PLF in Programmieren und Software Engineering

> Klasse: 7ABIF und 7ACIF
> Datum: DO, 4. Dezember 2025
> Arbeitszeit: 3 UE

Wichtiger Hinweis vor Arbeitsbeginn

Auf dem Laufwerk *Z* finden Sie die Datei *Plf7abifcif_20251204.7z*. Klicken Sie mit der rechten Maustaste auf die Datei und wählen Sie *Weitere Optionen* - *7-Zip* und *Extract here*. Gehen Sie dann in den Ordner *Plf7abifcif* und starten Sie die Datei *start_solution.cmd*. Diese Datei lädt zuerst alle Dependencies aus dem Internet und startet dann das Projekt in diesem Ordner. Sie müssen das Programm nicht abgeben, denn Sie arbeiten direkt am Netzlaufwerk.

Füllen Sie die Datei *README.md* in *Plf7abifcif/README.md* mit Ihren Daten (Klasse, Name und Accountname) aus. **Falls Sie dies nicht machen, kann Ihre Arbeit nicht zugeordnet und daher nicht bewertet werden!**

Während der Prüfung ist der Internetzugriff gesperrt. Für Visual Studio: Führen Sie niemals *Build* - *Rebuild Solution* aus, denn dadurch werden die lokalen Pakete gelöscht. Arbeiten Sie immer mit *Build* - *Build Solution* (F6).

Aufgabe: Object Relation Mapping

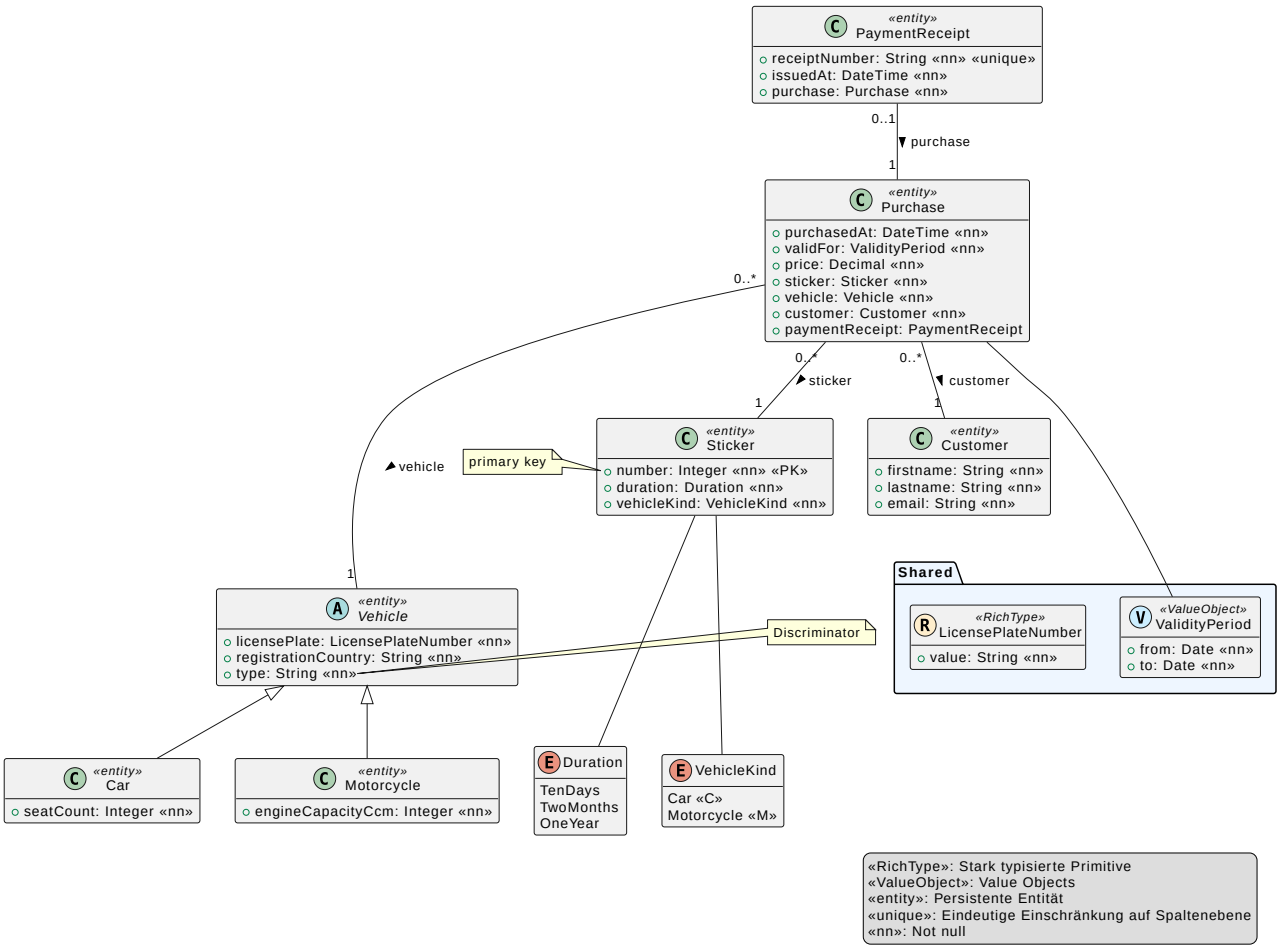
In Österreich benötigt man für die Benutzung von Autobahnen und Schnellstraßen eine sogenannte "Vignette" (Sticker). Das nachfolgende Domain Model zeigt ein Modell, wie der Verkauf dieser Vignetten in einem System abgebildet sein könnte. Die Kunden (Customer) werden mit Name und E-Mail erfasst. Das Fahrzeug (Vehicle) hat ein Kennzeichen (License Plate), das Land der Zulassung (Registration Country) wird ebenfalls gespeichert.

Da die Vignetten für PKWs und Motorräder unterschiedliche Preise haben, werden die Subtypen *Car* (PKW) und *Motorcycle* (Motorrad) definiert. Die Vignette (Sticker) hat eine eindeutige Nummer, die von einem Fremdsystem generiert wird. Es gibt Vignetten mit 10 Tagen, 2 Monaten und einem Jahr Gültigkeit. Sie wird für eine bestimmte Fahrzeugart (vehicle kind) ausgestellt.

Wird diese Vignette verkauft, wird der Verkauf in *Purchase* gespeichert. Es ist möglich, dass eine Vignette mehrfach verkauft wird. Dies ist z. B. bei einem Bruch der Scheibe der Fall.

Die Rechnung (payment receipt) wird auf Wunsch des Kunden ausgestellt und beinhaltet eine Rechnungsnummer (peceipt number).

Das Domain Model hat folgendes Aussehen:



Hinweis zu Naming Conventions

Das UML Diagramm ist plattformneutral. Implementieren Sie es in den für Ihre Sprache gebräuchlichen naming conventions wie z. B. die Großschreibung von Properties. Wählen Sie auch einen geeigneten Datentyp für Ihrer Sprache (z. B. in C# *int*, *string*, *DateOnly* für *Date* Werte, *DateTime*).

Arbeitsauftrag

Erstellung der Modelklassen

Implementieren Sie das dargestellte Diagramm Modelklassen für den OR Mapper.

Im Projekt *Plf7abifcif.Application* befinden sich leere Klassen sowie die Klasse *StickerContext* (C#), die Sie nutzen sollen.

Beachten Sie bei der Umsetzung folgende Punkte:

Allgemeine Anforderungen

- Legen Sie nötige Konstruktoren an. Ein *public* Konstruktor soll alle im Modell enthaltenen

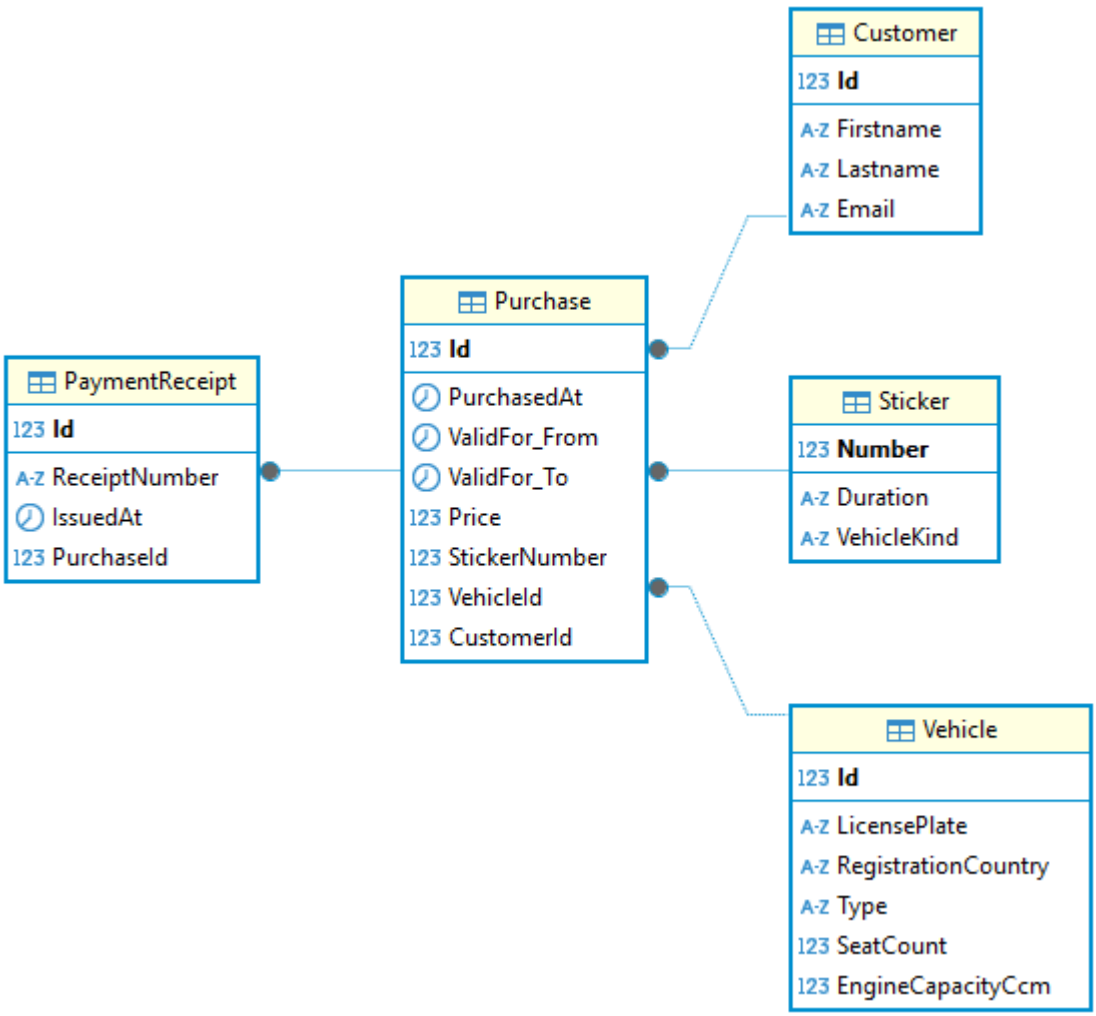
Properties initialisieren. Ergänzen Sie bei Bedarf die für den OR Mapper nötigen Konstruktoren.

- Beachten Sie Attribute Constraints wie *not null* (`<<nn>>`).
- Verwenden sie eigene primary keys mit dem Namen *id* (*Id* in C#) (autoincrement), außer im Modell ist mit *PK* explizit ein Schlüssel mit `<<PK>>` angegeben.
- Die Foreign Keys werden nach der Convention Propertyname + Name des PK generiert. Dies ist z. B. bei der Zuweisung des FKs der dependent Entities wichtig.

Fachliche Anforderungen

- Die generierten Tabellennamen sollen in Einzahl erzeugt werden (*Customer*, *Sticker*, *Vehicle*, *Purchase* und *PaymentReceipt*).
- Das Attribut *Sticker.number* soll als Primärschlüssel definiert werden. Achten Sie bei foreign keys, die auf diese Tabelle verweisen, auf die richtige naming convention.
- Das Attribut *PaymentReceipt.receiptNumber* soll ein *unique constraint* besitzen.
- Es dürfen nicht 2 Fahrzeuge mit den gleichen Werten in *licensePlate* und *registrationCounty* angelegt werden können. Stellen Sie dies durch ein unique constraint über diese 2 Spalten sicher.
- Die Enum in *Sticker.duration* soll als String mit den Stringwerten *TenDays*, *TwoMonths* und *OneYear* gespeichert werden.
- Die Enum in *Sticker.vehicleKind* soll als String mit den Stringwerten *C* für den Wert *Car* und *M* für den Wert *Motorcycle* gespeichert werden.
- Das Attribut *Vehicle.type* soll als discriminator definiert werden und folgende Werte annehmen: *C* für den Typ *Car* und *M* für den Typ *Motorcycle*.
- Das Attribut *Vehicle.licensePlate* soll als *rich type* definiert und als String gespeichert werden.
- Der rich type *LicensePlateNumber* soll prüfen, ob der Wert mindestens 5 Stellen lang ist. Falls nicht, werfen Sie mit `throw new StickerException("Invalid License Plate")` eine Exception.
- Das Attribut *Purchase.validFor* soll als *value object* (*Embedded* in Java) definiert werden. Die erzeugten Spalten sollen *ValidFor_From* und *ValidFor_To* in der Datenbank heißen (Standardverhalten in EF Core).

Das durch den OR Mapper erzeugte Datenbankschema soll so aussehen:



Der vorgegebene Test *T00_CanCreateDatabaseTest* in *GradingTests* prüft, ob mit Ihrer Implementierung überhaupt eine Datenbank erzeugt werden kann. **Läuft dieser Test nicht durch, können Sie keine positive Note erhalten.**

Der vorgegebene Test *T00_SchemaTest* in *GradingTests* verwenden, um das Gesamtmodell zu prüfen. Läuft dieser Test erfolgreich durch, sind Sie positiv.

Verfassen von Tests

In der Klasse *StickerContextTests* im Projekt *Plf7abifcif.Test* sollen Testmethoden verfasst werden, die die Richtigkeit der Konfiguration des OR Mappers beweisen sollen.

- *AddPurchaseWithPaymentReceiptTest* zeigt, dass Sie einen Verkauf (Purchase) samt Rechnung (Purchase) anlegen und persistieren können.
- *LicensePlateNumberThrowsStickerExceptionIfNumberIsNotValidTest* zeigt, dass die Validierung in *LicensePlateNumber* funktioniert. Sie müssen im Test nur eine Instanz dieser Klasse anlegen und das Fehlerverhalten prüfen.
- *LicensePlateAndRegistrationCountryThrowsExceptionWhenDuplicateTest* zeigt, dass beim Einfügen eines zweiten Fahrzeuges mit gleichen Werten in *licensePlate* und *registrationCountry* ein Fehler beim Persistieren geworfen wird.
- *DiscriminatorInVehicleHasRightValuesTest* zeigt, dass beim Speichern eines Objektes vom

Typ *Car* der Wert in *Vehicle.type* gleich *C* und beim Speichern eines Objektes vom Typ *Motorcycle* der Wert in *Vehicle.type* gleich *M* ist.

Bewertung

Um eine positive Beurteilung erreichen zu können, muss das Programm kompilieren und die implementierten Modelklassen müssen ein Erzeugen der Datenbank über den OR Mapper ermöglichen.

Aufgabe (36 Punkte in Summe)	**Ges**
Das Entity Customer wird mit den definierten Feldern Datenbank abgebildet.	1
Das Entity Sticker wird mit den definierten Feldern Datenbank abgebildet.	1
Das Entity Vehicle wird mit den definierten Feldern Datenbank abgebildet.	1
Das Entity Car wird mit den definierten Feldern Datenbank abgebildet.	1
Das Entity Motorcycle wird mit den definierten Feldern Datenbank abgebildet.	1
Das Entity Puchase wird mit den definierten Feldern Datenbank abgebildet.	1
Das Entity PaymentReceipt wird mit den definierten Feldern Datenbank abgebildet.	1
Sticker.Number ist als primary key definiert.	2
Die 1:1 Beziehung zwischen Purchase und PaymentReceipt ist korrekt konfiguriert.	2
PaymentReceipt.Number ist unique.	1
Das Tupel Vehicle.LicensePlate und Vehicle.RegistrationCountry ist unique.	2
Purchase.ValidityPeriod ist als value object definiert.	2
Vehicle.LicensePlateNumber ist als rich type definiert.	2
Vehicle.Type ist als discriminator definiert.	1
Vehicle.Type hat die korrekten Werte (C und M).	2
Sticker.Duration wird als String gespeichert.	1
Sticker.VehicleKind wird mit den korrekten Werten (C und M) gespeichert.	2
Der Test AddPurchaseWithPaymentReceiptTest hat den richtigen Aufbau.	2
Der Test AddPurchaseWithPaymentReceiptTest läuft durch.	1
Der Test LicensePlateNumberThrowsStickerExceptionIfNumberIsNotValidTest hat den richtigen Aufbau.	2
Der Test LicensePlateNumberThrowsStickerExceptionIfNumberIsNotValidTest läuft durch.	1
Der Test LicensePlateAndRegistrationCountryThrowsExceptionWhenDuplicateTest hat den richtigen Aufbau.	2
Der Test LicensePlateAndRegistrationCountryThrowsExceptionWhenDuplicateTest läuft durch.	1
Der Test DiscriminatorInVehicleHasRightValuesTest hat den richtigen Aufbau.	2
Der Test DiscriminatorInVehicleHasRightValuesTest läuft durch.	1

36 - 32 Punkte: Sehr gut, 31 - 28 Punkte: Gut, 27 - 23 Punkte: Befriedigend, 22 - 19 Punkte: Genügend, 18 - 0 Punkte: Nicht genügend.