Code Dokumentation:

Cash-Register-System

erstellt von:
Daniel Albrecht
Tobias Rieß
Karl-Erik Kley

1. **Funktionalitäten**
2. **Funktionsübersicht**
    a. **UML**
3. **Kommunikation**
4. **Software**
5. **Nutzung**
6. **Zukünftige Aussichten**

1. Funktionalitäten

Am Beginn des Projekt haben wir uns verschiedene Gedanken gemacht welche Kernfunktion sowie Hilfsfunktionen/Klassen benötigt werden um ein gut funktionierendes System zu Programmieren.

Als erstes wird der Grundprozess benötigt ein Produkt in den Bestand einzufügen sowie dieses auch wieder heraus zu löschen und im Zuge dieses Prozesses auch Parameter wie den Preis, die Menge und auch die Versteuerung festzulegen (19% oder 7%).
Darauf aufbauend sollte es auch die Möglichkeit zu geben bestehende Produkte im nachhinein anzupassen.

Natürlich müssen neu angelegte Produkte permanent speicherbar sein weshalb diese mittels einer write Inventory-Funktion in eine inventory.txt geschrieben wird, das heißt aber auch im Umkehrschluss das eine read Inventory-Funktion existiert welche den Bestand aus der inventory.txt in das Kassensystem liest.
Dieser Bestand wird in eine Liste gelesen mit welcher, in der Zeit wo Artikel gesucht werden, gearbeitet wird.

Auf diesem Prinzip ist es nun möglich ein Warenkorb für einen Kunden zu erstellen welchen wir füllen können mit den Artikeln die in der Liste liegen aber gleichzeitig auch der Preis errechnet wird mittels separater MwSt für die unterschiedlichen Artikel.

Dazu soll es möglich sein dem Kunden einen gesamten oder ein einzelnen Rabatt zu geben welche auch verrechnet werden muss.
Außerdem ist es notwendig für Produkte auch einen separaten Preis für ein einzelnes Produkt einzugeben.

Mittels Dieser Funktionen ist es nun möglich ein Kassen System zu erstellen welches verschieden Rechnungen für die Kunden erstellt gleichzeitig pro gekaufte Ware den Bestand verringert und durch dessen Differenzen einen Umsatz erzeugt welcher auch ausgegeben werden kann.

Das ganze wird durch einen Handscanner unterstützt welcher es erlaubt die Barcodes einfach zu scannen und somit schnell zu arbeiten.

2. Funktionsübersicht

Klasse:

- cart: article (List):
    - String sName
    - int iAmount
    - double dPrice
  - discount (int in percent)
  - fullPrice(double)
  - Pricew/oTax (double)
  - tax (double)
  - set/get -Funktionen
    - programmiert Karl

Cash-Register-System:

- invetoryArticle

- neuer Artikel in Rechnung (mit Anzahl der Artikel)
  - bool addArticle (cart ccart, char[] cBarcode,int iAmount)
  - programmiert: Karl

- Artikel entfernen (mit Anzahl der Artikel)
  - bool delArticle (cart ccart, char[] cBarcode, int iAmount)
  - programmiert: Karl

- Rabatt (auf alles)
  - bool discount (cart actualcart, int iDiscount)
  - programmiert: Daniel

- manuelle Preiseingabe(Rabatt auf kaputten Artikel, Artikel aus Warenbestand)
  - bool otherPrice(cart ccart,char[] cBarcode, double dPrice)
  - programmiert: Daniel

- berechnet Preis mit neuem Objekt
  - bool addToPrice(cart ccart,char[] cBarcode, int iAmount)

    - bool removeFromPrice(char[] cBarcode, int iAmount)
    - programmiert: Tobias

- Fehler falls Artikel nicht im Sortiment, Bestand(Kontrolle für addArticle)
    - void wrongArticle (char[] cBarcode)
    - programmiert: Tobias

- Artikel welcher hinzugefügt wurde (Preis, Name, Bild, aktuellen Preis)
    - void displayArticle (char[] cBarcode, cart ccart)
    - programmiert: Daniel

- Neue Ware anlegen
    - bool newItem(char[] cBarcode, String sName, double dPrice,int iAmount, boolean bisFood)
        - programmiert: Karl
- Ware entfernen
    - bool remove_item(char[] cBarcode)
        - programmiert: Tobias

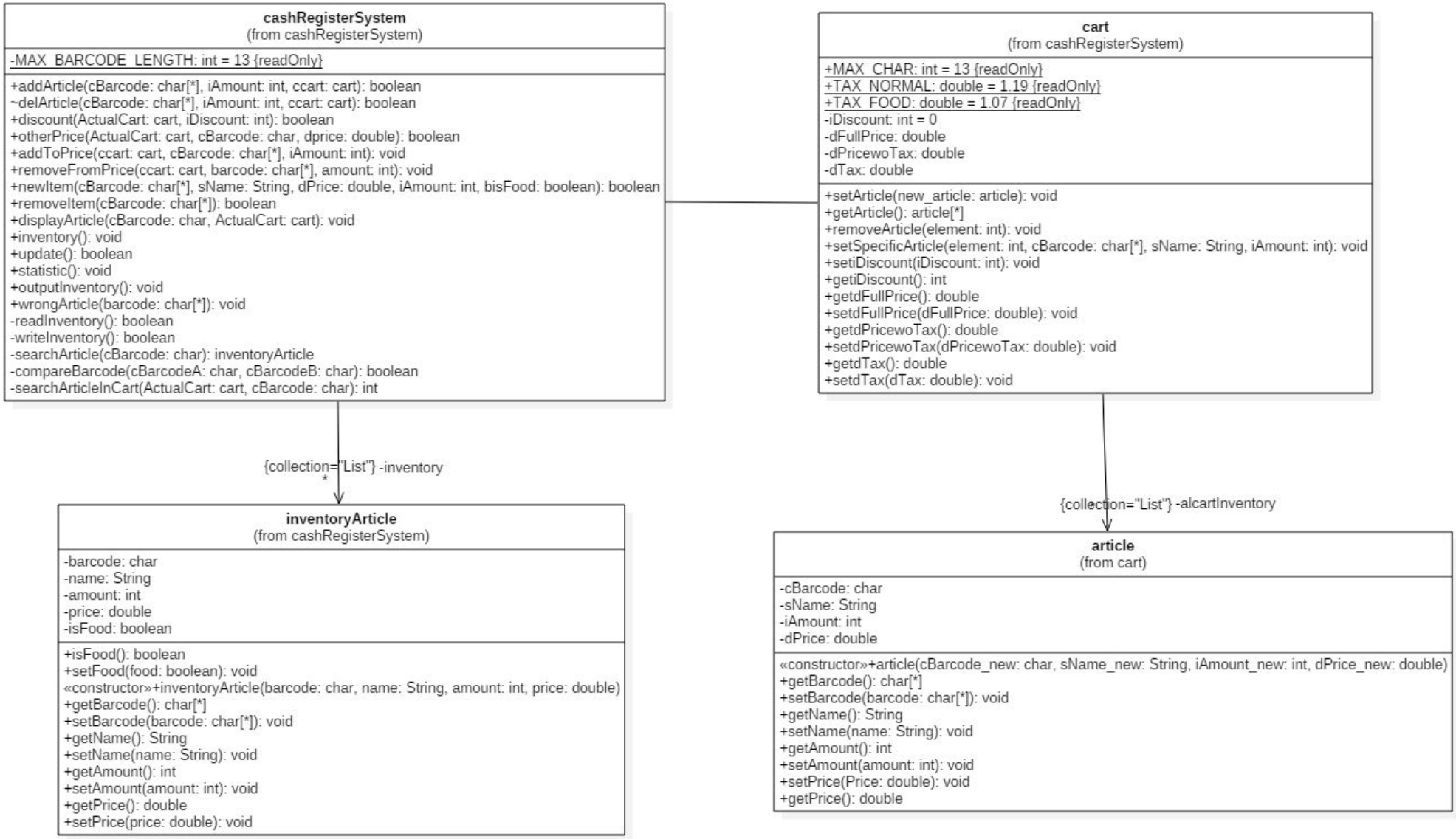
- Artikel anzeigen welche in der Kassen System registriert sind
    - void outputInventory
        - programmiert: Daniel

- Artikel vergleichen welche neu in der Kasse sind/ geändert wurden/gelöscht wurden
    - void update()
        - programmiert: Daniel

- Umsatz und Anzahl der Verkauften Artikel anzeigen lassen
  (Wird aus Differenz zur Inventory.txt berrechnet)
    - void statistic()
        - programmiert: Daniel

- Inventar aus Inventory.txt einlesen
    - boolean readInventory()
        - programmiert: Tobias

- Inventar auf inventory.txt kopieren
    - boolean writeInventory()
        - programmiert: Tobias


cashRegisterSystem
(from cashRegisterSystem)
-MAX_BARCODE_LENGTH: int = 13 {readOnly}
+addArticle(cBarcode: char[*], iAmount: int, ccart: cart): boolean
~delArticle(cBarcode: char[*], iAmount: int, ccart: cart): boolean
+discount(ActualCart: cart, iDiscount: int): boolean
+otherPrice(ActualCart: cart, cBarcode: char, dprice: double): boolean
+addToPrice(ccart: cart, cBarcode: char[*], iAmount: int): void
+removeFromPrice(ccart: cart, barcode: char[*], amount: int): void
+newItem(cBarcode: char[*], sName: String, dPrice: double, iAmount: int, bisFood: boolean): boolean
+removeItem(cBarcode: char[*]): boolean
+displayArticle(cBarcode: char, ActualCart: cart): void
+inventory(): void
+update(): boolean
+statistic(): void
+outputInventory(): void
+wrongArticle(barcode: char[*]): void
-readInventory(): boolean
-writeInventory(): boolean
-searchArticle(cBarcode: char): inventoryArticle
-compareBarcode(cBarcodeA: char, cBarcodeB: char): boolean
-searchArticleInCart(ActualCart: cart, cBarcode: char): int
cart
(from cashRegisterSystem)
+MAX_CHAR: int = 13 {readOnly}
+TAX_NORMAL: double = 1.19 {readOnly}
+TAX_FOOD: double = 1.07 {readOnly}
-iDiscount: int = 0
-dFullPrice: double
-dPricewoTax: double
-dTax: double
+setArticle(new_article: article): void
+getArticle(): article[*]
+removeArticle(element: int): void
+setSpecificArticle(element: int, cBarcode: char[*], sName: String, iAmount: int): void
+setiDiscount(iDiscount: int): void
+getiDiscount(): int
+getdFullPrice(): double
+setdFullPrice(dFullPrice: double): void
+getdPricewoTax(): double
+setdPricewoTax(dPricewoTax: double): void
+getdTax(): double
+setdTax(dTax: double): void
{collection="List"} -inventory
*
{collection="List"} -alcartInventory
*
inventoryArticle
(from cashRegisterSystem)
-barcode: char
-name: String
-amount: int
-price: double
-isFood: boolean
+isFood(): boolean
+setFood(food: boolean): void
«constructor»+inventoryArticle(barcode: char, name: String, amount: int, price: double)
+getBarcode(): char[*]
+setBarcode(barcode: char[*]): void
+getName(): String
+setName(name: String): void
+getAmount(): int
+setAmount(amount: int): void
+getPrice(): double
+setPrice(price: double): void
article
(from cart)
-cBarcode: char
-sName: String
-iAmount: int
-dPrice: double
«constructor»+article(cBarcode_new: char, sName_new: String, iAmount_new: int, dPrice_new: double)
+getBarcode(): char[*]
+setBarcode(barcode: char[*]): void
+getName(): String
+setName(name: String): void
+getAmount(): int
+setAmount(amount: int): void
+setPrice(Price: double): void
+getPrice(): double

3. Kommunikation

Die Kommunikation fand auf wöchentlicher Basis statt zu geeigneten Zeitpunkten wenn alle Gruppenmitglieder die Zeit aufbringen konnten um zu besprechen wie wir im Zeitplan stehen.
Die fand auf unseren zur Verfügung stehenden TeamSpeak Server statt.

4. Software

Als Softwareunterstützung hatten wir mehrere Systeme welche wir nutzen.
Als IDE nutzen wir IntelliJ mit Github als VCS und Google Docs/Drive als Speicher
und Office Lösung um gemeinsam an Dokumenten zu arbeiten.

5. Nutzung

Das Programm benötigt die inventory.txt welche wie folgt aufgebaut ist
“BARCODE(12 stellig)” : ”NAME” : ”PREIS” : “LEBENSMITTEL true/false”

Diese Datei wird mittels Update in die Kasse geladen und deren Einträge somit mit dem Kassensystem genutzt werden.

6. Zukünftige Aussichten

Als zukünftige Erweiterterungen haben geplant als Backend eine Datenbank anzubinden welche uns von der inventory.txt befreit und es uns einfacher macht das Lager zu verwalten.

Dazu kommt eine ansehnlich GUI welche das bedienen der Kasse einfacher gestalten soll und intuitiver macht.

Als Rechnungsausgabe hatten wir auch die Überlegung einen Bondrucker anzuschließen und diesen nutzbar zu machen.