Tatyana Merlo – Natalie Stalder – Nadja Stadelmann

Starbucks Manager

M120 – Benutzerschnittstelle entwickeln

# Einleitung

In einem vorherigen Projekt haben wir für Starbucks ein Menümanager geschrieben. Die Mitarbeiter können Produkte zum Menu hinzufügen, bearbeiten und wieder löschen. Weiter wird das Menü natürlich auch für die Kunden angezeigt.

Bisher wurde kein GUI eingesetzt, weshalb für die Klienten von Starbucks ein eher unschönes Menü angezeigt wurde. Weiter konnte das Programm nur über die Konsole bearbeitet werden, sodass eine spezielle Mitarbeiterschulung für den Gebrauch des Starbucks Managers nötig war.

Dies wollen wir in diesem Folgeprojekt nun ändern.

## Dokument

### Aufbau

* Analyse
  + Beschrieb und erste Aufsplittung in einzelne Teilbereiche
* Vorgehen
  + Abhängigkeiten, Aufgabenübersicht
* Design
  + Detaillierte Analyse DB-Design
  + Detaillierte Analyse Programmstruktur
* Implementation
* Testing
  + Testcases basieren auf UseCases

Inhalt

[Einleitung 1](#_Toc496680889)

[Dokument 1](#_Toc496680890)

[Aufbau 1](#_Toc496680891)

[Vision 4](#_Toc496680892)

[Analyse 4](#_Toc496680893)

[Anforderungen 4](#_Toc496680894)

[Muss-Kriterien 4](#_Toc496680895)

[Kann-Kriterien 4](#_Toc496680896)

[Use Cases 5](#_Toc496680897)

[Story-Bord 6](#_Toc496680898)

[Design 7](#_Toc496680899)

[Filestruktur 7](#_Toc496680900)

[Programm 7](#_Toc496680901)

[Model 7](#_Toc496680902)

[View 7](#_Toc496680903)

[Controller 8](#_Toc496680904)

[Klassendiagramm 8](#_Toc496680905)

[GUI 8](#_Toc496680906)

[Interaktion mit Model 8](#_Toc496680907)

[Überlegungen zum GUI 9](#_Toc496680908)

[Klärung der Ebenen 9](#_Toc496680909)

[Aufgaben-Ebene 9](#_Toc496680910)

[Semantische Ebene 9](#_Toc496680911)

[Syntaktische Ebene 9](#_Toc496680912)

[Interaktionsebene 9](#_Toc496680913)

[Benutzer-Arten 10](#_Toc496680914)

[Weitere Aspekte 10](#_Toc496680915)

[Aufbau 10](#_Toc496680916)

[Dialoge 10](#_Toc496680917)

[Styleguides 10](#_Toc496680918)

[Implementation 11](#_Toc496680919)

[Implementierung 11](#_Toc496680920)

[Verantwortlichkeiten 11](#_Toc496680921)

[Technischer Aufbau 11](#_Toc496680922)

[GUI 11](#_Toc496680923)

[DataHolder 11](#_Toc496680924)

[Verbindung zum Model 11](#_Toc496680925)

[Entwicklungsprozess 11](#_Toc496680926)

[Testing 13](#_Toc496680927)

# Vision

Im Modul 326 haben wir ein Starbucks-Menümanager mit Java erstellt. Bisher konnte der jedoch nur textbasiert verwendet werden (mittels Konsole).

Unser Ziel ist nun, den Mitarbeitern das Mutieren des Menüs mit einem GUI zu erleichtern. Das GUI soll alle bisherigen Userinteraktionen abdecken, sprich:

* Menü anzeigen
* Produkt hinzufügen
* Produkt bearbeiten
* Produkt löschen
* Programm beenden

Weiter möchten wir eine Login- sowie Logoutfunktion einbauen, um die Sicherheit der Daten zu gewährleisten.

Projektauftrag einfügen

# Analyse

## Anforderungen

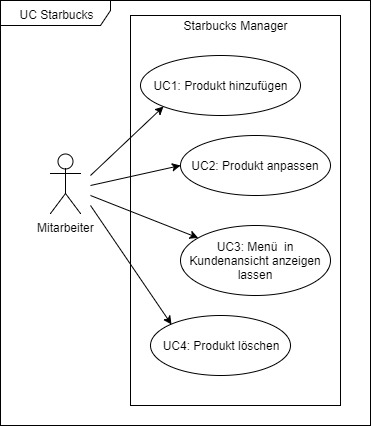
### Muss-Kriterien

1. Menüanzeige für die Kunden
   1. an Starbucks-Design angepasst
   2. gegliedert nach Produktkategorien
2. Menüanzeige für den Mitarbeiter
   1. Edit-Funktion zu jedem Produkt
   2. Delete-Funktion zu jedem Produkt
   3. selbe Gliederung wie bei der Kundenansicht
   4. Überbegriffe der einzelnen Felder sichtbar (bspw. Price)
3. Add-Funktion
   1. eigene Ansicht
   2. Auswahl der Produktkategorie
   3. Eingabe der Produktinformationen
      1. nur die relevanten Infos müssen eingegeben werden
4. Edit-Funktion
   1. Item bearbeiten
5. Delete-Funktion
   1. Item löschen
6. Layout
   1. Einheitlich
   2. Schlicht
   3. Gemäss Kapitel sowieso
7. Error-Handling
   1. genaue Informationen für den User

### Kann-Kriterien

1. Preis-Eingabe
   1. Dialogfenster mit Taschenrechnerlayout
2. Add-Funktion
   1. Speicherung relevante Daten beim Wechsel der Produktkategorie
3. Edit-Funktion
   1. Produktkategorie anpassen können
4. Help-Funktion
   1. Beschrieb der Funktionen
   2. Beschrieb des Files product.txt
5. Error-Handling
   1. Dialogfenster
      1. Message
      2. OK-Button als Userbestätigung

## Use Cases



|  |  |
| --- | --- |
| **Identifikation** | **UC1: Produkt hinzufügen** |
| **Beschreibung** | Ein Produkt wird hinzugefügt. |
| **Primärer Actor** | Mitarbeiter |
| **Vorbedingungen** | - |
| **Trigger** | Mitarbeiter wählt Menü „Add“ |
| **Standardablauf** | 1. Wählt Menü „Add“  2. Wählt Produktkategorie  3. Gibt Produktdaten ein  4. Produkt wird hinzugefügt |
| **Alternative Abläufe** | 2a. falsche Produktkategorie gewählt 🡪 User kann Kategorie anpassen ohne Datenverlust  3a. Produkt bereits vorhanden 🡪 Fehlermeldung ausgeben  3b. Falsche Zeichen eingegeben 🡪 Fehlermeldung ausgeben |

|  |  |
| --- | --- |
| **Identifikation** | **UC2: Produkt anpassen** |
| **Beschreibung** | Ein bestehendes Produkt wird editiert |
| **Primärer Actor** | Mitarbeiter |
| **Vorbedingungen** | Produkt vorhanden  Menü „Modify“ ausgewählt |
| **Trigger** | Mitarbeiter wählt Option „edit“ |
| **Standardablauf** | 1. Wählt Option „edit“  //noch anschauen wie natalie es handelt  2. Produkt wird angepasst |
| **Alternative Abläufe** | 2a. Falsche Zeichen eingegeben 🡪 Fehlermeldung ausgeben |

|  |  |
| --- | --- |
| **Identifikation** | **UC3: Menü in Kundensicht anzeigen** |
| **Beschreibung** | Das Menü wird strukturiert und im Starbucks-Design ausgegeben |
| **Primärer Actor** | Mitarbeiter |
| **Vorbedingungen** | - |
| **Trigger** | Mitarbeiter wählt Menü „Home“ / Programmstart |
| **Standardablauf** | 1. Wählt Menü „Home“ / startet Programm  2. Produkte werden nach Typ sortiert ausgegeben |
| **Alternative Abläufe** | - |

|  |  |
| --- | --- |
| **Identifikation** | **UC4: Produkt löschen** |
| **Beschreibung** | Ein bestehendes Produkt wird entfernt |
| **Primärer Actor** | Mitarbeiter |
| **Vorbedingungen** | Produkt vorhanden  Menü „Modify“ ausgewählt |
| **Trigger** | Mitarbeiter wählt Option „delete“ |
| **Standardablauf** | 1. Wählt Option „delete“  //Noch anschauen wie natalie es handelt  2. Produkt wird entfernt |
| **Alternative Abläufe** | - |

## Story-Bord

inklusive mockups (inklusive Dialogspezifikation)

🡪 hier erwartete Eingaben und Ausgaben genau beschreiben

# Design

## Filestruktur

Der bisherige Starbucks Manager hat keine Datenbank, welche er verwaltet. Die Daten werden in einem einfachen File abgespeichert. Dennoch ist die Struktur des Files natürlich extrem wichtig für die Datenverarbeitung, weshalb sie hier kurz deklariert ist.

Jede Zeile steht für einen Menü-Eintrag, der in einzelne Komponenten unterteilt ist. Je nachdem, welche Attribute der Eintrag mitführt, wird er in eine andere Kategorie unterteilt.

**Kategorie Recordstruktur**

Beverage Name | Preis | null | true (hot) / false (cold)  
beverage¦2.3¦null¦true

Coffee Name | Preis |Zutaten  
coffee¦1.4¦ingr

Extra Name | Preis  
extra¦2.0

Food Name | Preis |Zutaten | Vegi / Vegan / Glutenfrei / usw.  
food¦1.5¦Ingr¦Vegi

## Programm

### Model

Wie bereits erwähnt werden die Daten in einem einzigen File gehandhabt. Das File und sein Handling besteht bereits. Dennoch müssen am Datenmodel noch einiges angepasst werden, um die korrekte Zusammenarbeit mit dem GUI sicherzustellen.

### View

Das GUI ist komplett neu umzusetzen. Bisher waren nur Kommandozeileneingaben zur Steuerung des Programmes da. Dies soll neu über ein GUI geregelt werden, um den Mitarbeitern einen leichteren Einstieg zu ermöglichen.

So stellen wir uns das Ergebnis des GUIs vor:

Die Kundenansicht soll dem Starbucks-Design angepasst sein. In ihrem Logo ist ein spezielles Grün zu finden. Weiter zeigt es eine Weisse Figur auf schwarzem Hintergrund. Um den Kunden eine möglichst angenehme Ansicht bieten zu können, wird für das GUI das Grün als Hintergrund eingesetzt. Darauf positionieren werden die Artikel in weisser, serifenloser Schrift. Diese garantiert einen guten Lesefluss. Um dem Kunden die Auswahl möglichst zu vereinfachen, werden die Produkte in vier Kategorien unterteilt. Die Kaffees sind zu Oberst zu finden, da die meisten Kunden Starbucks wegen diesen Produkten besuchen. Danach folgen die Beverages (weitere Getränke), die Extras und zum Schluss noch die Muffins und Cookies unter der Kategorie Food.

Für die Mitarbeiteransicht wollen wir ein einheitliches Layout erstellen, damit der User sich schnell auf allen Seiten zurechtfinden. Es soll ausserdem übersichtlich sein, das heisst für uns, dass der User immer nur das angezeigt bekommt, was er wirklich braucht. Als Beispiel ist hier die Funktion ‘Produkt hinzufügen’ super geeignet. Ein Kaffee oder ein Extra benötigen weniger Angaben als ein neues Produkt der Kategorie Food. Also sollen auch nur diese Angaben abgefragt werden, welche notwendig sind.

Für all diese Views verwenden wir CSS für JavaFX, um die verschiedenen Teile des GUIs einheitlich gestalten zu können.

### Controller

Die Controller bestehen teilweise, es wurden bereits Usereingaben verarbeitet. Dennoch müssen sie an das GUI angepasst werden, da das Programm bisher nur über die Konsole bedient werden konnte. Teils müssen sie ganz ausgewechselt werden.

Einige Überprüfungen der Usereingaben werden neu vom GUI bereits abgefangen. So wird die Businesslogik entlastet, da nicht mehr alle Eingaben auf ihre Richtigkeit überprüft werden müssen. Ein Preis, welcher vom GUI weitergeliefert wird, ist beispielsweise immer ein double wenn er im Controllerbereich ankommt.

## Klassendiagramm

### GUI

### Interaktion mit Model

# Überlegungen zum GUI

Da sich dieses Modul vor allem um die GUI-Implementierung und das Design des GUIs dreht, gehen wir in diesem Kapitel genauer auf unsere Überlegungen dazu ein.

## Klärung der Ebenen

Als erstes müssen einige Aspekte geklärt werden, um genau eruieren zu können, was das GUI alles können muss. Dies geschieht mittels verschiedenen Ebenen, welche hier genauer erläutert und auf unser GUI bezogen bearbeitet werden.

### Aufgaben-Ebene

Als erstes stellen wir uns die Frage, welches nun die primären Aufgaben des Benutzers sind. Nur wenn wir wissen, für was der Benutzer das Programm im Endeffekt benützen wird, können wir es auch entsprechend gestalten. In unserem Fall sind dies folgende Aufgabe:

* Starbucks-Menü kundenfreundlich und aktuell anzeigen lassen

Um diese Endaufgabe erreichen zu können, muss der Benutzer auch folgende Aufgaben bewältigen können:

* Produkte hinzufügen
* Produkte bearbeiten
* Produkte löschen

### Semantische Ebene

Weiter ist es wichtig zu wissen, welche Werkzeuge dem Benutzer bereits zur Verfügung stehen und was noch bereitgestellt werden muss.

In unserem Fall besteht bereits das Model und die Businesslogik vom Programm, welches auf einem handelsüblichen PC laufen wird. Ein grosser Fernseher wird als Anzeige für die Kundenansicht verwendet.

Wir müssen nun noch das GUI zur Verfügung stellen.

### Syntaktische Ebene

Auf dieser Ebene ist die Frage nach dem Wissen des Benutzers extrem wichtig. Dies kann von Benutzer-Art zu Benutzer-Art unterschiedlich ausgeprägt sein. Je nach Komplexität des Programmes und nach Arbeitsteilung muss das Programm dementsprechend angepasst werden. Allenfalls müssen die Benutzer sogar geschult werden, damit das Programm im Arbeitsalltag (oder auch im Privatgebrauch) korrekt und effizient eingesetzt werden kann.

In unserem Fall handelt es sich um durchschnittliche End-User-Anwender. Einige Personen haben bereits Erfahrung mit dem bestehenden Programm, andere werden neu mit dem Programm arbeiten. Die Benutzer wissen, welche Produkte welcher Kategorie zuzuordnen sind und kennen ihre jeweiligen Eigenschaften.

### Interaktionsebene

Wie der Benutzer mit dem System interagieren kann, sprich wie er mit dem Programm kommunizieren kann ist für den Benutzer sehr zentral. Beim Starbucks Manager kann er dies per Maus und Tastatur machen.

## Benutzer-Arten

Grundsätzlich können verschiedene Benutzer sehr verschiedene Fähigkeiten und Wünsche mitbringen. Diese können sich im Laufe der Zeit verändern. Man unterscheidet meist Neulinge, geübte Benutzer und Profis.

Unser GUI wird sehr übersichtlich aufgebaut sein und für falsche Eingaben werden direkt mit Fehlermeldungen ausgegeben. So soll die Intuition und der Lerneffekt des Benutzer ihn schnell zu einem geübten Benutzer verändern. Daher legen wir das GUI auf diese Benutzergruppe aus (Reduktion). Für unerfahrene Benutzer stellen wir dafür den Menüpunkt ‘Help’ zur Verfügung, der die Anwendung kurz beschreibt.

Für Profis wäre allenfalls eine Möglichkeit, die Fehlerdialoge bei falschen Eingaben auszuschalten, um den User bei Tippfehlern nicht zu stören. Dies wird jedoch erst in einem weiteren Schritt implementiert – je nach Wunsch der User.

## Weitere Aspekte

### Aufbau

Damit sich ein neuer Benutzer in unserem Programm schnell zurechtfindet, halten wir uns an gewisse Standards. Die Menüleiste ist zuoberst zu finden, unten wird das Programm mit einem Footer abgeschlossen. Ganz links wird die Home-Seite eingeblendet, sie ist gleichzeitig der Einstiegspunkt. Die Kundenansicht wird nämlich jeden Morgen aufgestartet, Anpassungen am Menü werden tendenziell seltener gebraucht. Die Funktion ‘Help’ befindet sich wie bei den meisten Programmen ganz rechts in der Menüleiste.

Weiter verwenden wir ein Icon, damit der User in der Taskleiste auf den ersten Blick sehen kann, wo er den Starbucks Manager findet. Dies kann hilfreich sein, wenn er noch weitere Programme geöffnet hat und zwischen den Programmen wechseln will.

### Dialoge

Meldungen werden immer in einem Dialog geöffnet, welche der Benutzer mit einem Button bestätigen und gleichzeitig beenden kann. Dies ist er sich von den meisten bestehenden Programmen ebenfalls schon gewöhnt. Diese Dialoge sind modal, das bedeutet, der User muss zuerst den Dialog schliessen um weiterarbeiten zu können. Dies stellt sicher, dass die Fehlermeldung zur Kenntnis genommen wird.

Für die Eingabe des Preises wird eine Metapher verwendet. Eine Art Taschenrechner, welcher auf die benötigten Funktionen begrenzt ist, öffnet sich und mittels Klicks auf die Buttons füllt die Zahlen korrekt dem double-Format entsprechend ein. Auch dieser ist modal eingestellt, um keinen Datenverlust zu erleiden.

Die Menü-Option ‘Help’ öffnet einen nicht-modalen Dialog, in dem die Funktionen des Programmes erklärt sind. So kann der User im Help-Dialog nachschauen wie etwas funktioniert und es gleichzeitig im GUI ausführen.

### Styleguides

Für die Kundenansicht werden die Starbucks-Farben verwendet. So wird das Corporate Design eingehalten, der Kunde merkt sofort, dass dieses Menü zu einer Starbucks-Filiale gehören muss.

Als Icon verwenden wir das Starbucks-Logo, so findet der Mitarbeiter sofort das korrekte Programm, wenn er den Starbucks Manager sucht.

# Implementation

## Implementierung

### Verantwortlichkeiten

Für die Implementierung haben wir unser GUI in drei Verantwortlichkeitsbereiche unterteilt. Für den jeweiligen Verantwortlichen bedeutet dies, dass die Implementation sowie das Testing der jeweiligen Funktionen seine Aufgabe ist. Natürlich bestehen einige Abhängigkeiten zwischen den einzelnen Bereichen, die wir miteinander besprechen werden. So wird auch unsere Zusammenarbeit gewährleistet.

Die Kundenansicht, welche mittels WebView (???) gestaltet wird, ist unter Tatyana Merlo’s Verantwortlichkeit. Auch das Grundlayout gehört in ihren Bereich. Das Hinzufügen der Daten und die Anzeige der ‘Help’-Seite liegt unter Nadja Stadelmann’s Bereich. Natalie Stalder ist für die Mitarbeiteransicht zuständig, welche gleichzeitig die Funktionen ‘Produkt bearbeiten’ und ‘Produkt löschen’ beinhaltet.

### Technischer Aufbau

#### GUI

Unser Template beinhaltet das Fenster der Applikation inklusive Titel, Menüleiste und Footer. Je nach Auswahl der Menübuttons durch den User werden nun die eigentlichen Fensterinhalte ausgetauscht. Der Inhalt der Kundenansicht wird betreffend des Layouts speziell behandelt. Alle Mitarbeiteransichten werden mittels demselben CSS-Stylesheet formatiert. Dadurch stellen wir sicher, dass der User bei der Anwendung eine einheitliche Ansicht zu sehen bekommt.

#### DataHolder

Um die Datenintegrität sicherzustellen, wird ein DataHolder verwendet. Er beinhaltet die möglichen Daten eines einzelnen Produktes.

Die Views AddItem und Modify verwenden diesen, um für den User sicherzustellen, dass seine Daten zwischengespeichert sind. So klickt der User beispielsweise auf Add > Coffee und gibt den Namen eines neuen Produktes der Kategorie Beverage ein. Um ihm den Kategorienwechsel zu vereinfachen, werden die bereits eingegebenen Daten aus dem DataHolder in die neue Anzeige eingespeist.

#### Verbindung zum Model

Die Anbindung des GUIs an das Model wird über die Klasse ‘Menu’ gestaltet. In jener Klasse war zuvor die Konsoleneingaben und -überprüfungen zu finden. Sie hat bereits vorher schon mit dem Model kommuniziert, weshalb einige bestehende Methoden nun nur noch umgebaut werden müssen.

### Entwicklungsprozess

Bei der Implementation haben wir als erstes das Template erstellt. Parallel dazu wurden die Grundstrukturen der Menüoptionen ‘Modify’ und ‘Add’ festgelegt. Anschliessend folgte die Ausarbeitung dieser beider Menüs, sowie das Erstellen unserer Kundenansicht.

Zuerst wurden die notwendigen Elemente wie Labels, Textfelder und Buttons erstellt und ungefähr in die Grundstrukturen eingefügt. Dann ging es vor allem um das Datenhandling. Es folgte die Implementation der notwendigen Überprüfungen, sodass die Daten bereits im richtigen Format an die Businesslogik eingespeist werden.

Die Überprüfung des Preises wurde speziell implementiert. Wir bauten einen bestehenden Taschenrechner um, sodass er unseren Ansprüchen genügt. So kann nun mittels einiger Überprüfungen sichergestellt werden, dass der Preis immer als double gespeichert wird.

Je nach Useraktion müssen die Daten in verschiedenen Feldern beibehalten werden (bspw. genau bei unserer Preiseingabe: der eingegebene Preis muss auf das Preis-Textfeld in der jeweiligen Ansicht übertragen werden. Dafür haben wir eine eigene Klasse erstellt, den ‘DataHolder’. Diese ist zuständig für die Daten eines einzelnen Produktes und wurde zunächst static implementiert. Die Überlegung dafür war, dass der DataHolder grundsätzlich nicht als einzelne Instanz benutzt wird, sondern nur die Daten darin relevant sind. Dies hat jedoch anschliessend nicht so funktioniert wie gedacht. Deshalb bauten wir für diesen Datenaustausch das Observer-Pattern ein. Der DataHolder ist observable, die Views sind seine jeweiligen Observer. So wird die Datenintegrität zwischen dem DataHolder und den Views sichergestellt.

Am Schluss folgte die definitive Anbindung an das Model, sodass das Programm am Ende auch wirklich funktioniert und die verschiedenen Elemente hinzugefügt, angezeigt, bearbeitet und gelöscht werden können.

Parallel zu diesem Prozess schrieben wir das Error-Handling mittels des Errors-Dialogs, welcher überall verwendet werden kann. So wird die Einheitlichkeit für die Meldungen sichergestellt. Das Layout haben wir ebenfalls parallel zum Prozess verbessert.

Ganz am Ende fand die Implementation des ‘Help’-Dialoges statt für unerfahrene User, welche eventuell etwas nicht verstehen.

# Testing

Neben Tests auch ein Benutzer-Feedback verfassen