C# Workshop LINQ

Einleitung

Herzlich willkommen zum C# Workshop mit Schwerpunkt LINQ! Du fragst dich vielleicht: "Was genau ist LINQ und wozu brauche ich das überhaupt?" Keine Sorge – genau diese Fragen werden wir heute gemeinsam beantworten.

LINQ (Language Integrated Query) ist ein mächtiges Werkzeug in der Programmiersprache C#, das entwickelt wurde, um Abfragen auf Daten einfacher und intuitiver zu gestalten. Anstatt komplexe Schleifen und Bedingungen zu schreiben, kannst du mit LINQ deine Daten mit nur wenigen verständlichen Befehlen durchsuchen, filtern und sortieren.

In diesem Workshop wirst du Schritt für Schritt lernen, wie du LINQ effektiv einsetzt, um deine tägliche Arbeit als Programmierer deutlich zu vereinfachen. Egal, ob du Datenbanken abfragst, Listen durchsuchst oder komplexe Strukturen vereinfachst – mit LINQ wird all das viel leichter.

Freue dich auf spannende Inhalte und praktische Beispiele, die dir einen optimalen Einstieg ermöglichen und deine Programmierfähigkeiten bereichern!

Idee hinter LINQ

LINQ wurde entwickelt, um komplexe Abfragen in Programmiersprachen einfacher, klarer und schneller zu gestalten. Die Idee dahinter ist, dass du mit nur wenigen Zeilen Code komplexe Datenoperationen durchführen kannst, ohne komplizierte Schleifen und manuelle Vergleiche schreiben zu müssen. LINQ bietet dir eine einheitliche Syntax, mit der du verschiedene Datenquellen wie Listen, Arrays oder sogar Datenbanken abfragen kannst.

Funktionsweise

Die Funktionsweise von LINQ ist leicht zu verstehen. Im Wesentlichen besteht eine LINQ-Abfrage aus drei Schritten:

1. Datenquelle angeben: Du sagst, welche Daten abgefragt werden sollen (zum Beispiel eine Liste oder ein Array).
2. Bedingungen definieren: Du legst fest, welche Bedingungen erfüllt sein müssen (z.B. nur Elemente mit bestimmten Werten).
3. Ausführung der Abfrage: LINQ führt die Abfrage automatisch aus und liefert die Ergebnisse in gewünschter Form zurück.

Matthias Bohn, Tim Deimel, Sebastian Schmalz, Marc Jaworski

Beispiele

Hier ein einfaches Beispiel: Angenommen, du möchtest aus einer Liste von Zahlen nur die geraden Zahlen auswählen:

```
LIST<INT> ZAHLEN = NEW LIST<INT> { 1, 2, 3, 4, 5, 6 };
VAR GERADEZAHLEN = FROM ZAHL IN ZAHLEN
                   WHERE ZAHL % 2 == 0
                   SELECT ZAHL;
```

Oder eine unsortierte Liste von Strings sortieren:

```
List<string> namen = new List<string> { "Anna", "Ben", "Clara", "David" };
var sortierteNamen = from name in namen
                     orderby name
                     select name;
```

Oder Eine Liste Filtern und Gruppieren

```
List<string> tiere = new List<string> { "Hund", "Katze", "Hamster", "Hund",
"Katze" };
var gruppierteTiere = from tier in tiere
                      group tier by tier into gruppe
                      select new { Tier = gruppe.Key, Anzahl = gruppe.Count()
};

foreach (var gruppe in gruppierteTiere)
{
    Console.WriteLine($"Tier: {gruppe.Tier}, Anzahl: {gruppe.Anzahl}");
}
```

Methoden Schreibweise

```
List<int> list = new List<int> {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
List<int> neueListe = list
    .Select(item => item * 2)
    .Where(item => item >= 5)
    . ToList();
```

Operationen ausführen

```
int summe = list.Select(item => item)
        .Where (item => item % 2 == 0)
        .Sum();

int length = personArray.Select(person => person)
        .Where(person => person.getAge() >= 18)
        .Count() ;
```

Grobe Übersicht der Operationen:

LINQ-Operation	**Beschreibung**
Count()	Zählt Elemente
Sum()	Summiert Werte
Average()	Durchschnitt berechnen
Min()	Gibt Minimum zurück
Max()	Gibt Maximum zurück
First()	Erster Wert
FirstOrDefault()	Erster oder Standardwert
Last()	Letzter Wert
LastOrDefault()	Letzter oder Standardwert
Where()	Filtert Elemente
Select()	Projektion von Elementen
OrderBy()	Aufsteigend sortieren
OrderByDescending()	Absteigend sortieren

GroupBy()	Gruppiert Elemente
Distinct()	Entfernt Duplikate
Any()	Prüft auf Vorhandensein
All()	Prüft alle Bedingungen
Take()	Nimmt erste N Elemente
Skip()	Überspringt erste N Elemente
Join()	Verbindet zwei Sequenzen
Concat()	Fügt zwei Sequenzen zusammen
SelectMany()	Flacht verschachtelte Sequenzen ab
Aggregate()	Aggregiert Werte
ToList()	Konvertiert zu Liste
ToArray()	Konvertiert zu Array
ElementAt()	Gibt Element an Index zurück
ElementAtOrDefault()	Gibt Element oder Standardwert
Intersect()	Gibt Schnittmengen zurück
Union()	Vereinigung von Sequenzen

Einsatzgebiet

LINQ wird vielseitig eingesetzt:

- Listen und Arrays: Schnelles Filtern, Sortieren oder Gruppieren von Daten
- Datenbanken: Einfache und schnelle Abfragen ohne kompliziertes SQL
- XML und JSON: Einfacher Zugriff auf Datenstrukturen zur schnellen Verarbeitung
- Allgemeine Programmierung: Vereinfachung komplexer Logiken durch verständliche und wartbare Abfragen

Fazit

LINQ ist ein unverzichtbares Werkzeug für jeden C#-Entwickler, das dir hilft, komplexe Abfragen klar, einfach und effizient umzusetzen. Mit wenig Aufwand kannst du große und komplexe Datenmengen verwalten und bearbeiten. Nutze die Möglichkeiten von LINQ, um deinen Code sauberer, übersichtlicher und leistungsfähiger zu gestalten – deine Produktivität wird davon profitieren!

Matthias Bohn, Tim Deimel, Sebastian Schmalz, Marc Jaworski