





# Language-Integrated Query (LINQ)

# What is LINQ?

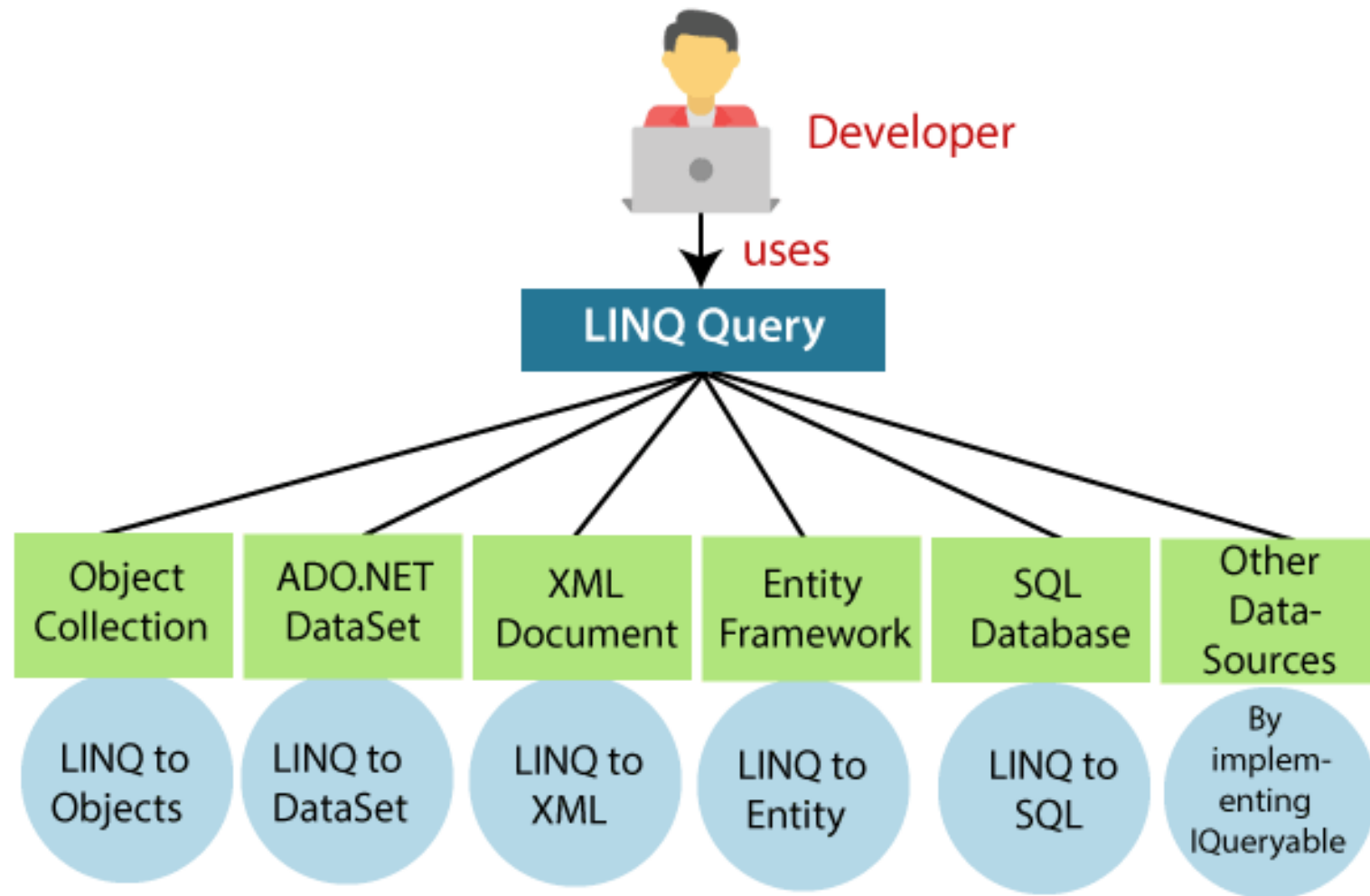
LINQ stands for Language Integrated Query. It's a feature in .NET programming languages like C# and Visual Basic that allows developers to query data from different types of data sources using a uniform syntax.



LINQ provides a set of standard query operators that enable developers to perform queries on collections of objects, databases, XML documents, and other data sources directly from within their code.



LINQ provides a set of standard query operators that allow you to perform filtering, sorting, grouping, joining, and aggregating operations on data. These operators are available through LINQ query syntax (using query expressions) or method syntax (using extension methods).



# Example of a LINQ query

```
using System;
using System.Linq;
class Program
{
    static void Main()
    {
        string[] words = {"hello", "wonderful", "LINQ", "beautiful", "world"};
        //Get only short words
        var shortWords = from word in words where word.Length <= 5 select word;
        //Print each word out
        foreach (var word in shortWords)
        {
            Console.WriteLine(word);
        }
        Console.ReadLine();
    }
}
```

# Syntax of LINQ

There are two syntaxes of LINQ.

## Lamda (Method) Syntax

```
var longWords = words.Where( w  $\Rightarrow$  w.length > 10);
```

## Query Syntax

```
var longwords = from w in words where w.length > 10;
```

# LINQ Query Syntax

- It is similar to SQL and is often more readable for those familiar with SQL. It starts with a from clause followed by a range variable and includes standard query operations like where, select, group, join, etc.

```
from <range variable> in <IEnumerable<T> or IQueryable<T> Collection>  
<Standard Query Operators> <lambda expression> <select or groupBy operator>  
<result formation>
```

Example:

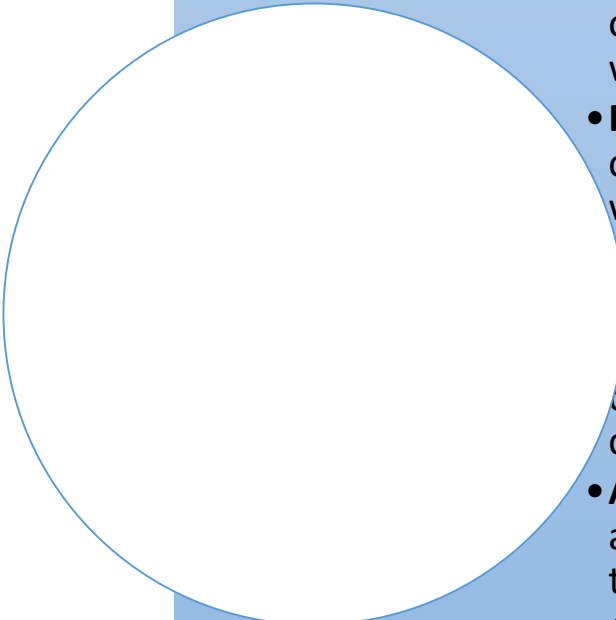
```
var query = from c in customers where c.City == "London" select c.Name;
```

# Method Syntax(Fluent Syntax)

- It uses extension methods and lambda expressions. It can be more concise and is preferred when writing complex queries because it can be easier to read and compose.

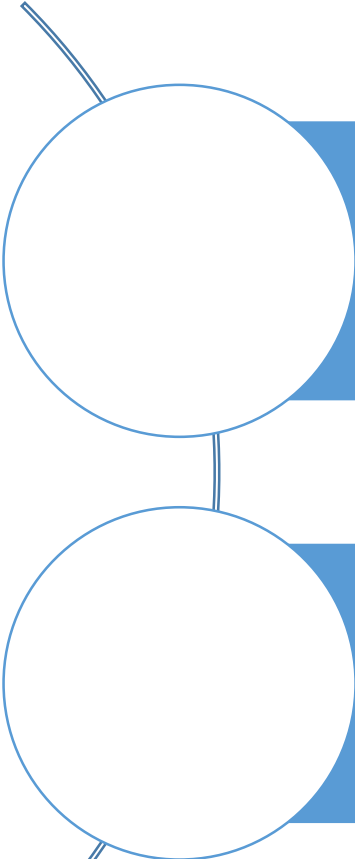
```
var query = customers.Where(c => c.City ==  
"London").Select(c => c.Name);
```

# When to Use LINQ?

- 
- **Querying Collections:** LINQ is ideal for querying collections like arrays, lists, or any other types that implement IEnumerable. It simplifies the process of filtering, sorting, and grouping data.
  - **Database Operations:** With LINQ to SQL or Entity Framework, you can perform database operations. LINQ queries are automatically translated into SQL queries, making it easier to interact with databases without writing raw SQL.
  - **Readability and Maintainability:** LINQ queries often result in more readable and maintainable code compared to traditional loops and conditional statements. The syntax is declarative, specifying what you want to do rather than how to do it.
  - **Working with XML:** LINQ to XML provides a simple and efficient way to handle XML documents. It allows you to query, modify, and navigate XML data in a more readable and concise way.
  - **Joining Data Sources:** If you need to join data from different sources (like different collections, databases, or XML files), LINQ can be a powerful tool. It simplifies the syntax for joining and correlating data from multiple sources.
  - **Aggregations and Calculations:** When you need to perform calculations or aggregations (like sum, average, min, max) on a collection of items, LINQ offers straightforward methods to accomplish these tasks.
  - **Converting Data Types:** LINQ provides easy-to-use methods for converting one type of data into another, such as converting an array to a list or vice versa.



# LINQ Lambda Expression Syntax



In LINQ, Lambda Expression is a function without a name. It makes the syntax short and clear. Though lambda expression is not readable as the LINQ query, it is equally important as the LINQ query, and it converts into lambda internally. The scope of the lambda expression is limited when we use it as an expression. Lambda Expression cannot be used afterward

**The syntax of defining the lambda expression in LINQ is as:**  
(Input Parameter) => Method Expression

# LINQ Element Operators

---

First

---

FirstOrDefault

---

Last

---

LastOrDefault

---

ElementAt

---

ElementAtOrDefault

---

Single

---

SingleOrDefault

---

DefaultIfEmpty



Thank you