Unidad 2

Fundamentos de los Lenguajes

Capitulo 3

Sintaxis Avanzada

Contenidos

- Colecciones
- Generics
- LINQ
- Local Variable Type Inference
- Object Initialisers
- Anonymous Types
- Lambda Expressions
- Extension Methods
- Query Expressions
- Eventos, Handlers de Eventos, Delegados

Colecciones

- System.Collections namespace
  - Una colección puede contener un número de items sin necesidad de especificar este número previamente.
  - Los elementos de una colección no tienen que ser necesariamente del mismo tipo.
  - La posición de un objeto en la colección puede cambiar en base a un cambio en la colección, debido a esto la posición de un objeto en la colección puede variar.
  - Es muy fácil agregar ítems a una colección pero al consultarlos normalmente nos vemos obligados a hacer un cast desde Object al tipo en particular.

Colecciones mas utilizadas

Clase	Descripción
ArrayList	• Colección de proposito general • Colección lineal de objetos
BitArray	• Colección de boolean • Util para operaciones entre bits (AND, NOT, and XOR)
Hashtable	• Diccionario de proposito general. • Guarda pares de key/value
Queue	• FIFO
SortedList	• Diccionario ordenado por key. • Se pueden obtener items por key o indice.
Stack	• LIFO

Usando colecciones

- Agregar objetos

```
Auto auto1 = new Auto(“Volkswagen", “Gol");
ArrayList vehiculos = new ArrayList();
vehiculos.Add(auto1);
```

- Recuperar por indice

```
Auto primerAuto = (Auto) vehiculos[0];
```

- Iterar usando un foreach

```
foreach(Auto a in vehiculos)
{
  // Console.WriteLine(a.Marca);
}
```

Usando un Diccionario

- Incluir key y valor cuando se agrega un nuevo item a la colección

```
Hashtable ingredientes = new Hashtable();
ingredientes.Add("Café Moca", "Cafe, Leche, Chocolate");
```

- Recuperar por key

```
string recetaCafeMoca = ingredientes["Café Moca"];
```

- Iterar en la colección de keys

```
foreach(string key in ingredientes.Keys)
{
  Console.WriteLine(ingredientes[key]);
}
```

Generics

- Permiten personalizar un método, clase, estructura o interfaz para el tipo de datos preciso sobre el que actúa.
- Ejemplo: en lugar de utilizar la clase ArrayList, que permite que los objetos sean de cualquier tipo, puede usar la clase genérica List<T> y especificar el tipo permitido.

```
List<int> listaInt = new List<int>();

listaInt.Add(1);
listaInt.Add(2);
listaInt.Add("Tres");   //Error de compilación

int i = listaInt[0];      //No requiere cast
```

Generics

- Proporciona código independiente de la clase
  - Elegante, productivo y óptimo
- Utilizado en la BCL
  - Colecciones genéricas, tipos nullables, etc.

```
// Sin Generics
Collection emps = new Collection();

emps.Add(new Employee("1", "Ap1"));
emps.Add(new Employee("2", "Ap2"));

foreach(object emp in emps)
{
  Employee employee = (Employee)emp;
  Console.WriteLine(employee.Nombre);
  Console.WriteLine(employee.Apellido);
}
```

```
// Con Generics
Collection<Employee> emps = new
Collection<Employee>();

emps.Add(new Employee("1", "Ap1"));
emps.Add(new Employee("2", "Ap2"));

foreach(Employee employee in emps)
{
  Console.WriteLine(employee.Nombre);
  Console.WriteLine(employee.Apellido);
}
```

Language INtegrated Query (LINQ)

- Considerando que usamos distintos tipos de forma de consultar información según el tipo de la misma:
  - SQL, XQuery/XPath, foreach, etc.
- Tal vez se pueda mejorar la productividad si…
  - Elegimos una sola forma/lenguaje de consulta
  - Habilitamos al compilador a chequear las consultas y los resultados
  - Permitir que este lenguaje sea extensible a cualquier tipo de datos

Que es LINQ?

- Language Integrated Query
- Integra consultas a C#
- Desde .NET Framework 3.5

Consultas sin LINQ

- Usando loops y condiciones

```
foreach(Cliente c in clientes)
   if (c.Provincia == “Santa Fe")
```

- Bases de datos usando SQL

```
SELECT * FROM Clientes WHERE Provincia=‘Santa Fe'
```

- XML usando XPath/XQuery

```
//Clientes/Cliente[@Provincia=‘Santa Fe']
```

Consultas con LINQ

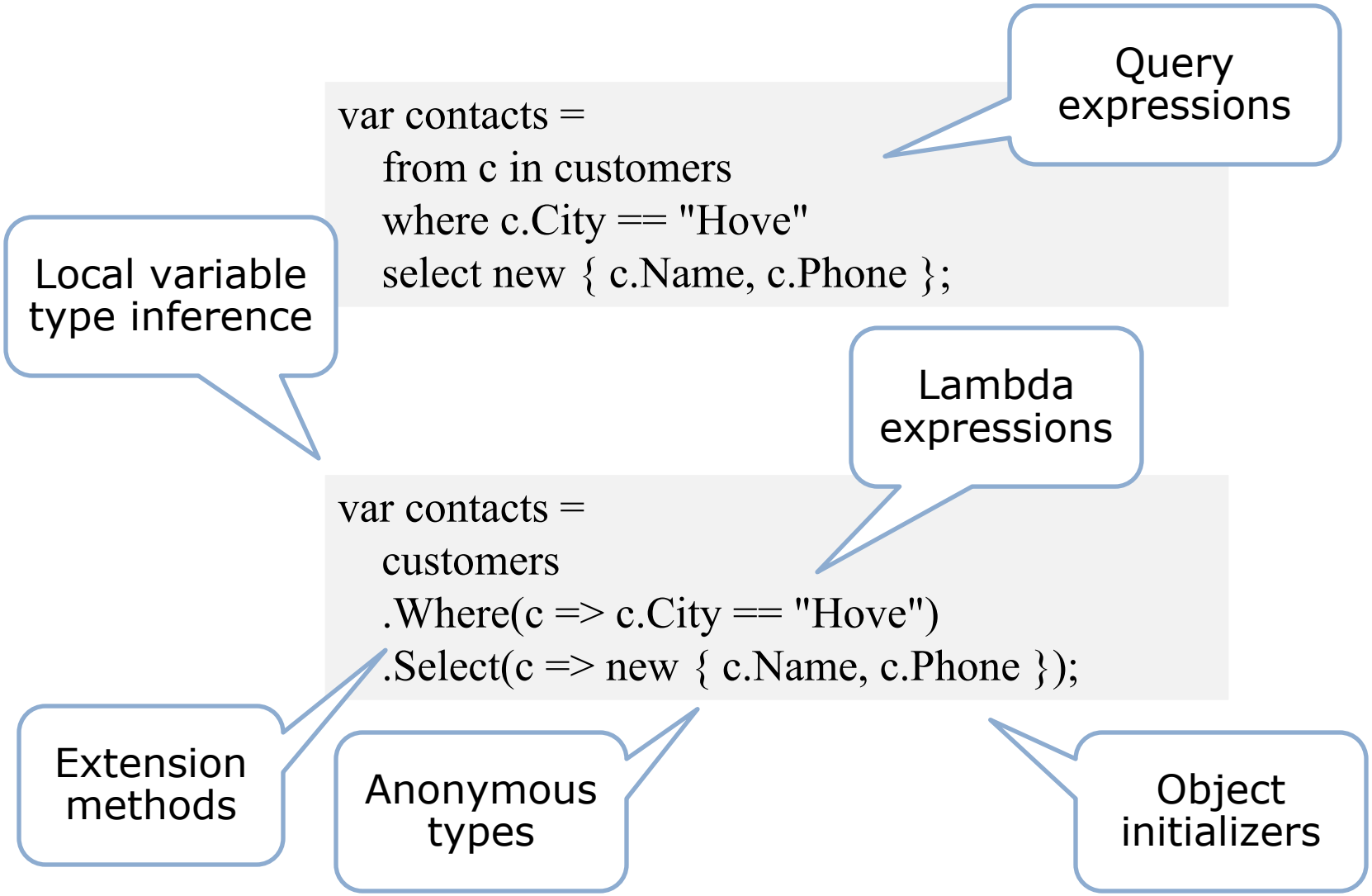
```
var misClientes = from c in Clientes
    where c.Provincia == “Santa Fe"
    select c;
```

```
var mejoresClientes = (from c in db.Clientes
    where c.CodigoPostal.StartsWith(“S")
    orderby c.Ventas descending
    select c).Skip(10).Take(10);
```

Ventajas

- Acceso a datos unificado, una sola sintaxis para aprender y recordar.
- Tipado. Detecta errores al compilar
- IntelliSense.
- Los resultados pueden ser bindeados

Caracteristicas de C# utilizadas



Local Variable Type Inference

```
int i = 666;
string s = "Goodbye";
double d = 3.14;
int[] numbers = new int[] {1, 2, 3};
Dictionary<int,Order> orders = new Dictionary<int,Order>();
```

```
var i = 666;
var s = "Goodbye";
var d = 3.14;
var numbers = new int[] {1, 2, 3};
var orders = new Dictionary<int,Order>();
```

El tipo se define en el lado derecho

Object Initialisers

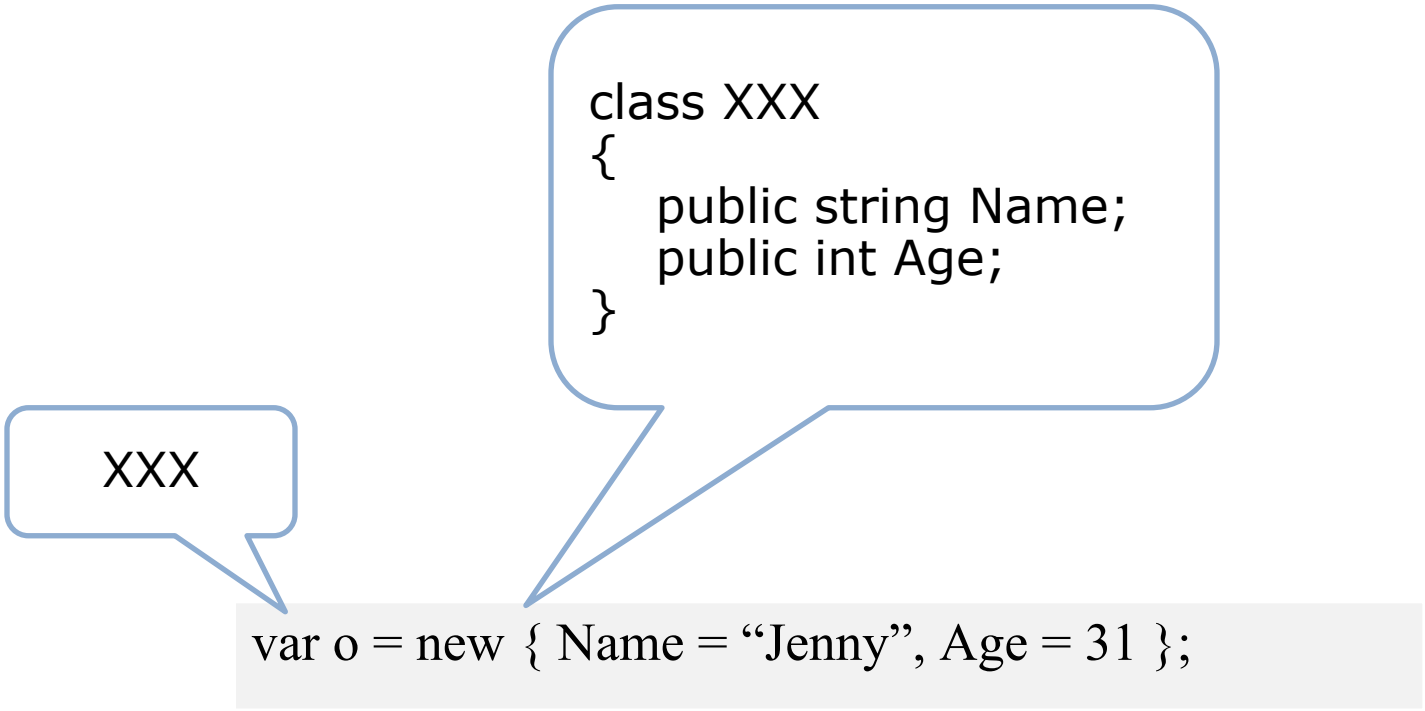
```
public class Point
{
    private int x, y;

    public int X { get { return x; } set { x = value; } }
    public int Y { get { return y; } set { y = value; } }
}
```

```
Point a = new Point { X = 0, Y = 1 };
```

Asignacion a campos o propiedades

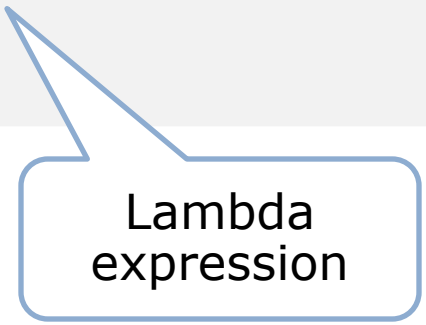
```
Point a = new Point();
a.X = 0;
a.Y = 1;
```

Anonymous Types



Lambda Expressions

```
public class MyClass
{
    public static void Main() {
        List<Customer> customers = GetCustomerList();
        List<Customer> locals =
            customers.FindAll(c => c.City == "Hove");
    }
}
```



Extension Methods

```
namespace MyStuff
{
    public static class Extensions
    {
        public static string Concatenate(this IEnumerable<string> strings,
            string separator) {…}
    }
}
```

Extension method

```
using MyStuff;
```

Habilitar Extension

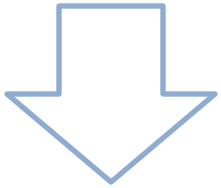
```
string[] names = new string[] { "Jenny", "Daniel", "Rita" };
string s = names.Concatenate(", ");
```

IntelliSense

Query Expressions

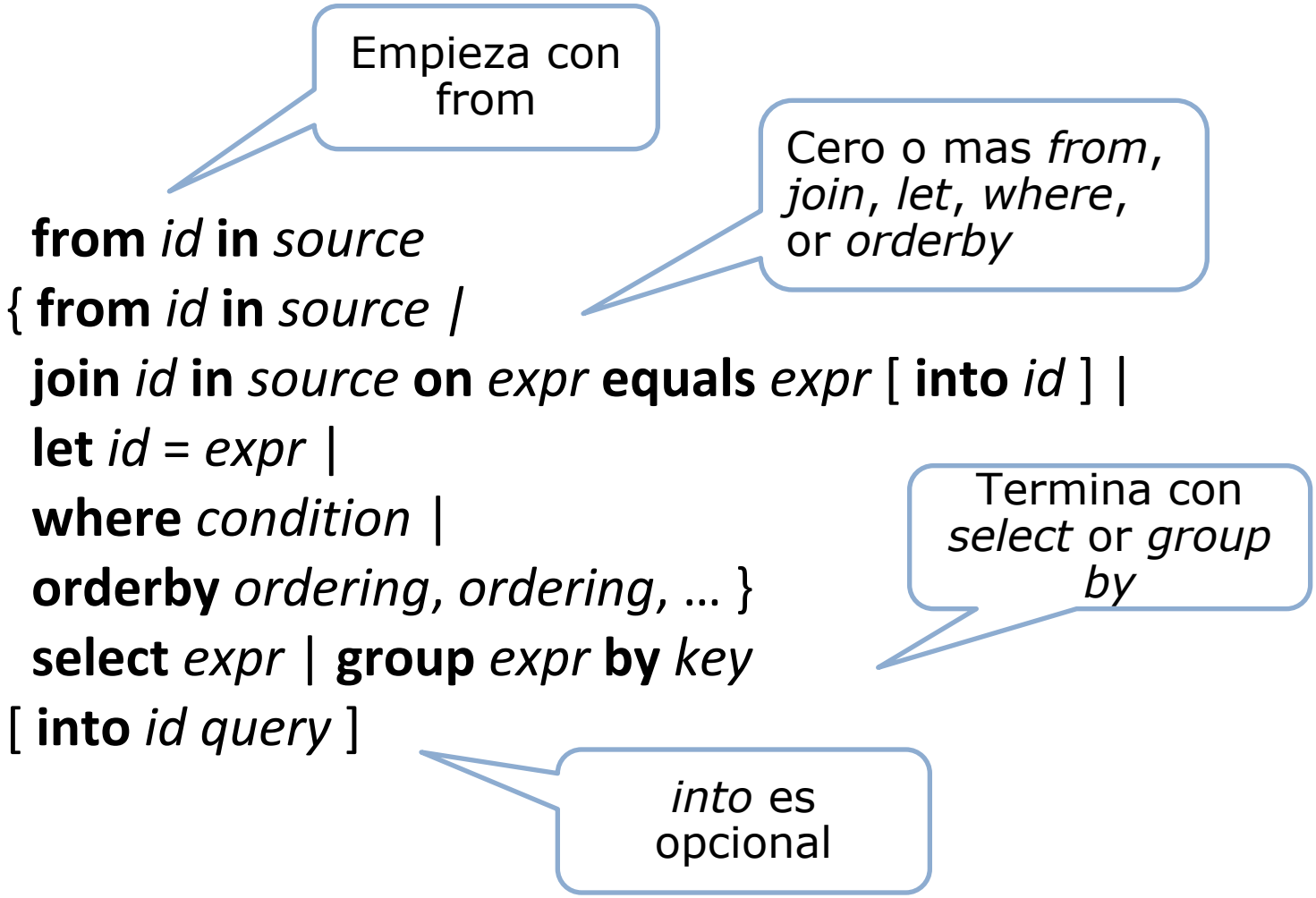
- Consultas expresadas con metodos
  - Where, Join, OrderBy, Select, GroupBy, …

```
from c in customers
where c.City == "Hove"
select new { c.Name, c.Phone };
```



```
customers
.Where(c => c.City == "Hove")
.Select(c => new { c.Name, c.Phone });
```

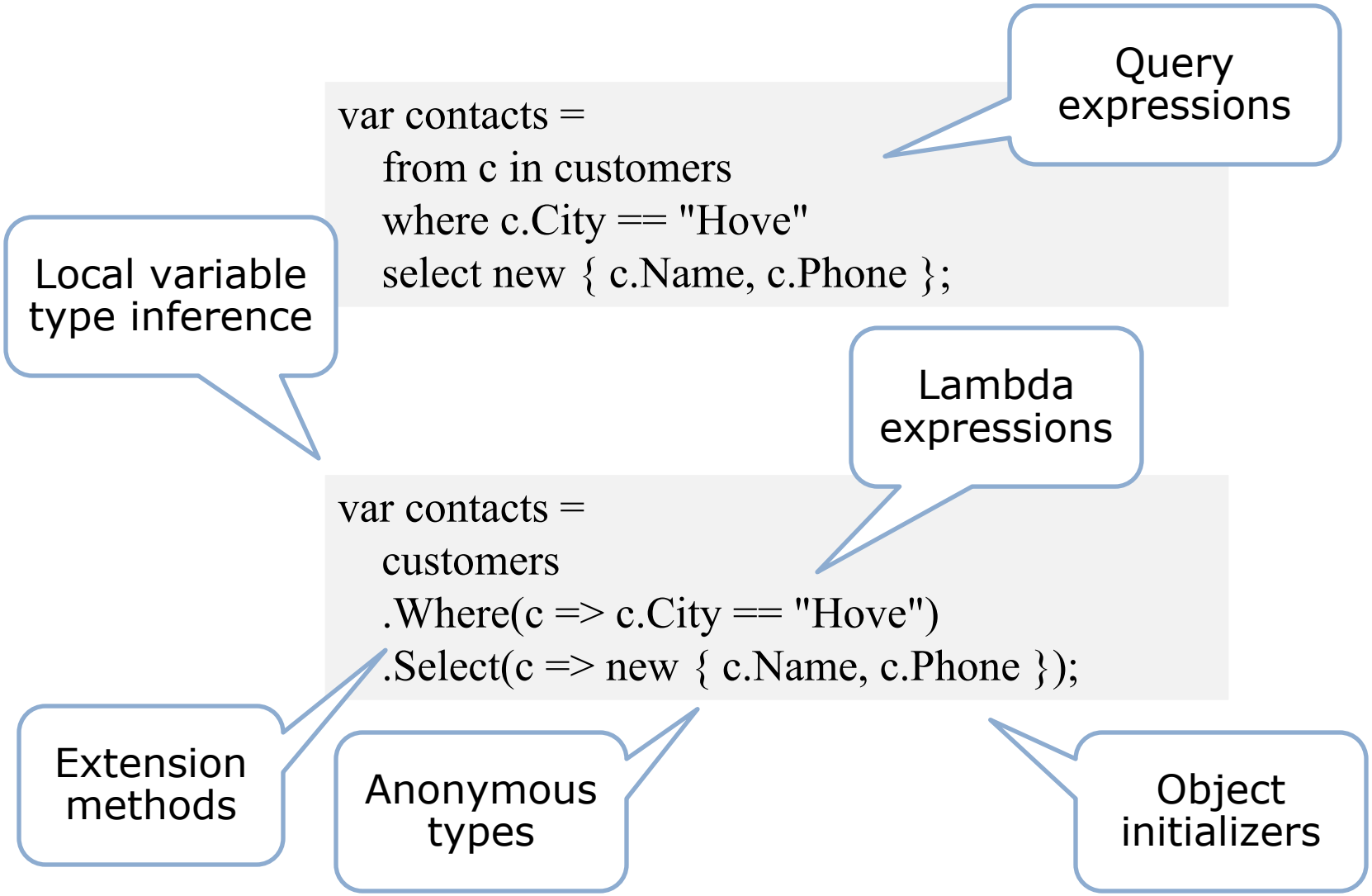
Query Expressions



Query Expressions

Project	**Select** *<expr>*
Filter	**Where** *<expr>*, **Distinct**
Test	**Any(***<expr>***)**, **All(***<expr>***)**
Join	*<expr>* **Join** *<expr>* **On** *<expr>* **Equals** *<expr>*
Group	**Group By** *<expr>*, *<expr>* **Into** *<expr>*, *<expr>* **Group Join** *<decl>* **On** *<expr>* **Equals** *<expr>* **Into** *<expr>*
Aggregate	**Count(***<expr>***)**, **Sum(***<expr>***)**, **Min(***<expr>***)**, **Max(***<expr>***)**, **Avg(***<expr>***)**
Partition	**Skip** [**While**] *<expr>*, **Take** [**While**] *<expr>*
Set	**Union**, **Intersect**, **Except**
Order	**Order By** *<expr>*, *<expr>* [**Ascending** | **Descending**]

Funcionando en conjunto para habilitar LINQ



Demostracion

Demostración 1

LINQ

Kahoot

Laboratorios