PLF in Programmieren und Software Engineering

> Klasse: 5CAIF
> Datum: DI, 9. Dezember 2025
> Arbeitszeit: 3 UE

Wichtiger Hinweis vor Arbeitsbeginn

Auf dem Laufwerk *Z* finden Sie die Datei *Plf5caif_20251209.7z*. 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 *Plf5caif* und starten Sie die Datei *start_solution.cmd*. Diese Datei lädt zuerst alle Dependencies aus dem Internet und startet dann die sln Datei in diesem Ordner. Sie müssen das Programm nicht abgeben, denn Sie arbeiten direkt am Netzlaufwerk.

Füllen Sie die Datei *README.md* in *Plf5caif/README.md* mit Ihren Daten (Klasse, Name und Accountname) aus. Sie sehen die Datei in Visual Studio unter *Solution Items* nach dem Öffnen der Solution. **Falls Sie dies nicht machen, kann Ihre Arbeit nicht zugeordnet und daher nicht bewertet werden!**

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

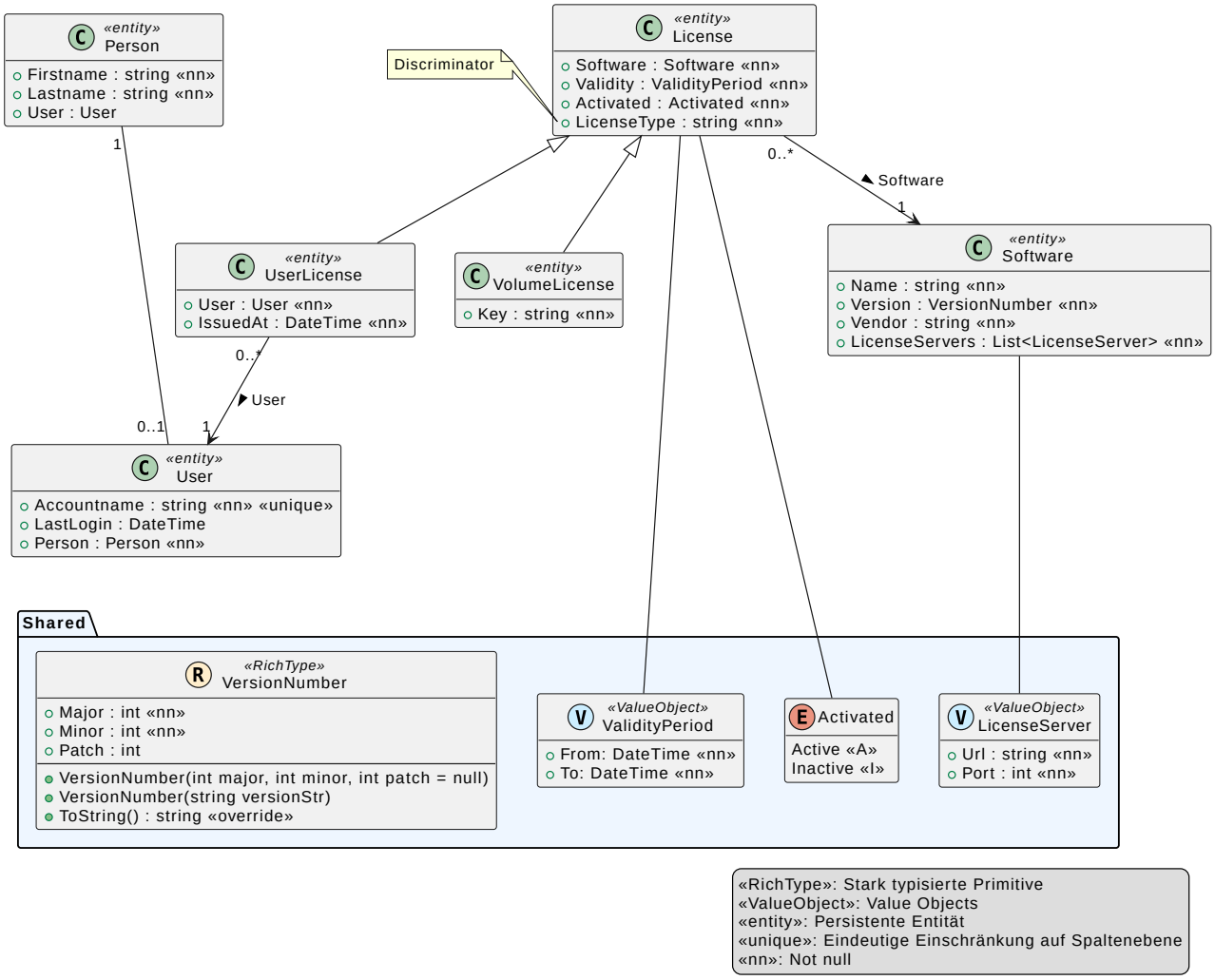
Aufgabe: Object Relation Mapping

An unserer Schule können Personen Softwarelizenzen für verschiedene Programme erwerben. Zuerst wird die Software mit Name, Version und Hersteller (Vendor) erfasst. Viele Softwareprodukte arbeiten mit einem Lizenzserver, der die Lizenzen verwaltet. In der Klasse *Software* wird daher eine Liste von Lizenzservern (*LicenseServer*) gespeichert.

Die Personen (*Person*) werden mit Vorname und Nachname aus einem Fremdsystem übernommen. Jede Person kann maximal einen Benutzeraccount (*User*) besitzen, der mit einem Accountnamen und dem letzten Login gespeichert wird.

Eine Lizenz kann entweder eine Userlizenz (*UserLicense*) oder eine Volumenlizenz (*VolumeLicense*) sein. Die Userlizenz ist an einen Benutzer (*User*) gebunden, während die Volumenlizenz nur über einen allgemeinen Lizenzschlüssel (*Key*) verfügt. Da Lizenzen eine Gültigkeitsdauer (*ValidityPeriod*) besitzen, wird diese als Value Object modelliert. Um kurzfristig Lizenzen deaktivieren zu können, wird ein Enum (*Activated*) verwendet, das den Status der Lizenz angibt.

Um dies in einem System abzubilden, wurde folgendes Domain Model entworfen.



Arbeitsauftrag

Erstellung der Modelklassen

Implementieren Sie das dargestellte Diagramm Modelklassen für den OR Mapper.

Im Projekt *Plf5caif.Application* befinden sich leere Klassen sowie die Klasse *LicenseContext*, 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* (autoincrement), außer im Modell ist

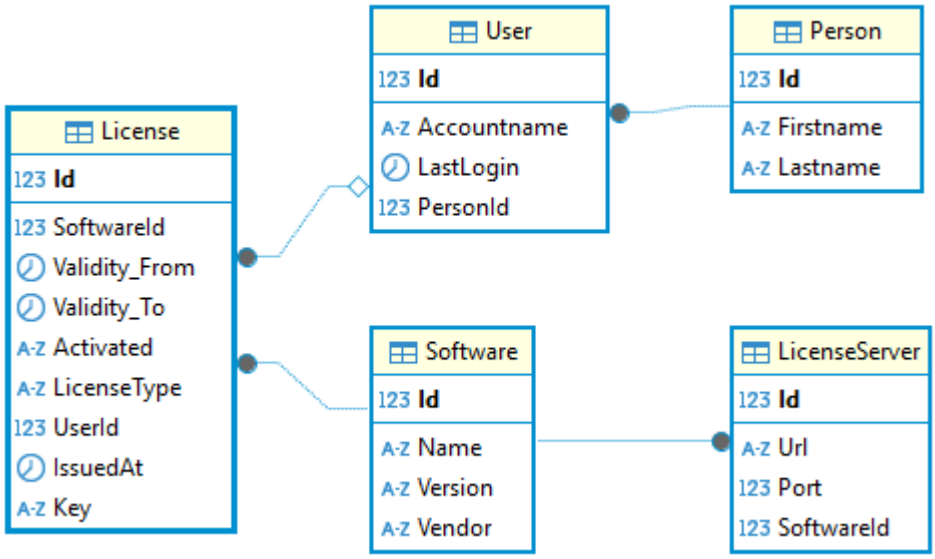
mit *PK* explizit ein Schlüssel mit `<<PK>>` angegeben.
- Die Foreign Keys werden nach der Convention Propertyname + Name des PK generiert. Dies ist z. B. bei der Zuweisung des FKs der dependent Entities wichtig.

Fachliche Anforderungen

- Die generierten Tabellennamen sollen in Einzahl erzeugt werden (*Person*, *Software*, *User*, ...).
- Eine Person kann maximal einen User besitzen. Implementieren Sie die *1:0..1* Beziehung entsprechend.
- Das Attribut *User.Accountname* soll ein *unique constraint* besitzen.
- Es dürfen nicht 2 Userlizenzen mit den gleichen Werten in *Software* und *User* angelegt werden können. Stellen Sie dies durch ein unique constraint über diese 2 Spalten (also die darunterliegenden Fremdschlüssel) sicher.
- Das Attribut *License.LicenseType* soll als discriminator definiert werden.
- Das Attribut *License.Validity* soll als *value object* definiert werden. Die erzeugten Spalten sollen *Validity_From* und *Validity_To* in der Datenbank heißen (Standardverhalten in EF Core).
- Das Attribut *Software.LicenseServers* soll als Liste von *value objects* definiert werden. Achten Sie darauf, dass der Key der erzeugten Tabelle für die value objects *Id* heißt.
- Die Enum in *License.Activated* soll als String mit den Stringwerten *A* für den Wert *Active* und *I* für den Wert *Inactive* gespeichert werden.
- Das Attribut *Software.Version* soll als *rich type* definiert und als String gespeichert werden.
- Implementieren Sie die Klasse *VersionNumber* so, dass sie eine Versionsnummer im Format *Major.Minor.Patch* (z. B. *1.0.0* oder *2.1*) repräsentiert. Implementieren Sie dazu einen Konstruktor, der die drei Ganzzahlen Major, Minor und Patch (optional) entgegennimmt, sowie einen weiteren Konstruktor, der eine String-Repräsentation der Versionsnummer entgegennimmt. Sie können die `Split(string)`-Methode des Strings verwenden, um die einzelnen Teile der Versionsnummer zu extrahieren. Mit `int.Parse(string)` können Sie einen String in eine Ganzzahl umwandeln. Achten Sie darauf, dass der String aus 2 oder 3 Teilen besteht und dass Major und Minor immer gesetzt sind. Ist dies nicht der Fall, werfen Sie mit `new LicenseException("Invalid version string.")` eine Exception. Überschreiben Sie die Methode *ToString()*, sodass diese die Versionsnummer im Format *Major.Minor.Patch* bzw. *Major.Minor* (wenn Patch nicht gesetzt wurde) zurückgibt.

Falls Sie Probleme mit der Klasse *VersionNumber* haben, können Sie diese auch als einfaches Stringfeld in *Software.Version* implementieren.

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 *LicenseContextTests* im Projekt *Plf5caif.Test* sollen Testmethoden verfasst werden, die die Richtigkeit der Konfiguration des OR Mappers beweisen sollen.

- *T01_InsertSoftwareWithUserLicenceTest* zeigt, dass Sie eine Userlizenz mit zugehöriger Software anlegen und speichern können.
- *T02_UserLicenceUserAndSoftwareIsUniqueTest* zeigt, dass beim Einfügen einer zweiten Userlizenz mit gleichen Werten in *User* und *Software* ein Fehler beim Persistieren (*DbUpdateException*) geworfen wird.
- *T03_AddSoftwareWithLicenceServersTest* zeigt, dass Sie eine Software mit mehreren Lizenzservern anlegen und speichern können. Achten Sie im Assert darauf, dass Sie ohne *Include* beim Laden der Software auch auf die Liste der Lizenzserver zugreifen können (Anzahl prüfen).
- *T04_VersionNumberFromNumericArgumentsReturnsCorrentStringTest* prüft, ob die *ToString()* Methode von *VersionNumber* mit numerischen Argumenten den korrekten String zurückgibt. Beispiele: `new VersionNumber(1,0)` liefert "1.0", `new VersionNumber(2,1,3)` liefert "2.1.3"

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 Person wird mit den definierten Feldern Datenbank abgebildet.	1
Das Entity Software wird mit den definierten Feldern Datenbank abgebildet.	1
Das Entity User wird mit den definierten Feldern Datenbank abgebildet.	1
Die Liste in Software.LicenseServer wird mit den definierten Feldern Datenbank abgebildet.	1
Das Entity License wird mit den definierten Feldern Datenbank abgebildet.	1
Das Entity UserLicense wird mit den definierten Feldern Datenbank abgebildet.	1
Das Entity VolumeLicense wird mit den definierten Feldern Datenbank abgebildet.	1
Die 1:1 Beziehung zwischen User und Person ist korrekt konfiguriert.	1
User.Accountname ist unique	1
Das Tupel UserLicence.Software und UserLicence.User ist unique.	2
Licence.Validity ist als value object definiert.	1
Software.LicenceServer ist als Liste von value objects definiert.	2
Software.VersionNumber ist als rich type mit converter definiert.	2
License.LicenseType ist als discriminator definiert.	1
License.Activated wird mit den korrekten Werten (A und I) gespeichert.	2
Mit der Klasse VersionNumber können Stringwerte aus der Datenbank korrekt eingelesen werden.	2
Die Konstruktoren, Properties und die Methoden von VersionNumber arbeiten korrekt.	3
Der Test T01_InsertSoftwareWithUserLicenceTest hat den richtigen Aufbau.	2
Der Test T01_InsertSoftwareWithUserLicenceTest läuft durch.	1
Der Test T02_UserLicenceUserAndSoftwareIsUniqueTest hat den richtigen Aufbau.	2
Der Test T02_UserLicenceUserAndSoftwareIsUniqueTest läuft durch.	1
Der Test T03_AddSoftwareWithLicenceServersTest hat den richtigen Aufbau.	2
Der Test T03_AddSoftwareWithLicenceServersTest läuft durch.	1
Der Test T04_VersionNumberFromNumericArgumentsReturnsCorrentStringTest hat den richtigen Aufbau.	2
Der Test T04_VersionNumberFromNumericArgumentsReturnsCorrentStringTest 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.