Programmentwicklung Advanced II
Modulübergreifende Projektarbeit

LINQ

Ausgangslage

Im Zuge der Aufgabe zum Thema «Generics» haben Sie ein generisches Repository-Pattern implementiert. In den einzelnen Repositories haben Sie mittels ADO.NET direkt SQL-Statements erstellt und gegen die Datenbank ausgeführt.

Auftrag

Die SQL-Statements sollen nun durch LINQ-Abfragen ersetzt werden. Verwenden Sie dazu LINQ-to-SQL. Damit die bestehenden Repositories noch erhalten bleiben, verwenden Sie das Strategy-Pattern. So können Sie die neue Variante, welche LINQ verwendet, als zusätzliche Strategie zur bestehenden Variante implementieren. Die Darstellung des Strategy-Patterns ist in Abbildung 1 ersichtlich. Weitere Informationen sowie ein Beispiel finden Sie bei Wikipedia: https://en.wikipedia.org/wiki/Strategy_pattern

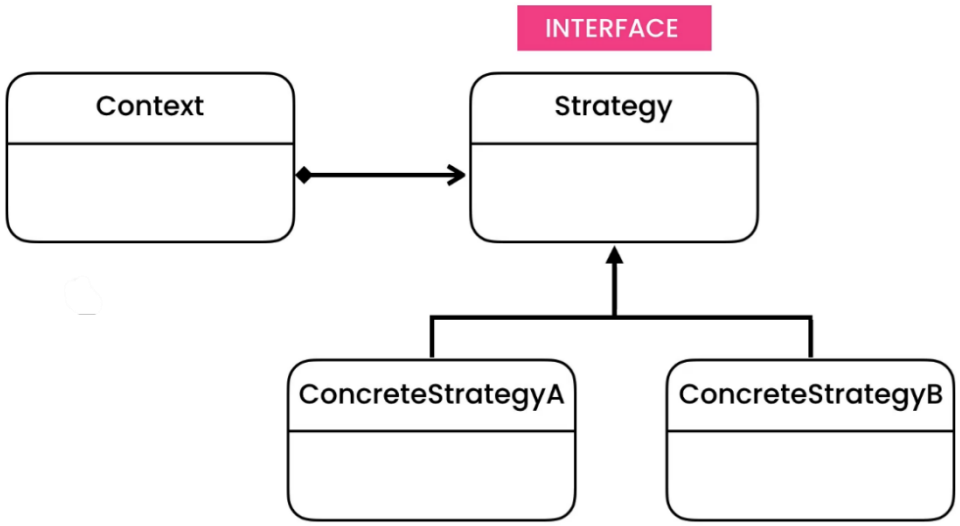

Abbildung 1: Strategy-Pattern

Zusätzlich können Sie das Factory-Pattern um sich abhängig von Regeln oder Einstellungen die korrekte Strategie erstellen zu lassen. Weitere Informationen zum Factory-Pattern finden Sie wiederum in Wikipedia (inkl. C#-Beispiel):
https://en.wikipedia.org/wiki/Factory_method_pattern

Im Modul «Datenbanken Advanced» werden Sie eine weitere Strategie implementieren, welche für den Datenbankzugriff Entity Framework verwendet.

Nebenbei: Indem Sie die verschiedenen Implementierungsvarianten mittels Strategien umsetzen, könnten Sie anschliessend z.B. Benchmarking durchführen und so feststellen, welche Variante besser performed.

Nun aber zur konkreten Umsetzung mittels LINQ:

In einem ersten Schritt müssen Sie die benötigen Tabellen mappen. Erstellen Sie dazu für jede Tabelle ein DTO gemäss nachfolgendem Muster (siehe Abbildung 2 sowie Listing 1).

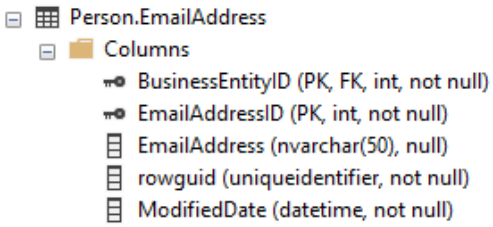

Abbildung 2: Tabelle Person.EmailAddress

```
[Table(Name = "Person.EmailAddress")] // needs: using System.Data.Linq.Mapping;
public class Email {
    [Column(IsPrimaryKey = true, Name = "BusinessEntityID")]
    public int PersonID { get; set; }

    [Column(IsPrimaryKey = true, Name = "EmailAddressID", IsDbGenerated = true)]
    public int ID { get; set; }

    [Column(Name = "EmailAddress")]
    public string Address { get; set; }

    [Column(Name = "RowGuid", IsDbGenerated = true)]
    public Guid RowGuid { get; set; }

    [Column(Name = "ModifiedDate", IsDbGenerated = true)]
    public DateTime ModifiedAt { get; set; }
}
```

Listing 1: Mapping für Tabelle EmailAddress

Die Klasse `DataContext`[1] stellt Ihnen die die gemappten Entities als Source (`IQueryable<TEntity>`) zur Verfügung. Sie ermöglicht Ihnen in der Folge die Daten via LINQ anstatt mit hardcodierten SQL-Statements abzufragen. Dies impliziert, dass Sie die `whereCondition` in den Methoden `GetAll()` sowie `Count()` durch Predicates[2] ersetzen, die Sie anschliessend für die LINQ-Abfrage verwenden können. Ebenfalls soll die Methode `GetAll()` (inkl. Überladungen) keine `List<M>` sondern neu ein `IQueryable<M>` zurückliefern, damit «Deferred Evaluation» möglich ist.

Informieren Sie sich, wie Sie die Klasse `DataContext` verwenden müssen (Internetrecherche bzw. Sample «Linq-To-Sql») und setzen Sie anschliessend die CRUD[3]-Operationen in Ihren Repositories gemäss Anforderungen mittels LINQ bzw. Linq-To-Sql um.

Lieferergebnis

Als Abgabe wird der komplette Sourcecode als ZIP-Datei erwartet. Zusätzlich soll eine Kurzdokumentation erstellt werden, aus der die Handhabung Ihrer Anwendung hervorgeht.

Bewertung

Gemäss Initialdokument

Termin

Gemäss Moodle-Abgabe

[1] Klasse `DataContext`: https://docs.microsoft.com/en-us/dotnet/api/system.data.linq.datacontext
[2] Klasse `Predicate`: https://docs.microsoft.com/en-us/dotnet/api/system.predicate-1
[3] CRUD ist ein Akronym für die vier fundamentalen Operationen des Datenmanagements: Create (Insert), Read (Select), Update, Delete. Siehe https://de.wikipedia.org/wiki/CRUD