LINQ – Teil 4

Ingo Köster

Diplom-Informatiker (FH)

join-Klausel

- Mittels `join` Elemente aus unterschiedlichen Quellen, die keine direkte Beziehung im Objektmodell haben, miteinander verbinden

- Anforderung
  - Die Elemente in jeder Quelle nutzen Werte gemeinsam
  - Diese können auf Gleichheit verglichen werden

join-Beispiel

```
class Person {
 public string FirstName { get; set; }
 public string LastName { get; set; }
}

class Pet {
 public string Name { get; set; }
 public Person Owner { get; set; }
}
```

join-Klausel

```
var query = from person in people    // Eine Person
            join pet in pets         // Ein Tier
                // Sind Person und Besitzer gleich?
                on person equals pet.Owner
                // Neuer anonymer Typ
                // mit Vorname und Tiername
                select new
                {
                  OwnerName = person.FirstName,
                  PetName = pet.Name
                };
```

join-Klausel – Alternative Syntax

```
var query = people.Join(pets,          // Die zu joinende Liste
             person => person,         // Eine Person
             pet => pet.Owner,         // Ein Besitzer
             (pers1, pet1) =>          // Für gleiche Objekte
            // anonymer Typ mit Vornamen und Tiername
            new
                {
                    OwnerName = pers1.FirstName,
                    PetName = pet1.Name
                });
```

Bereichsvariable

- Mittels `let` das Ergebnis eines Unterausdrucks speichern, um es in nachfolgenden Klauseln zu verwenden
  - Sogenannte Bereichsvariable

- Nachdem eine Bereichsvariable mit einem Wert initialisiert wurde, kann sie nicht mehr zum Speichern eines anderen Werts verwendet werden

Bereichsvariable

```
string[] sprichwörter = {
         "A penny saved is a penny earned.",
         "The early bird catches the worm.",
         "The pen is mightier than the sword." };

// Unterausdruck: Wörter eines Satzes
var query = from satz in sprichwörter
            let wörter = satz.Split(' ')
              // Unterausdruck verwenden
            from einWort in wörter
            where einWort[0] == 'T' // 1. Wort mit T?
            select satz;
```

Bereichsvariable

```
char[] vokale = { 'a', 'e', 'i', 'o', 'u' };

var vowelsQuery =
  from satz in sprichwörter
  let wörter = satz.Split(' ')
  from einWort in wörter
  let wortKlein = einWort.ToLower()
  // Ein Wort das mit einem Vokal anfängt
  where vokale.Contains(wortKlein[0])
  select einWort;
```

Klasse Enumerable – Methode Range

- Methode Range
  - Generiert eine Sequenz von ganzen Zahlen in einem angegebenen Bereich
- Beispiele:
- Zahlen von 1 bis 99

```
int[] zahlen = Enumerable.Range(1, 100).ToArray();
```

  - 1. Wert -> Startwert (inklusive)
  - 2. Wert -> Anzahl

- Quadrate der ersten 10 Zahlen

```
var quadrate = Enumerable.Range(1, 10).Select(x => x * x);
```

Klasse Enumerable – Methode Range

› Array (Größe 100) mit Zufallszahlen zwischen 1 und 1000 füllen

```
Random random = new Random();

int[] zahlen = Enumerable.Range(0, 100)
               .Select(_ => random.Next(1, 1_001))
               .ToArray();
```

Klasse Enumerable – Methode Repeat

- Erzeugt eine Sequenz mit einer angegebenen Anzahl von Elementen in welcher jedes Element denselben Wert enthält
- Die `Repeat`-Methode hat 2 Parameter
  - Der erste Parameter gibt den zu wiederholenden Wert an
  - Der zweite Parameter gibt an, wie oft wiederholt werden soll
- Der Rückgabetyp der Methode ist `IEnumerable<T>`
  - T ist der Datentyp des zu wiederholenden Wertes

```
IEnumerable<string> someStrings =
Enumerable.Repeat(@"¯\_(ツ)_/¯", 10);
```