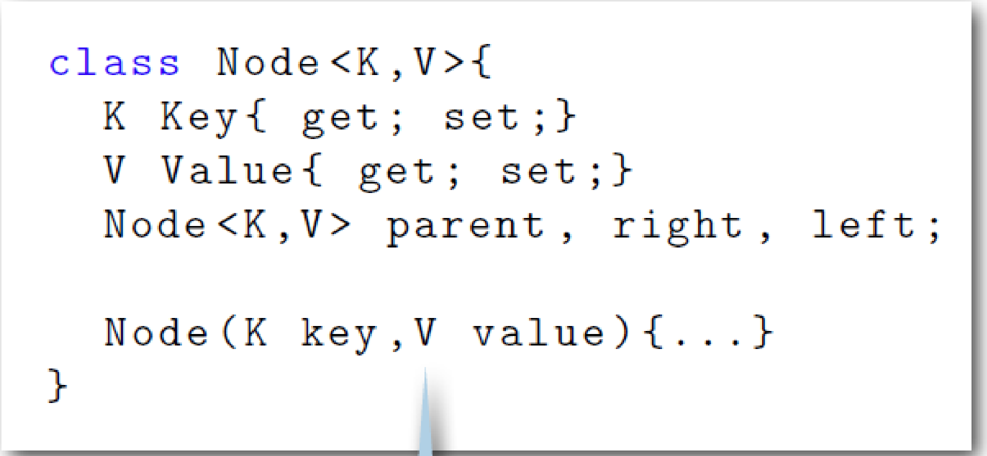
RAWDATA
Section 2

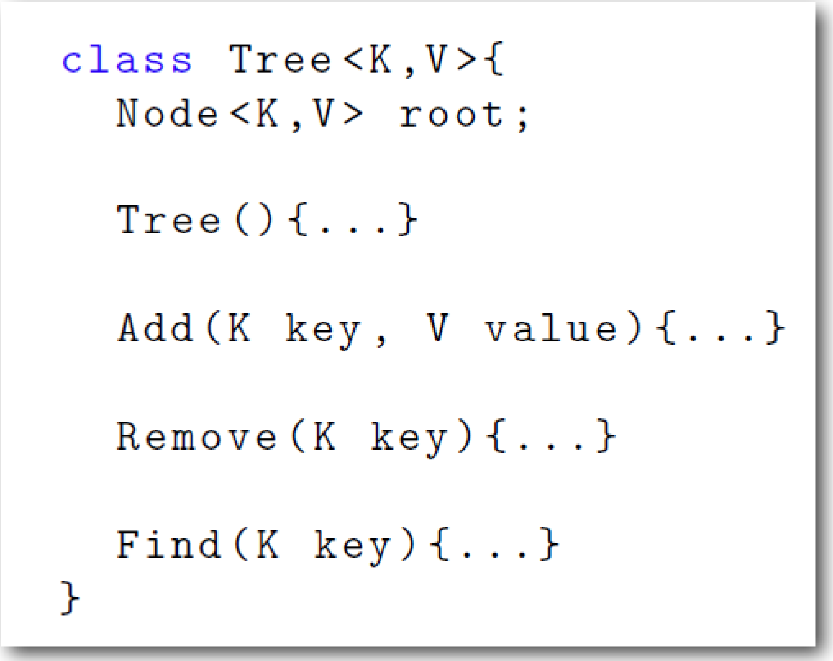
Troels Andreasen & Henrik Bulskov

Generic Classes



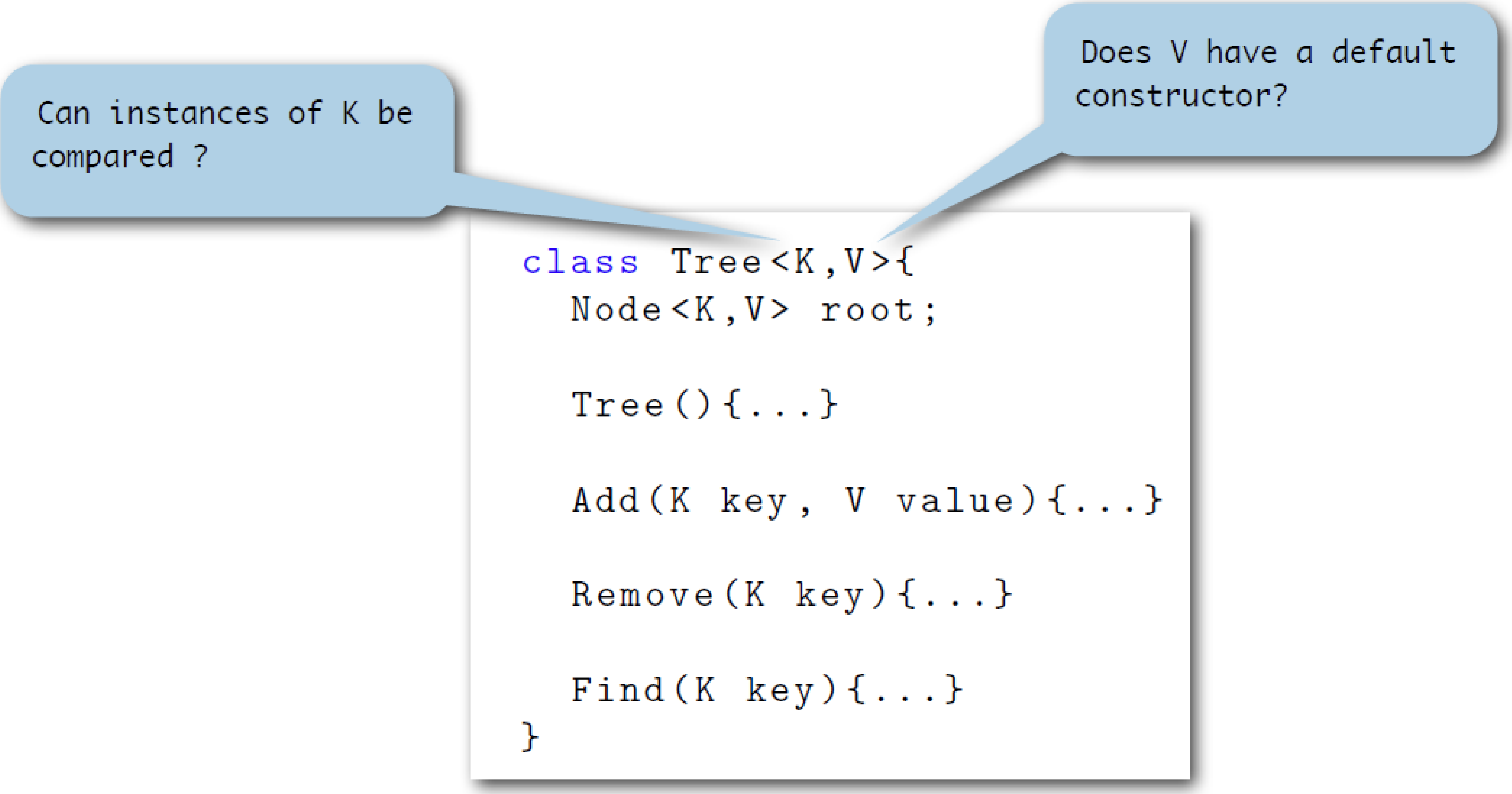
```
class Node<K,V>{
  K Key{ get; set;}
  V Value{ get; set;}
  Node<K,V> parent, right, left;

  Node(K key,V value){...}
}
```

Any type but not void



```
class Tree<K,V>{
  Node<K,V> root;

  Tree(){...}

  Add(K key, V value){...}

  Remove(K key){...}

  Find(K key){...}
}
```

Constraints on Type Parameters



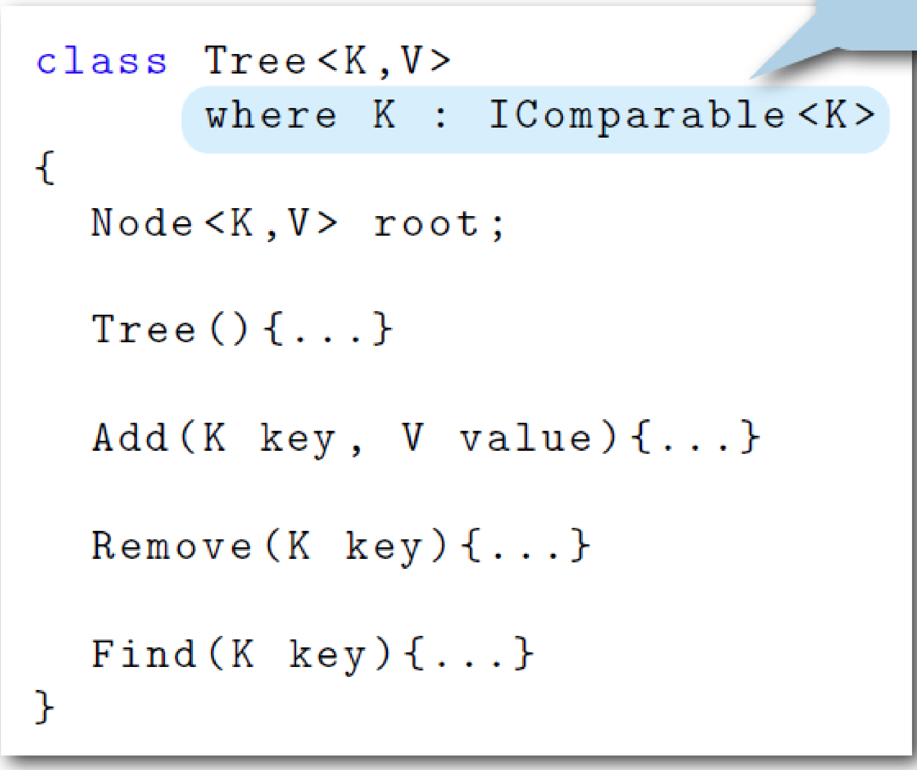
Can instances of K be compared ?

Does V have a default constructor?

```
class Tree<K,V>{
  Node<K,V> root;

  Tree(){...}

  Add(K key, V value){...}

  Remove(K key){...}

  Find(K key){...}
}
```

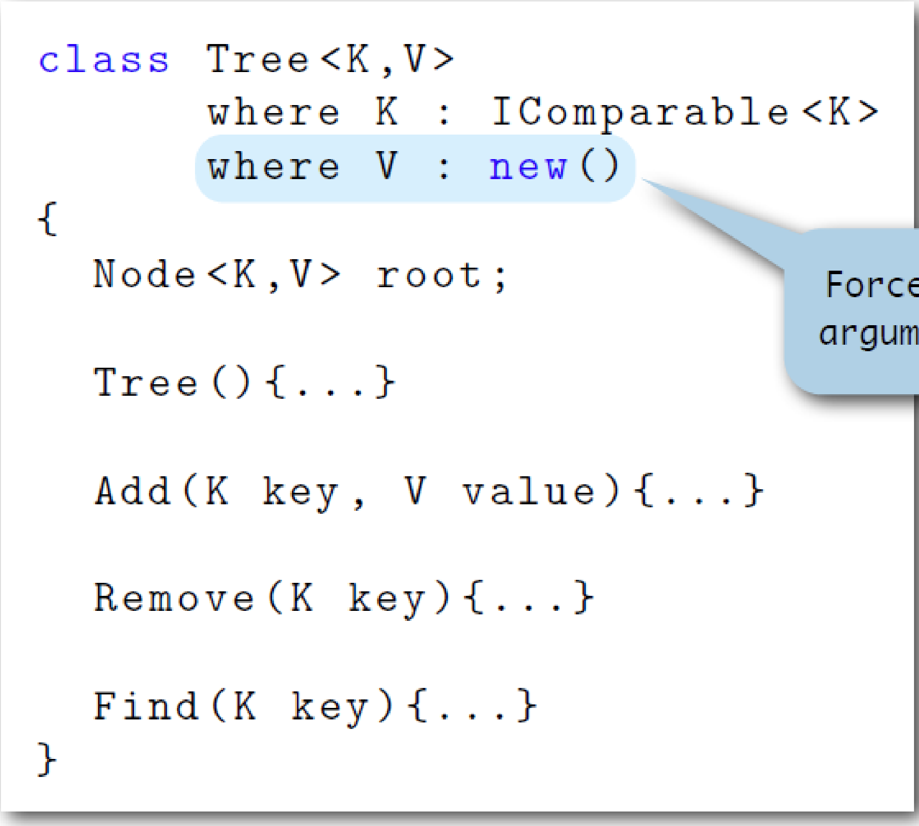
Constraints on Type Parameters

Forces K to implement the IComparable<K> interface



```
class Tree<K,V>
      where K : IComparable<K>
{
  Node<K,V> root;

  Tree(){...}

  Add(K key, V value){...}

  Remove(K key){...}

  Find(K key){...}
}
```

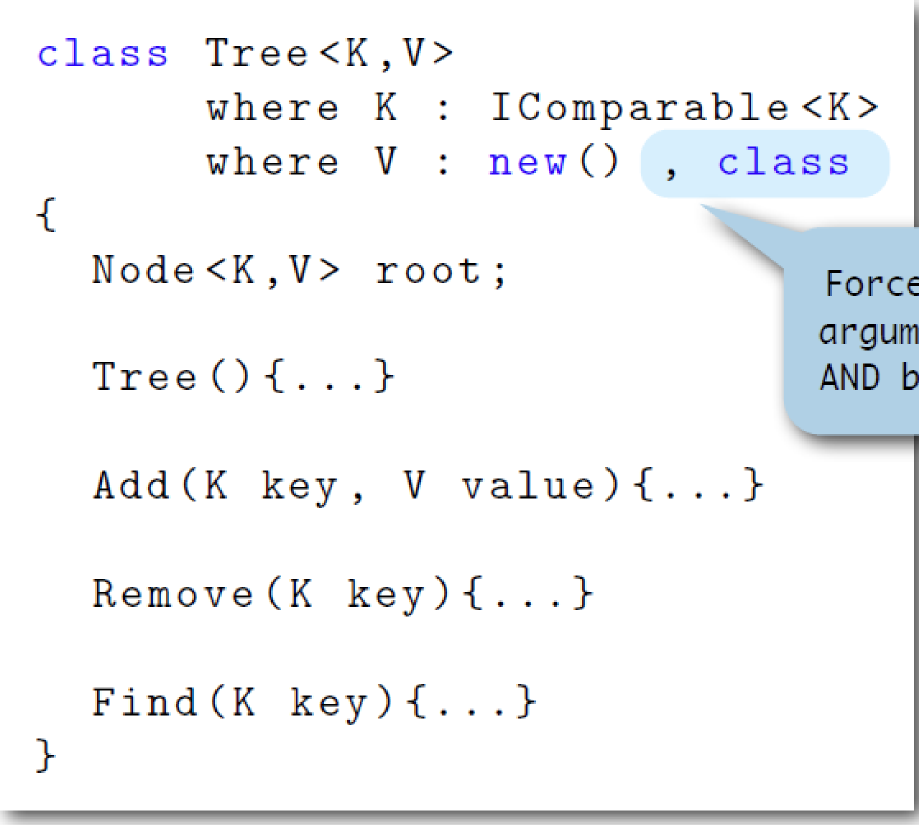
Constraints on Type Parameters



```
class Tree<K,V>
      where K : IComparable<K>
      where V : new()
{
  Node<K,V> root;

  Tree(){...}

  Add(K key, V value){...}

  Remove(K key){...}

  Find(K key){...}
}
```

Forces V to have an argumentless constructor

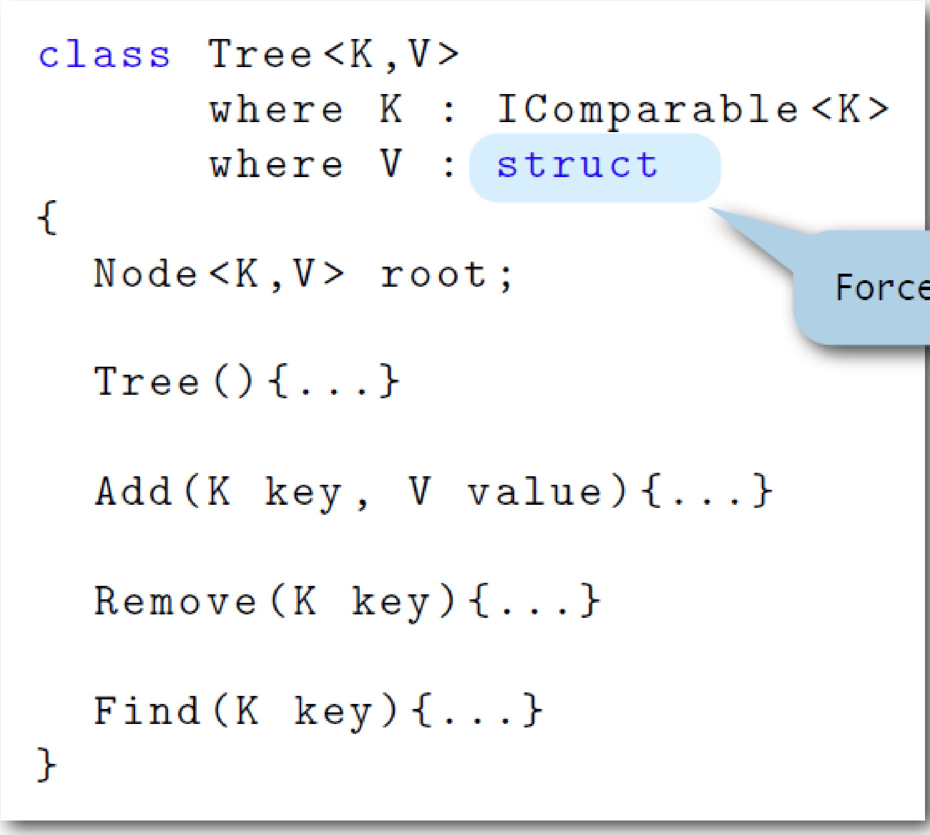
Constraints on Type Parameters



```
class Tree<K,V>
      where K : IComparable<K>
      where V : new() , class
{
  Node<K,V> root;

  Tree(){...}

  Add(K key, V value){...}

  Remove(K key){...}

  Find(K key){...}
}
```

Forces V to have an argumentless constructor AND be a reference type

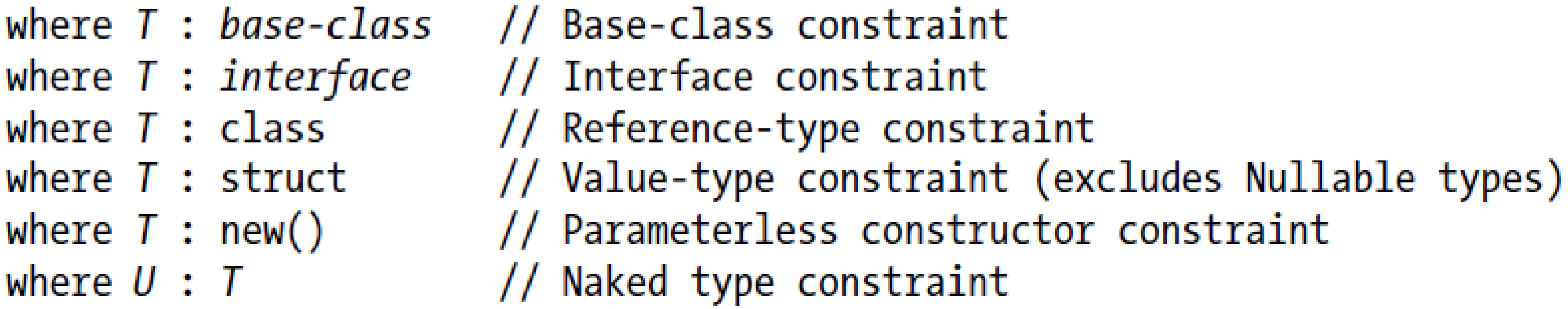
Constraints on Type Parameters



```
class Tree<K,V>
      where K : IComparable<K>
      where V : struct
{
  Node<K,V> root;

  Tree(){...}

  Add(K key, V value){...}

  Remove(K key){...}

  Find(K key){...}
}
```

Forces V be a value type

Generic Constraints

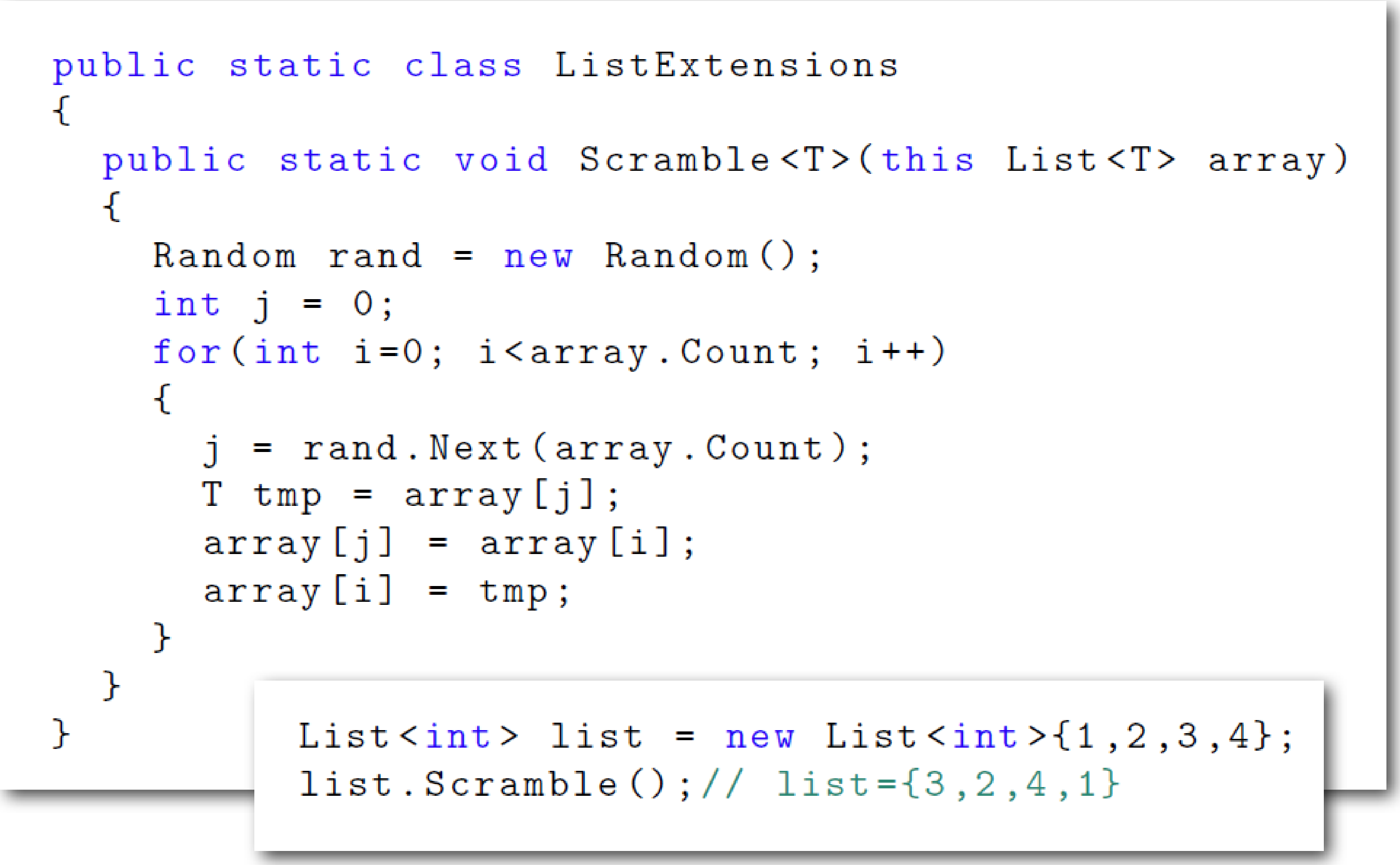


```
where T : base-class    // Base-class constraint
where T : interface     // Interface constraint
where T : class         // Reference-type constraint
where T : struct        // Value-type constraint (excludes Nullable types)
where T : new()         // Parameterless constructor constraint
where U : T             // Naked type constraint
```

Use of Type Parameters

- Use it as type of fields, variables, properties, method parameters and return types
- Use it to create arrays e.g. `new T[10]`
- Call `default(T)` to get the appropriate default value
- Create a new instance with `new T()` if the new() constraint is specified
- Use methods of the interfaces or base classes in the constraint specification.
- CANNOT call static methods

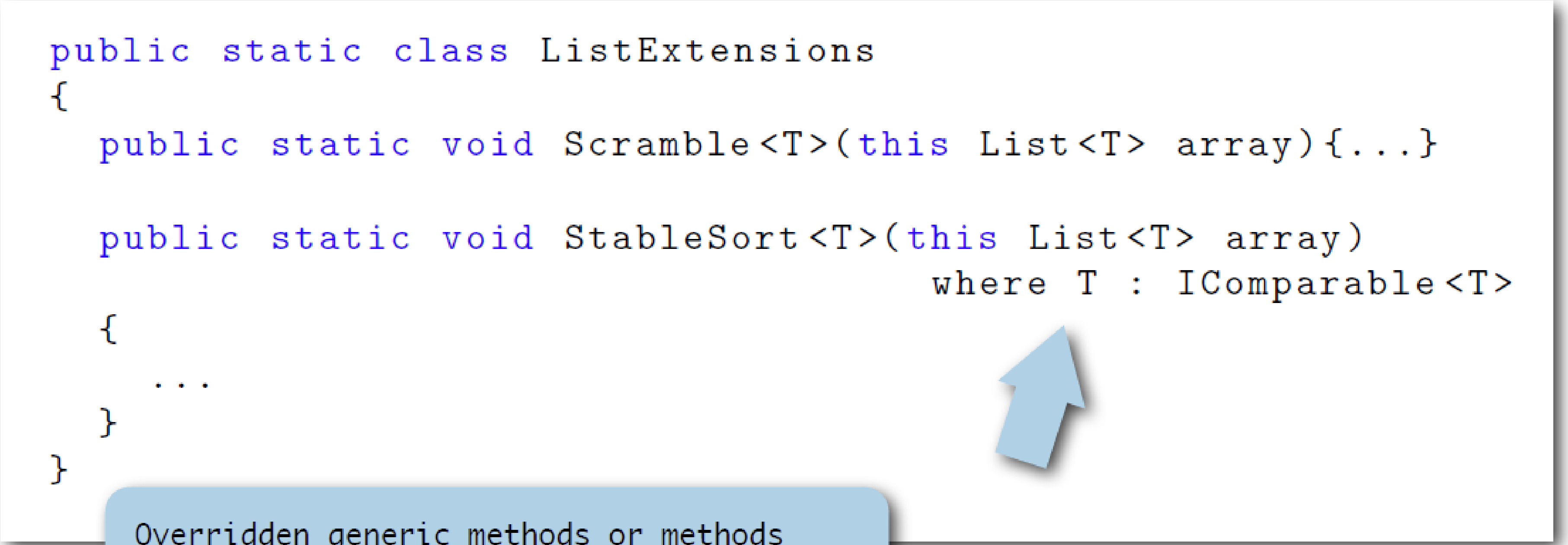
Generic Methods



```
public static class ListExtensions
{
  public static void Scramble<T>(this List<T> array)
  {
    Random rand = new Random();
    int j = 0;
    for(int i=0; i<array.Count; i++)
    {
      j = rand.Next(array.Count);
      T tmp = array[j];
      array[j] = array[i];
      array[i] = tmp;
    }
  }
}
```

```
List<int> list = new List<int>{1,2,3,4};
list.Scramble();// list={3,2,4,1}
```

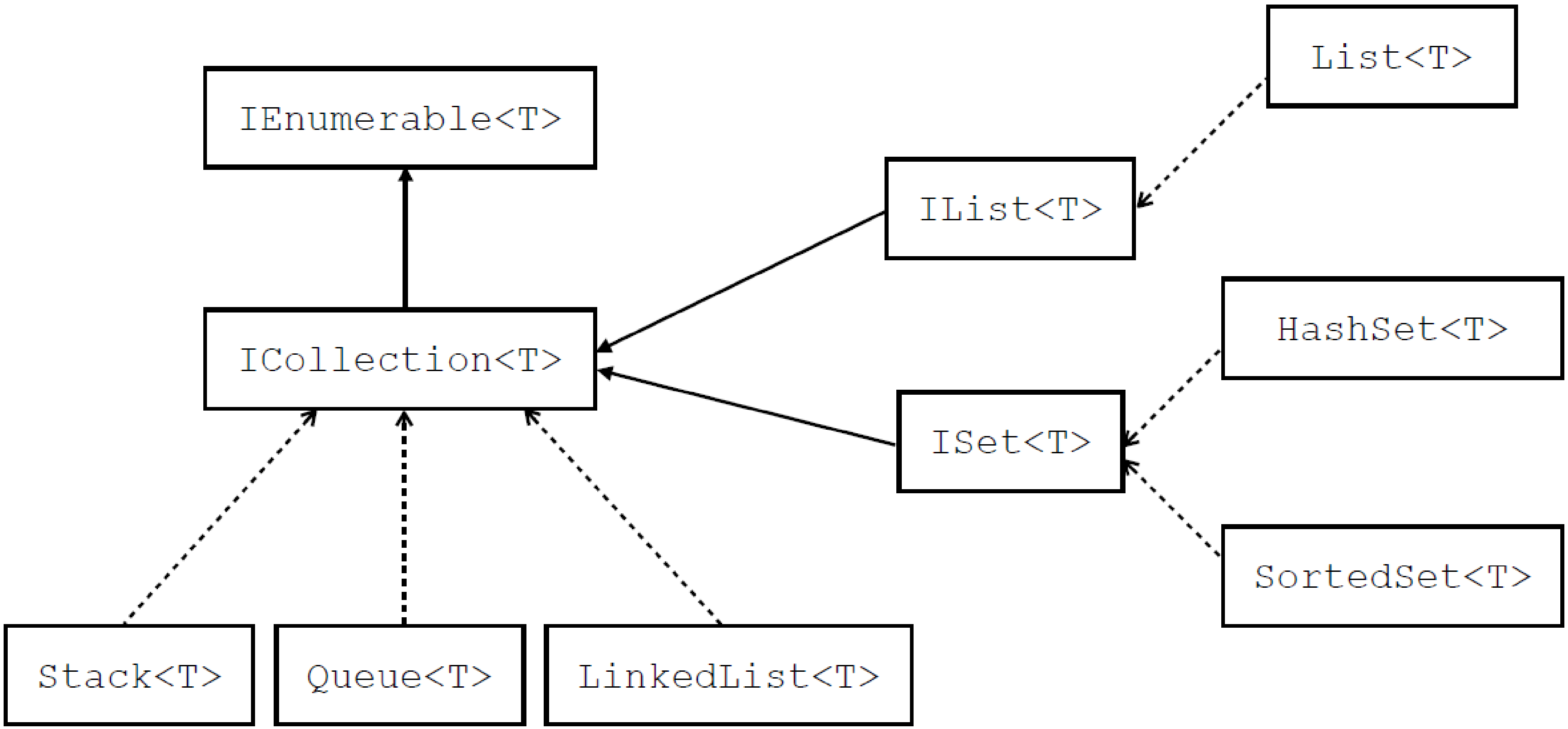
Generic Methods



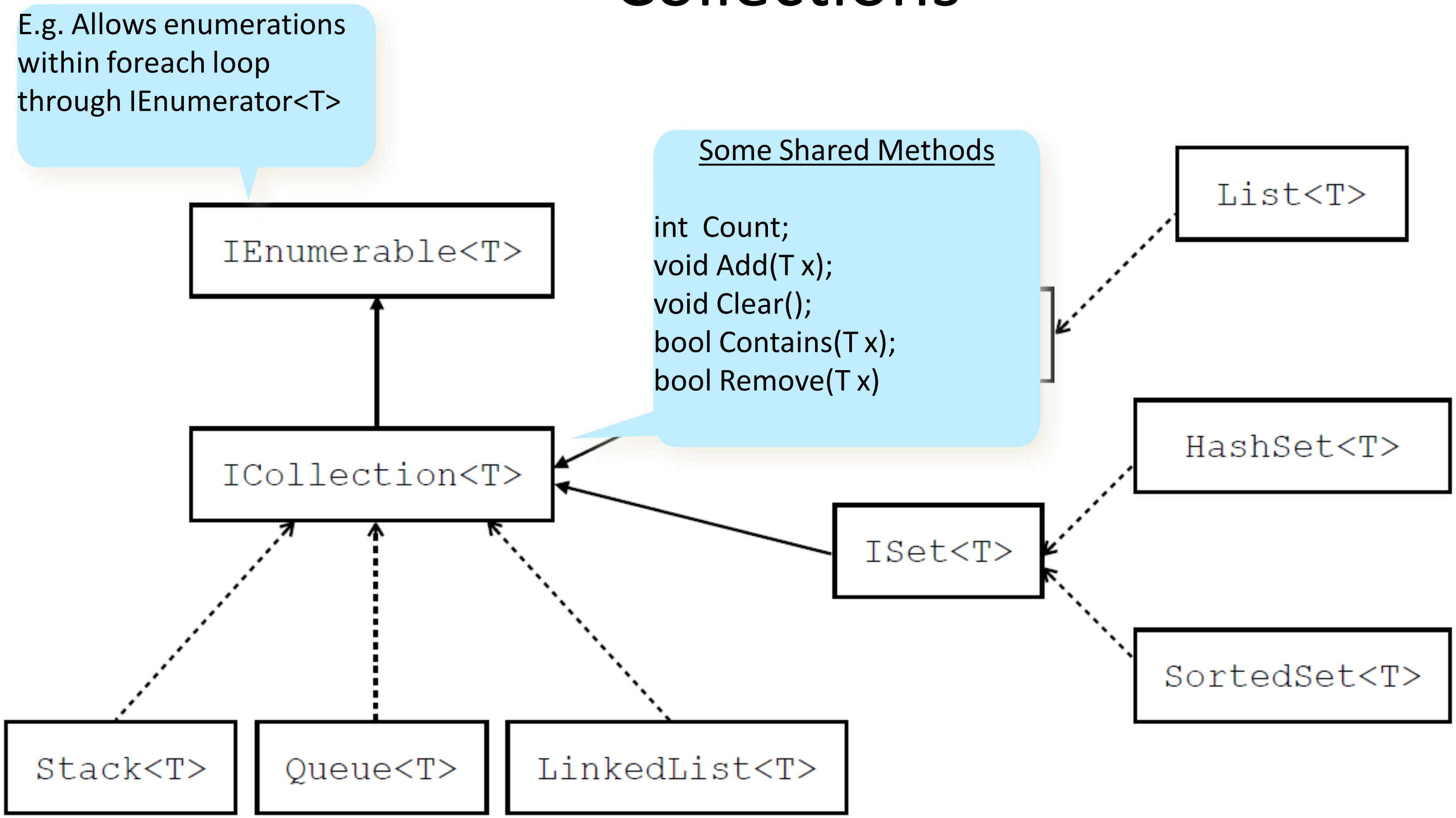
```
public static class ListExtensions
{
  public static void Scramble<T>(this List<T> array){...}

  public static void StableSort<T>(this List<T> array)
                                        where T : IComparable<T>
  {
    ...
  }
}
```

Overridden generic methods or methods implementing generic interfaces must have the same parameter constraints.

Collections



Collections

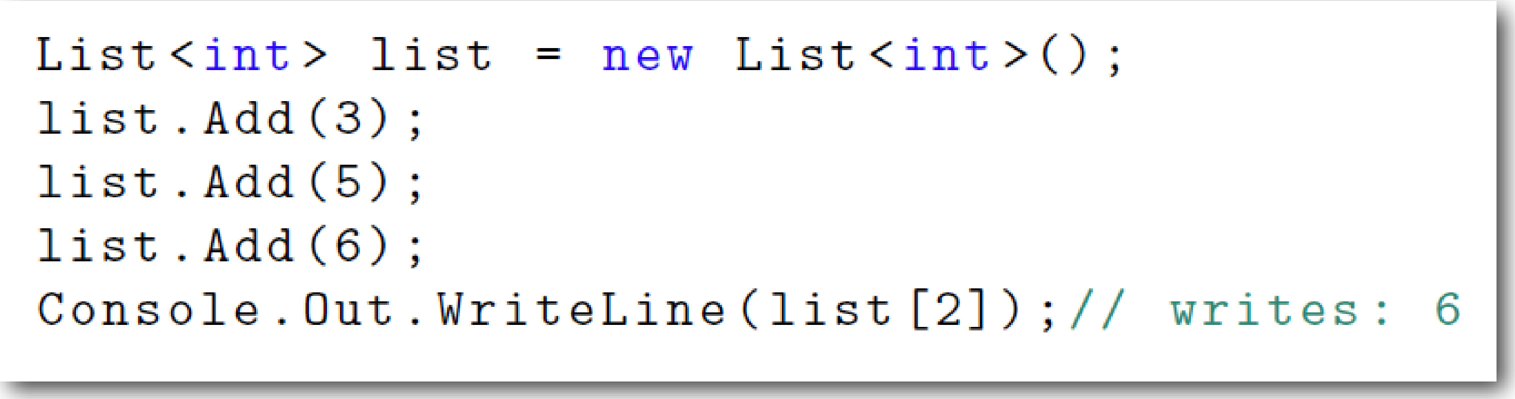


E.g. Allows enumerations within foreach loop through IEnumerator<T>

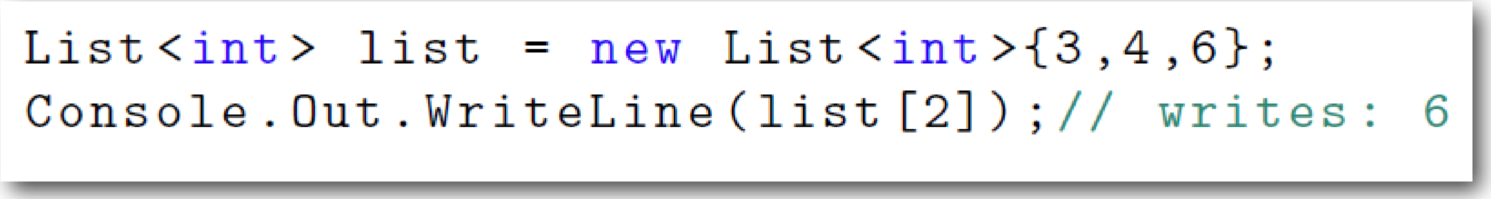
Some Shared Methods

int Count;
void Add(T x);
void Clear();
bool Contains(T x);
bool Remove(T x)

List<T>

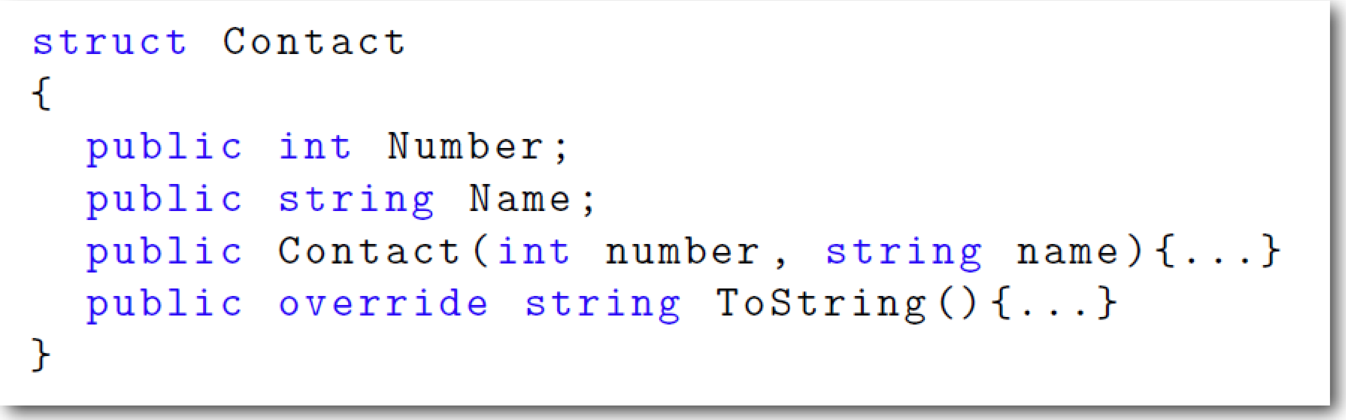


```
List<int> list = new List<int>();
list.Add(3);
list.Add(5);
list.Add(6);
Console.Out.WriteLine(list[2]);// writes: 6
```

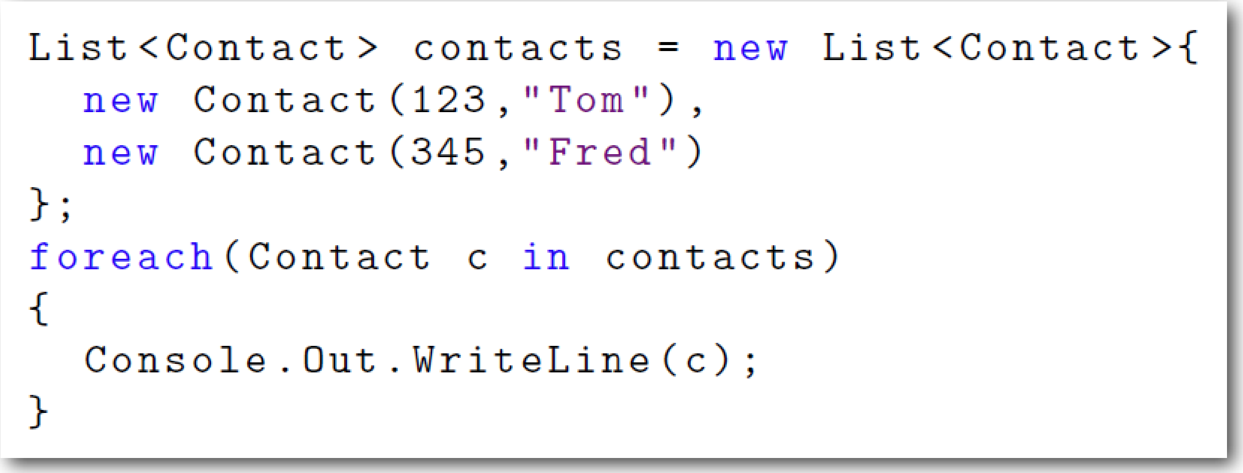


```
List<int> list = new List<int>{3,4,6};
Console.Out.WriteLine(list[2]);// writes: 6
```

List<T>

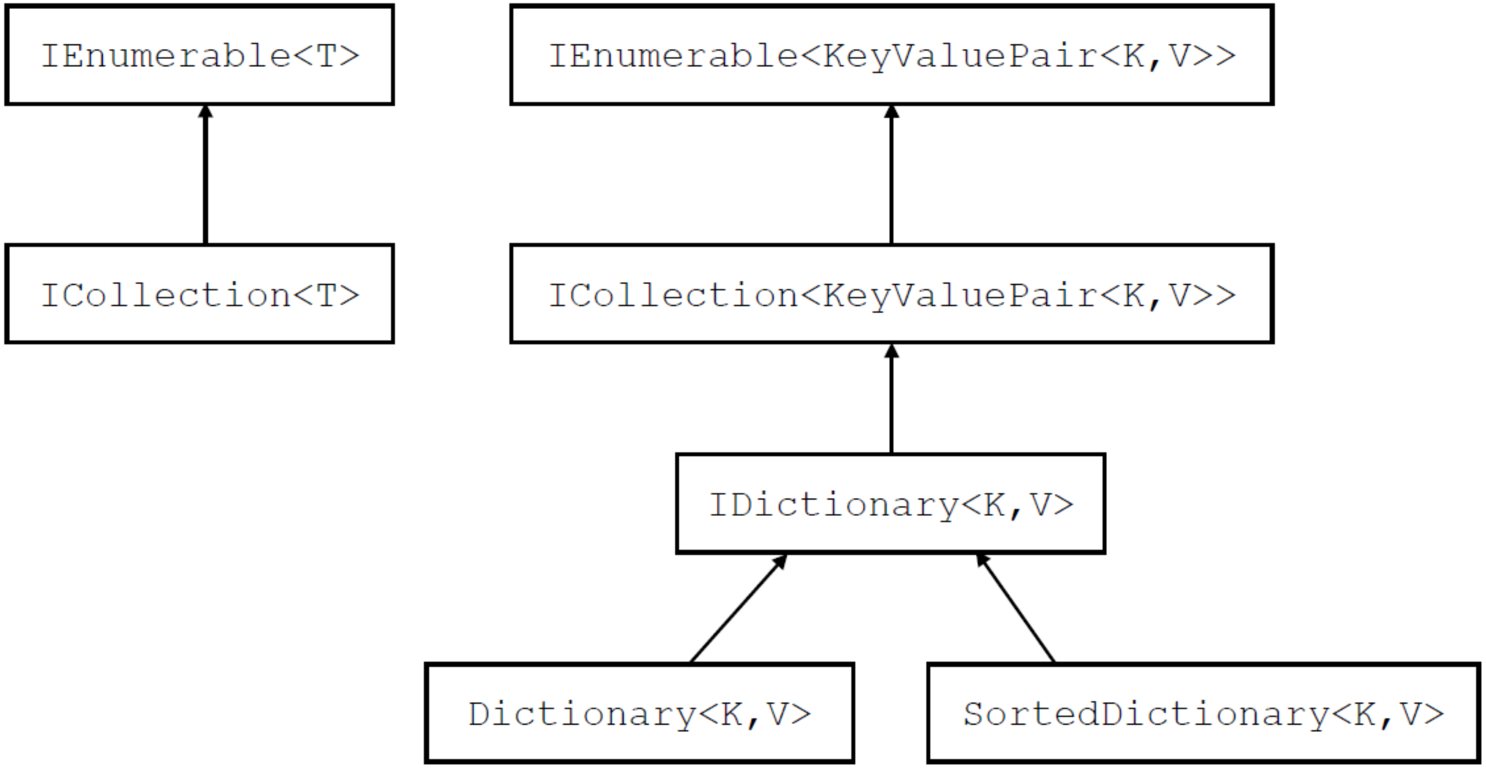


```
struct Contact
{
  public int Number;
  public string Name;
  public Contact(int number, string name){...}
  public override string ToString(){...}
}
```



```
List<Contact> contacts = new List<Contact>{
  new Contact(123,"Tom"),
  new Contact(345,"Fred")
};
foreach(Contact c in contacts)
{
  Console.Out.WriteLine(c);
}
```

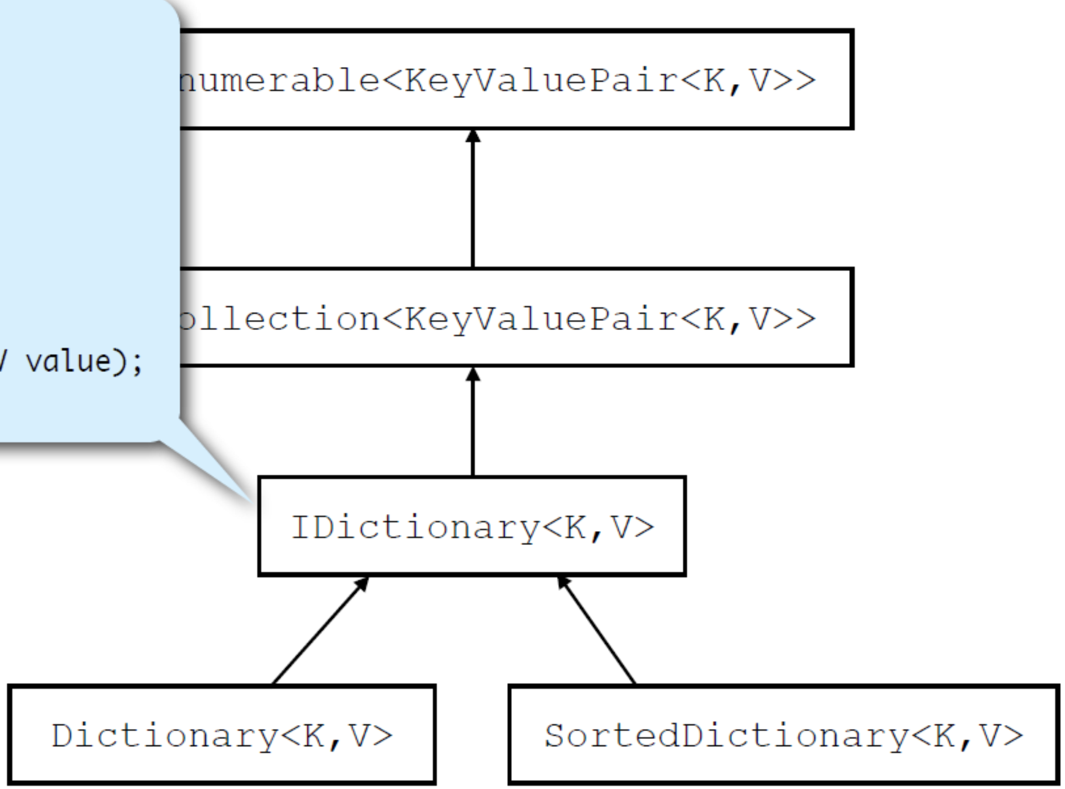
KeyValue Collections



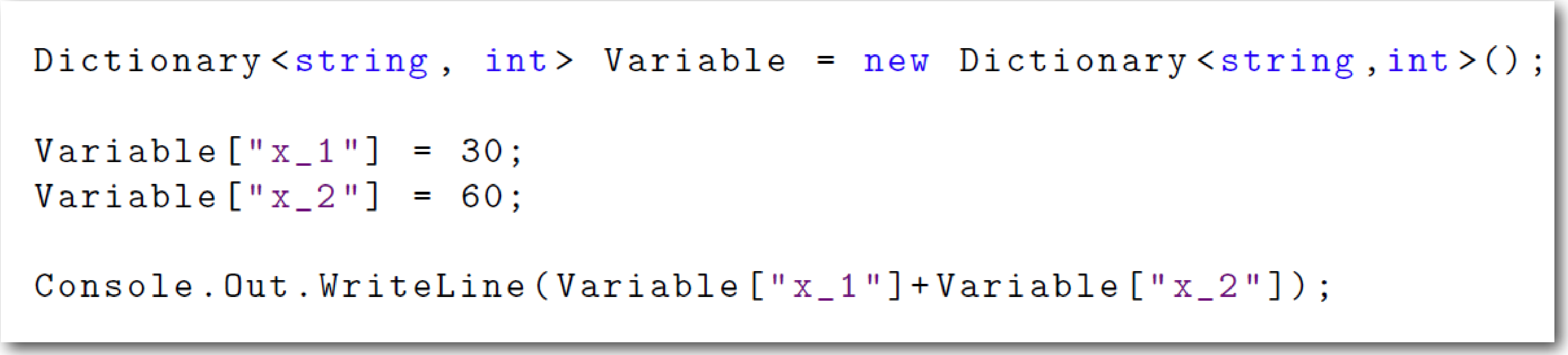
KeyValue Collections

Some Shared Methods

```
ICollection<K>  Keys;
ICollection<V>  Values;
V this[K key];
void Add(K key,V value);
bool Remove(K key);
bool ContainsKey(K key);
bool TryGetValue(K key, out V value);
```

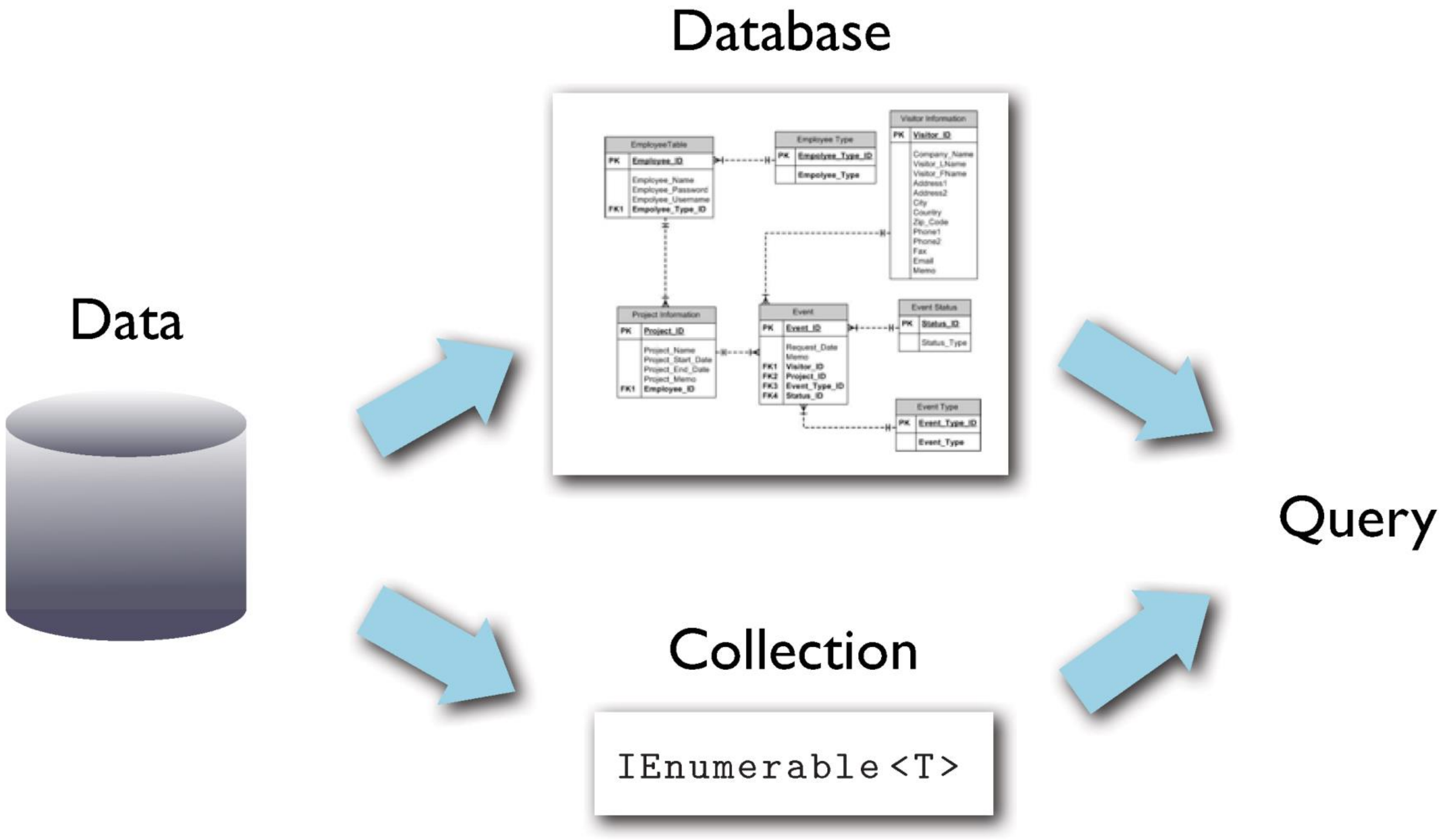


Dictonary<K,V> SortedDictionary<K,V>



```
Dictionary<string, int> Variable = new Dictionary<string,int>();

Variable["x_1"] = 30;
Variable["x_2"] = 60;

Console.Out.WriteLine(Variable["x_1"]+Variable["x_2"]);
```

LINQ



Database

Data

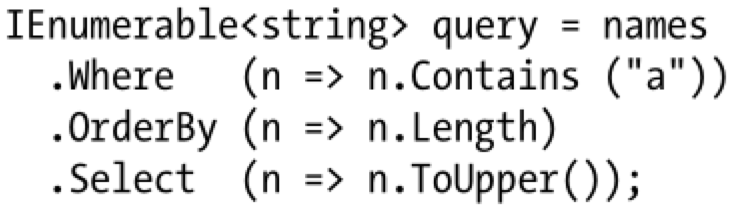
Query

Collection

`IEnumerable<T>`

Fluent Syntax

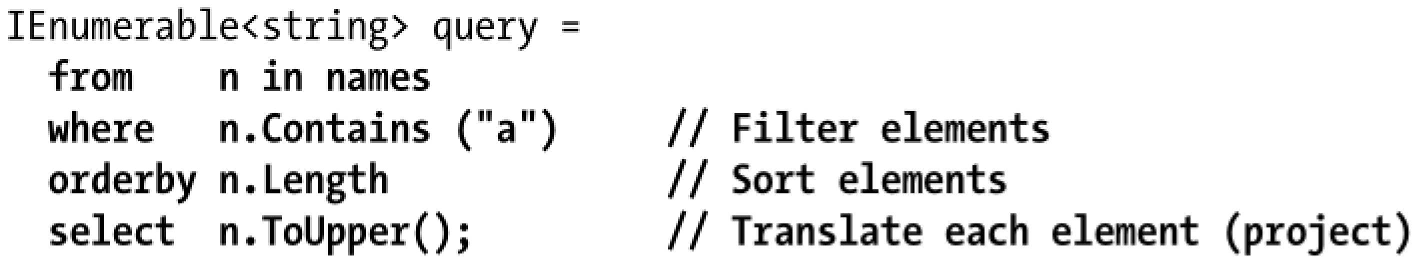
- Chaining Query Operators



```
IEnumerable<string> query = names
  .Where   (n => n.Contains ("a"))
  .OrderBy (n => n.Length)
  .Select  (n => n.ToUpper());
```

Query Expressions

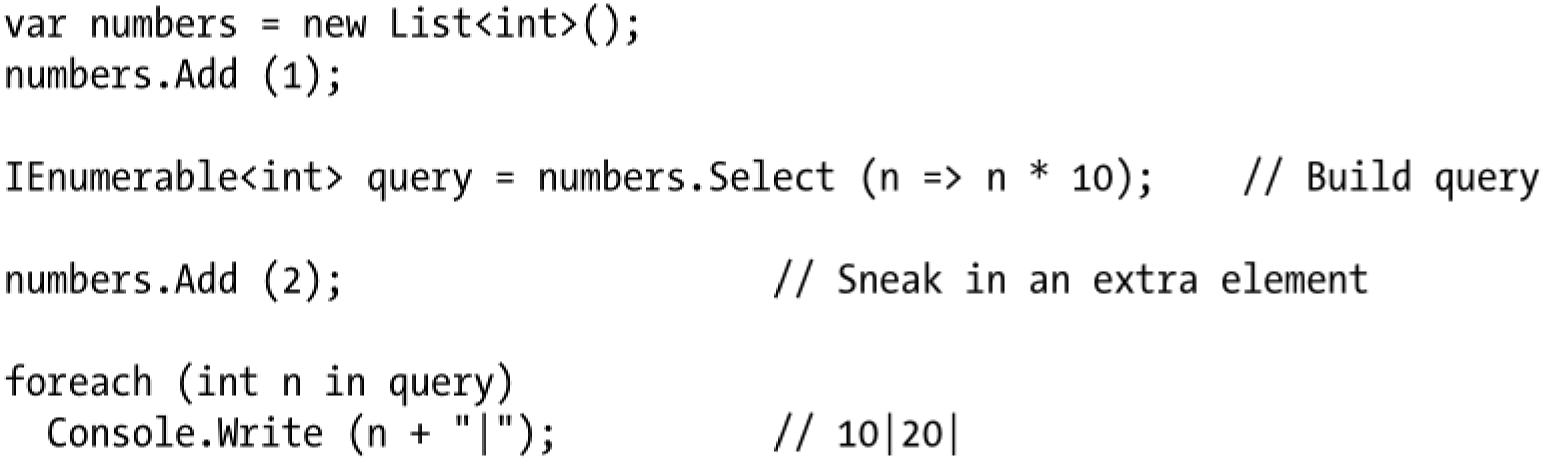
- C# provides a syntactic shortcut for writing LINQ queries, called query expressions



```
IEnumerable<string> query =
  from    n in names
  where   n.Contains ("a")     // Filter elements
  orderby n.Length             // Sort elements
  select  n.ToUpper();         // Translate each element (project)
```

Deferred Execution

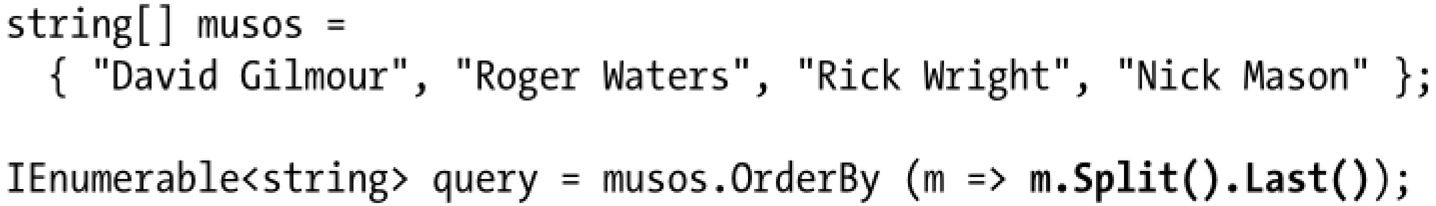
- An important feature of most query operators is that they execute not when constructed, but when enumerated



```
var numbers = new List<int>();
numbers.Add (1);

IEnumerable<int> query = numbers.Select (n => n * 10);    // Build query

numbers.Add (2);                    // Sneak in an extra element

foreach (int n in query)
  Console.Write (n + "|");          // 10|20|
```

Subqueries

- A subquery is a query contained within another query’s lambda expression



```
string[] musos =
  { "David Gilmour", "Roger Waters", "Rick Wright", "Nick Mason" };

IEnumerable<string> query = musos.OrderBy (m => m.Split().Last());
```

Strategies

- Composition Strategies
  – Progressive Query Building

- Projection Strategies
  – Anonymous Types

Clean Code*

- What Are We Aiming For?

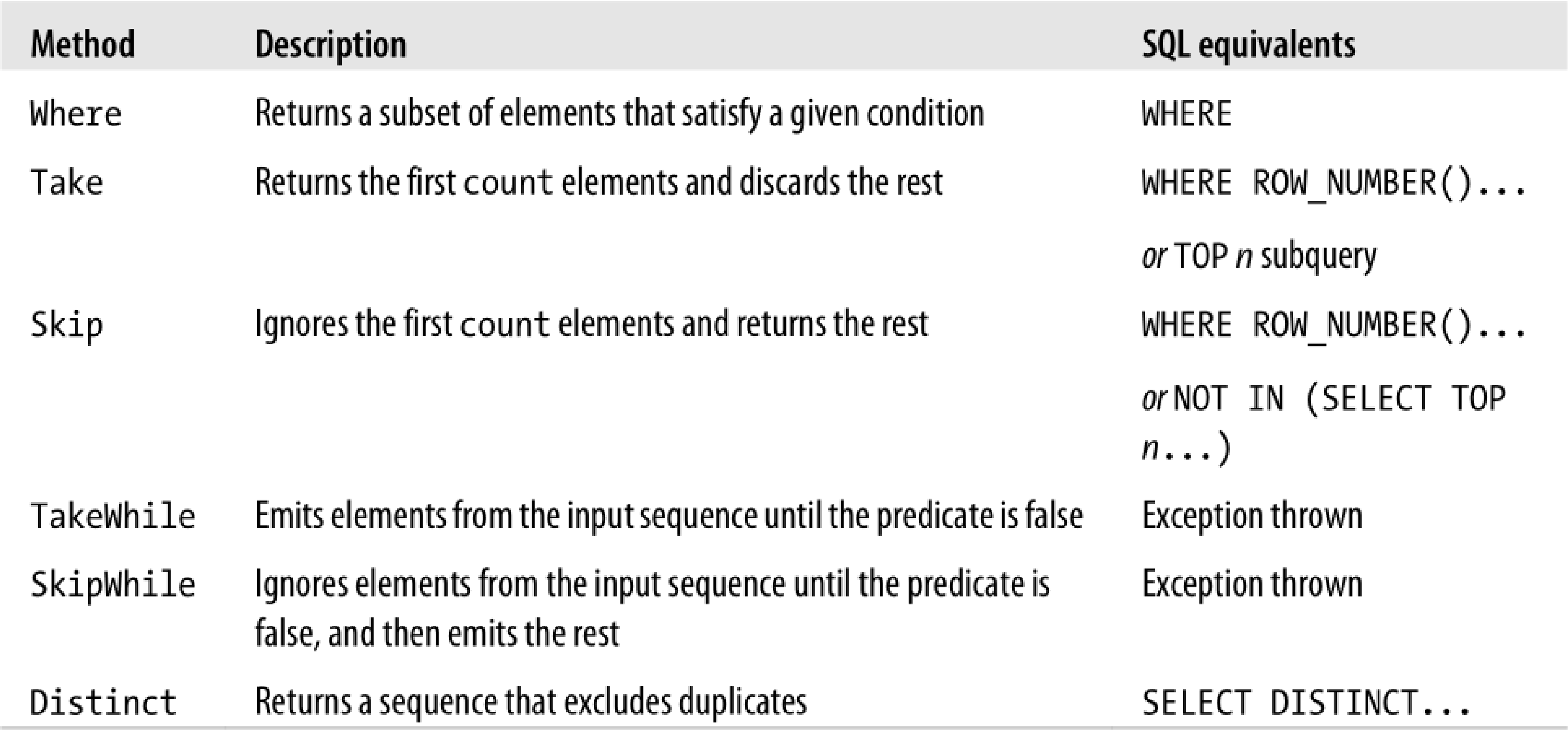
Most Concise
Solves the problem in the fewest lines of code

Most Readable
More code, but easier to understand what's going on

Fastest
More complicated but produces results quickly

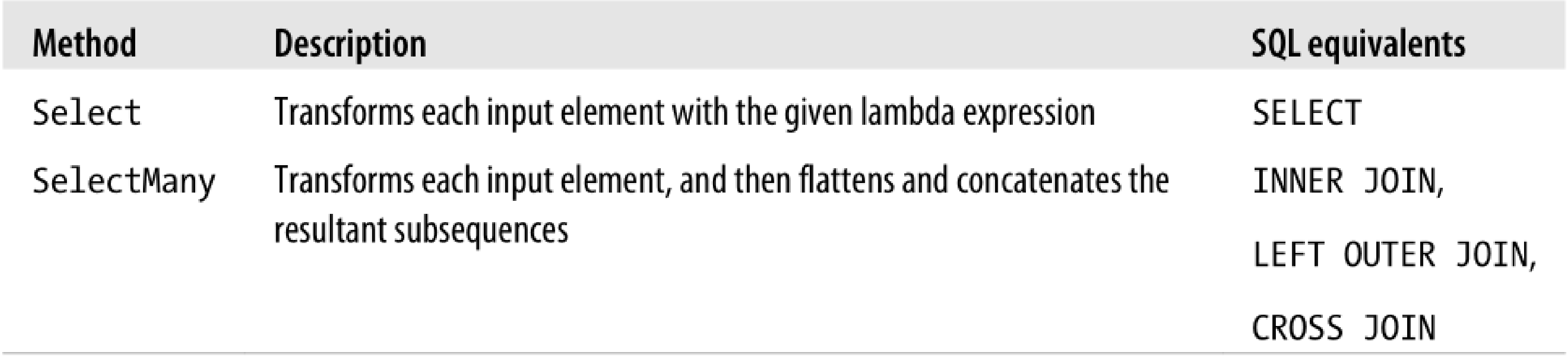
* Book of Robert C. Martin – Read It!

Filtering



Method	Description	SQL equivalents
`Where`	Returns a subset of elements that satisfy a given condition	`WHERE`
`Take`	Returns the first `count` elements and discards the rest	`WHERE ROW_NUMBER()...` *or* TOP *n* subquery
`Skip`	Ignores the first `count` elements and returns the rest	`WHERE ROW_NUMBER()...` *or* `NOT IN (SELECT TOP` *n*`...)`
`TakeWhile`	Emits elements from the input sequence until the predicate is false	Exception thrown
`SkipWhile`	Ignores elements from the input sequence until the predicate is false, and then emits the rest	Exception thrown
`Distinct`	Returns a sequence that excludes duplicates	`SELECT DISTINCT...`

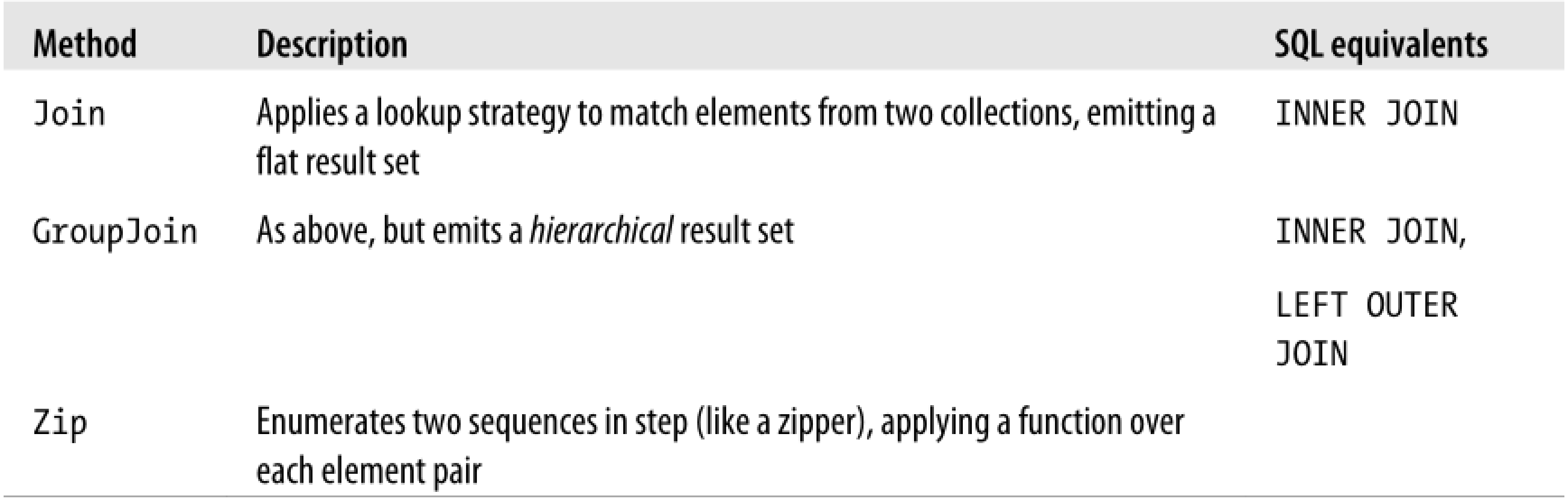
Projecting



Method	Description	SQL equivalents
`Select`	Transforms each input element with the given lambda expression	`SELECT`

`SelectMany`	Transforms each input element, and then flattens and concatenates the resultant subsequences	`INNER JOIN,`
		`LEFT OUTER JOIN,`
		`CROSS JOIN`

Joining

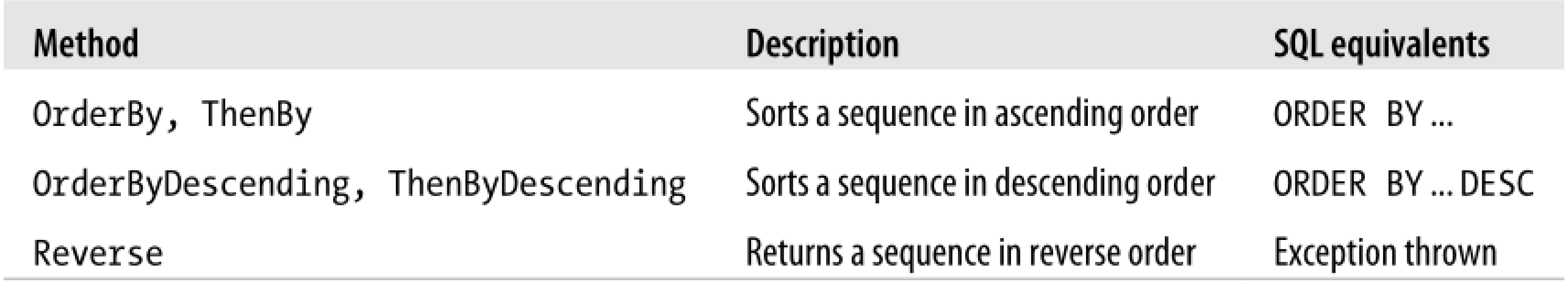


Method	Description	SQL equivalents
`Join`	Applies a lookup strategy to match elements from two collections, emitting a flat result set	`INNER JOIN`

`GroupJoin`	As above, but emits a *hierarchical* result set	`INNER JOIN,`
		`LEFT OUTER JOIN`

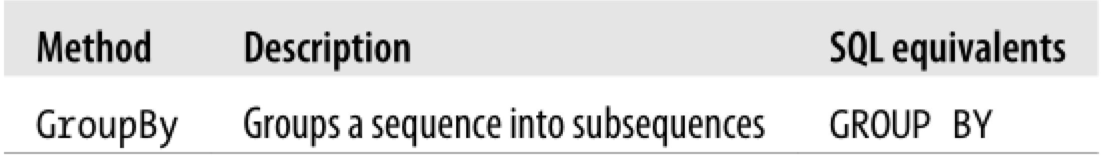
`Zip`	Enumerates two sequences in step (like a zipper), applying a function over each element pair	

Ordering



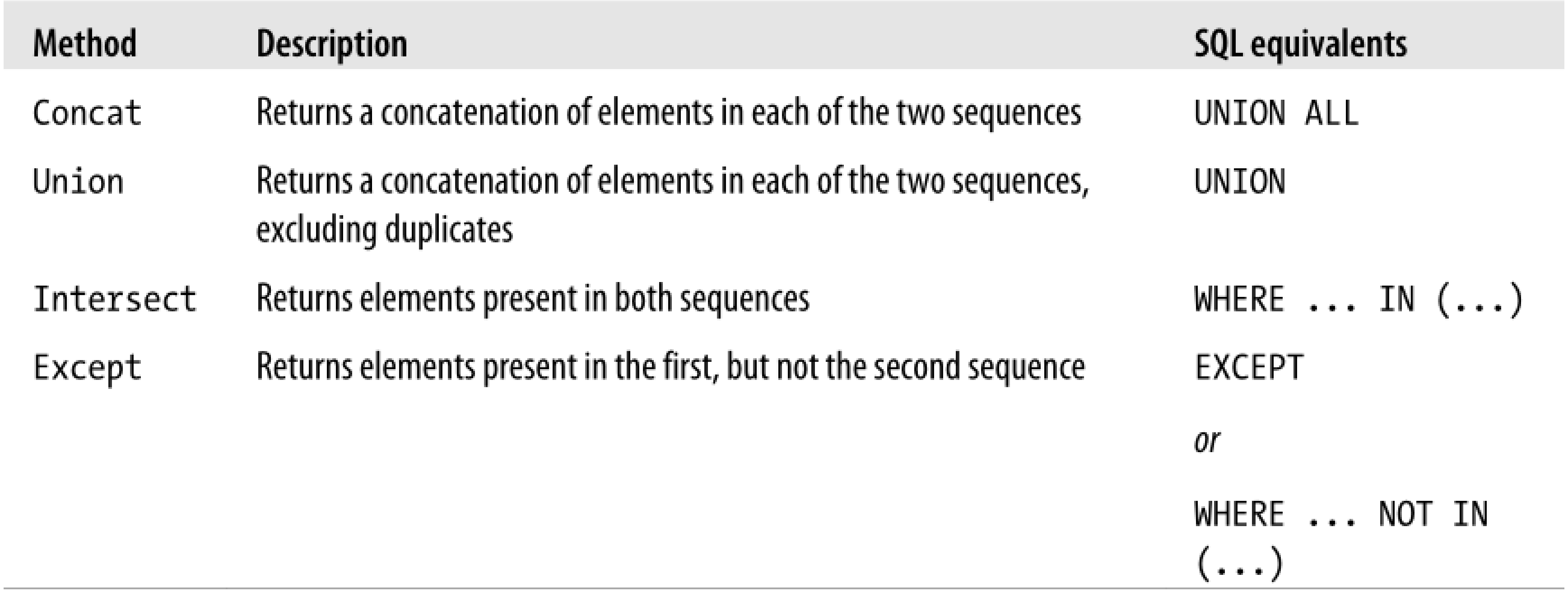
Method	Description	SQL equivalents
`OrderBy, ThenBy`	Sorts a sequence in ascending order	`ORDER BY` ...
`OrderByDescending, ThenByDescending`	Sorts a sequence in descending order	`ORDER BY` ... DESC
`Reverse`	Returns a sequence in reverse order	Exception thrown

Grouping



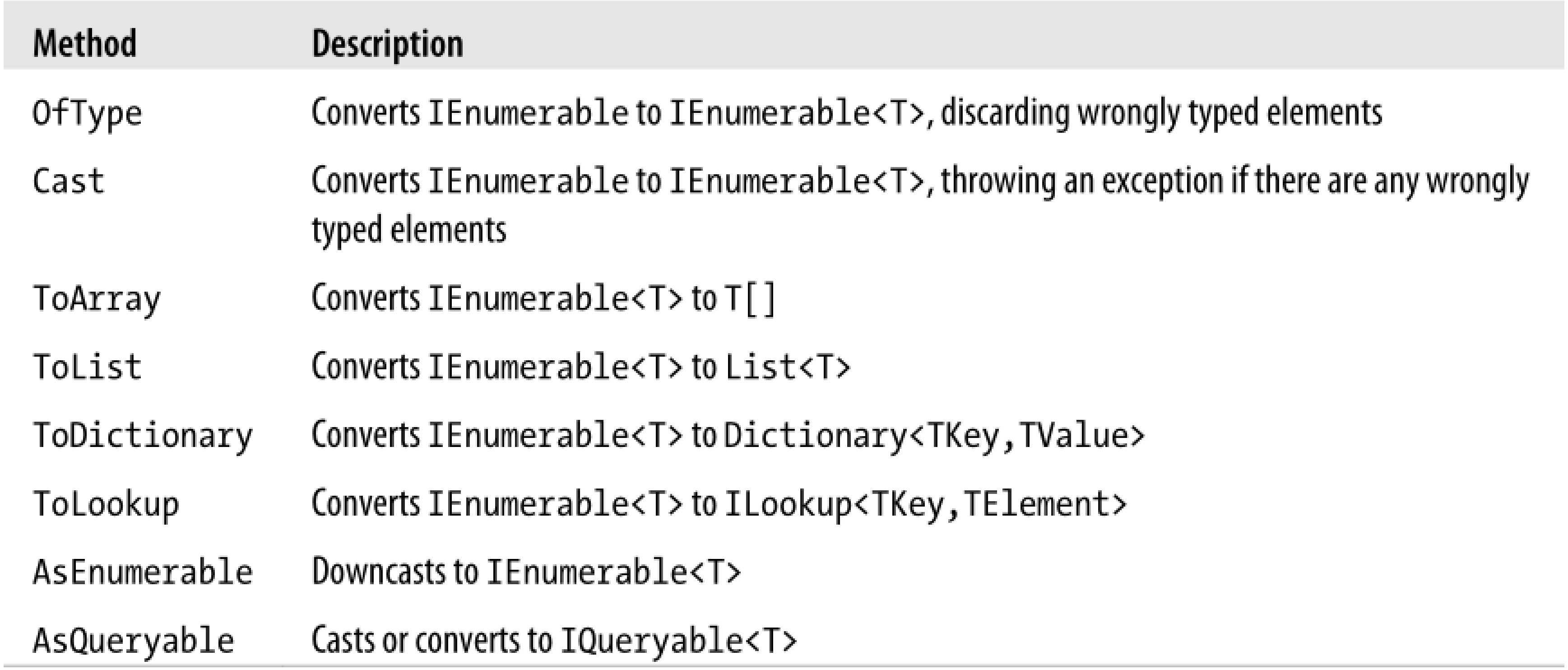
Method	Description	SQL equivalents
GroupBy	Groups a sequence into subsequences	GROUP BY

Set Operators



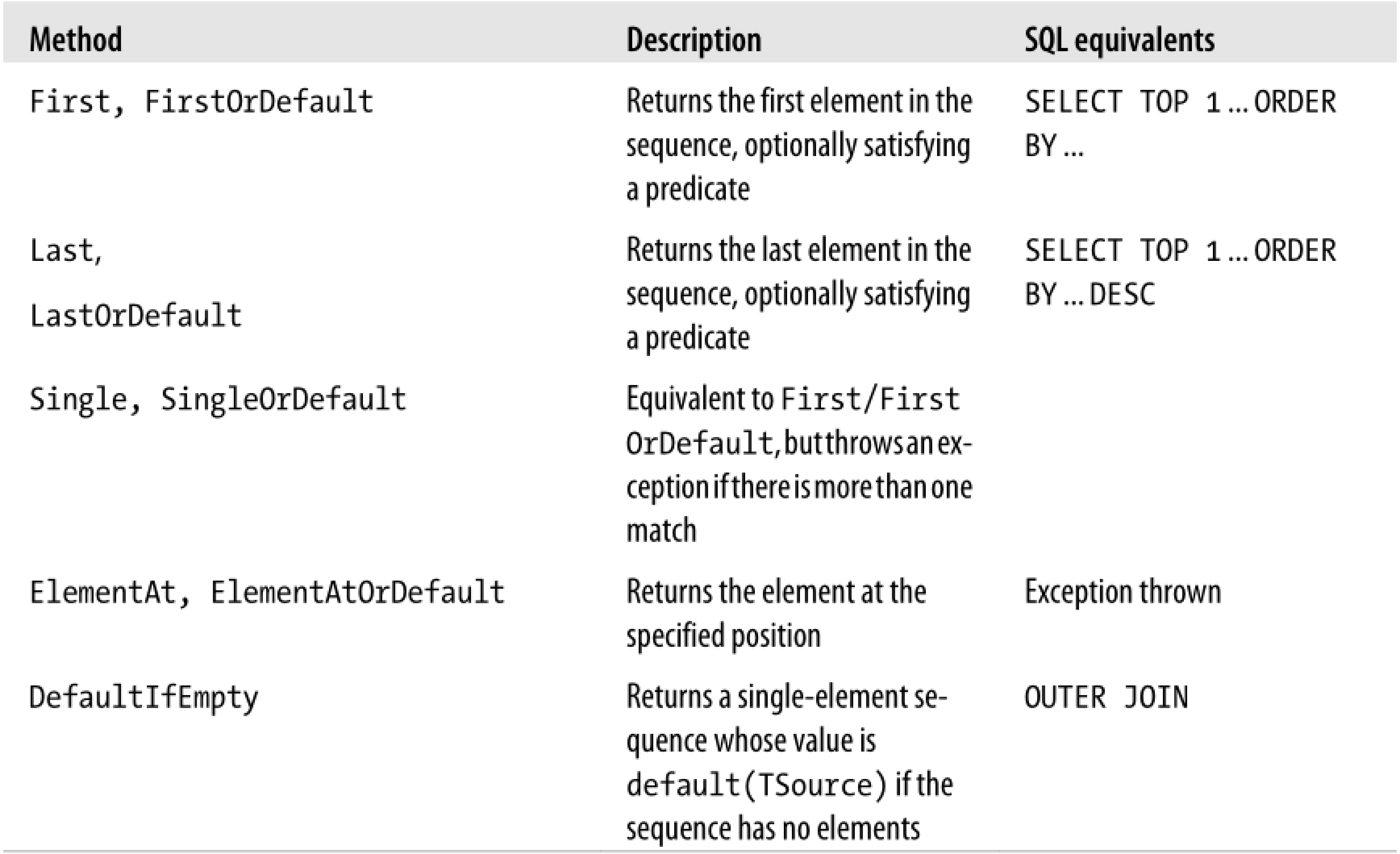
Method	Description	SQL equivalents
Concat	Returns a concatenation of elements in each of the two sequences	UNION ALL
Union	Returns a concatenation of elements in each of the two sequences, excluding duplicates	UNION
Intersect	Returns elements present in both sequences	WHERE ... IN (...)
Except	Returns elements present in the first, but not the second sequence	EXCEPT *or* WHERE ... NOT IN (...)

Conversion Methods



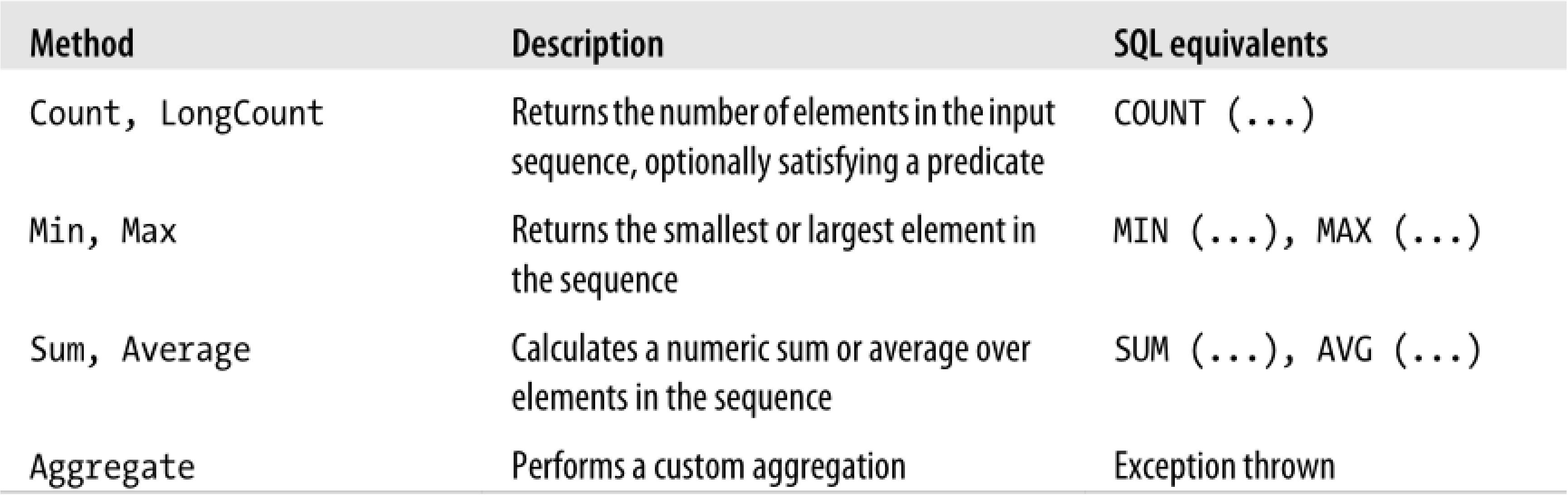
Method	Description
OfType	Converts IEnumerable to IEnumerable<T>, discarding wrongly typed elements
Cast	Converts IEnumerable to IEnumerable<T>, throwing an exception if there are any wrongly typed elements
ToArray	Converts IEnumerable<T> to T[]
ToList	Converts IEnumerable<T> to List<T>
ToDictionary	Converts IEnumerable<T> to Dictionary<TKey,TValue>
ToLookup	Converts IEnumerable<T> to ILookup<TKey,TElement>
AsEnumerable	Downcasts to IEnumerable<T>
AsQueryable	Casts or converts to IQueryable<T>

Element Operators



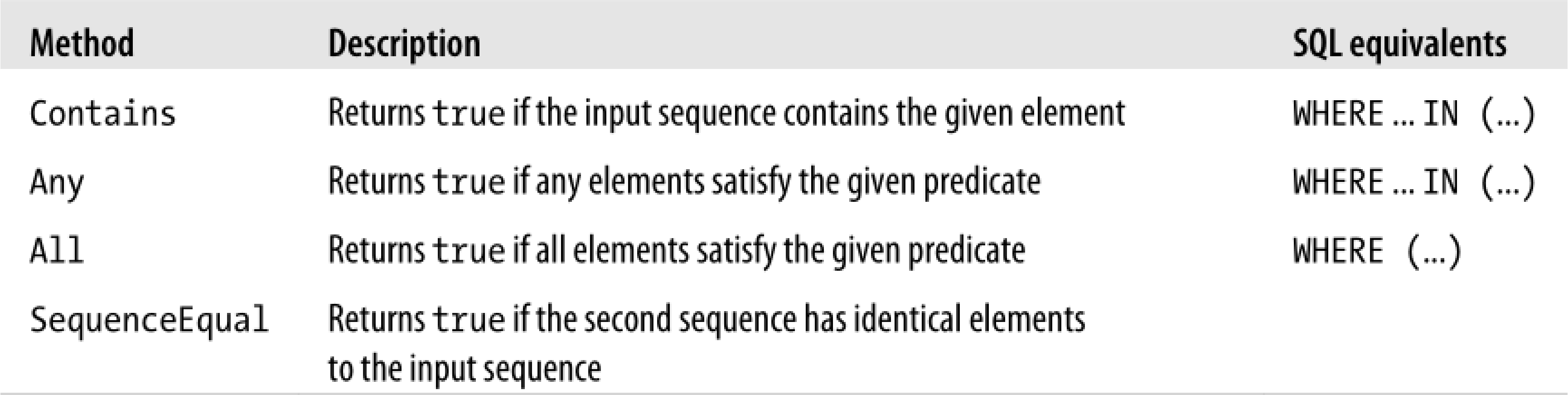
Method	Description	SQL equivalents
First, FirstOrDefault	Returns the first element in the sequence, optionally satisfying a predicate	SELECT TOP 1 ... ORDER BY ...
Last, LastOrDefault	Returns the last element in the sequence, optionally satisfying a predicate	SELECT TOP 1 ... ORDER BY ... DESC
Single, SingleOrDefault	Equivalent to First/FirstOrDefault, but throws an exception if there is more than one match	
ElementAt, ElementAtOrDefault	Returns the element at the specified position	Exception thrown
DefaultIfEmpty	Returns a single-element sequence whose value is default(TSource) if the sequence has no elements	OUTER JOIN

Aggregation Methods



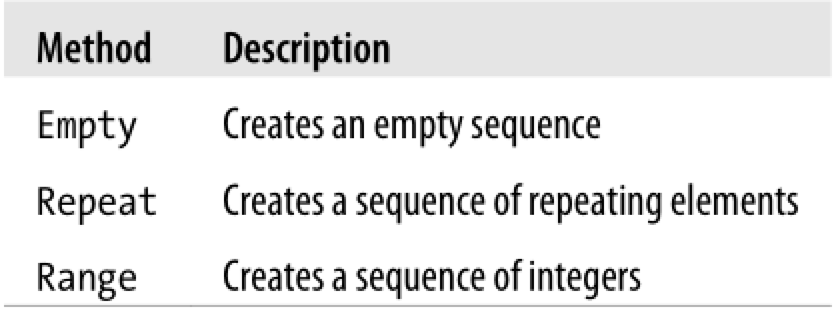
Method	Description	SQL equivalents
`Count, LongCount`	Returns the number of elements in the input sequence, optionally satisfying a predicate	`COUNT (...)`
`Min, Max`	Returns the smallest or largest element in the sequence	`MIN (...), MAX (...)`
`Sum, Average`	Calculates a numeric sum or average over elements in the sequence	`SUM (...), AVG (...)`
`Aggregate`	Performs a custom aggregation	Exception thrown

Quantifiers



Method	Description	SQL equivalents
`Contains`	Returns `true` if the input sequence contains the given element	`WHERE` ... `IN (`...`)`
`Any`	Returns `true` if any elements satisfy the given predicate	`WHERE` ... `IN (`...`)`
`All`	Returns `true` if all elements satisfy the given predicate	`WHERE (`...`)`
`SequenceEqual`	Returns `true` if the second sequence has identical elements to the input sequence	

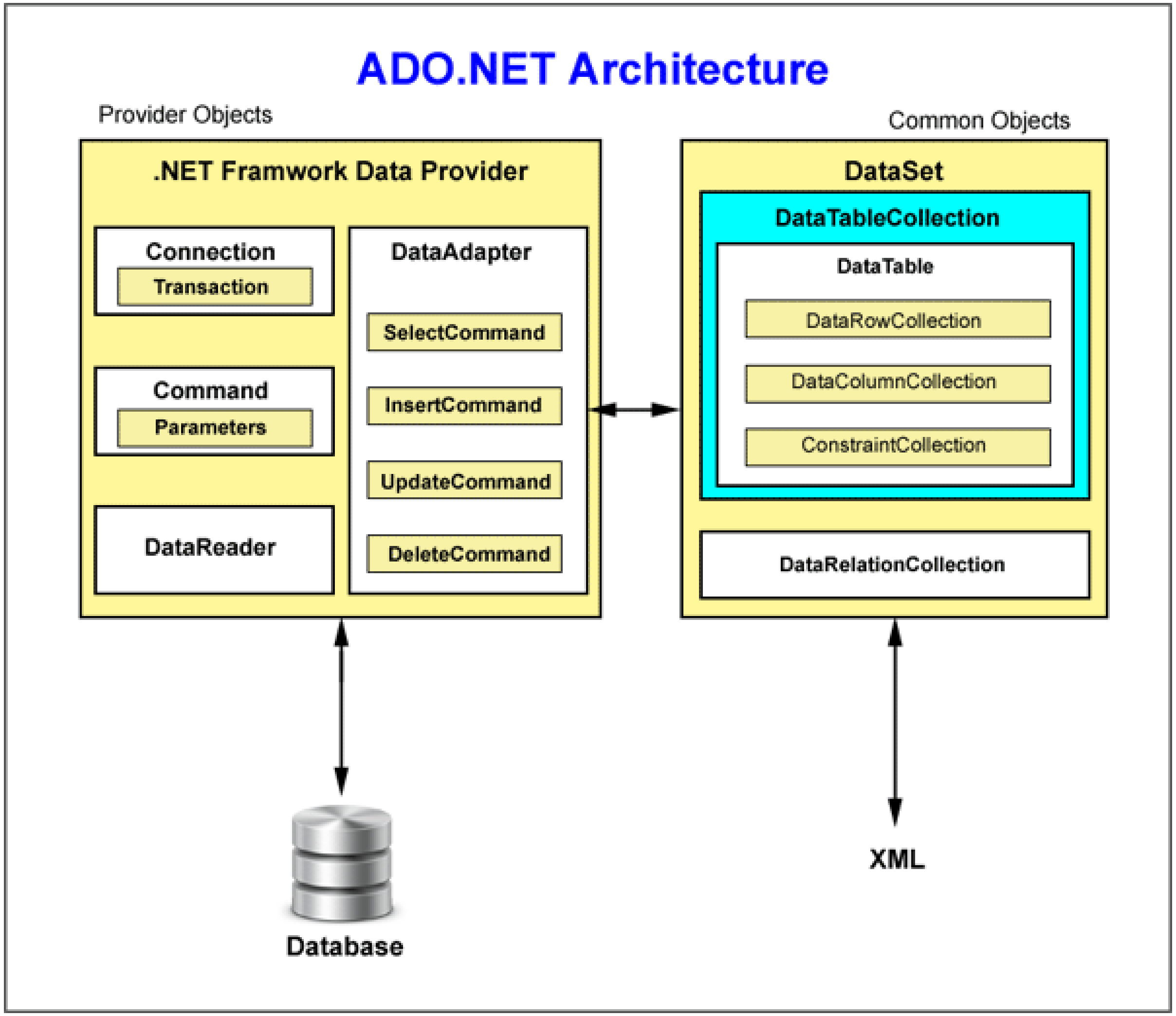
Generation Methods



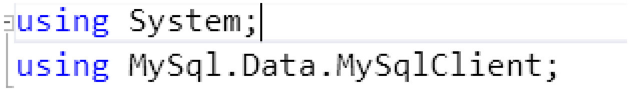
Method	Description
`Empty`	Creates an empty sequence
`Repeat`	Creates a sequence of repeating elements
`Range`	Creates a sequence of integers

ADO

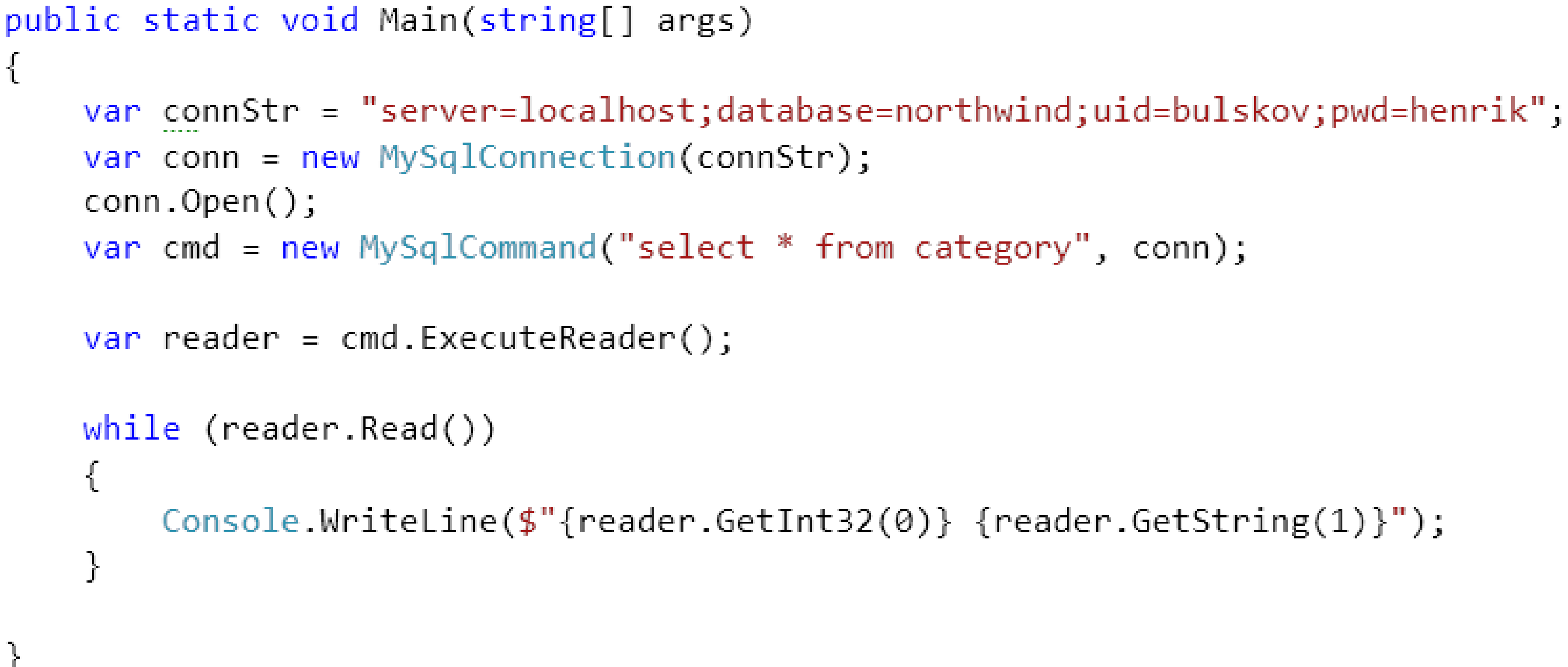
ADO.NET


ADO.NET Architecture
Provider Objects
Common Objects
.NET Framwork Data Provider
Connection
Transaction
Command
Parameters
DataReader
DataAdapter
SelectCommand
InsertCommand
UpdateCommand
DeleteCommand
DataSet
DataTableCollection
DataTable
DataRowCollection
DataColumnCollection
ConstraintCollection
DataRelationCollection
Database
XML

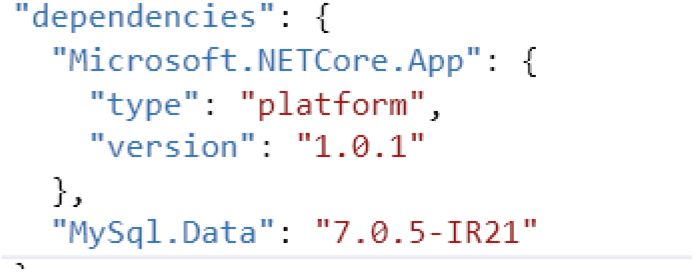
ADO.NET Example



```
using System;
using MySql.Data.MySqlClient;
```



```
public static void Main(string[] args)
{
    var connStr = "server=localhost;database=northwind;uid=bulskov;pwd=henrik";
    var conn = new MySqlConnection(connStr);
    conn.Open();
    var cmd = new MySqlCommand("select * from category", conn);

    var reader = cmd.ExecuteReader();

    while (reader.Read())
    {
        Console.WriteLine($"{reader.GetInt32(0)} {reader.GetString(1)}");
    }

}
```

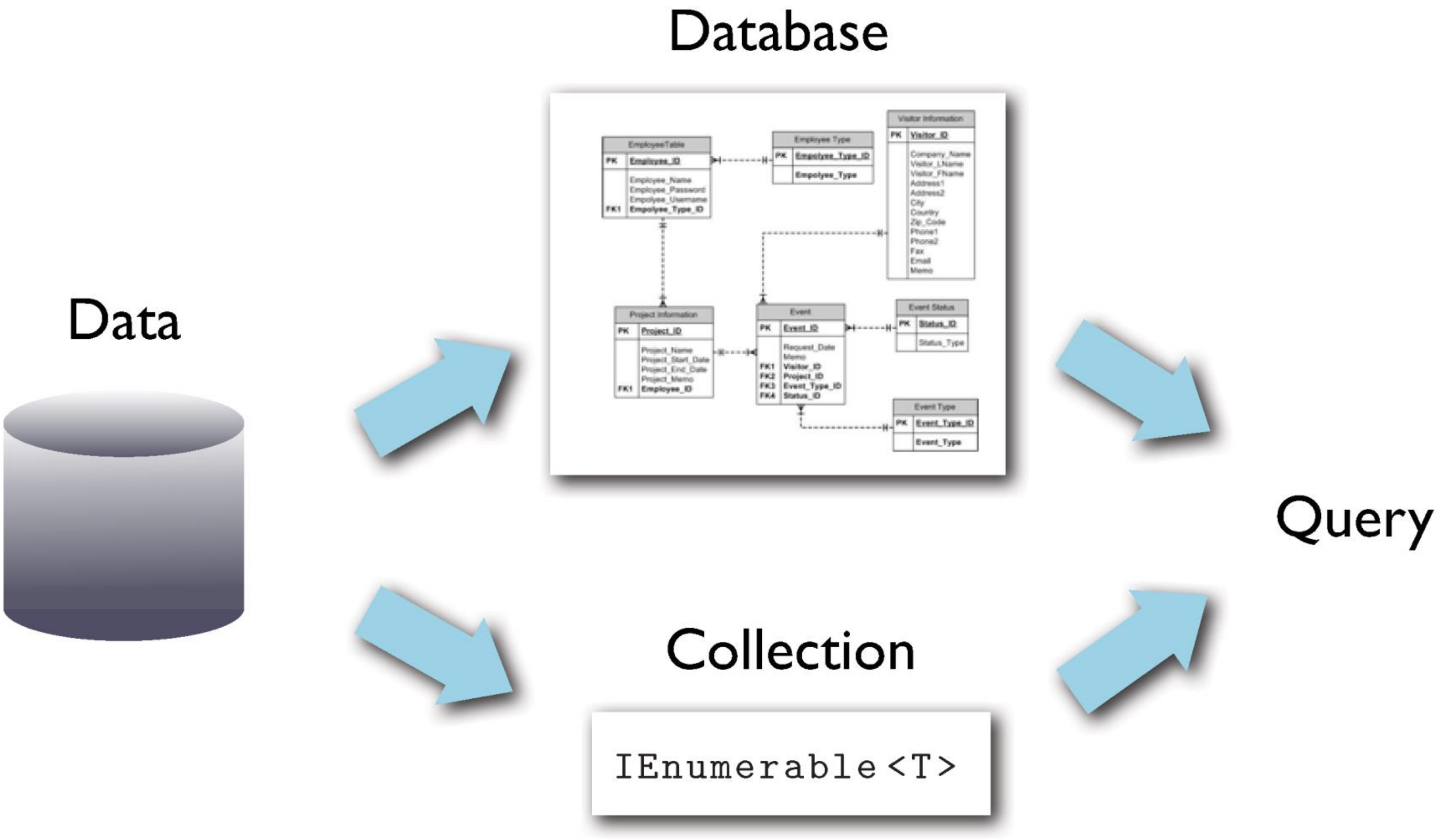


```
"dependencies": {
  "Microsoft.NETCore.App": {
    "type": "platform",
    "version": "1.0.1"
  },
  "MySql.Data": "7.0.5-IR21"
```

Language INtegrated Query

LINQ

LINQ



Database

Data

Query

Collection

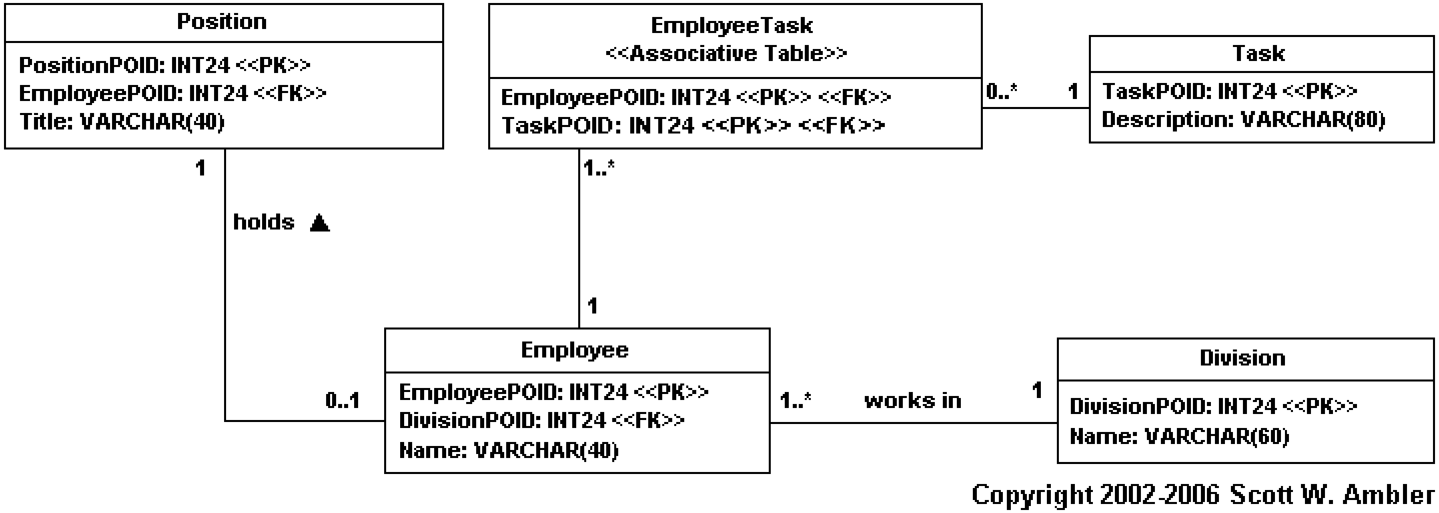
`IEnumerable<T>`

Object-Relational Mapping

- Object-relational mapping (ORM, O/RM, and O/R mapping tool) is a programming technique for converting data between incompatible type systems in object-oriented programming languages.

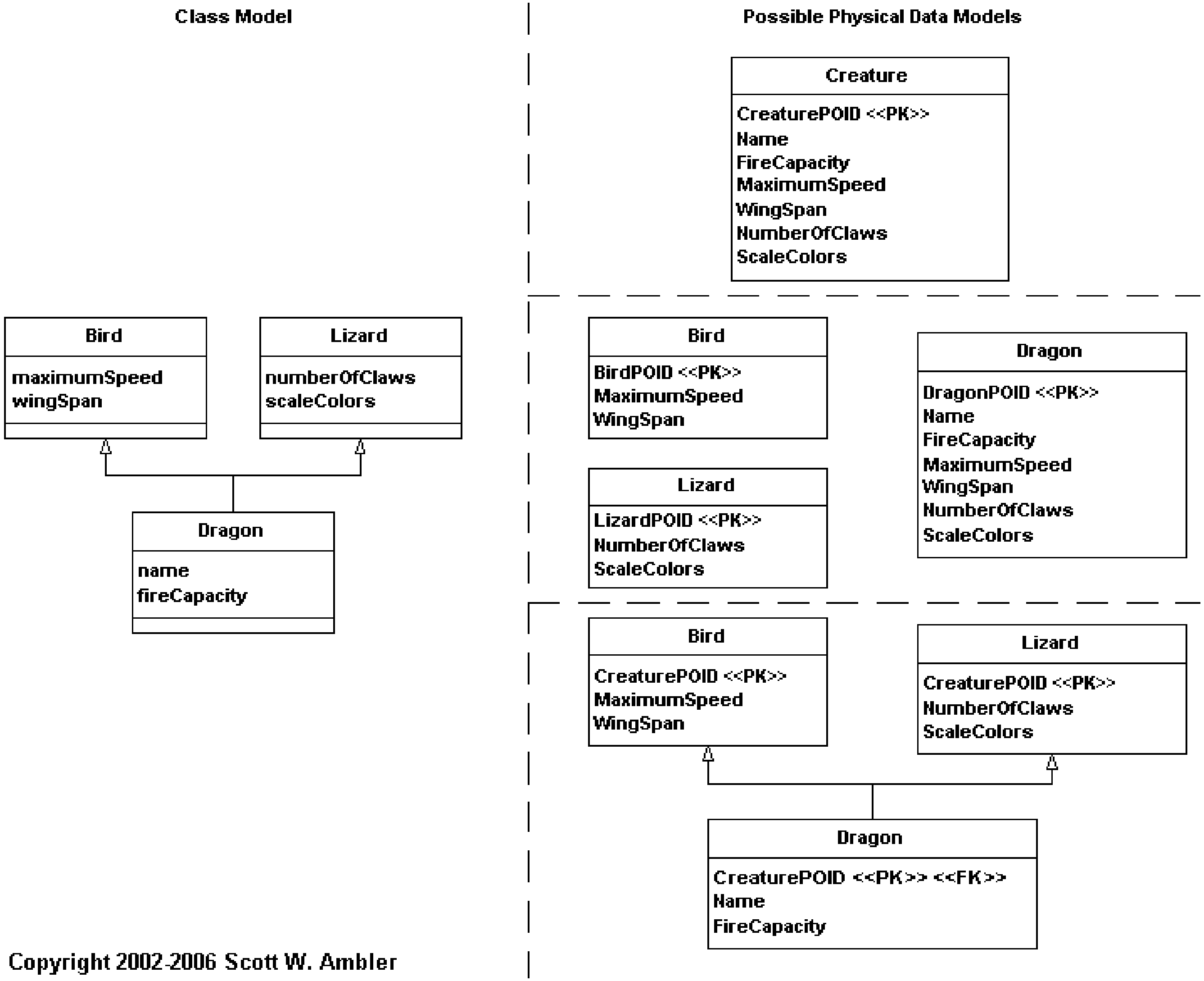
Mapping Object Relationships

- One-to-one relationships
- One-to-many relationships
- Many-to-many relationships



Copyright 2002-2006 Scott W. Ambler

Mapping Inheritance Structures



Class Model

Bird
- maximumSpeed
- wingSpan

Lizard
- numberOfClaws
- scaleColors

Dragon
- name
- fireCapacity

Possible Physical Data Models

Creature
- CreaturePOID <<PK>>
- Name
- FireCapacity
- MaximumSpeed
- WingSpan
- NumberOfClaws
- ScaleColors

Bird
- BirdPOID <<PK>>
- MaximumSpeed
- WingSpan

Lizard
- LizardPOID <<PK>>
- NumberOfClaws
- ScaleColors

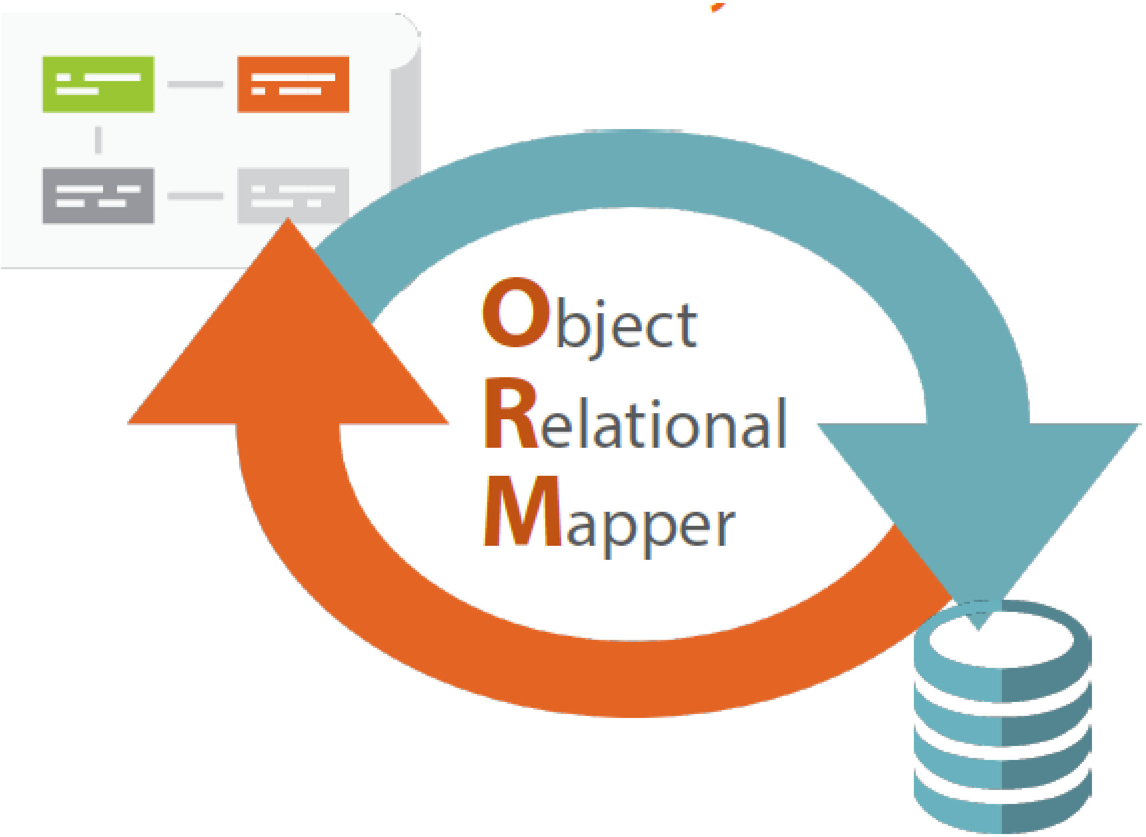
Dragon
- DragonPOID <<PK>>
- Name
- FireCapacity
- MaximumSpeed
- WingSpan
- NumberOfClaws
- ScaleColors

Bird
- CreaturePOID <<PK>>
- MaximumSpeed
- WingSpan

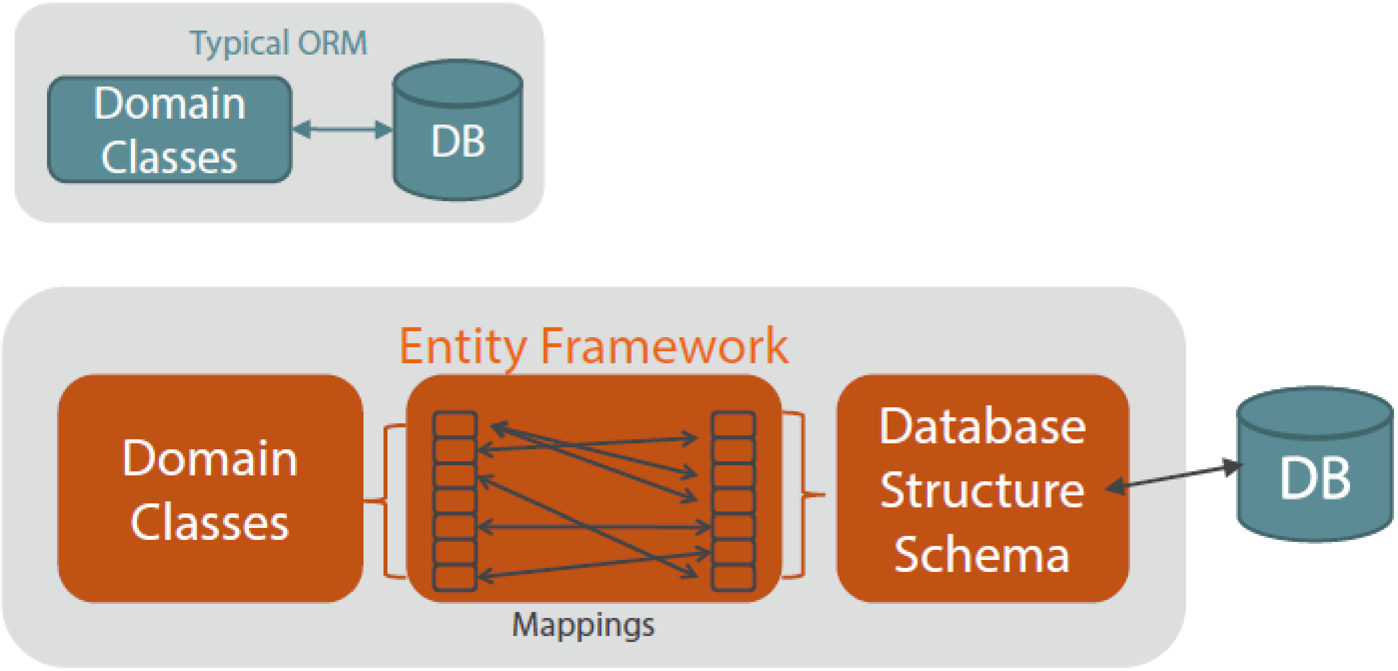
Lizard
- CreaturePOID <<PK>>
- NumberOfClaws
- ScaleColors

Dragon
- CreaturePOID <<PK>> <<FK>>
- Name
- FireCapacity

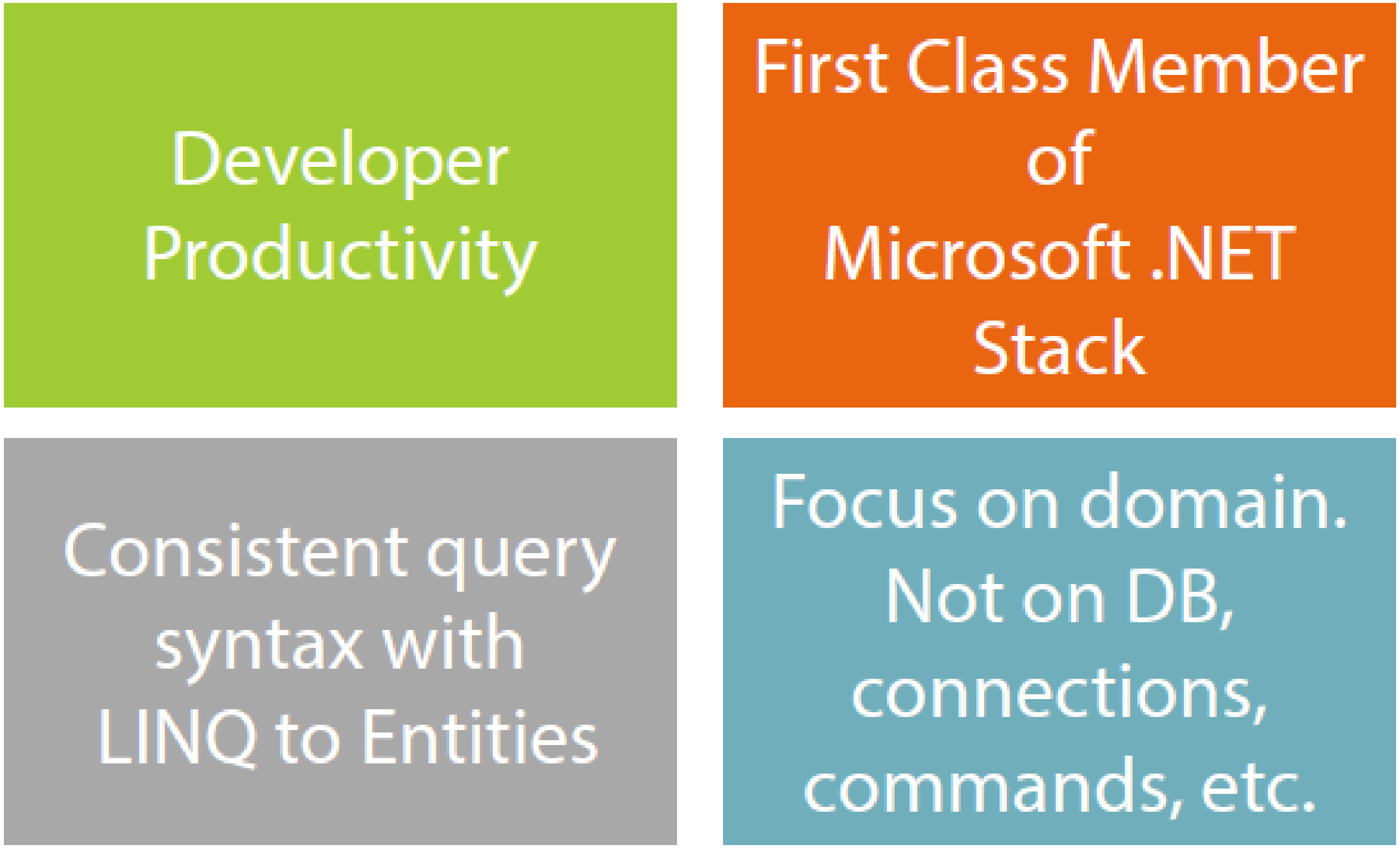
What is Entity Framework



Entity Framework vs. Other ORMs



Why Entity Framework?



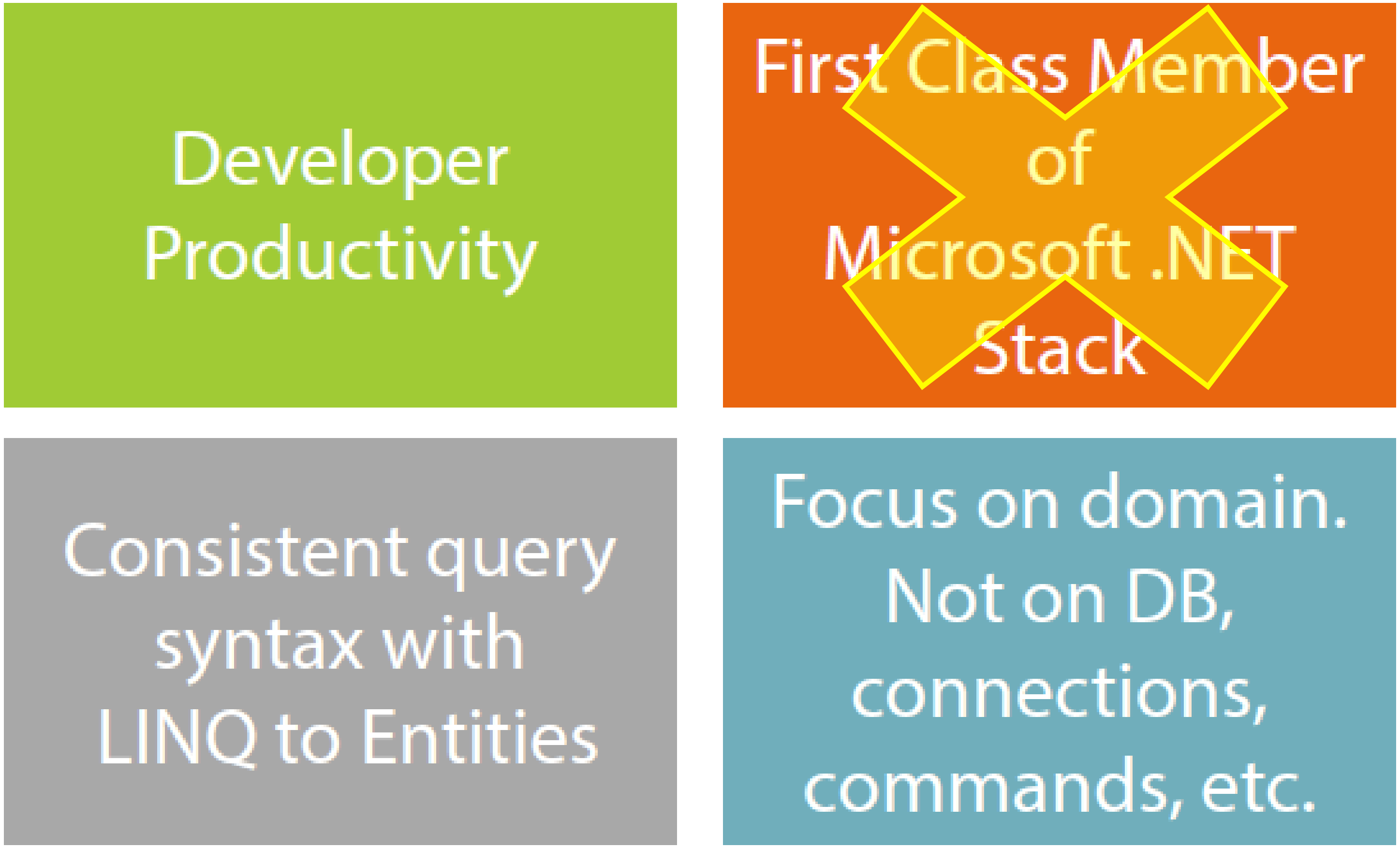
Developer Productivity

First Class Member of Microsoft .NET Stack

Consistent query syntax with LINQ to Entities

Focus on domain. Not on DB, connections, commands, etc.

Why Entity Framework Core?



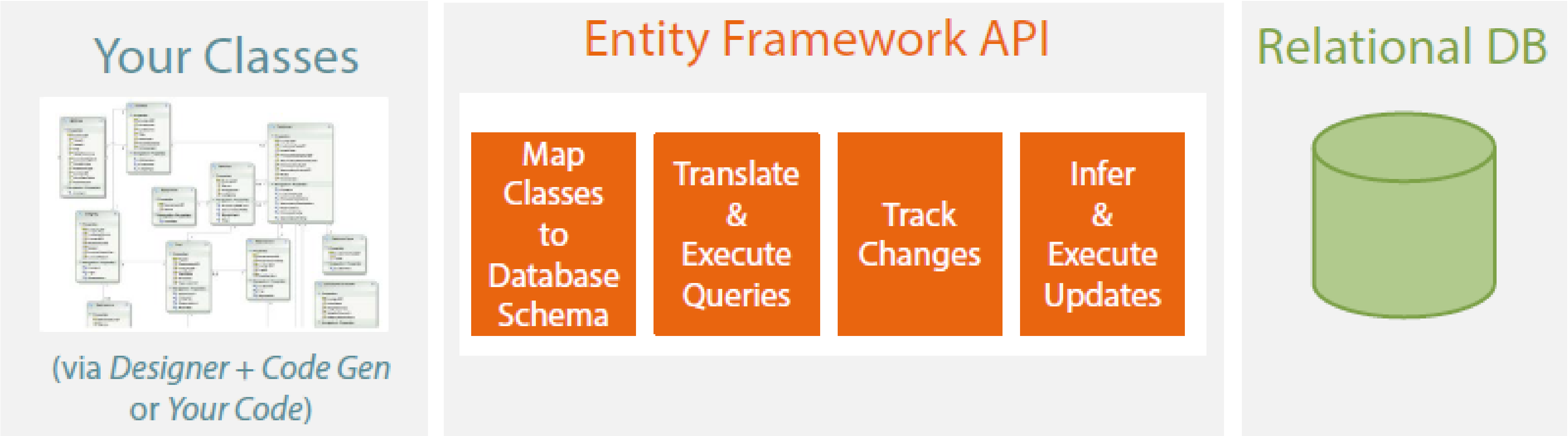
Developer Productivity

~~First Class Member of Microsoft .NET Stack~~

Consistent query syntax with LINQ to Entities

Focus on domain. Not on DB, connections, commands, etc.

How EF Works



Your Classes

(via *Designer* + *Code Gen* or *Your Code*)

Entity Framework API

Map Classes to Database Schema

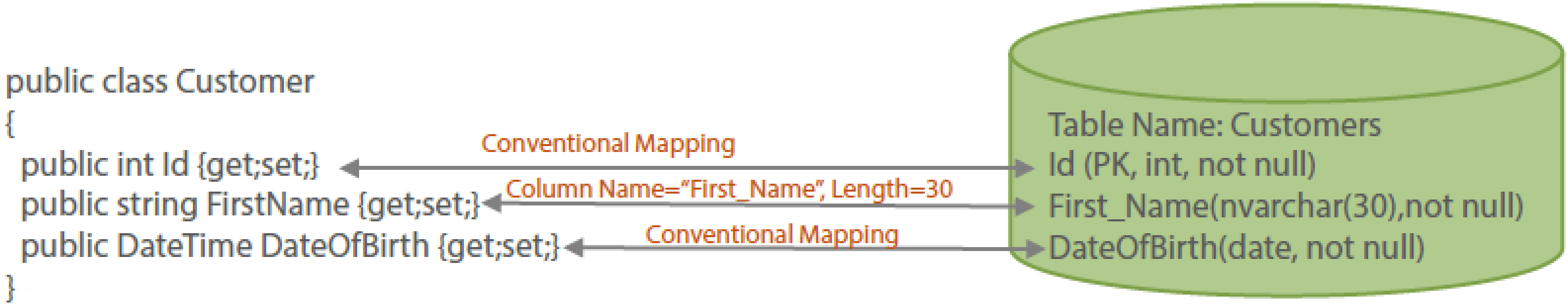
Translate & Execute Queries

Track Changes

Infer & Execute Updates

Relational DB

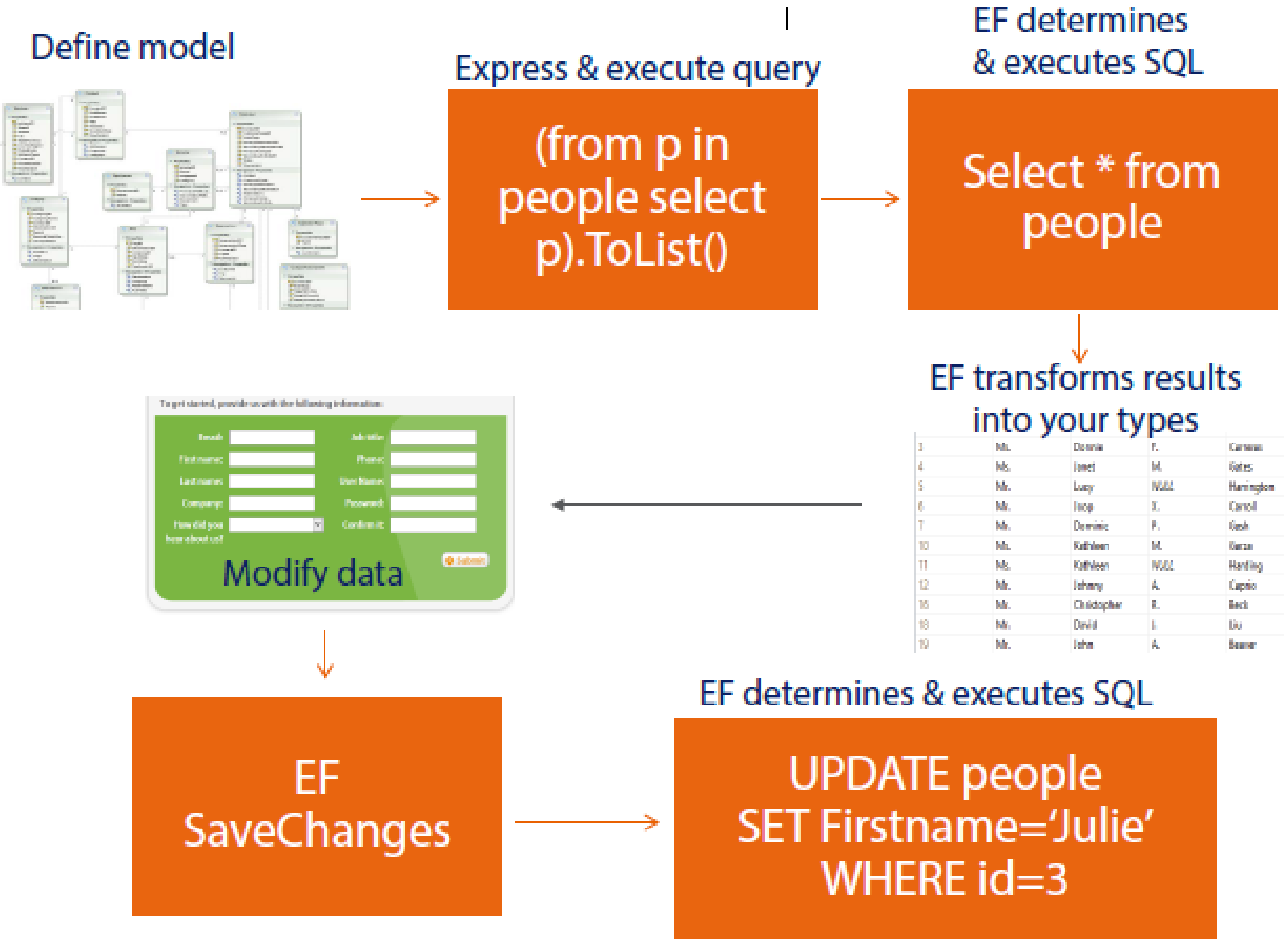
Mappings



```
public class Customer
{
  public int Id {get;set;}
  public string FirstName {get;set;}
  public DateTime DateOfBirth {get;set;}
}
```

Table Name: Customers
Id (PK, int, not null)
First_Name(nvarchar(30),not null)
DateOfBirth(date, not null)

Basic Workflow



Define model

Express & execute query

(from p in people select p).ToList()

EF determines & executes SQL

Select * from people

EF transforms results into your types

3	Ms.	Donnie	F.	Carreras
4	Ms.	Janet	M.	Gates
5	Mr.	Lucy	NULL	Harrington
6	Mr.	[illegible]	J.	Carroll
7	Mr.	Dominic	P.	Gash
10	Ms.	Kathleen	M.	Garza
11	Ms.	Kathleen	NULL	Harding
12	Mr.	Johnny	A.	Caprio
16	Mr.	Christopher	R.	Beck
18	Mr.	David	J.	Liu
19	Mr.	John	A.	Beaver

To get started, provide us with the following information:

Email:
First name:
Last name:
Company:
How did you hear about us?
Job title:
Phone:
User Name:
Password:
Confirm it:

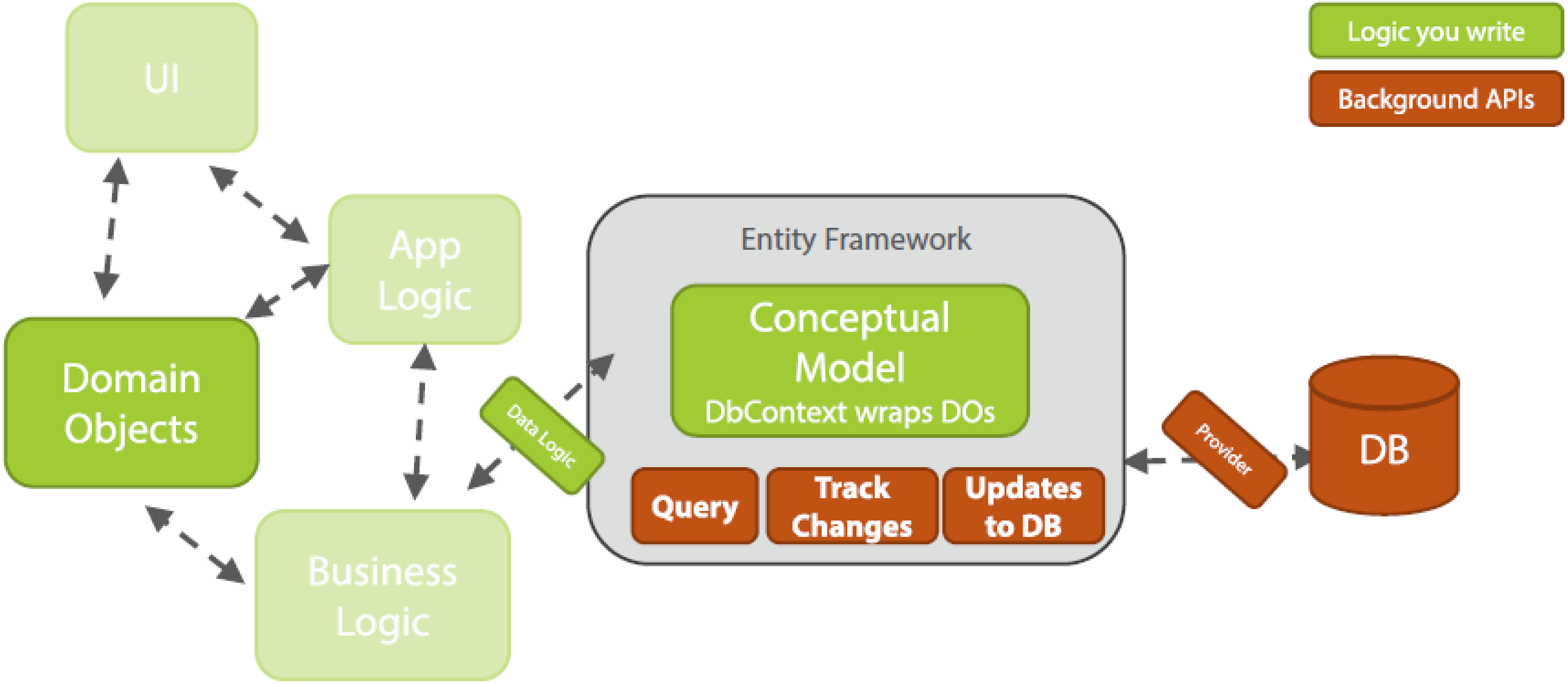
Modify data

EF SaveChanges

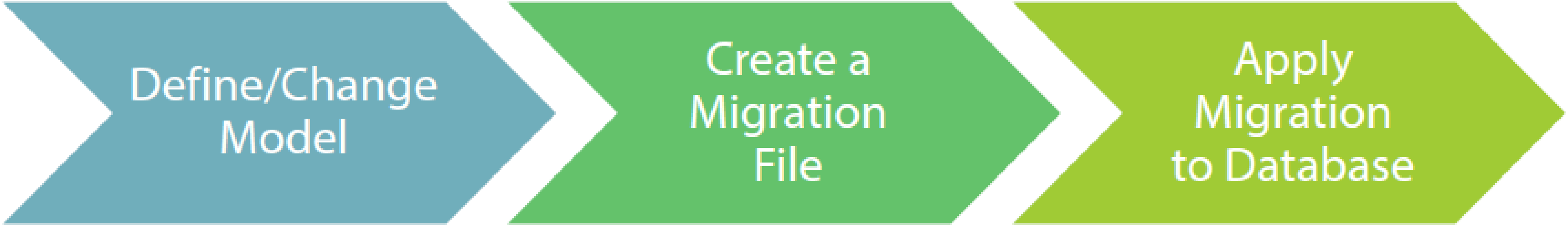
EF determines & executes SQL

UPDATE people SET Firstname='Julie' WHERE id=3

EF in Your Software Architecture

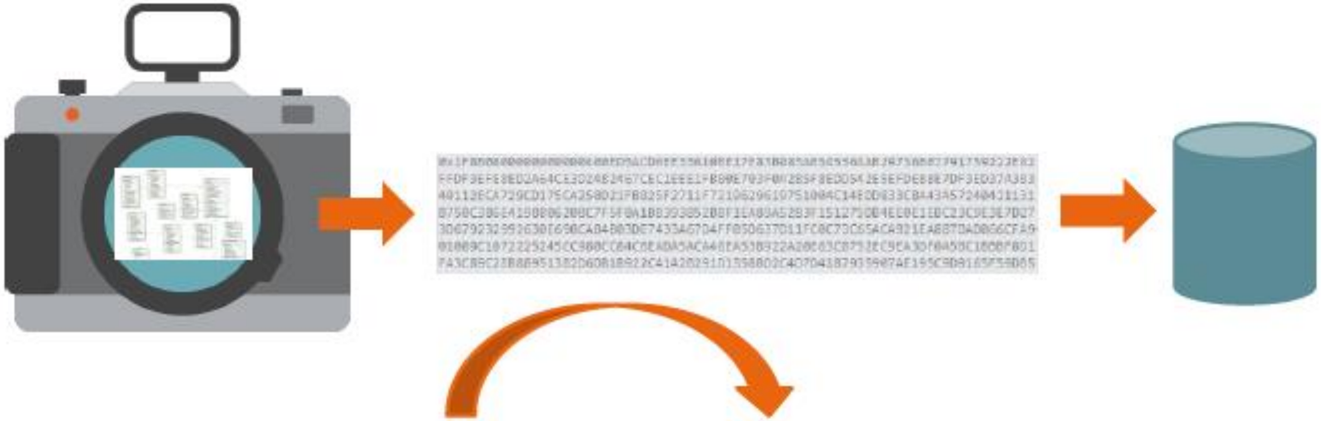


Code First Database Migrations

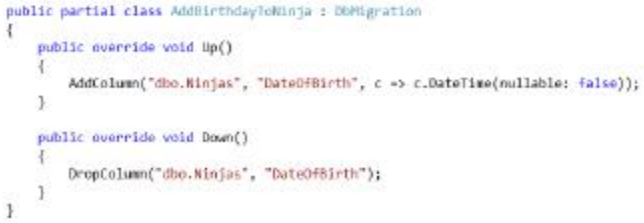


Determining Migrations

“Update-Database”

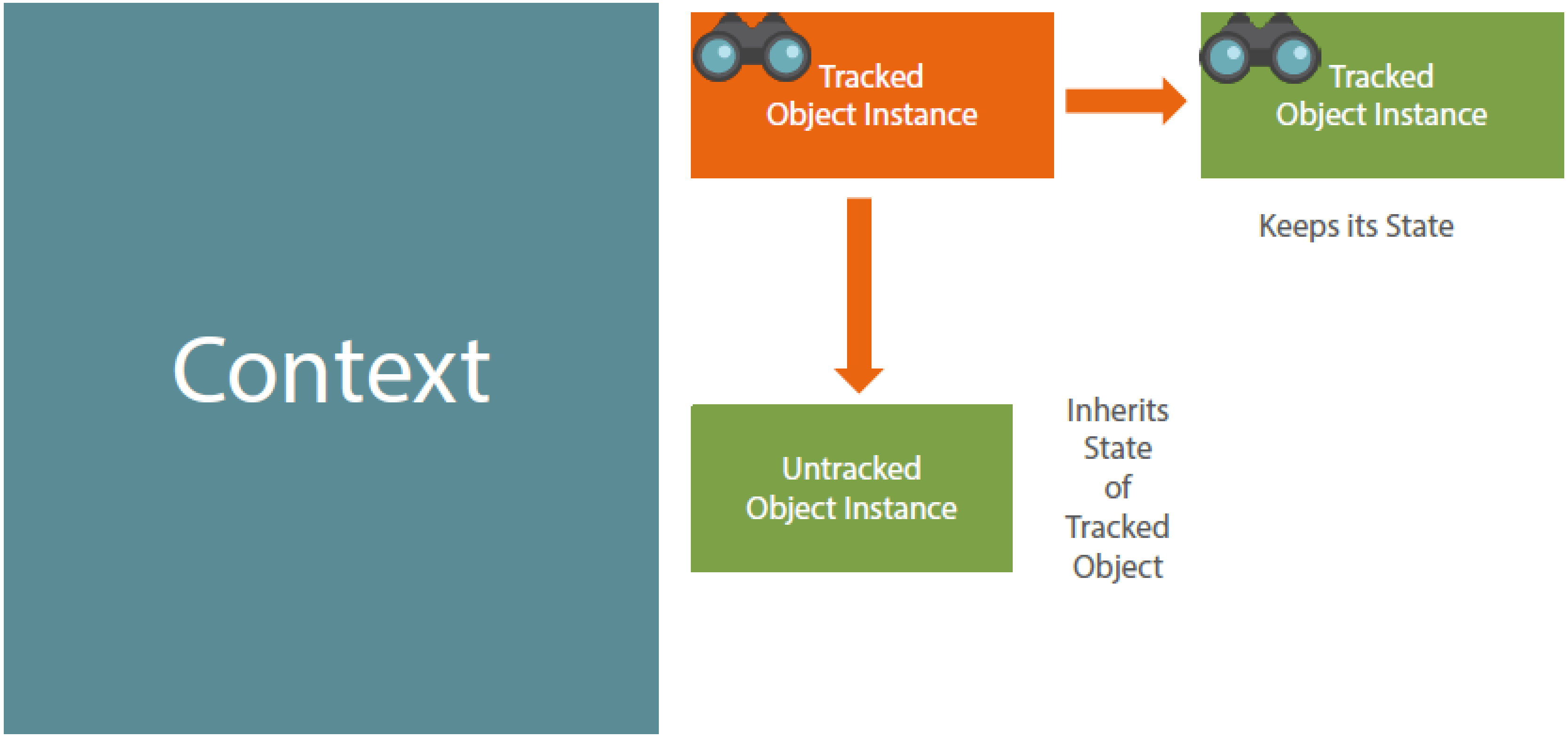


“Add-Migration”



```
public partial class AddBirthdayToNinja : DbMigration
{
    public override void Up()
    {
        AddColumn("dbo.Ninjas", "DateOfBirth", c => c.DateTime(nullable: false));
    }

    public override void Down()
    {
        DropColumn("dbo.Ninjas", "DateOfBirth");
    }
}
```

Tracking



Context

Tracked Object Instance

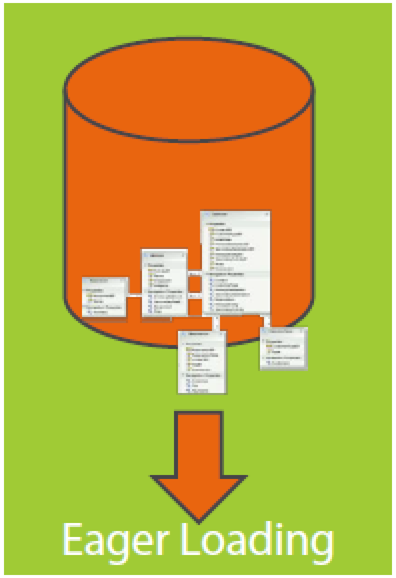
Tracked Object Instance

Keeps its State

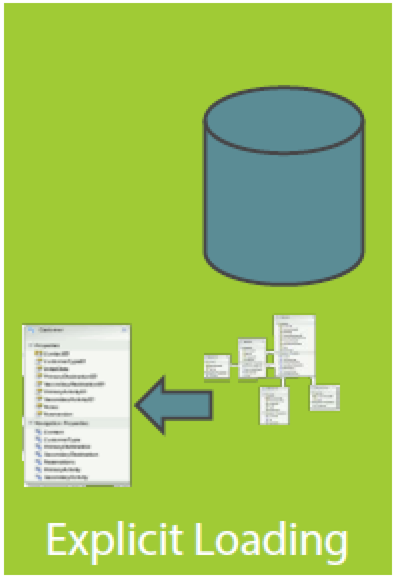
Untracked Object Instance

Inherits State of Tracked Object

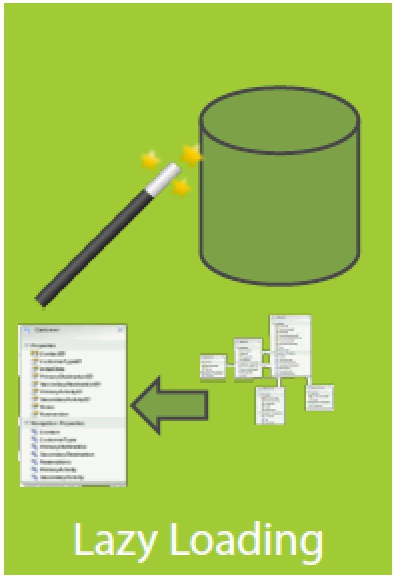
Loading Related Data



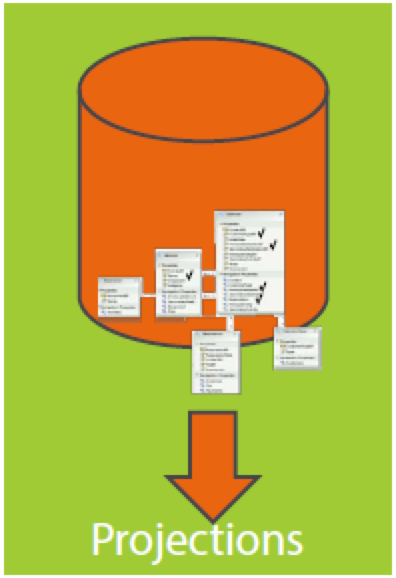
Eager Loading



Explicit Loading

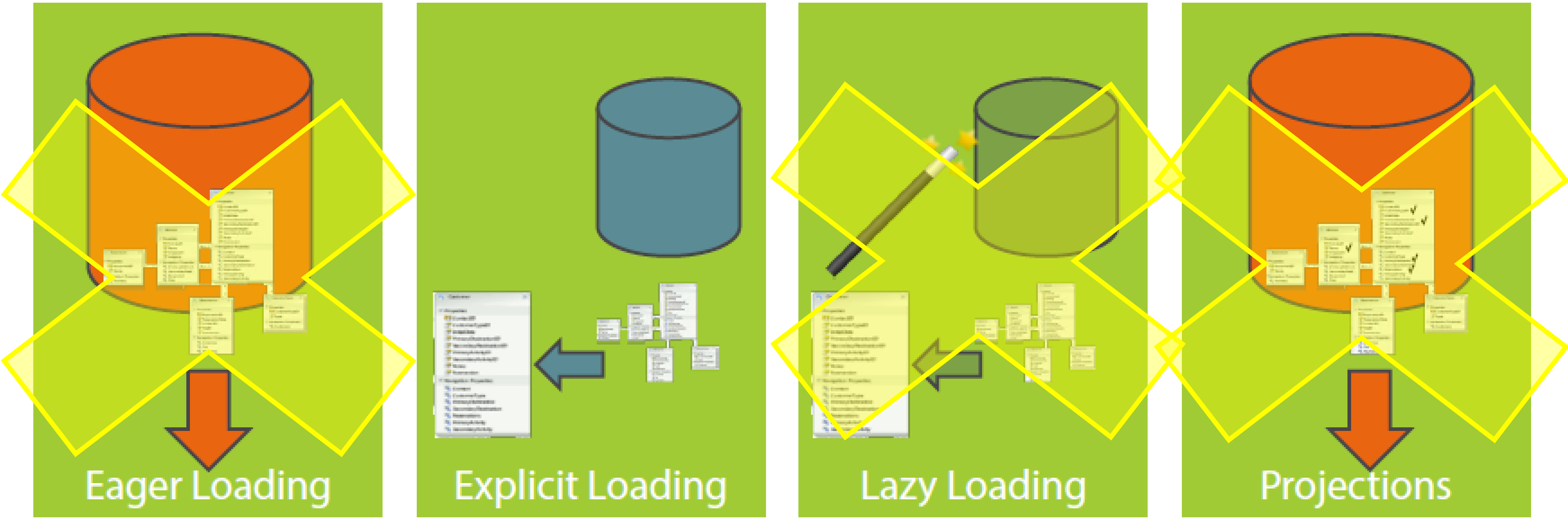


Lazy Loading



Projections

(Core) Loading Related Data


Eager Loading
Explicit Loading
Lazy Loading
Projections