# Teilaufgabe 1: Erstellen von EF Core Modelklassen

## **Eine Kassenverwaltung**

Es ist das Domain Model für eine Verwaltungssoftware für Kassen z. B. in einem Supermarkt zu erstellen. Es werden die Mitarbeiterinnen (Employee) verwaltet. Diese Klasse ist abstrakt, denn die Mitarbeiterinnen unterteilen sich in 2 Gruppen: Manager und Cashier. Bei der Zahlung an der Kassa (CashDesk) entsteht ein Payment. Das Payment besteht aus mehreren Instanzen der Klasse PaymentItem. Das PaymentItem ist ein Artikel, der gekauft wurde. Hier wird die Menge (Amount) und der Preis (Price) erfasst.

Folgendes Klassendiagramm ist gegeben:

Ein Bild, das Text, Screenshot, Diagramm, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

## **Arbeitsauftrag**

### **Erstellung der Modelklassen**

Im Projekt SPG\_Fachtheorie.Aufgabe1 befinden sich im Ordner Model leere Klassendefinitionen. Bilden Sie jede Klasse gemäß dem UML Diagramm ab, sodass EF Core diese persistieren kann. Beachten Sie folgendes:

* Wählen Sie selbst notwendige Primary keys. Vorgegebene Primary keys sind im Modell mit "PK" gekennzeichnet. Vorgegebene Keys werden nicht von der Datenbank generiert, sondern werden im Konstruktor übergeben.
* Die Klasse Employee ist abstrakt, stellen Sie dies durch eine entsprechende Klassendefinition sicher.
* Definieren Sie Stringfelder mit vernünftigen Maximallängen (z. B. 255 Zeichen für Namen, etc.).
* Durch das nullable Feature werden alle Felder als NOT NULL angelegt. Verwenden Sie daher nullable Typen für optionale Felder. Sie sind mit [0..\*] im Diagramm gekennzeichnet.
* Address ist ein value object. Stellen Sie durch Ihre Definition sicher, dass kein Mapping diese Klasse in eine eigene Datenbanktabelle durchgeführt wird.
* Legen Sie Konstruktoren mit allen Feldern an. Erstellen Sie die für EF Core notwendigen default Konstruktoren als protected.
* Das Feld Type in Employee ist als Discrimiator Feld vorgesehen. Es wird von EF Core initialisiert, diese sind natürlich nicht im Konstruktor aufzunehmen. Mappen Sie in der Konfiguration das Discriminator Feld in Employee in das Feld Type
* Implementieren Sie die Vererbung korrekt, sodass eine (1) Tabelle Employee entsteht.
* Legen Sie die erforderlichen DB Sets im Datenbankcontext an.

### **Verfassen von Tests**

Im Projekt SPG\_Fachtheorie.Aufgabe1.Test ist in Aufgabe1Test.cs der Test CreateDatabaseTest vorgegeben. Er muss erfolgreich durchlaufen und die Datenbank erzeugen. Sie können die erzeugte Datenbank in C:/Scratch/Aufgabe1\_Test/Debug/net6.0/cash.db in SQLite Studio öffnen.

Implementieren Sie folgende Tests selbst, indem Sie die minimalen Daten in die (leere) Datenbank schreiben. Leeren Sie immer vor dem Assert die nachverfolgten Objekte mittels db.ChangeTracker.Clear().

* Der Test AddCashierSuccessTest beweist, dass Sie einen Kassier (Cashier) in die Datenbank einfügen können. Prüfen Sie im Assert, ob die eingegebene RegistrationNumber auch korrekt gespeichert wurde.
* Der Test AddPaymentSuccessTest beweist, dass Sie eine Zahlung (Payment) speichern können. Legen Sie dafür eine Instanz von Payment an.
* Der Test EmployeeDiscriminatorSuccessTest beweist, dass der Typ in Employee korrekt von EF Core geschrieben wird. Gehen Sie dabei so vor:
  + Fügen Sie einen neuen Cashier oder Manager in die Datenbank ein.
  + Prüfen Sie in der Assert Bedingung, ob das Feld Type den Wert "Cashier" oder "Manager" hat.

## **Bewertung (26P, 37.1% der Gesamtpunkte)**

Jedes der folgenden Kriterien wird mit 1 Punkt bewertet.

* Die Stringfelder verwenden sinnvolle Längenbegrenzungen.
* Die Klasse Employee beinhaltet die im UML Diagramm abgebildeten Felder und korrekte public bzw. protected Konstruktoren.
* Die Klasse Employee wurde korrekt im DbContext registriert.
* Die Klasse Employee besitzt ein korrekt konfiguriertes value object Address.
* Die Klasse Employee besitzt einen korrekt konfigurierten Discriminator Type.
* Die Klasse Employee besitzt einen korrekt konfigurierten Schlüssel RegistrationNumber.
* Die Klasse Address beinhaltet die im UML Diagramm abgebildeten Felder und einen korrekten Konstruktor.
* Die Klasse Address ist ein value object, d. h. sie besitzt keine Schlüsselfelder.
* Die Klasse Manager beinhaltet die im UML Diagramm abgebildeten Felder und korrekte public bzw. protected Konstruktoren.
* Die Klasse Manager erbt korrekt von der Klasse Employee.
* Die Klasse Manager wurde korrekt im DbContext registriert.
* Die Klasse Cashier beinhaltet die im UML Diagramm abgebildeten Felder und korrekte public bzw. protected Konstruktoren.
* Die Klasse Cashier erbt korrekt von der Klasse Employee.
* Die Klasse Cashier wurde korrekt im DbContext registriert.
* Die Klasse CashDesk beinhaltet die im UML Diagramm abgebildeten Felder und korrekte public bzw. protected Konstruktoren.
* Die Klasse CashDesk wurde korrekt im DbContext registriert.
* Die Klasse Payment beinhaltet die im UML Diagramm abgebildeten Felder und korrekte public bzw. protected Konstruktoren.
* Die Klasse Payment wurde korrekt im DbContext registriert.
* Die Klasse PaymentItem beinhaltet die im UML Diagramm abgebildeten Felder und korrekte public bzw. protected Konstruktoren.
* Die Klasse PaymentItem wurde korrekt im DbContext registriert.
* Der Test AddCashierSuccessTest ist korrekt aufgebaut.
* Der Test AddCashierSuccessTest läuft erfolgreich durch.
* Der Test AddPaymentSuccessTest ist korrekt aufgebaut.
* Der Test AddPaymentSuccessTest läuft erfolgreich durch.
* Der Test EmployeeDiscriminatorSuccessTest ist korrekt aufgebaut.
* Der Test EmployeeDiscriminatorSuccessTest läuft erfolgreich durch.