht der Datenbank würde sich auch anbieten, diese drei Entitäten als eigene Relationen aufzubauen. Wegen dem begrenzten Umfang dieser Arbeit wurde aber darauf verzichtet. Die Operationen, die in dem Webshop vor oder nach der Interaktion mit dem Benutzer - wie Artikel und einen Bestand anlegen, Benutzer verwalten oder Bestellungen bearbeiten wurden übersprungen oder vom Testprogramm simuliert. Die Benutzeroberfläche ist stark begrenzt und ausschließlich auf Testen orientiert.

Literaturverzeichnis

Oracle Corporation (o.J.)
Learning the Java Language,
https://docs.oracle.com/javase/tutorial/java/TOC.html (Zugriff am 23.03.2019)

Oracle Corporation (o.J.)
Primitive Data Types
https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html (Zugriff am 23.03.2019)

Zukowski, John (2007)
The need for BigDecimal
https://blogs.oracle.com/corejavatechtips/the-need-for-bigdecimal (Zugriff am 23.03.2019)

Oracle Corporation (o.J.)
Java Platform, Standard Edition & Java Development Kit Version 10 API Specification
https://docs.oracle.com/javase/10/docs/api/overview-summary.html (Zugriff am 23.03.2019)

Oracle Corporation (o.J.)
Overriding and Hiding Methods
https://docs.oracle.com/javase/tutorial/java/IandI/override.html (Zugriff am 23.03.2019)