Name: Jan-Oliver Lehnen Mat: 70467507

Dokumentation:

Damit diese Dokumentation übersichtlicher ist teile ich sie in Oberpunkte ein.

Grundstruktur:

Zunächst erstellte ich alle Klassen und ihre Abhängikeiten um die Grundstruktur des Kaufhauses abzubilden. Angefangen habe ich mit der Lagerklasse gefolgt von der Regalklasse. Bevor ich den Artikel einbauen konnte, fing ich nun an den zweiten Baum zu implementieren: Abteilungsleiter bzw. Mitarbeiter. Hier hatte ich nun zwei Optionen: Eine abstrakte Klasse Aperson, welche grundlegende Attribute an Mitarbeiter, Abteilungsleiter und später Kunden vererbt, oder drei getrennte Klassen. In diesem Fall entschied ich mich für getrennte Klassen. Grund war einerseits eine zu der Zeit bestehende unsicherheit gegenüber Vererbung, andererseits der Gedanke dass drei Klassen nicht soviele währen, und eine Implementierung über Vererbung fast schon zu aufwändig wäre. Also implementierte ich Mitarbeiter und Abteilungsleiter als zwei separate Klassen. Darauf folgenden kam die Abteilungsklasse, welche die Bedingung hatte mindestens 2 Mitarbeiter und einen Abteilungsleiter zu haben. Aufgrund dieser Bedingung wurden besagte Mitarbeiter und Abteilungsleiter fest im Konstruktor eingebaut. Eine „List<Artikel>“ wurde hinzugefügt nachdem die Artikelklasse selbst implementiert wurde. Dazu nun mehr. Eine Abteilung kann ohne Artikel existieren, ein Artikel aber in diesem Fall nicht ohne Abteilung. Also baute ich die Abteilung in welche der Artikel gehört in den Konstruktor ein und übergab im späteren Verlauf der Abteilung den Artikel im Konstruktor. Damit stand nun das Grundgerüst des Kaufhauses.

Verwaltung:

Als nächstes ging es um die Implementierung einer Verwaltungsklasse zur Umsetzung einiger Funktionen: Ziel war es sowohl die PrintOverview()-Methode (Verwaltung.cs, Z. 53), als auch später die Log-Funktionen zu nutzen. Diese Klasse enthält ausserdem sämtliche relevanten Objekte (Mitarbeiter, Einkäufe etc.) in Listen, welchen die Objekte bei erstellung zugefügt werden. Ausnahme sind Abteilungen (Diese liegen nach Anforderung in der Kaufhausklasse) und Artikel (welche in der Abteilungsklasse liegen).

Kunden und Einkäufe:

Das implementieren von Kunden und Einkäufen stellte sich zunächst als schwieriger als Gedacht dar. Nach einigem Überlegen entschied ich mich dazu jedem Kunden eine eindeutige Id (über die statische Variable lfdNr [Kunde.cs, Z. 19]) zuzuweisen. Desweiteren wird für jeden Kunden eine Liste für seine Einkäufe erstellt.

Name: Jan-Oliver Lehnen Mat: 70467507

Log-Funktion:

Das Umsetzen der Log-Funktionen lief reibungslos. Per Directory-Klasse von C# ließ sich auch der Ordner einfach realisieren.

Simulation:

Die Simulation brachte einige Probleme mit sich, namentlich:

- Wahrscheinlichkeiten der Kunden und der Kunden vom Vortag
- Tagesanfang und Tagesende mit Vergleichswerten am Ende
- Steuerung der zufälligen Einkaufe von Kunden

Zunächst: Das umändern der API (bzw. DynamicCostumer-Klasse) lief Problemlost. Für die obigen Probleme ließ sich auch je eine Lösung finden.

Die Wahrscheinlichkeiten fragte ich sehr simple mit einer Random()-Variablen zwischen 0 und 100 (%) ab. Das funktionierte einwandfrei. Nach Abgabe der 3. Studienleistung wurde allerdings ein Problem festgestellt was ich nun behoben habe. Der Tag wurde nicht erhöht, bzw. die Kunden vom Vortag waren die heutigen Kunden. Grund dafür ist die Funktionsweise der DateTime.AddDays()-Funktion. Ich dachte sie erhöht die Tage des DateTime-Objekts, jedoch wird ein neues DateTime-Objekt erstellt. Sprich bei z.B. „_datum.AddDays(1)“ passiert nichts. Nutze ich allerdings „_datum = _datum.AddDays(1)“ wird der Tag um eins erhöht. Dieser Fehler führte zu einigen weiteren Fehlern im Code, welche ich glücklicherweise beheben konnte.

Tagesanfang und Tagesende mit Vergleichswerten am Ende nutzte ich eine Referenz-Liste welche zum Tagesstart alle Artikel bekommt in den Mengen in denen sie zu dieser Zeit vorhanden sind. Am Tagesende konnten damit die entsprechenden Differenzen ausgerechnet werden.

Die Einkäufe der Kunden ließ sich ähnlich implementieren wie die Wahrscheinlichkeit einers Einkauf, jedoch nutzte ich zwei Random()-Objekte. Eines für die Abteilung, eines für den Artikel. Beim dritten Einkauf wird geprüft ob die gewählte Abteilung gleich den zwei vorherigen Abteilungen ist, um zu verhindern dass alle drei Artikel aus der selben Abteilung stammen.