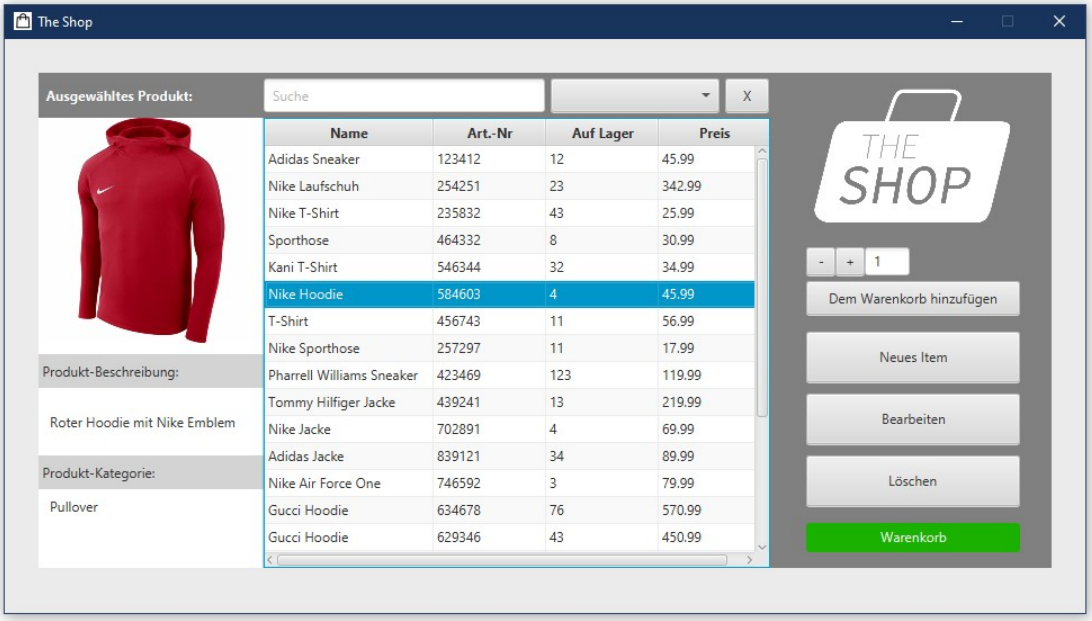
Software-Entwicklung 2

Sommersemester 2020

Lager- und Einzelhandelsmanager
The Shop



https://gitlab.mi.hdm-stuttgart.de/ns144/se2

Hannes Koksch (hk058)

Nikolas Schaber (ns144)

Florian Fallscheer (ff054)

1. Kurzbeschreibung

Bei unserem Projekt handelt es sich um einen Lagermanager bzw. ein kleines Shop-System welches z. B. bei Einzelhändlern zum Einsatz kommen könnte.

Der Lagermanager umfasst die Anzeige und Verwaltung von Artikeln mit den Funktionen, Artikel neu anzulegen, zu bearbeiten, zu löschen und auch zu verkaufen.
Auf dem Main-Screen können die Details der Artikel angezeigt werden und man hat durch eine nach den verschiedenen Artikel-Attributen sortierbare Liste eine Übersicht über die vorhandenen Artikel. Als zusätzliche Filtermöglichkeit kann man auch die Liste durchsuchen oder eine einzelne Artikelkategorie auswählen.
Artikel können mit einer einzigartigen 6-stelligen Id und weiteren Attributen in den entsprechenden Fenstern hinzugefügt und bearbeitet werden.
Beim Hinzufügen von Artikeln zum Warenkorb werden diese aus dem Lager entfernt und dem Warenkorb hinzugefügt. Nach Auswahl der Rechnungsart im Fenster des Warenkorbs kann man die Bestellung abschließen und es wird modellhaft ein Kassenzettel bzw. eine Rechnung in die Konsole geprintet.

2. Startklasse

Die Main Methode befindet sich in der Klasse App.

3. Besonderheiten

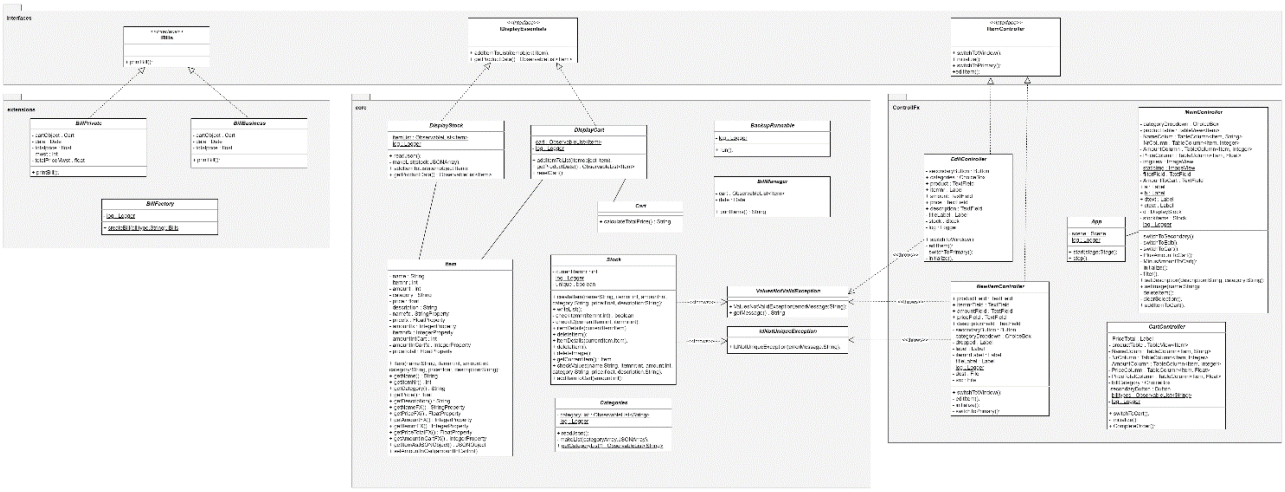
- Nachdem neue Artikel mit Bildern hinzugefügt wurden, muss die Anwendung neu gestartet werden, damit die Bilder mit geladen werden. Ist während der Laufzeit der Anwendung wohl nicht möglich.

4. Diagramme

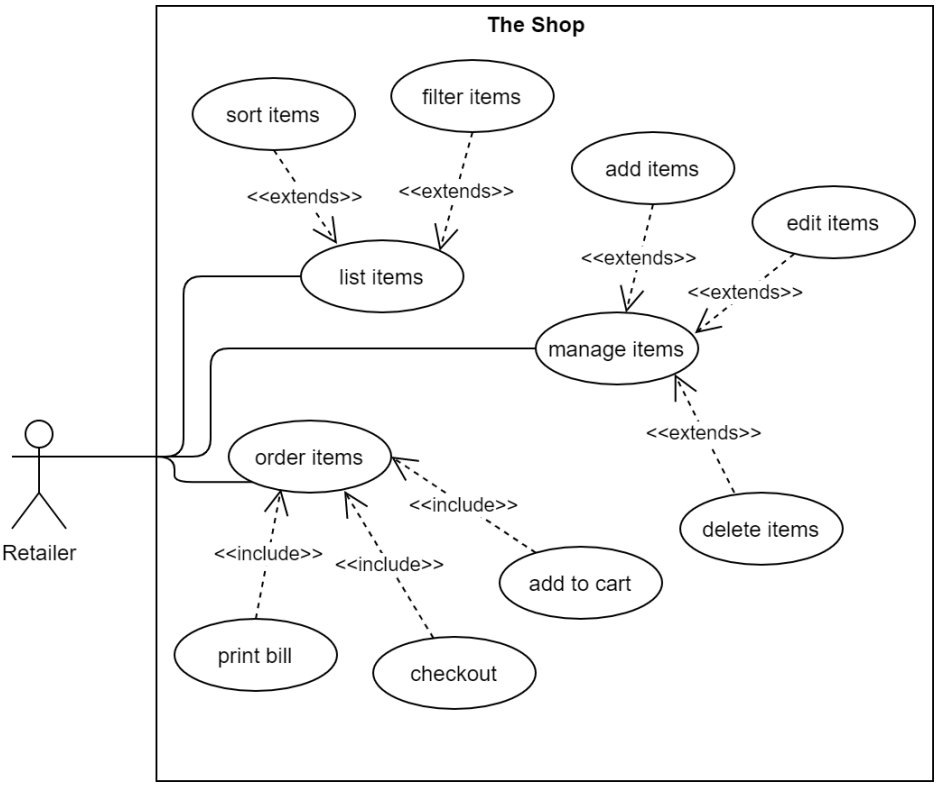
Beide Diagramme befinden sich als *.html*- oder als *.drawio*-Datei zusätzlich auch noch im Repository unter **/Diagramme.**

Klassendiagramm

➔ Für höhere Auflösung bitte Dateien in Repository benutzen!



Use-Case Diagramm



5. Stellungnahme

Architektur

Interfaces:

- **IBills** (wird implementiert von BillBusiness, BillPrivate)
- **IDisplayEssentials** (wird implementiert von DisplayCart, DisplayStock)
- **IItemController** (wird implementiert von NewItemController, EditController)

Die Architektur ist dahingehend erweiterbar, als dass es mit dem Interface IBills und der Factory BillFactory möglich ist, andere Formen der Rechnungserstellung und andere Rechnungsarten zu implementieren. Denkbar sind hier weitere Rechnungen im Stil von BillPrivate oder BillBusiness, z.B. für Angebote, andere Steuersätze oder allgemein für andere Kunden oder aber ganz andere Ausgabemethoden, wie z.B. eine Rechnungserstellung als PDF.

Clean Code

Es werden keine schreibbaren Referenzen auf Objekte übergeben, Referenzen werden kopiert vor Übergaben. In den Kernklassen werden keine public member verwendet.
Statics waren nötig bei der Verwaltung von Daten (ObservableList) sowie einzelnen GUI Elementen.

Tests

- **TestBackup** testet die korrekte Funktionsweise des Backup-Threads, der in regelmäßigen Abständen (alle 10 Sekunden) die Liste der lagernden Items als *json*-Datei abspeichert.
  Der Test löscht diese Datei und überprüft anschließend deren Existenz nach einer Wartezeit von 11 Sekunden. Der Test muss während der Laufzeit der Anwendung ausgeführt werden!
- **TestItem** testet die Funktionen der Item Klasse, explizit die Verwaltung der Anzahl der Artikel im Warenkorb.
- **TestStock** testet die Funktionen der Stock Klasse insbesondere in Bezug auf die beiden Exceptions. Hier wurden unter anderem auch Tests zu falsch Eingaben (**Negativtests**) eingebaut.
- **TestJSON** testet ob JSON Files erfolgreich beschrieben und ausgelesen werden können.

GUI (JavaFX)

Die Grafische Oberfläche wurde anhand von *fxml*-Dateien erstellt, die sich im Ordner **resources/ControllFx** befinden. Im Package **ControllFx** befinden sich die Controller für die verschiedenen Scenes. Mit Einbindung von *CSS* wurden manche Standard-Elemente etwas angepasst, um die Usabillity zu verbessern.

Exceptions

Es wurden zwei Exceptions angelegt:

- **IdNotUniqueException** wird ausgegeben, wenn die gewünschte ID/Artikelnummer beim Erstellen eines Artikels bereits vergeben ist.

- **ValuesNotValidException** wird ausgegeben, wenn die Angaben bei der Erstellung oder Bearbeitung eines Artikels nicht den Vorgaben entsprechen. Zusätzlich lässt sich eine message abrufen (getMessage()) um die Ursache/den Fehler zu erhalten.

Logging

Konfiguration durch **log4j2.xml** im Ordner **resources**.
Allgemein werden alle Logs mit root gesammelt und anhand ihres Log-Levels an die verschiedenen Appenders verteilt.

- **Exception-Logging** (normale Exceptions und oben beschriebene Business-Exception)
  → **log.warn()** logging in Konsole
- **Stockchanges** (Alle Veränderungen, die im Programm an Items im Stock getätigt werden)
  → **log.info()** logging in **stockchanges.log**
- **Andere Informationen der Applikation** (Applikation gestartet, Thread gestartet…)
  → **log.debug()** logging in Konsole

UML

Das Klassendiagram beschreibt alle Zusammenhänge/Beziehungen zwischen den verschiedenen Klassen des Projekts.
Zudem wurde die Zugehörigkeit der Klassen zu den jeweiligen Packages kenntlich gemacht.

Das Use-Case-Diagramm zeigt auf, wer etwas in unserem System tut und was getan wird (Anwendungsfälle). Diese Anwendungsfälle wurden um die beteiligten Akteure erweitert. Akteure sind widerum Anwendungsfälle, die entweder ein Logischer Teil eines Anwendungsfalls sein können (<<include>>), oder diesen optional erweitern (<<extends>>).

Threads

BackupRunnable führt alle 10 Sekunden ein „Backup“ im Hintergrund aus, indem es die ganze Observable List itemlist aus DisplayStock mit der writeList() Methode der Klasse Stock in einer *json*-Datei abspeichert.

Die Threads (BackupThread und ApplicationThread) werden in **start()** der Main-Klasse **App** gestartet.

Streams und Lambda-Funktionen

Unter anderem wurden Streams in der Suchfunktion für Produkte innerhalb der Observable List verwendet – siehe MainController-Klasse – filter-Methode. Parallel Streams wurden in der Methode checkitemnr innerhalb der Stock-Klasse verwendet, um die bisherigen Daten auf die neue Artikelnummer zu durchsuchen, sodass man sicherstellen kann, dass die ID eindeutig ist.

Factories

Die **BillFactory** ist Teil des Packages **extentions** und gibt das passende Rechnungsobjekt zum im Warenkorb ausgewählten Billtype, dem Rechnungstyp, zurück.
Weitere Implementierungen sind durch anpassungen im extentions Package gut möglich. Siehe auch oben bei Stellungnahme zu erweiterbarer Architektur.

6. Bewertungsbogen

Arc.	Clean Code	Doku	Tests	GUI	Logging/Except.	UML	Threads	Streams	Profiling	Summe - Projekt
3	3	3	3	3	3	3	3	3	3	30,00