Advanced C# type construction, LINQ & delegates

- Delegates
- Events
- Pointers in C#
- Operator overload
- LINQ
- Lambda operators
- Extension methods
- Partial methods
- Type conversions (explicit – implicit)
- Anonymous Types

Delegates

.NET delegates

- From plain C
    - function pointers
        - worked on raw address in memory
    - Problems : crash !!
- in .NET callbacks
    - are safe
    - Called: delegates

Informations for the delegates

- The name of the method to be called
- arguments
- return values

- When the delegate is created it can invoke the method it represents

- Keyword: **delegate**

- Delegate is a sealed class
    - Inheriting from System.MulticastDelegate

What happen ??

```
...

public delegate void MyDelegate(int i);
...
```

- Transformed into a sealed class

```
...

sealed class MyDelegate: System.MulticastDelegate
{
   public MyDelegate(object target,uint
                                functionAddress);
   public void Invoke(int i);
   public IAsyncResult BeginInvoke( int i,
                    AsyncCallback cb, object state);
   public void EndInvoke(IAsyncResult result);
}
...
```

- 3 auto generated methods,(parameters and return values depending on the declaration of the delegate

- Creates a type inheriting from System.MulticastDelegate

Delegates can also "point to" methods that contain any number of out or ref parameters (as well as array parameters marked with the params keyword). For example, assume the following delegate type:

```
public delegate string MyOtherDelegate(out bool a, ref bool b, int c);
```

The signatures of the Invoke() and BeginInvoke() methods look as you would expect; however, check out the following EndInvoke() method, which now includes the set of all out/ref arguments defined by the delegate type:

```
public sealed class MyOtherDelegate : System.MulticastDelegate
{
  public string Invoke(out bool a, ref bool b, int c);
  public IAsyncResult BeginInvoke(out bool a, ref bool b, int c,
    AsyncCallback cb, object state);
  public string EndInvoke(out bool a, ref bool b, IAsyncResult result);
}
```

Table 10-1. *Select Members of System.MultcastDelegate/System.Delegate*

Member	Meaning in Life
Method	This property returns a System.Reflection.MethodInfo object that represents details of a static method maintained by the delegate.
Target	If the method to be called is defined at the object level (rather than a static method), Target returns an object that represents the method maintained by the delegate. If the value returned from Target equals null, the method to be called is a static member.
Combine()	This static method adds a method to the list maintained by the delegate. In C#, you trigger this method using the overloaded += operator as a shorthand notation.
GetInvocationList()	This method returns an array of System.Delegate objects, each representing a particular method that may be invoked.

Remove()
RemoveAll()
	These static methods remove a method (or all methods) from the delegate's invocation list. In C#, the Remove() method can be called indirectly using the overloaded -= operator.

- A delegate can point at several methods

Example:

```
using System;




public class UseDelegates
{
  private delegate void EnSimpelDelegat( string msg);

  public static void Udskriv( string msg)

  {
       Console.WriteLine("Udskriv siger {0}",msg);

  }



  public static void Snyd( string msg)

  {
       Console.WriteLine("Snydt - hva??");

  }




  public static void Main()

  {

     EnSimpelDelegat esd;
     esd = new EnSimpelDelegat(Udskriv);
     esd("via delegaten.."); // Invoke is called


     Console.WriteLine("Metode: {0}",esd.Method);
     Console.WriteLine("Target: {0}",esd.ToString());

     esd = new EnSimpelDelegat(Snyd);
     esd("via delegaten..");  // Invoke is called

     Console.WriteLine("Metode: {0}",esd.Method);
```

```
      Console.WriteLine("Target: {0}",esd.ToString());



      Console.ReadLine();
   }





}
```

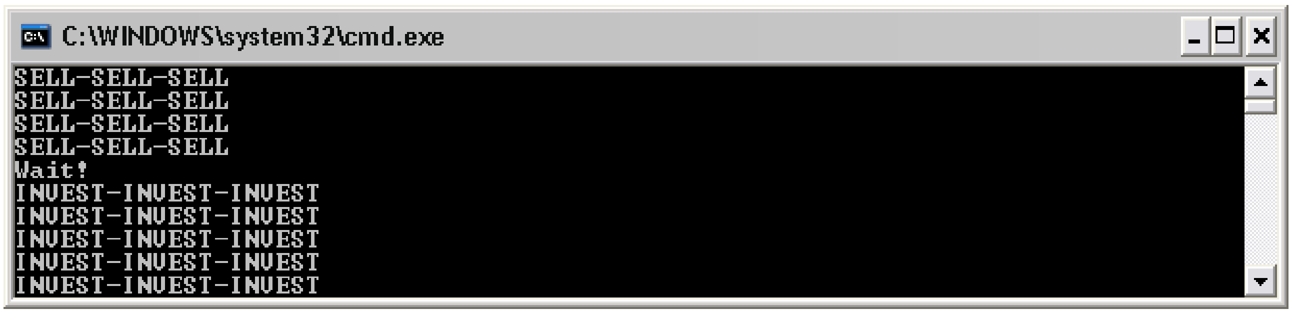
Output:

```
Udskriv siger via delegaten..
Metode: Void Udskriv(System.String)
Target: UseDelegates+EnSimpelDelegat
Snydt - hva??
Metode: Void Snyd(System.String)
Target: UseDelegates+EnSimpelDelegat
```

- Notice that the delegate object does not care about the name of the method – must have correct parameters

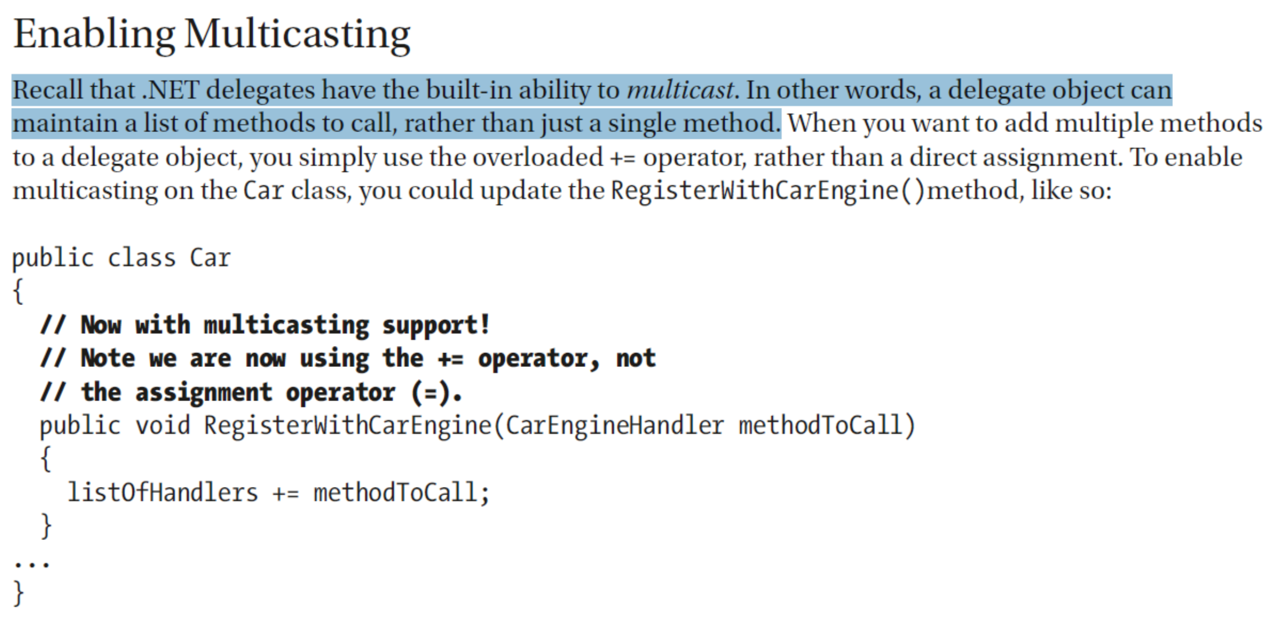
Object notification using delegates

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace DelegatePlay
{
    public class Economy
    {
        private const double interestRate = 2.00;

        public delegate void InterestRateHandler(String interestMessage);

        private InterestRateHandler IRHandler;

        public void RegisterInterest(InterestRateHandler metode)
        {
            IRHandler = metode;
        }


        public void SetNewRate(double irate)
        {
            if (irate > interestRate)
                IRHandler("INVEST-INVEST-INVEST");
            if(irate < interestRate)
                IRHandler("SELL-SELL-SELL");
            if (irate == interestRate)
                IRHandler("Wait!");


        }



    }
    public class Program
    {
        static void Main(string[] args)
        {

            Economy econ = new Economy();
            econ.RegisterInterest(new Economy.InterestRateHandler(OnEconomyChange));

            for (double i = 0.0; i < 5.00; i = i + 0.50)

                econ.SetNewRate(i);

        }

        public static void OnEconomyChange(string msg)
        {
            Console.WriteLine(msg);
        }
    }
}
```


C:\WINDOWS\system32\cmd.exe

```
SELL-SELL-SELL
SELL-SELL-SELL
SELL-SELL-SELL
SELL-SELL-SELL
Wait!
INVEST-INVEST-INVEST
INVEST-INVEST-INVEST
INVEST-INVEST-INVEST
INVEST-INVEST-INVEST
INVEST-INVEST-INVEST
```

Multicast delegates



Enabling Multicasting

Recall that .NET delegates have the built-in ability to *multicast*. In other words, a delegate object can maintain a list of methods to call, rather than just a single method. When you want to add multiple methods to a delegate object, you simply use the overloaded += operator, rather than a direct assignment. To enable multicasting on the Car class, you could update the RegisterWithCarEngine()method, like so:

```
public class Car
{
  // Now with multicasting support!
  // Note we are now using the += operator, not
  // the assignment operator (=).
  public void RegisterWithCarEngine(CarEngineHandler methodToCall)
  {
    listOfHandlers += methodToCall;
  }
...
}
```

```
...

 MulticastDelegate d = wash + rotate;
 // Send the new delegate into the ProcessCars() method.
 g.ProcessCars((Car.CarDelegate)d);
...
```

- Instead of the + operator can **Delegate.Combine(wash, rotate)** be used

- Removing a method:

```
...
 MulticastDelegate d = wash + rotate;
...
 Delegate washOnly = MulticastDelegate.Remove(d,rotate);
...
 g.ProcessCars((Car.CarDelegate)washOnly);
...
```

- List of invoctive methods:

```
...
foreach(Delegate d in proc.GetInvocationList())
{
   Console.WriteLine("***** Calling: {0} *****",

          d.Method.ToString());
}
...
```

Example 2: Multicast delegates

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace MulicastDelegates
{


    public class FabriksChef
    {
        public delegate void ProduktionsLinie(int antal);

    }

    public class Fabrik
    {
        public static void StartLinie1(int antal)
        {
            // Producer et antal varer på bånd 1
            Console.WriteLine("Linie 1, antal :" + antal);
        }

        public static void StartLinie2(int antal)
        {
            // Producer et antal varer på bånd 2
            Console.WriteLine("Linie 2, antal :" + antal);
        }

        public static void StartLinie3(int antal)
        {
            // Producer et antal varer på bånd 3
            Console.WriteLine("Linie 3, antal :" + antal);
        }

    }



    public class MultiDelegatBruger
    {
        private static FabriksChef.ProduktionsLinie
                                    linie1, linie2, linie3;
        private static FabriksChef.ProduktionsLinie
                                    planlægger;


        public static void Main()
        {
            linie1 = new FabriksChef.ProduktionsLinie(
                                    Fabrik.StartLinie1);
            linie2 = new FabriksChef.ProduktionsLinie(
                                    Fabrik.StartLinie2);
            linie3 = new FabriksChef.ProduktionsLinie(
                                    Fabrik.StartLinie3);
            // linie 1 opsættes
            Console.WriteLine("Linie 1 aktiveres med 300 enheder");
            planlægger += linie1;
            planlægger(300);
            //Det går strygende:
            //Start linie 2 og 3. Alle skal producere 400)
            Console.WriteLine("Linie 1,2,3 aktiveres med 400 enheder");
            planlægger += linie2 + linie3;
            planlægger(400);

            // økonomien i landet svigter - stop overproduktion
            Console.WriteLine("Alle frakobles");
            planlægger-=linie1;
            planlægger-=linie2;
            planlægger-=linie3;


            // økonomien kører på vågeblus - start linie 2 med 150
            Console.WriteLine("Linie 2 aktiveres med 150 enheder");
            planlægger += linie2;
            planlægger(150);
            Console.ReadLine();

        }


    }
}
```

Which methods works with callbacks?

- In C only callbacks to static members
    - In C# we can have callbacks to
        - static members
        - and object instances

Generic delegates

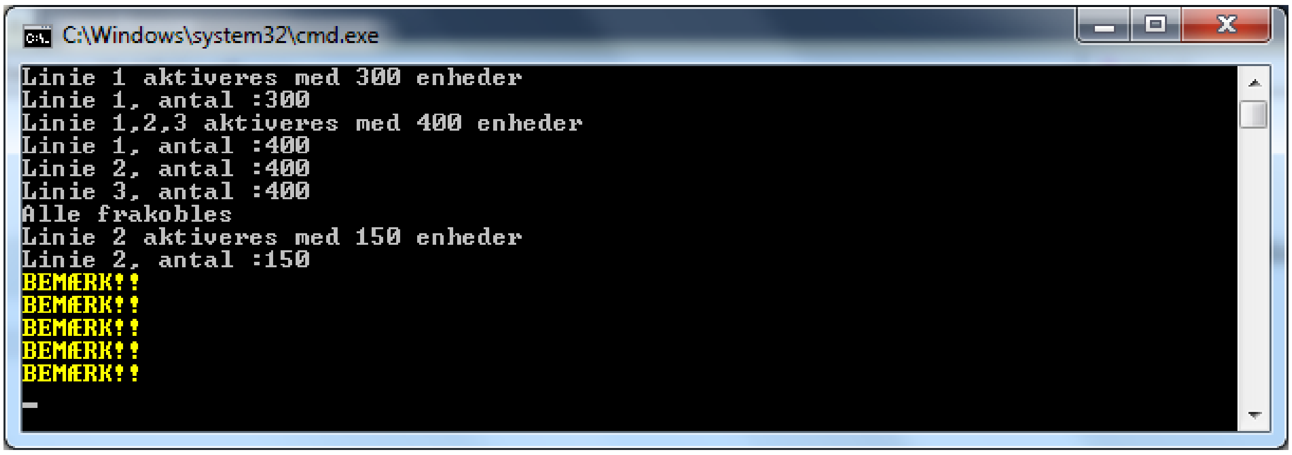
```
...

public delegate void MyGenericDelegate<T>(T arg);

...
static void Main()
{

 MyGenericDelegate<string> strTarget =
    new MyGenericDelegate<string>(StringTarget);

 strTarget("Some string data");

 MyGenericDelegate<int> intTarget = IntTarget;
 intTarget(9);

...
 static void StringTarget(string arg)
 {
   Console.WriteLine("arg in uppercase is: {0}",
                              arg.ToUpper());
 }

 static void IntTarget(int arg)
 {
   Console.WriteLine("++arg is: {0}", ++arg);
 }
...
```

Action<> and Func<> generic delegates

- Action<> can point at a method that returns void and take up to 16 arguments
- Action<> is a generic delegate

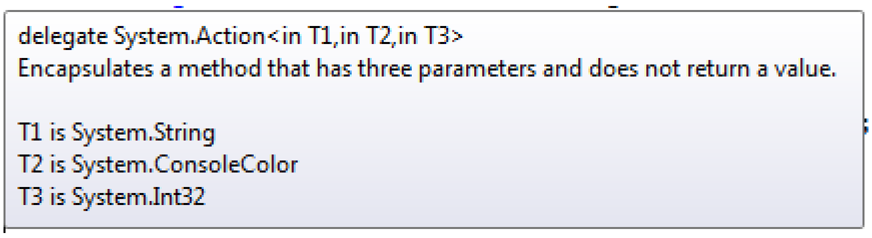
- Example

```
…
        static void DisplayMessage(string msg, ConsoleColor txtColor, int printCount)
        {
            // Ændrer tekstfarve og skriver en tekst på konsollen et antal gange.
            ConsoleColor previous = Console.ForegroundColor;
            Console.ForegroundColor = txtColor;
            for (int I = 0; I < printCount; i++)
            {
                Console.WriteLine(msg);
            }
            // Sætter tidligere farve tilbage
            Console.ForegroundColor = previous;
        }

…

// I Main
            Action<string, ConsoleColor, int> actionTarget =
                          new Action<string, ConsoleColor, int>(DisplayMessage);
            actionTarget(“BEMÆRK!!”, ConsoleColor.Yellow, 5);

            Console.ReadLine();

…
```



```
Linie 1 aktiveres med 300 enheder
Linie 1, antal :300
Linie 1,2,3 aktiveres med 400 enheder
Linie 1, antal :400
Linie 2, antal :400
Linie 3, antal :400
Alle frakobles
Linie 2 aktiveres med 150 enheder
Linie 2, antal :150
BEMÆRK!!
BEMÆRK!!
BEMÆRK!!
BEMÆRK!!
BEMÆRK!!
_
```

- Action<> returns void
- All parameters is **in**


delegate System.Action<in T1,in T2,in T3>
Encapsulates a method that has three parameters and does not return a value.

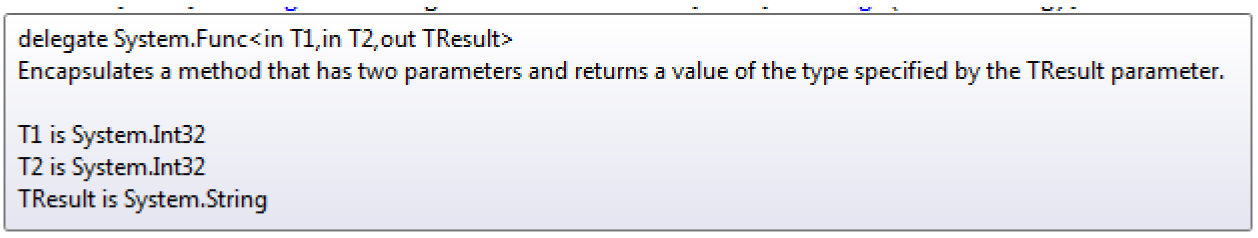
T1 is System.String
T2 is System.ConsoleColor
T3 is System.Int32

- Func<> **can return a value**
- Can also point up to 16 arguments

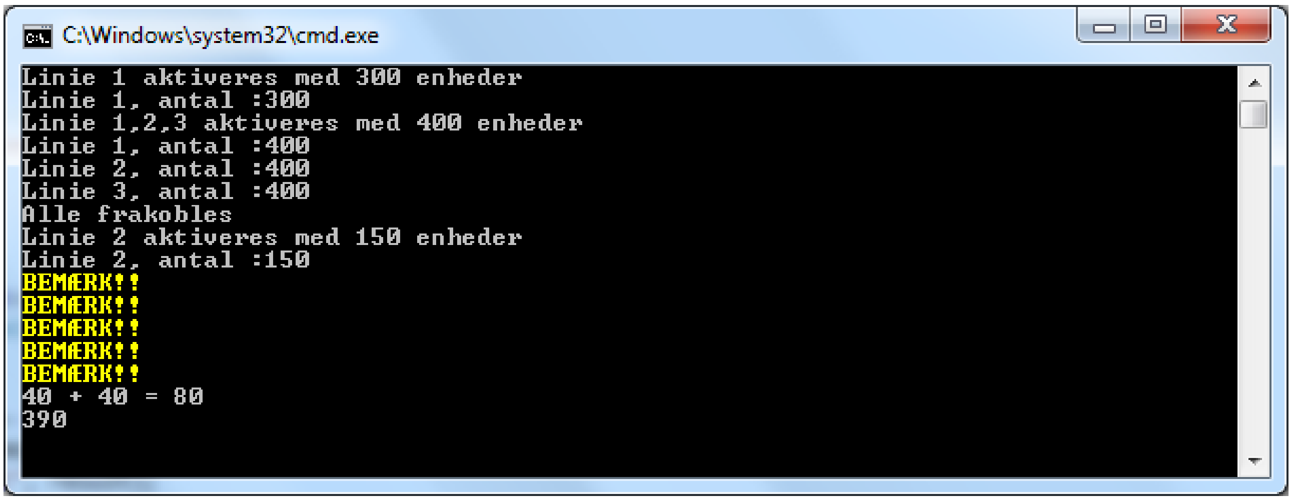
- Example

```
…

        static int Add(int x, int y)
        {
            return x + y;
        }
        static string SumToString(int x, int y)
        {
            return (x + y).ToString();
        }
…

// i Main
            Func<int, int, int> funcTarget = new Func<int, int, int>(Add);
            int result = funcTarget.Invoke(40, 40);
            Console.WriteLine("40 + 40 = {0}", result);

            Func<int, int, string> funcTarget2 = new Func<int, int, string>(SumToString);
            string sum = funcTarget2(90, 300);
            Console.WriteLine(sum);

            Console.ReadLine();

…
```

- Func<> the last parameter is out


delegate System.Func<in T1,in T2,out TResult>
Encapsulates a method that has two parameters and returns a value of the type specified by the TResult parameter.

T1 is System.Int32
T2 is System.Int32
TResult is System.String


C:\Windows\system32\cmd.exe

```
Linie 1 aktiveres med 300 enheder
Linie 1, antal :300
Linie 1,2,3 aktiveres med 400 enheder
Linie 1, antal :400
Linie 2, antal :400
Linie 3, antal :400
Alle frakobles
Linie 2 aktiveres med 150 enheder
Linie 2, antal :150
BEMÆRK!!
BEMÆRK!!
BEMÆRK!!
BEMÆRK!!
BEMÆRK!!
40 + 40 = 80
390
```

Events

- Events are often private
- An event is created in two steps:
    - First, a delegate is created – that points at a method
    - Second, an event is created

- Example (from the book)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace MulicastDelegates
{
    public class Car
    {
        // This delegate works in conjunction with the
        // Car's events.
        public delegate void CarEngineHandler(string msg);
          // This car can send these events.
         private event CarEngineHandler Exploded = new CarEngineHandler(MessageMethod);
         private event CarEngineHandler AboutToBlow = new CarEngineHandler(MessageMethod);
         bool carIsDead= false;
         int CurrentSpeed;
         int MaxSpeed =50;

         public void Accelerate(int delta)
         {
             // If the car is dead, fire Exploded event.
             if (carIsDead)
             {
                 if (Exploded != null)
                     Exploded("Sorry, this car is dead...");
```

```
            }
            else
            {
                CurrentSpeed += delta;
                // Almost dead?
                if (10 >= MaxSpeed - CurrentSpeed
                && AboutToBlow != null)
                {
                    AboutToBlow("Careful buddy! Gonna blow!");
                }
                // Still OK!
                if (CurrentSpeed >= MaxSpeed)
                    carIsDead = true;
                else
                    Console.WriteLine("CurrentSpeed = {0}", CurrentSpeed);
            }
        }

        public static void MessageMethod(string msg)
        {
            Console.WriteLine(msg);
        }
    }


}
```

- Kald til Accelerate

```
…

           Car car = new Car();

           for(int i = 0;i<60;i++)
              car.Accelerate(i);
           Console.ReadLine();

…
```

public ??

What if?

```
    // We can now assign to a whole new object...
    // confusing at best.
    myCar.listOfHandlers = new Car.CarEngineHandler(CallHereToo);
    myCar.Accelerate(10);

    // The caller can also directly invoke the delegate!
    myCar.listOfHandlers.Invoke("hee, hee, hee...");
    Console.ReadLine();
  }

  static void CallWhenExploded(string msg)
  { Console.WriteLine(msg); }

  static void CallHereToo(string msg)
```

Events

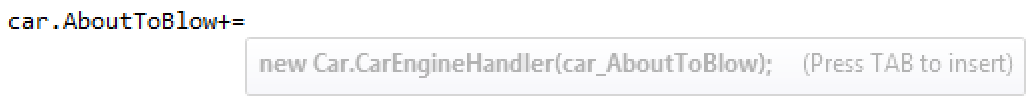
- Members are private!

The C# event Keyword

As a shortcut, so you don't have to build custom methods to add or remove methods to a delegate's invocation list, C# provides the event keyword. When the compiler processes the event keyword, you are automatically provided with registration and unregistration methods, as well as any necessary member variables for your delegate types. These delegate member variables are *always* declared private, and, therefore, they are not directly exposed from the object firing the event. To be sure, the event keyword can be used to simplify how a custom class sends out notifications to external objects.

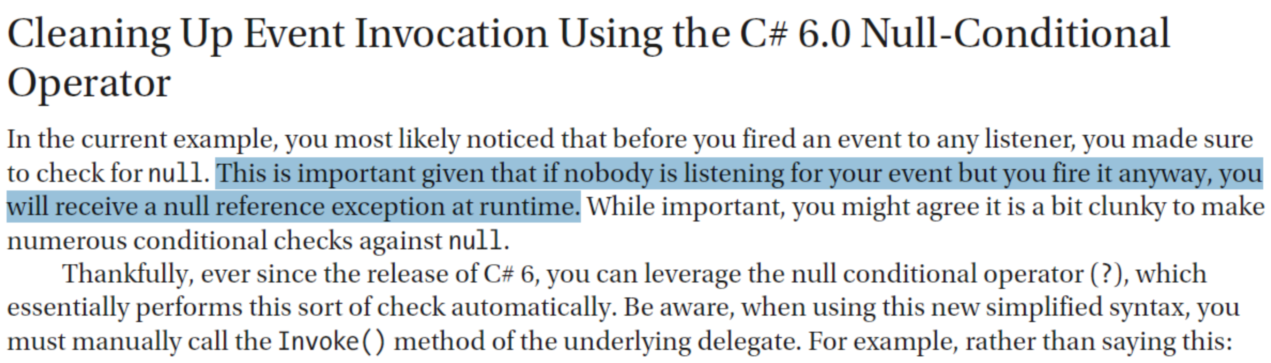
Defining an event is a two-step process. First, you need to define a delegate type (or reuse an existing one) that will hold the list of methods to be called when the event is fired. Next, you declare an event (using the C# event keyword) in terms of the related delegate type.

VS2019 can add events easily



```
car.AboutToBlow+=
                 new Car.CarEngineHandler(car_AboutToBlow);    (Press TAB to insert)
```

Cleaning up event notification

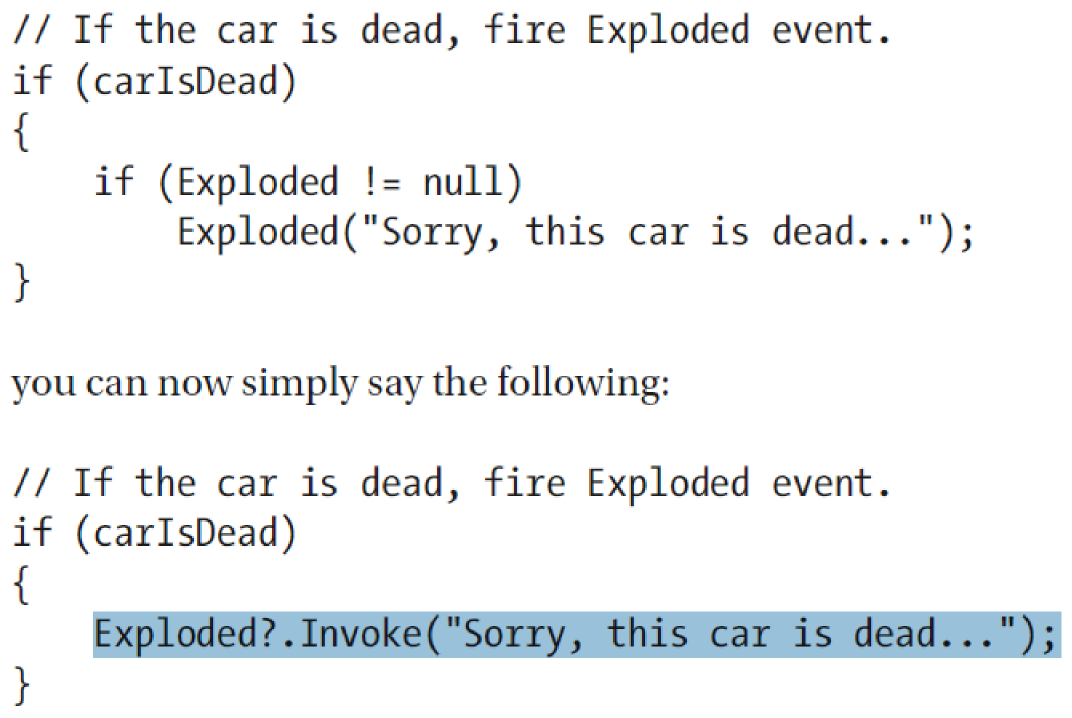


Cleaning Up Event Invocation Using the C# 6.0 Null-Conditional Operator

In the current example, you most likely noticed that before you fired an event to any listener, you made sure to check for `null`. This is important given that if nobody is listening for your event but you fire it anyway, you will receive a null reference exception at runtime. While important, you might agree it is a bit clunky to make numerous conditional checks against `null`.

Thankfully, ever since the release of C# 6, you can leverage the null conditional operator (`?`), which essentially performs this sort of check automatically. Be aware, when using this new simplified syntax, you must manually call the `Invoke()` method of the underlying delegate. For example, rather than saying this:

Solution:



```
// If the car is dead, fire Exploded event.
if (carIsDead)
{
    if (Exploded != null)
        Exploded("Sorry, this car is dead...");
}
```

you can now simply say the following:

```
// If the car is dead, fire Exploded event.
if (carIsDead)
{
    Exploded?.Invoke("Sorry, this car is dead...");
}
```

Microsoft Event pattern (the Microsoft way)

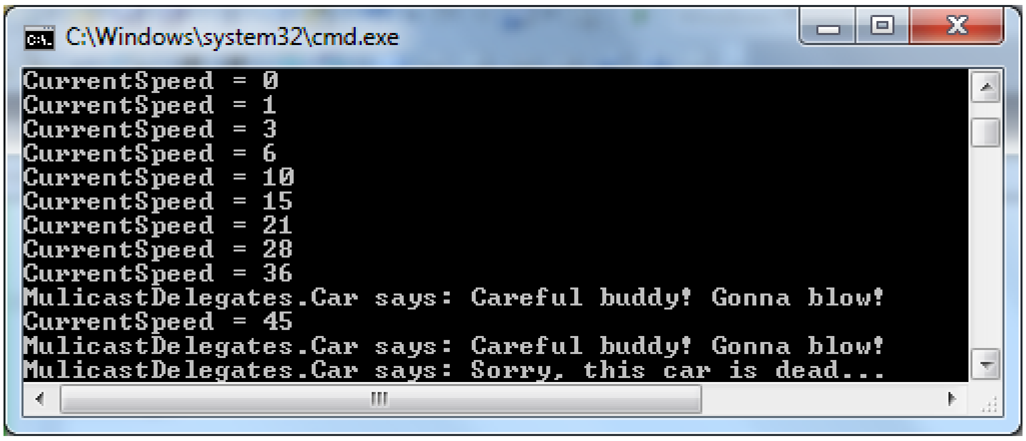
- Uses a class inheriting from EventArgs

- Example – the Microsoft way

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace MulicastDelegates
{
    public class Car
    {


        public delegate void CarEngineHandler(object sender, CarEventArgs e);

        public event CarEngineHandler Exploded;
        public event CarEngineHandler AboutToBlow;
         bool carIsDead= false;
         int CurrentSpeed;
         int MaxSpeed =50;

         public void Accelerate(int delta)
         {
             // If the car is dead, fire Exploded event.
             if (carIsDead)
             {
                 if (Exploded != null)
                     Exploded(this, new CarEventArgs("Sorry, this car is dead..."));
             }
             else
             {
                 CurrentSpeed += delta;
                 // Almost dead?
                 if (10 >= MaxSpeed - CurrentSpeed
                 && AboutToBlow != null)
                 {
                     AboutToBlow(this,new CarEventArgs("Careful buddy! Gonna blow!"));
                 }
                 // Still OK!
                 if (CurrentSpeed >= MaxSpeed)
                     carIsDead = true;
                 else
                     Console.WriteLine("CurrentSpeed = {0}", CurrentSpeed);
             }
         }


    }
    public class CarEventArgs : EventArgs
    {
        public readonly string msg;
        public CarEventArgs(string message)
        {
            msg = message;
        }
    }



}
```

- Now the caller is creating the events

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace MulicastDelegates
{



    public class MultiDelegatBruger
    {



        public static void Main()
        {


            Car car = new Car();

            car.AboutToBlow += CarAboutToBlow;
            car.Exploded += Exploded;

            for(int i = 0;i<60;i++)
               car.Accelerate(i);
            Console.ReadLine();




        }


        public static void CarAboutToBlow(object sender, CarEventArgs e)
        {
            Console.WriteLine("{0} says: {1}", sender, e.msg);
        }
        public static void Exploded(object sender, CarEventArgs e)
        {
            Console.WriteLine("{0} says: {1}", sender, e.msg);
        }
    }
}
```



```
C:\Windows\system32\cmd.exe
CurrentSpeed = 0
CurrentSpeed = 1
CurrentSpeed = 3
CurrentSpeed = 6
CurrentSpeed = 10
CurrentSpeed = 15
CurrentSpeed = 21
CurrentSpeed = 28
CurrentSpeed = 36
MulicastDelegates.Car says: Careful buddy! Gonna blow!
CurrentSpeed = 45
MulicastDelegates.Car says: Careful buddy! Gonna blow!
MulicastDelegates.Car says: Sorry, this car is dead...
```

Generic event handlers

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace MulicastDelegates
{
    public class Car
    {



         public event EventHandler<CarEventArgs> Exploded;
         public event EventHandler<CarEventArgs> AboutToBlow;
         bool carIsDead= false;
         int CurrentSpeed;
         int MaxSpeed =50;

…
```

```
…

           Car car = new Car();

            car.AboutToBlow += CarAboutToBlow;
            car.Exploded += Exploded;

            for(int i = 0;i<60;i++)
```

```
                car.Accelerate(i);

...
```

Asynchronous delegates

- In professionel application is very often multi threads
  - More actions on (nearly)the same time

- An async delegate starts a method in a thread
  - and reports when work is done
- .NET delegates supports threads

Example:

```
using System;
using System.Threading ;
using System.Windows.Forms ;

public class AsyncDemo
{
 string LongRunningMethod (int iCallTime, out int iExecThread)
 {
     Thread.Sleep (iCallTime) ; // sover i "iCallTime" mS
     iExecThread = AppDomain.GetCurrentThreadId ();
     return "MyCallTime was " + iCallTime.ToString() ;
 }

 delegate string MethodDelegate(int iCallTime, out int
iExecThread)  ;

public void DemoSyncCall() // Synkront kald
{
  string s ;
  int iExecThread;

// En wrapper til metoden
MethodDelegate dlgt =
         new MethodDelegate (this.LongRunningMethod) ;

// Kald LongRunningMethod via delegaten.
```

```
 s = dlgt(3000, out iExecThread);

 MessageBox.Show (string.Format ("The delegate call returned the
                                  string:   \"{0}\",
                                  and the thread ID {1}", s,
                                  iExecThread.ToString() ) );
  }

public void DemoEndInvoke()
{
MethodDelegate dlgt =
                 new MethodDelegate (this.LongRunningMethod) ;
 string s ;
 int iExecThread;

 // Start  asynkront kald.
 IAsyncResult ar =
            dlgt.BeginInvoke(3000, out iExecThread, null,null);

// Arbejd her i normalt flow
// Afvikles sammen med den asynkrone kode
// Afvent svaret via EndInvoke

s = dlgt.EndInvoke (out iExecThread, ar) ;
MessageBox.Show (string.Format ("The delegate call returned the
string:   \"{0}\",
              and the number {1}", s,
              iExecThread.ToString() ) );

}


}



 static void Main(string[] args)
 {
    // Synkront kald
    AsyncDemo ad = new AsyncDemo () ;
    ad.DemoSyncCall() ;
    // Asynkront kald
    AsyncDemo ad = new AsyncDemo () ;
    ad.DemoEndInvoke() ;

 }
```

- Application will not lock

Event handlers as anonymous methods

```
…

class Program
{
static void Main(string[] args)
{
Console.WriteLine("***** Anonymous Methods *****\n");
Car c1 = new Car("SlugBug", 100, 10);
// Register event handlers as anonymous methods.
c1.AboutToBlow += delegate
{
Console.WriteLine("Eek! Going too fast!");
};
c1.AboutToBlow += delegate(object sender, CarEventArgs e)
{
Console.WriteLine("Message from Car: {0}", e.msg);
};
c1.Exploded += delegate(object sender, CarEventArgs e)
{
Console.WriteLine("Fatal Message from Car: {0}", e.msg);
};
…
```

Anonymous methods and access to private variables

- Anonymous methods cannot access ref or out parameters
- An anonymous method cannot have a local variable with the same name as local variable in 'outer method'
- An anonymous method can access instance variables (and static variables) in the outer class

The Lambda operators

- List<> has a FindAll method
- A lambda expression could be:

```
...
List<int> evenNumbers = anotherList.FindAll( i => (i%2) == 0);
...
```

- => is the same as =
- is right associative

Delegate example

```
...
   public class Program
    {
        private delegate void TestDelegate(string s);

        static void Main(string[] args)
        {


TestDelegate myDel = n => { string s = n + " " + "World"; Console.WriteLine(s); };

        myDel("Hello");



        }

...
```

- Notice: Several lines with semicolon in expression!
- string s = **n** + ” ” + ”World”
- **n** is in parameter to **myDel**

Example with Func<>

```
…

        Func<int, bool> myFunc = x => x == 5;
        bool result = myFunc(4);
        Console.WriteLine(result.ToString());

…
```

- Return value is always **the last** argument
- others are in parameters
- myFunc is function name

Another simple example with Func<>

```
…
        Func<double, double> expr = (x) =>  x / 2;

        Console.WriteLine(expr(22.4));

…
```

Count

```
...
      int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
        int oddNumbers = numbers.Count(n => n % 2 == 1);
        Console.WriteLine("Number of odd numbers {0}",oddNumbers);

…
```

- (mod 2)

Advanced Lambda

```
…
using System.Linq.Expressions;
…

ParameterExpression parameter1 = Expression.Parameter(typeof(int), "x");
BinaryExpression multiply = Expression.Multiply(parameter1, parameter1);
Expression<Func<int, int>> square = Expression.Lambda<Func<int, int>>(
            multiply, parameter1);

Func<int, int> lambda = square.Compile();

Console.WriteLine(lambda(5));


…
```

Explanation (step-by-step)

- Parameter of int type

```
ParameterExpression parameter1 = Expression.Parameter(typeof(int), "x");
```

- Binary lambda expression with 2 input: parameter1

```
BinaryExpression multiply = Expression.Multiply(parameter1, parameter1);
```

- Build lambda expression with multiply and parameter1

```
Expression<Func<int, int>> square = Expression.Lambda<Func<int, int>>(
            multiply, parameter1);
```

- Convert expression to a delegate

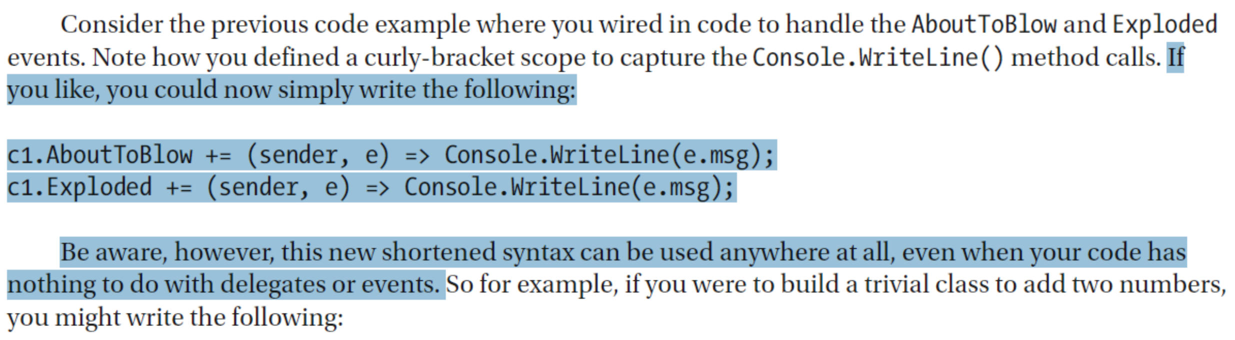
```
Func<int, int> lambda = square.Compile();
```

- Execute

```
Console.WriteLine(lambda(5));
```

- Result (of course) 25

Simplified expression in C# 7



Consider the previous code example where you wired in code to handle the `AboutToBlow` and `Exploded` events. Note how you defined a curly-bracket scope to capture the `Console.WriteLine()` method calls. If you like, you could now simply write the following:

```
c1.AboutToBlow += (sender, e) => Console.WriteLine(e.msg);
c1.Exploded += (sender, e) => Console.WriteLine(e.msg);
```

Be aware, however, this new shortened syntax can be used anywhere at all, even when your code has nothing to do with delegates or events. So for example, if you were to build a trivial class to add two numbers, you might write the following:

Advanced C#

- Operator overload & indexers
- Type conversion
- Extension methods
- Partial methods
- Pointers

Operator overloading & indexers

Indexer –a pseudo operator overload

- in C++ can [ ] be overloaded
- in C# ”indexers” exists

Example:

```
public class Cars : IEnumerable
         {
// This class maintains an array of cars.
       private ArrayList carArray;

 public Cars()
 {
  carArray = new ArrayList();
}

// The indexer.
 public Car this[int pos]
 {
 get
 {
   if(pos < 0)
     throw new
    IndexOutOfRangeException("Hey! Index out of range");
   else
    return (Car)carArray[pos];
 }
 set
 {
    carArray.Insert(pos, value);
    // Or simply call carArray.Add(value);
 }
}
...
```

- ”Indexers” gives a more safe manipulation of own types

A variation

- Uses:
    - `System.Collections.Specialized.ListDictionary`
    - can use a string (key token)

Example:

```
...
using System;
using System.Collections;
using System.Collections.Specialized;

namespace DictionaryIndexer
{
public class Cars
{
 // This class maintains an dictionary of cars.
 private ListDictionary carDictionary;

 public Cars()
 {
  carDictionary = new ListDictionary();
 }

// The string indexer.
 public Car this[string name]
 {
  get { return (Car)carDictionary[name];}
  set { carDictionary.Add(name, value);}
 }

 // The int indexer.
 public Car this[int item]
 {
```

```
 get { return (Car)carDictionary[item];}
 set { carDictionary.Add(item, value);}

 }
}
}
...
```

- indexers are CLS compatible

Multi dimensionel indexers

```
…
public class SomeContainer
{
private int[,] my2DintArray = new int[10, 10];
public int this[int row, int column]
{ /* get or set value from 2D array */ }
}
…
```

Indexers as interface type

```
…

public interface IStringContainer
{
string this[int index] { get; set; }
}

//implementation

class SomeClass : IStringContainer
{
private List<string> myStrings = new List<string>();
public string this[int index]
{
get { return myStrings[index]; }
set { myStrings.Insert(index, value); }
}
}
…
```

Operator overload

- Very useful
    - example:

```
...
string a = ”Hello”;
string b = “ World”;
string c = a+b;

//or ( Depending on the implementation  )
Kompleks k1 = new Kompleks(4,3);
Kompleks k2 = new Kompleks(6,6);
Kompleks k3 = k1+k2;

...
```

- Exists in C++

Example with a Point:

```
using System;

class Point
{

  private int x;
  private int y;

  public Point( int x, int y)
  {
     this.x = x;
     this.y = y;

  }
  public static Point operator + (Point p1, Point p2)
  {

         return new Point(p1.x+p2.x, p1.y+p2.y);
  }

  public static Point operator - (Point p1, Point p2)
  {
         return new Point(p1.x-p2.x, p1.y-p2.y);
  }
  public override string ToString()
  {
      return string.Format("X = {0} ,Y={1}",
                                           this.x,this.y);
  }
}

public class PointUser
{


  public static void Main()

  {

   Point p1 = new Point(4,3);
   Point p2 = new Point(4,3);
   Point p4 = p1+p2;

   Console.WriteLine("p4: {0}",p4.ToString());

   Point p5 = new Point(4,3);
   Point p6 = new Point(1,1);
   Point p7 = p5-p6;

   Console.WriteLine("p7: {0}",p7.ToString());
   Console.ReadLine();

  }

}
```

Output:

```
p4: X = 8 ,Y= 6
p7: X = 3 ,Y= 2
```

Operator overload MUST BE STATIC!

- Apply extra logic
  - If a Point must not be in the negative quadrants

Example:

```
...

// in the Point class

  public static Point operator - (Point p1, Point p2)
  {
         int tmpX = p1.x - p2.x;
         if(tmpX<0)
           throw new ArgumentOutOfRangeException();

         int tmpY = p1.y - p2.y;
         if(tmpY<0)
           throw new ArgumentOutOfRangeException();


         return new Point(p1.x-p2.x, p1.y-p2.y);
  }
...

// in Main


   Point p1 = new Point(4,3);
   Point p2 = new Point(4,3);
   Point p4 = p1+p2;

   Console.WriteLine("p4: {0}",p4.ToString());

   Point p5 = new Point(4,3);
   Point p6 = new Point(1,5);
   Point p7 = new Point(0,0);



   try
   {
     p7= p5-p6;
   }catch(ArgumentOutOfRangeException aoore)
   {
    Console.WriteLine(aoore.Message);
   }
   Console.WriteLine("p7: {0}",p7.ToString());
   Console.ReadLine();
```

Output:

```
p4: X = 8 ,Y= 6
```

```
Specified argument was out of the range of valid values.
p7: X = 0 ,Y= 0
```

Overload of equality operators

- First: overload Equals and GetHashCode
- Next == and !=

Example:

```
using System;

class Point
{

  private int x;
  private int y;

  public Point( int x, int y)
  {
     this.x = x;
     this.y = y;

  }
  public static Point operator + (Point p1, Point p2)
  {

           return new Point(p1.x+p2.x, p1.y+p2.y);
  }

  public static Point operator - (Point p1, Point p2)
  {
         int tmpX = p1.x - p2.x;
         if(tmpX<0)
           throw new ArgumentOutOfRangeException();

         int tmpY = p1.y - p2.y;
         if(tmpY<0)
           throw new ArgumentOutOfRangeException();


         return new Point(p1.x-p2.x, p1.y-p2.y);
  }

  public override string ToString()
  {
        return string.Format("X = {0} ,Y=
{1}",this.x,this.y);
  }

  public override int GetHashCode()
  {
      return this.ToString().GetHashCode();
  }

  public override bool Equals(object o)
  {
         if( ((Point) o).x == this.x &&
             ((Point) o).y == this.y )
              return true;
         else
             return false;
  }

  public static bool operator ==( Point p1,Point p2)
  {


           return (p1.Equals(p2));
 }

  public static bool operator !=( Point p1,Point p2)
  {


           return (!p1.Equals(p2));
 }

}


public class PointUser
{


  public static void Main()

  {


   Point p1 = new Point(4,3);
   Point p2 = new Point(4,3);
   Point p4 = p1+p2;

   Console.WriteLine("p4: {0}",p4.ToString());
   Point p5 = new Point(10,6);
   Point p6 = new Point(2,0);
   Point p7 = new Point(0,0);
   try
   {
    p7= p5-p6;
   }catch(ArgumentOutOfRangeException aoore)
   {
    Console.WriteLine(aoore.Message);
   }
   Console.WriteLine("p7: {0}",p7.ToString());


   if(p4==p7)
        Console.WriteLine("Equal..");

   Console.ReadLine();

  }

}
```

Output:

```
p4: X = 8 ,Y= 6
p7: X = 8 ,Y= 6
Equal..
```

Comparison operators

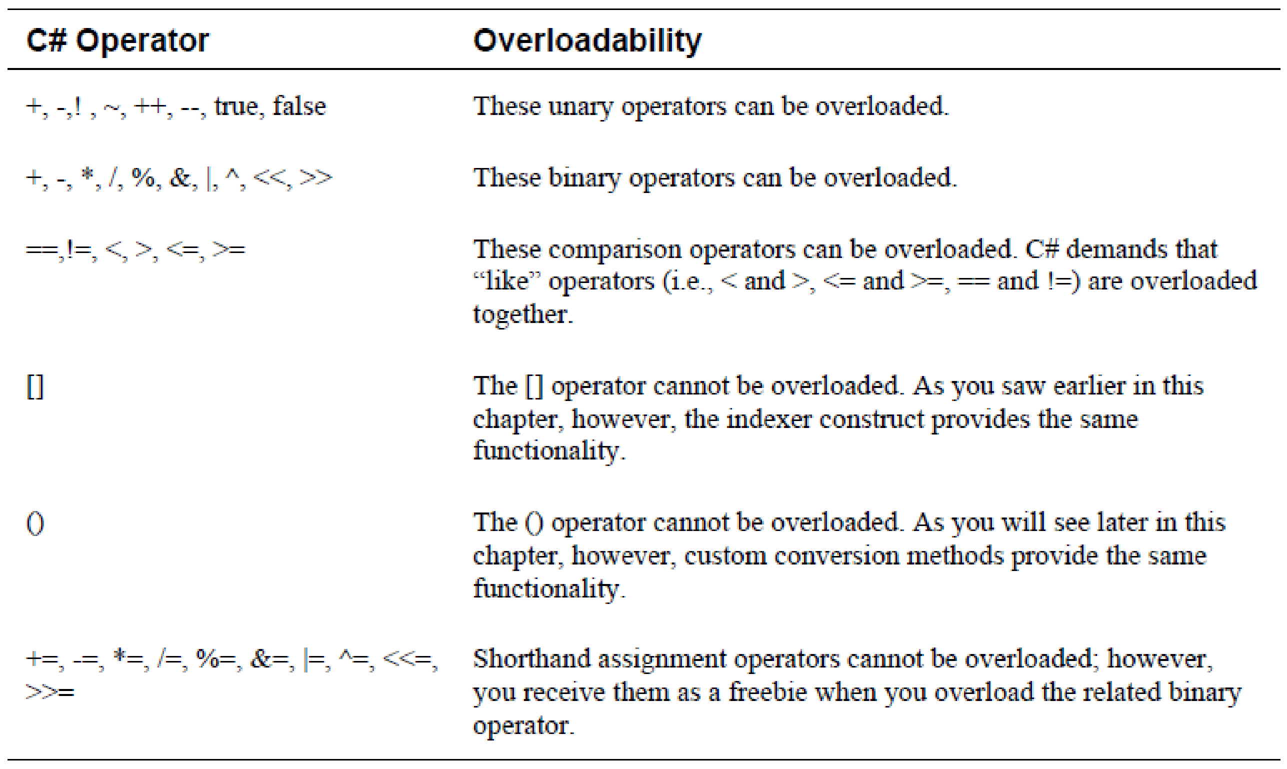
- Assumption: A ”size” (”length”) is defined as
  - Math.Sqrt((x*x)+(y*y))

- Use the IComparable interface
- MUST implement the method CompareTo()

```
...

// in Point
public int CompareTo(object o)
 {
          Point pt = (Point) o;

          if(PointLen(this)>PointLen(pt))
             return 1;
           if(PointLen(this)<PointLen(pt))
             return -1;
          else
             return 0;

 }

 public double PointLen(Point p)  // helper method

 {
       return Math.Sqrt((p.x*p.x)+(p.y*p.y));

 }

 public static bool operator > (Point p1, Point p2)
 {
         IComparable itfPoint = (IComparable) p1;
         return( itfPoint.CompareTo(p2) >0);

 }

 public static bool operator < (Point p1, Point p2)
 {
         IComparable itfPoint = (IComparable) p1;
         return( itfPoint.CompareTo(p2) < 0);

 }
...

// in Main
   Point p8 = new Point(4,9);
   Point p9 = new Point(4,10);
   if(p9>p8)
        Console.WriteLine("p9>p8");
...
```

Which operators can be overloaded?

Table 11-1. *Overloadability of C# Operators*



C# Operator	Overloadability
+, -,! , ~, ++, --, true, false	These unary operators can be overloaded.
+, -, *, /, %, &, |, ^, <<, >>	These binary operators can be overloaded.
==,!=, <, >, <=, >=	These comparison operators can be overloaded. C# demands that "like" operators (i.e., < and >, <= and >=, == and !=) are overloaded together.
[]	The [] operator cannot be overloaded. As you saw earlier in this chapter, however, the indexer construct provides the same functionality.
()	The () operator cannot be overloaded. As you will see later in this chapter, however, custom conversion methods provide the same functionality.
+=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=	Shorthand assignment operators cannot be overloaded; however, you receive them as a freebie when you overload the related binary operator.

Notice:

- if +, - and = = operators are overloaded
    - (for free) += and -= operators are given

Type conversion (explicit / implicit)

- C# has **two keyword** for type conversion
    - explicit
    - implicit

- ASSUME we have two types
    - Rectangle class
    - Square class

- No inheritance is used

- But: We will translate a Rectangle object to a Square object

explicit

We would like to do like this:

```
...
Rectangle r = new Rectangle(14,4);
Square s = (Square) r;
...
```

- Then apply this:

```
...

public static explicit operator Square(Rectangle r)
{
   Square s = new Square();
   s.Height = r.Height;
   return s;
}
...
```

- Notice: No return value
- Method must be static

implicit

- What if...

```
...

Square s = new Square();
Rectange r = s;
...
```

- Compiler error
- Here can implicit type conversion be used

```
...

public static implicit operator Rectangle(Square s)
{
   Rectangle r = new Rectangle();
   r.Height = s.Height;
   // and other stuff
  return r;

}
...
```

Extension Methods

- Allows compiled types (classes, structs, interfaces) to obtain new functionality

Use of methods that does not exists in the class

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Udvidning
{
    static class SomeExtensions
    {
        public static void VisAssemblyInfo(this object o)
        {
            Console.WriteLine("Member name: {0}", o.GetType().Name);
        }

    }



    public class Program
    {
        static void Main(string[] args)
        {
            String s = "Hallo World";
            s.VisAssemblyInfo();



        }
    }
}
```

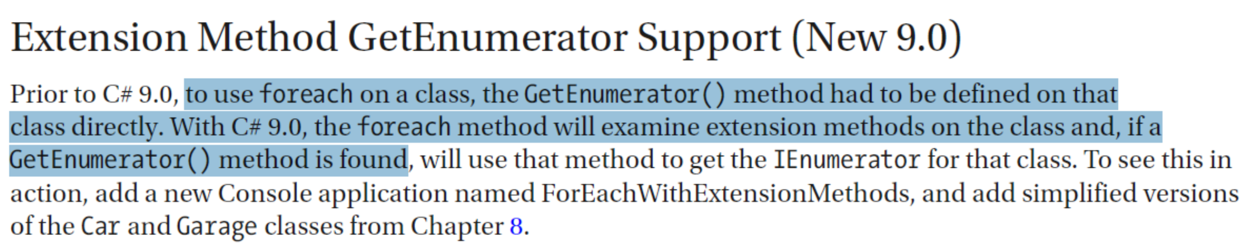
- Must be static

Extension methods with interface implementation

- Here: IEnumerable

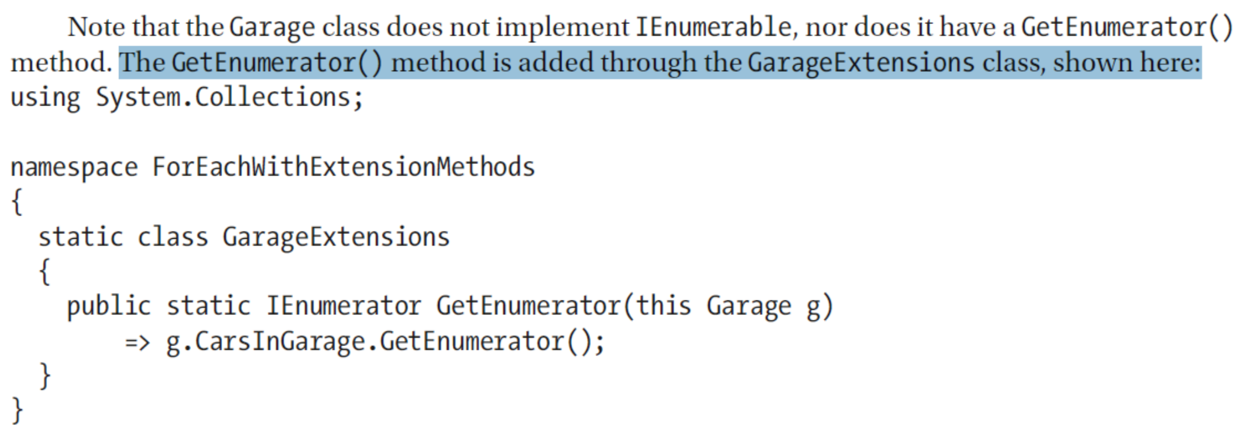
```
…
static class AnnoyingExtensions
{
public static void PrintDataAndBeep(this System.Collections.IEnumerable iterator)
{
foreach (var item in iterator)
{
Console.WriteLine(item);
Console.Beep();
}
}
}
…
```

New in C#9: GetEnumerator as Extension Method



Extension Method GetEnumerator Support (New 9.0)

Prior to C# 9.0, to use foreach on a class, the GetEnumerator() method had to be defined on that class directly. With C# 9.0, the foreach method will examine extension methods on the class and, if a GetEnumerator() method is found, will use that method to get the IEnumerator for that class. To see this in action, add a new Console application named ForEachWithExtensionMethods, and add simplified versions of the Car and Garage classes from Chapter 8.



Note that the Garage class does not implement IEnumerable, nor does it have a GetEnumerator() method. The GetEnumerator() method is added through the GarageExtensions class, shown here:

```
using System.Collections;

namespace ForEachWithExtensionMethods
{
  static class GarageExtensions
  {
    public static IEnumerator GetEnumerator(this Garage g)
        => g.CarsInGarage.GetEnumerator();
  }
}
```

Anonymous types

- Use it for data containers:

- Using the
  - var keyword

```
...
var myCar = new {Color = "blue", Make= "SAAB"};
...
```

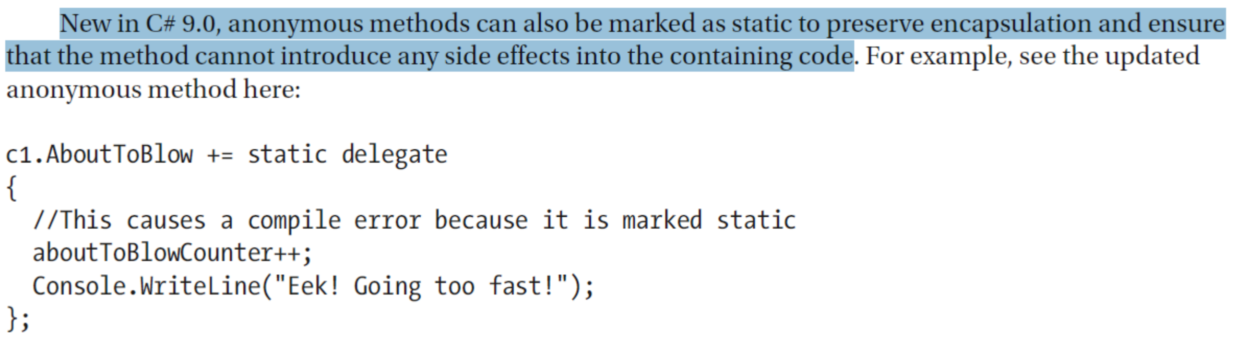
Example:

```
…
static void BuildAnonType( string make, string color, int currSp )
{
// Build anon type using incoming args.
var car = new { Make = make, Color = color, Speed = currSp };
// Note you can now use this type to get the property data!
Console.WriteLine("You have a {0} {1} going {2} MPH",
car.Color, car.Make, car.Speed);
// Anon types have custom implementations of each virtual
// method of System.Object. For example:
Console.WriteLine("ToString() == {0}", car.ToString());
}
…

// Call

// Make an anonymous type representing a car.
var myCar = new { Color = "Bright Pink", Make = "Saab", CurrentSpeed = 55 };
…
```

Anonymous types can contain other anonymous types

- See page 441

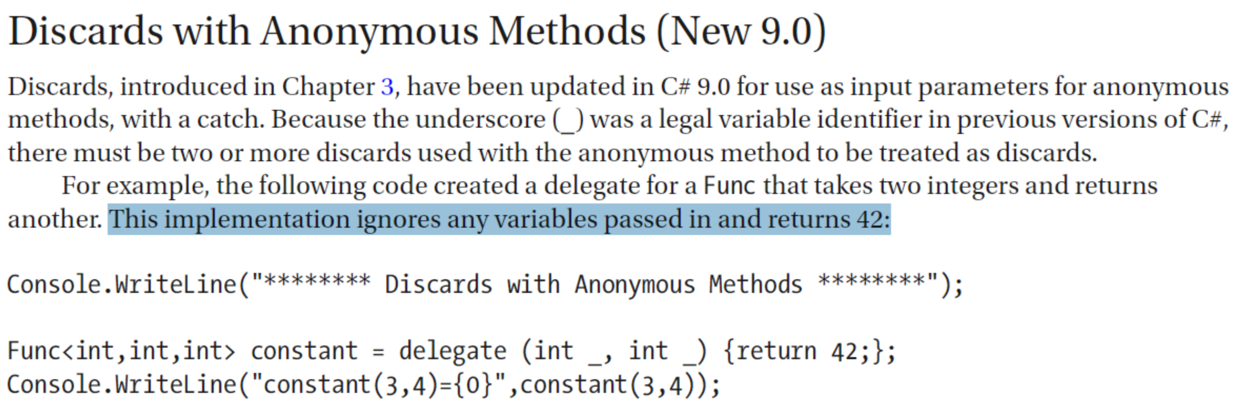
New in C#9: static anonymous methods



New in C# 9.0, anonymous methods can also be marked as static to preserve encapsulation and ensure that the method cannot introduce any side effects into the containing code. For example, see the updated anonymous method here:

```
c1.AboutToBlow += static delegate
{
  //This causes a compile error because it is marked static
  aboutToBlowCounter++;
  Console.WriteLine("Eek! Going too fast!");
};
```

New in C#9: Discards



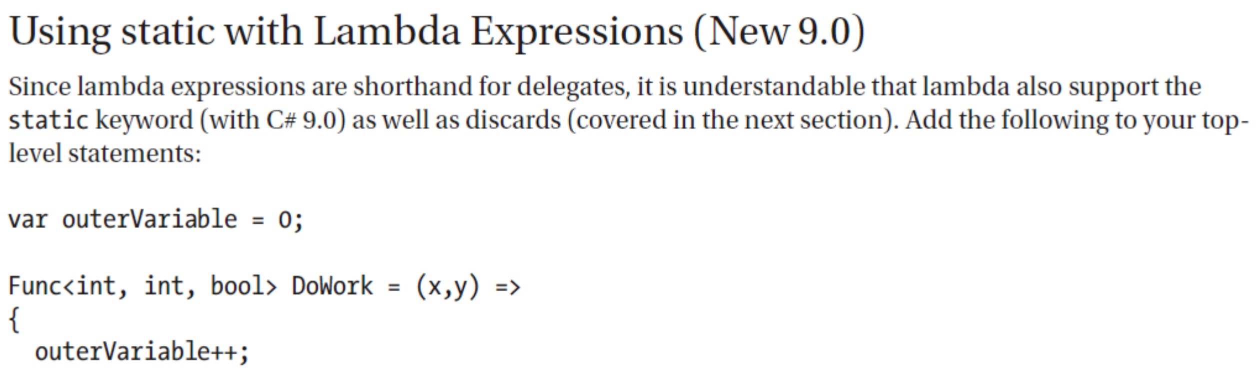
Discards with Anonymous Methods (New 9.0)

Discards, introduced in Chapter 3, have been updated in C# 9.0 for use as input parameters for anonymous methods, with a catch. Because the underscore (_) was a legal variable identifier in previous versions of C#, there must be two or more discards used with the anonymous method to be treated as discards.

For example, the following code created a delegate for a Func that takes two integers and returns another. This implementation ignores any variables passed in and returns 42:

```
Console.WriteLine("******** Discards with Anonymous Methods ********");

Func<int,int,int> constant = delegate (int _, int _) {return 42;};
Console.WriteLine("constant(3,4)={0}",constant(3,4));
```

Lambda Expressions (C#9)



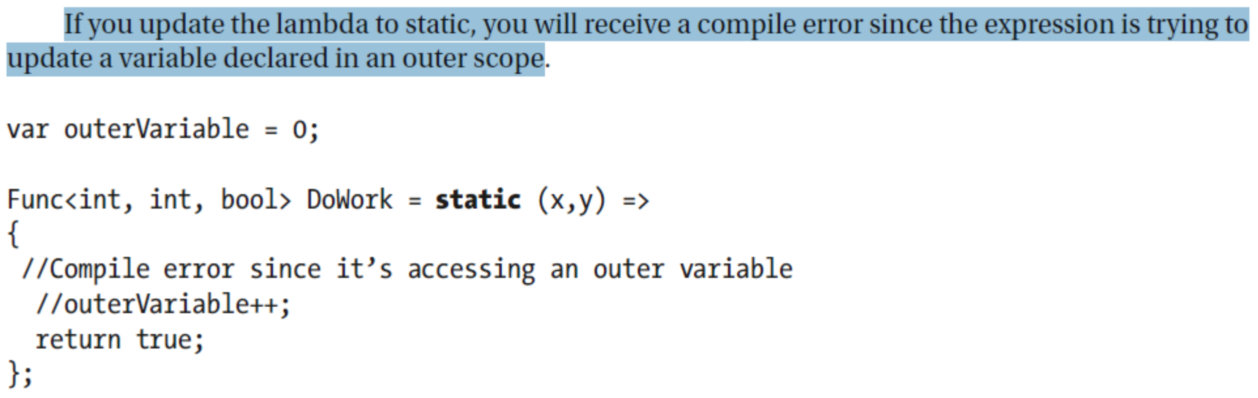
Using static with Lambda Expressions (New 9.0)

Since lambda expressions are shorthand for delegates, it is understandable that lambda also support the static keyword (with C# 9.0) as well as discards (covered in the next section). Add the following to your top-level statements:

```
var outerVariable = 0;

Func<int, int, bool> DoWork = (x,y) =>
{
  outerVariable++;
```

- outerVariable is now 1

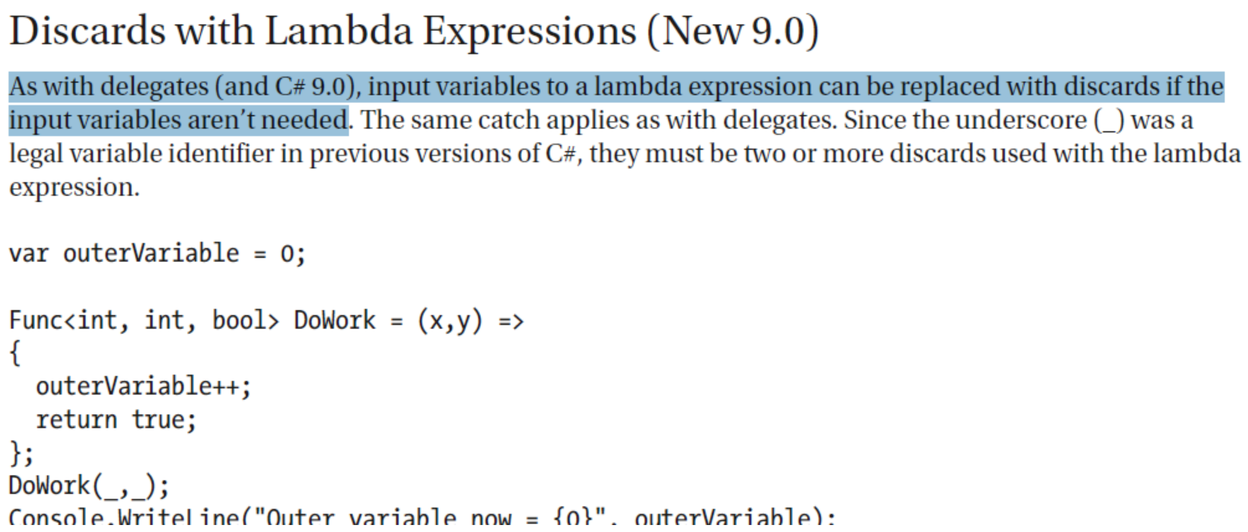
But if the Func<> is static:



If you update the lambda to static, you will receive a compile error since the expression is trying to update a variable declared in an outer scope.

```
var outerVariable = 0;

Func<int, int, bool> DoWork = static (x,y) =>
{
 //Compile error since it’s accessing an outer variable
  //outerVariable++;
  return true;
};
```

Duscards..



Discards with Lambda Expressions (New 9.0)

As with delegates (and C# 9.0), input variables to a lambda expression can be replaced with discards if the input variables aren’t needed. The same catch applies as with delegates. Since the underscore (_) was a legal variable identifier in previous versions of C#, they must be two or more discards used with the lambda expression.

```
var outerVariable = 0;

Func<int, int, bool> DoWork = (x,y) =>
{
  outerVariable++;
  return true;
};
DoWork(_,_);
Console.WriteLine("Outer variable now = {0}", outerVariable);
```

Partial methods

- Can only be created in partial classes
- Method must return void

- Restrictions

  - Method must return void
  - is implicit private

- Example simple partial class

```
public partial class Employee
{
    public void DoWork()
    {
    }
}

public partial class Employee
{
    public void GoToLunch()
    {
    }
}
```

- About partial methods

Partial methods are especially useful as a way to customize generated code. **They allow for a method name and signature to be reserved, so that generated code can call the method but the developer can decide whether to implement the method.** Much like partial classes, partial methods enable code created by a code generator and code created by a human developer to work together without run-time costs.

A partial method declaration consists of two parts: the definition, and the implementation. These may be in separate parts of a partial class, or in the same part. If there is no implementation declaration, then the compiler optimizes away both the defining declaration and all calls to the method.

Microsoft msdn

Example:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace PartialMethods
{
    public partial class Printing
    {
        partial void PrintInfo(string info) // implementation
        {
            Console.WriteLine(info);
        }
    }

}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace PartialMethods
{

    public partial class Printing
    {
        partial void PrintInfo(string info);  // declaration
        public void UsePrintInfo(string info)
        {
            PrintInfo(info);
        }
    }
    public class Program
    {
        static void Main(string[] args)
        {
            Printing p = new Printing();
            p.UsePrintInfo("Uses partial method");

        }
    }
}
```

Watch out:

```
    public partial class PartMethods
    {
        partial void Amethod();
    }

   public  partial class PartMethods
    {
        // Amethod() not implemented

        public void UseAmethod()
        {
            try
            {
                Amethod(); // but called here
            }
            catch (NotImplementedException nie)
            {
                Console.WriteLine(nie.Message);
            }

        }

    }
```

- Here: Method implementation not implemented
- You should think that exception is thrown
- It is not – actually nothing happen

```
…
  public  partial class PartMethods
    {

        partial void Amethod()
        {
           throw new NotImplementedException("Ups");
        }

        public void UseAmethod()
        {
            try
            {
                Amethod();
            }
            catch (NotImplementedException nie)
            {
                Console.WriteLine(nie.Message);
            }

        }

    }


…
```

- Now it throws an exception

Pointers

Arguments for using pointers:

- Optimization of code
- Use of C based DLL’s

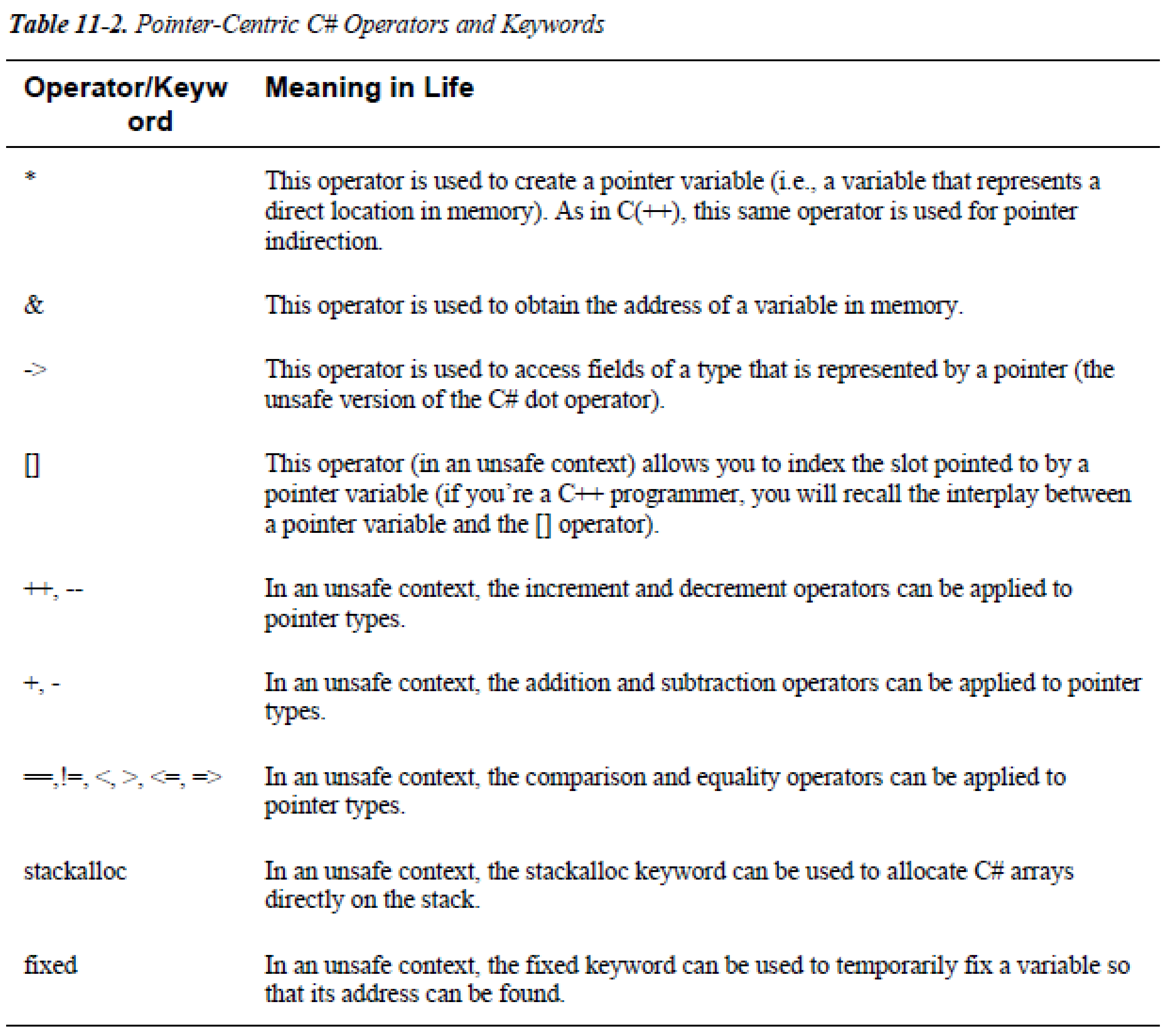
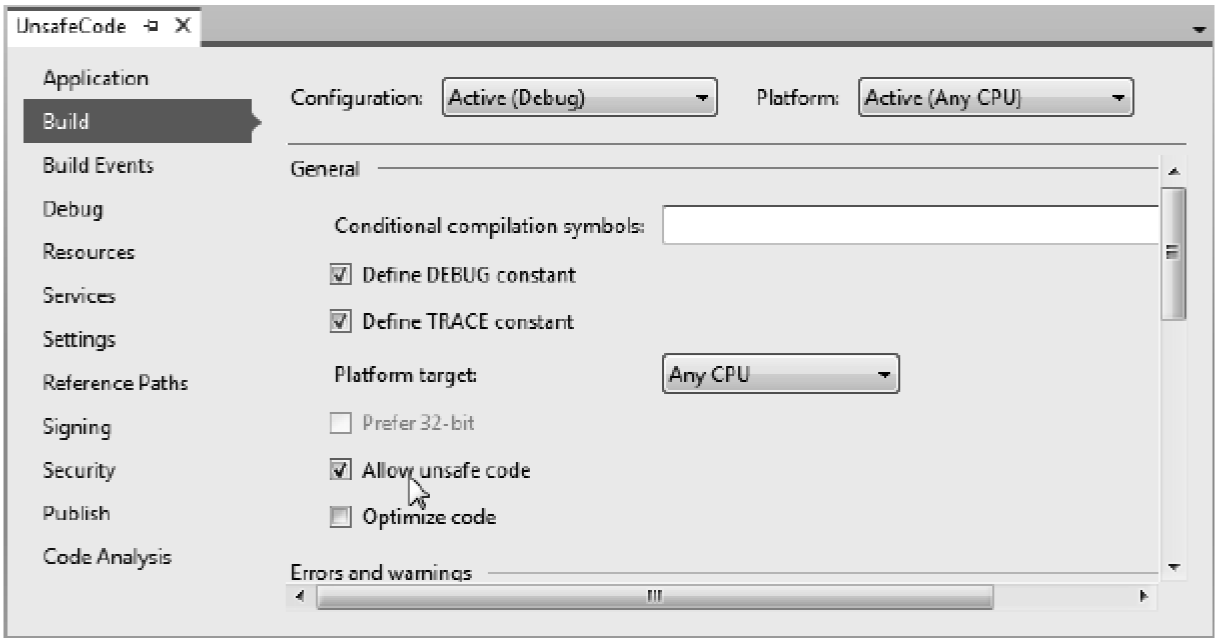


Table 11-2. Pointer-Centric C# Operators and Keywords

Operator/Keyword	Meaning in Life
*	This operator is used to create a pointer variable (i.e., a variable that represents a direct location in memory). As in C(++), this same operator is used for pointer indirection.
&	This operator is used to obtain the address of a variable in memory.
->	This operator is used to access fields of a type that is represented by a pointer (the unsafe version of the C# dot operator).
[]	This operator (in an unsafe context) allows you to index the slot pointed to by a pointer variable (if you're a C++ programmer, you will recall the interplay between a pointer variable and the [] operator).
++, --	In an unsafe context, the increment and decrement operators can be applied to pointer types.
+, -	In an unsafe context, the addition and subtraction operators can be applied to pointer types.
==,!=, <, >, <=, =>	In an unsafe context, the comparison and equality operators can be applied to pointer types.
stackalloc	In an unsafe context, the stackalloc keyword can be used to allocate C# arrays directly on the stack.
fixed	In an unsafe context, the fixed keyword can be used to temporarily fix a variable so that its address can be found.

- Set the **unsafe** flag to the compiler
- Project Settings ”Allow unsafe code blocks”



Configuration: Active (Debug)
Platform: Active (Any CPU)

Conditional compilation symbols:

☑ Define DEBUG constant

☑ Define TRACE constant

Platform target: Any CPU

☐ Prefer 32-bit

☑ Allow unsafe code

☐ Optimize code

Figure 11-2. *Enabling unsafe code using Visual Studio*

Basic example – pointers:

```
using System;

public class PointereErHerlige
{


  public static void Main()

  {

   unsafe
      {

       int x = 100;
       int* px = &x;
       Console.WriteLine("px adr: {0:x}",(int)px);
       Console.WriteLine("px content: {0}",*px);
       *px = 418;
       Console.WriteLine("px content is now: {0}",*px);
      }

         Console.ReadLine();

  }

}
```

- Are methods used –
  - o The call to the method must be unsafe

```
using System;

public class PointereErHerlige
{


  public static void Main()

   {


         unsafe{ LegMedPointere();}

         Console.ReadLine();

   }


 unsafe public static void LegMedPointere()
 {
        int x = 100;
        int* px = &x;
        Console.WriteLine("px adr: {0:x}",(int)px);
        Console.WriteLine("px indhold: {0}",*px);
        *px = 418;
        Console.WriteLine("px indhold er nu: {0}",*px);


 }


}
```

- Avoid this by
  - o moving the unsafe block

```
...

public static void LegMedPointere()
```

```
 {
      unsafe {
        int x = 100;
        int* px = &x;
        Console.WriteLine("px adr: {0:x}",(int)px);
        Console.WriteLine("px indhold: {0}",*px);
        *px = 418;
        Console.WriteLine("px indhold er nu: {0}",*px);

      }
 }

...
```

Declarations of pointers

```
...

int *pa,*pb; // Will not work
...
```

```
l6_3.cs(24,19): error CS1001: Identifier expected
```

But:

```
...
int* pa,pb; // is ok!
...
```

Example: dot and pointer-field (->) operators

```
using System;



struct Auto
{
   public int serNum;
   public int volume;

  public override string ToString()
  {

    return "Nummer: "+serNum+"VOLUME: "+volume;
  }

}

public class PointereErHerlige
{


   public static void Main()
  {

        unsafe
        {
              Auto skoda;
              Auto* pAuto= &skoda;
              pAuto->serNum =120120;
              pAuto->volume= 1200;
              Console.WriteLine(skoda.ToString());


              Auto lada;
              Auto* pAuto2= &lada;
              (*pAuto2).serNum= 418418;
              (*pAuto2).volume= 1300;
              Console.WriteLine(lada.ToString());

        }

          Console.ReadLine();

  }


}
```

Declaration of local variables with direct memory allocation

- keyword:
    - stackalloc

```
...

   unsafe
        {
             char* p = stackalloc char[256];
             for ( int k = 0; k<256;k++)
                    p[k] = (char) k;

        }
...
```

Lock of reference types – the fixed keyword

- It is not possible to set pointers to objects on the heap (due to the Garbage Collection):

But we can as follows:

Example:

```
using System;



class Auto
{
   public int serNum;  // showing fixed statement
   public int volume;

  public override string ToString()
  {

    return "Number: "+serNum+"VOLUME: "+volume;
  }

}

public class PointereErHerlige
{


   public static void Main()

  {

        unsafe
        {
              Auto skoda = new Auto();

              skoda.serNum =120120;
              skoda.volume= 1200;
              fixed ( int* p = &skoda.serNum)
              {

                 Console.WriteLine("p: {0}",*p);
              }
```

```
                Console.WriteLine(skoda.ToString());

         }


          Console.ReadLine();

   }


}
```

- The reference is **locked to the variable**
  - Address stay constant in the fixed statement

Volatile keyword

- Defines a type,that can be accessed by several threads

sizeof keyword

- As in C++

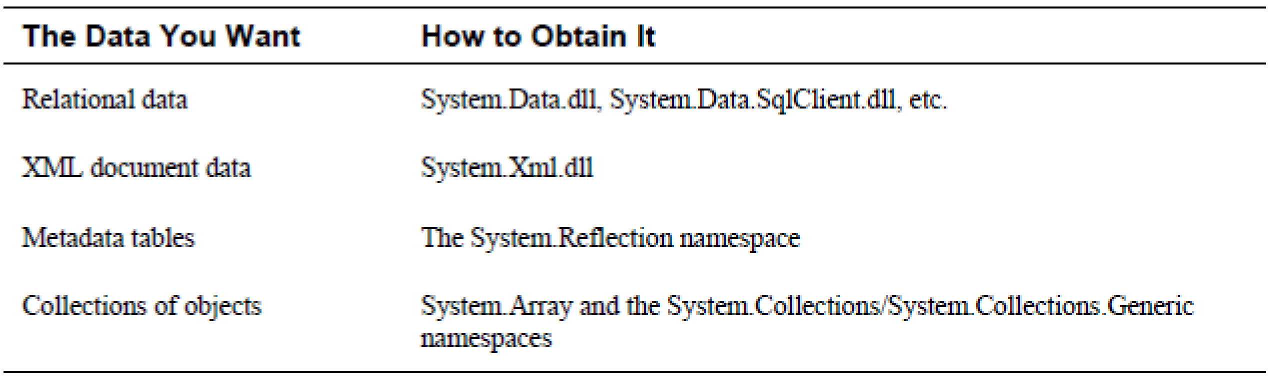
Example:

```
...
Console.WriteLine("SHORTY: {0}",sizeof(short));
...
```

Introduction to LINQ (Language Integrated Query)

- Easy way to work on data
    - earlier in .NET
        - System.data namespace
        - System.XML namespace
        - plus plus
    - From version 3.5 we have
        - LINQ

Table 12-1. *Ways to Manipulate Various Types of Data*

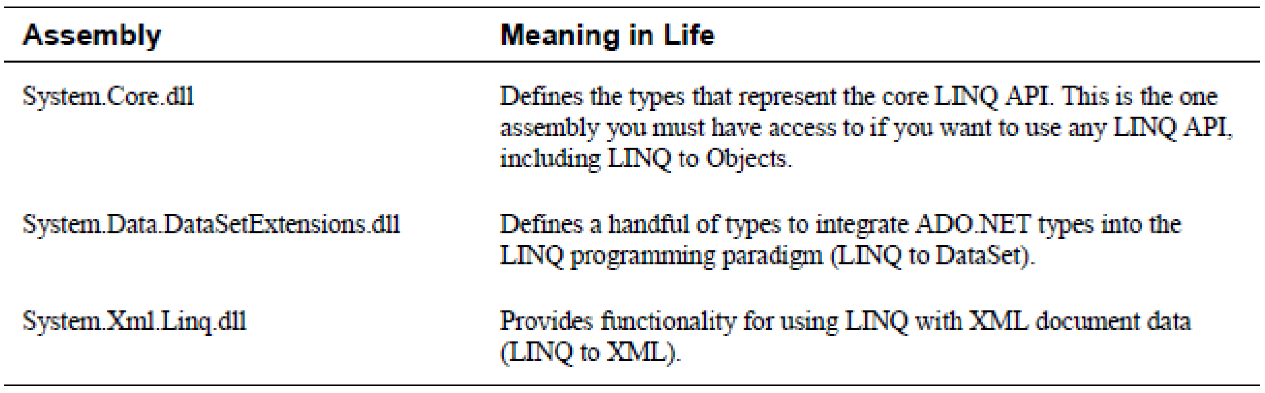


The Data You Want	How to Obtain It
Relational data	System.Data.dll, System.Data.SqlClient.dll, etc.
XML document data	System.Xml.dll
Metadata tables	The System.Reflection namespace
Collections of objects	System.Array and the System.Collections/System.Collections.Generic namespaces

- LINQ makes queries possible – looks like SQL
- A LINQ query is ”strongly typed”
    - The C# compiler can check an query
    - It is possible to create new operators and redefine existing operators

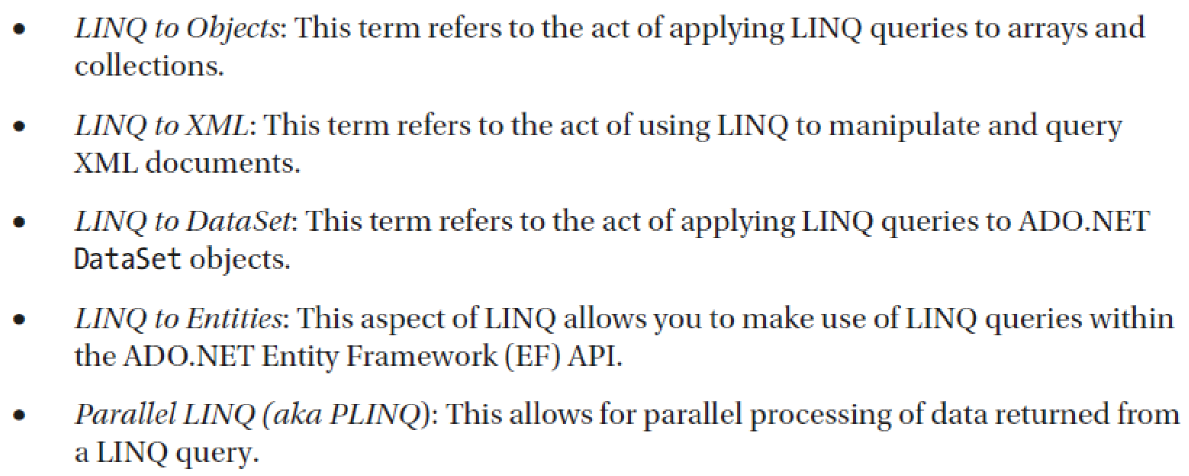
- Visual Studio creates automatically reference to the LINQ assembly (If the compiler is >= the .NET 4.5 version)

- Assemblies:



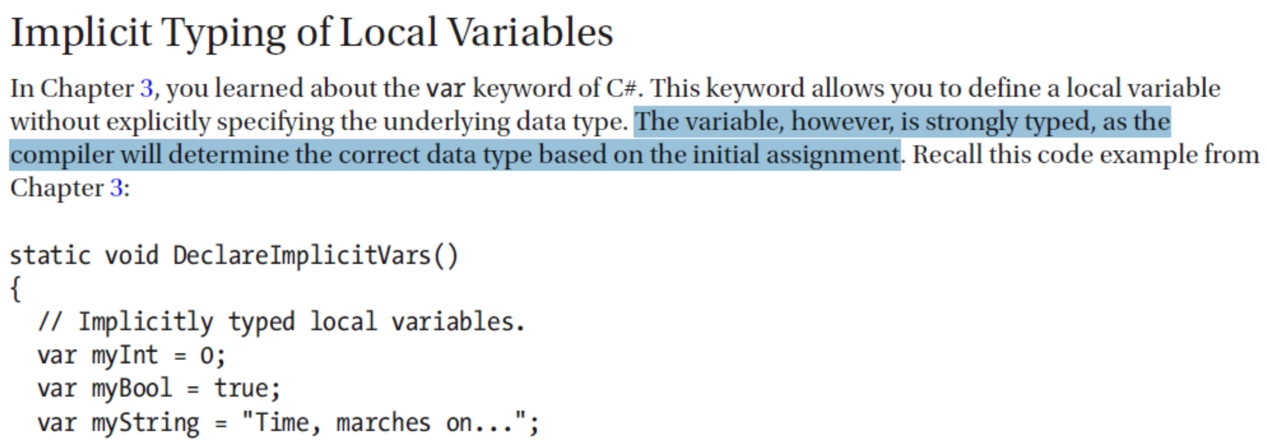
Assembly	Meaning in Life
System.Core.dll	Defines the types that represent the core LINQ API. This is the one assembly you must have access to if you want to use any LINQ API, including LINQ to Objects.
System.Data.DataSetExtensions.dll	Defines a handful of types to integrate ADO.NET types into the LINQ programming paradigm (LINQ to DataSet).
System.Xml.Linq.dll	Provides functionality for using LINQ with XML document data (LINQ to XML).

LINQ to ?



- *LINQ to Objects*: This term refers to the act of applying LINQ queries to arrays and collections.
- *LINQ to XML*: This term refers to the act of using LINQ to manipulate and query XML documents.
- *LINQ to DataSet*: This term refers to the act of applying LINQ queries to ADO.NET `DataSet` objects.
- *LINQ to Entities*: This aspect of LINQ allows you to make use of LINQ queries within the ADO.NET Entity Framework (EF) API.
- *Parallel LINQ (aka PLINQ)*: This allows for parallel processing of data returned from a LINQ query.

Implicit typing



Implicit Typing of Local Variables

In Chapter 3, you learned about the `var` keyword of C#. This keyword allows you to define a local variable without explicitly specifying the underlying data type. The variable, however, is strongly typed, as the compiler will determine the correct data type based on the initial assignment. Recall this code example from Chapter 3:

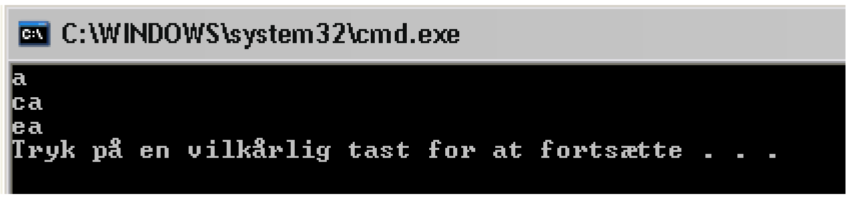
```
static void DeclareImplicitVars()
{
  // Implicitly typed local variables.
  var myInt = 0;
  var myBool = true;
  var myString = "Time, marches on...";
```

LINQ

- **LINQ uses lambda operators!**

Introduction example: LINQ

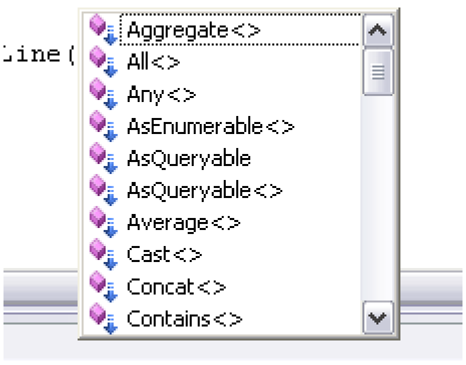
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LINQ1
{
    class Program
    {
        static void Main(string[] args)
        {
            String[] ar = { "a", "b", "ca", "d","ea","f" };

            IEnumerable<String> sæt = from c in ar where c.Contains("a") select
c;
            foreach( String s in sæt)
            {
                Console.WriteLine("{0}",s);
            }
        }
    }
}
```



```
C:\WINDOWS\system32\cmd.exe
a
ca
ea
Tryk på en vilkårlig tast for at fortsætte . . .
```

Members

- An INumerable set has many members:

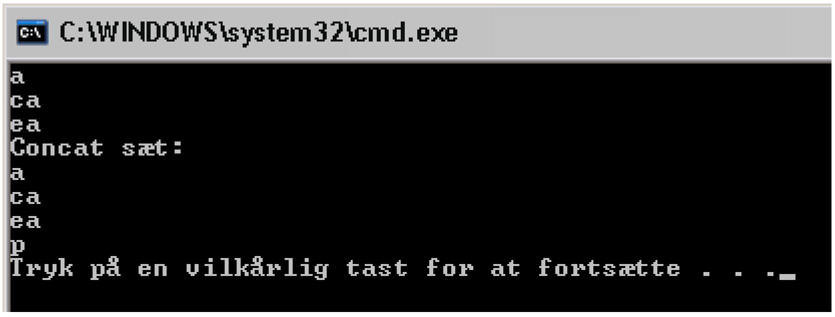


```
Aggregate<>
All<>
Any<>
AsEnumerable<>
AsQueryable
AsQueryable<>
Average<>
Cast<>
Concat<>
Contains<>
```

Example on Concat:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LINQ1
{
    class Program
    {
      static void Main(string[] args)
      {
       String[] ar = { "a", "b", "ca", "d","ea","f" };
       String[] ar2 = { "p", "n", "m" };

       IEnumerable<String> sæt = from c in ar where c.Contains("a") select c;
            foreach( String s in sæt)
            {
                Console.WriteLine("{0}",s);

            }
        Console.WriteLine("Concat sæt:");
        IEnumerable<String> etAndetSæt = from c in ar2
                                         where c.Contains("p") select c;
        IEnumerable<String> resultatSæt = sæt.Concat<String>(etAndetSæt);

            foreach (String s in resultatSæt)
            {
                Console.WriteLine("{0}", s);

            }

        }
    }
}
```

Output:

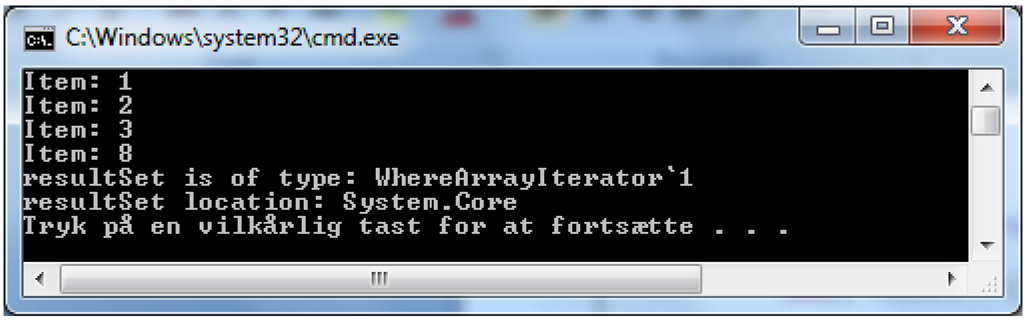


```
a
ca
ea
Concat sæt:
a
ca
ea
p
Tryk på en vilkårlig tast for at fortsætte . . .
```

Implicit typed

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LinqTest
```

```
{
    class Program
    {
        static void Main(string[] args)
        {
            int[] numbers = { 10, 20, 30, 40, 1, 2, 3, 8 };


            var subset = from i in numbers where i < 10 select i;
            foreach (int i in subset)
                Console.WriteLine("Item: {0}", i);

            Console.WriteLine("resultSet is of type: {0}",subset.GetType().Name);
            Console.WriteLine("resultSet location: {0}",
                                           subset.GetType().Assembly.GetName().Name);
        }
    }
}
```



```
Item: 1
Item: 2
Item: 3
Item: 8
resultSet is of type: WhereArrayIterator`1
resultSet location: System.Core
Tryk på en vilkårlig tast for at fortsætte . . .
```

- Is of the type WhereArrayIterator

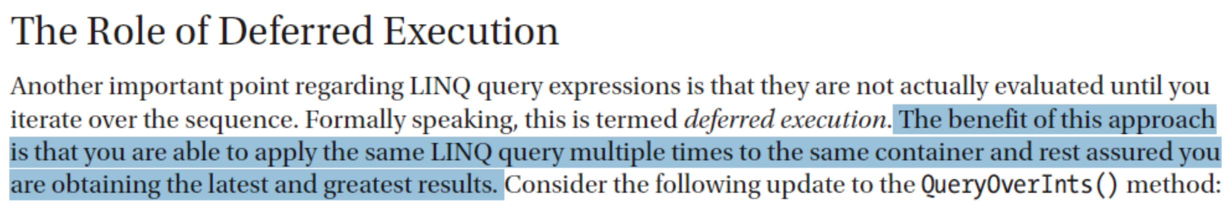
Deferred and immediate execution

- When evaluated in the iteration
  - **Is called deferred execution**
- Evaluationen outside an iteration:
  - ToArray<T>
  - ToDictionary<TSource, TKey>
  - ToList<T>
  - **Is called immidiate execution**

When is the expression evaluated?

- Is evaluated in the foreach construction

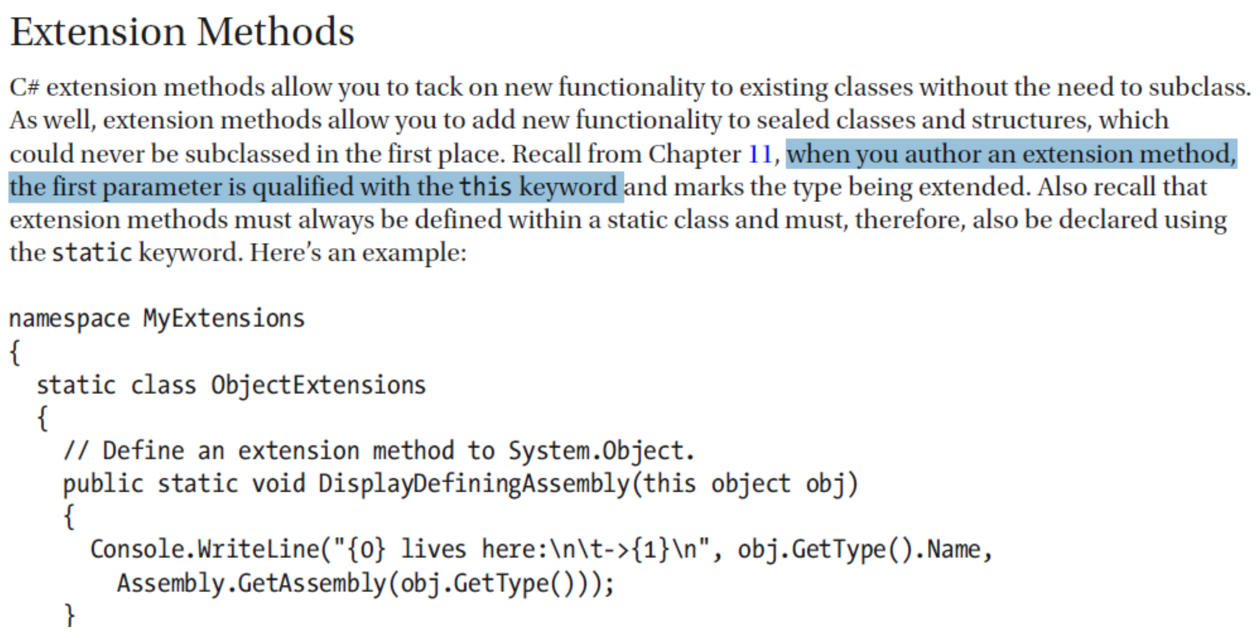
```
...
            foreach (String s in resultatSæt)
            {
                Console.WriteLine("{0}", s);
                }

...
```



The Role of Deferred Execution

Another important point regarding LINQ query expressions is that they are not actually evaluated until you iterate over the sequence. Formally speaking, this is termed *deferred execution*. The benefit of this approach is that you are able to apply the same LINQ query multiple times to the same container and rest assured you are obtaining the latest and greatest results. Consider the following update to the `QueryOverInts()` method:

LINQ has extension methods



Extension Methods

C# extension methods allow you to tack on new functionality to existing classes without the need to subclass. As well, extension methods allow you to add new functionality to sealed classes and structures, which could never be subclassed in the first place. Recall from Chapter 11, when you author an extension method, the first parameter is qualified with the `this` keyword and marks the type being extended. Also recall that extension methods must always be defined within a static class and must, therefore, also be declared using the `static` keyword. Here's an example:

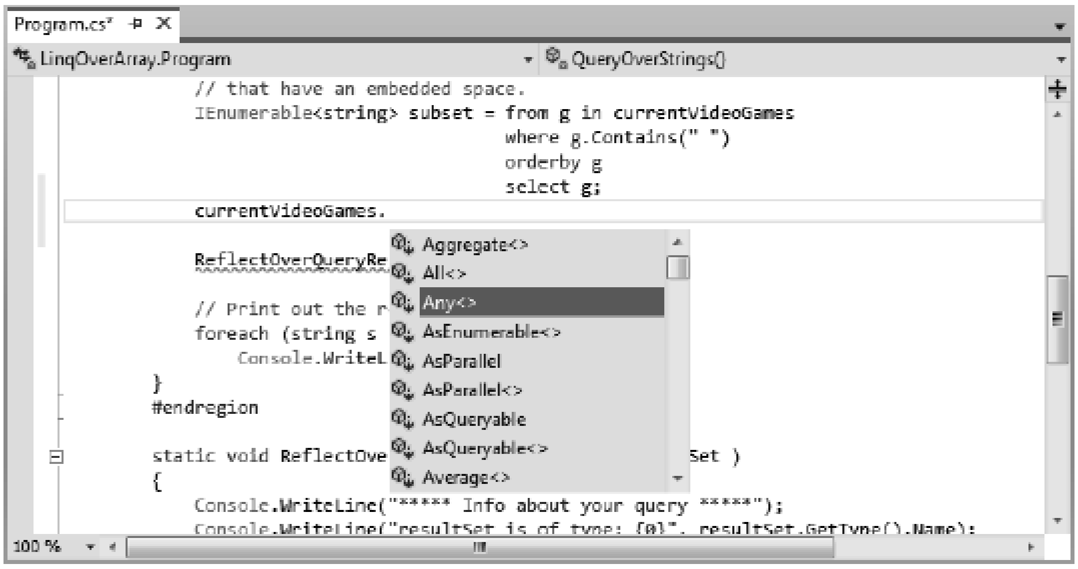
```
namespace MyExtensions
{
  static class ObjectExtensions
  {
    // Define an extension method to System.Object.
    public static void DisplayDefiningAssembly(this object obj)
    {
      Console.WriteLine("{0} lives here:\n\t->{1}\n", obj.GetType().Name,
        Assembly.GetAssembly(obj.GetType()));
    }
```

Immediate Execution

Example (`.ToList<String>()`) :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LINQ1
{
    class Program
    {
      static void Main(string[] args)
      {
        String[] ar = { "a", "b", "ca", "d","ea","f" };
        String[] ar2 = { "p", "n", "m" };

        IEnumerable<String> sæt = from c in ar where c.Contains("a") select c;
         foreach( String s in sæt)
           {
                Console.WriteLine("{0}",s);

           }
         Console.WriteLine("Concat sæt:");
         IEnumerable<String> etAndetSæt =
                          from c in ar2 where c.Contains("p") select c;
         IEnumerable<String> resultatSæt = sæt.Concat<String>(etAndetSæt);

          foreach (String s in resultatSæt)
           {
                Console.WriteLine("{0}", s);

           }





          Console.WriteLine("List<>:");
          List<String> enListe = (from c in ar where c.Contains("f")
                                    select c).ToList<String>();

           foreach (String s in enListe)
            {
                Console.WriteLine("{0}", s);

            }



        }
    }
}
```

VS2019 shows Extension types



```
        // that have an embedded space.
        IEnumerable<string> subset = from g in currentVideoGames
                                     where g.Contains(" ")
                                     orderby g
                                     select g;
        currentVideoGames.

        ReflectOverQueryRe

        // Print out the r
        foreach (string s
            Console.WriteL
    }
    #endregion

    static void ReflectOve                                  Set )
    {
        Console.WriteLine("***** Info about your query *****");
        Console.WriteLine("resultSet is of type: {0}", resultSet.GetType().Name);
```

Figure 12-1. The System.Array type has been extended with members of System.Linq.Enumerable

A method cannot return a var (can only be used internal in a method)

- Use Extension methods:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LinqTest
{
    class Program
    {
        static void Main(string[] args)
        {

            foreach (string item in GetStringSubsetAsArray())
            {
                Console.WriteLine(item);
            }
        }


        static string[] GetStringSubsetAsArray()
        {
            string[] colors = {"Light Red", "Green",
                               "Yellow", "Dark Red", "Red", "Purple"};
            var theRedColors = from c in colors
                               where c.Contains("Red")
                               select c;

            return theRedColors.ToArray();
        }


    }
}
```

LINQ and generic collections

- LINQ works for instance also on a list

```
...

static void GetFastCars(List<Car> myCars)
{
// Find all Car objects in the List<>, where the Speed is
// greater than 55.
var fastCars = from c in myCars where c.Speed > 55 select c;
  foreach (var car in fastCars)
  {
    Console.WriteLine("{0} is going too fast!", car.PetName);
  }

}
...
```

LINQ and ArrayList

- Again: Need to specify the type to be fetched
    - anArrayList.OfType< ObjectType>
    - See example. p. 512

LINQ Query operators

Operators:

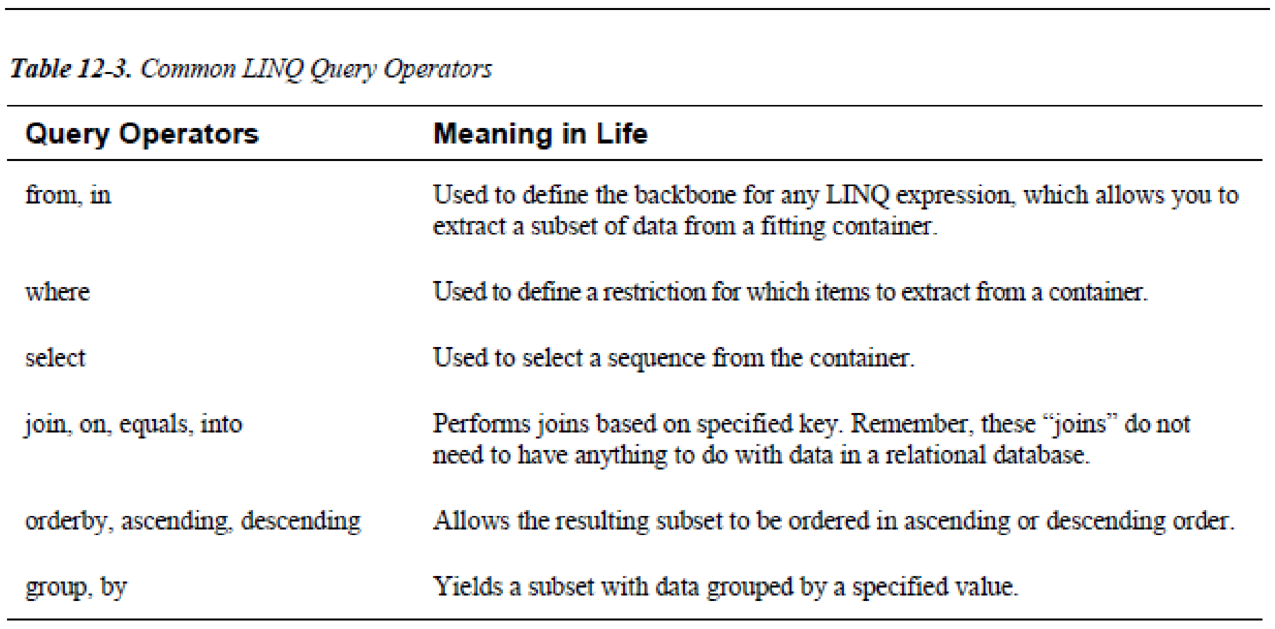


Table 12-3. Common LINQ Query Operators

Query Operators	Meaning in Life
from, in	Used to define the backbone for any LINQ expression, which allows you to extract a subset of data from a fitting container.
where	Used to define a restriction for which items to extract from a container.
select	Used to select a sequence from the container.
join, on, equals, into	Performs joins based on specified key. Remember, these "joins" do not need to have anything to do with data in a relational database.
orderby, ascending, descending	Allows the resulting subset to be ordered in ascending or descending order.
group, by	Yields a subset with data grouped by a specified value.

- See examples page 489-491

Count

```
...

static string[] GetStringSubsetAsArray()
{
     string[] colors = {"Light Red", "Green",
                        "Yellow", "Dark Red", "Red", "Purple"};
      var theRedColors = from c in colors
                               where c.Contains("Red")
                               select c;

      int numb =
                (from g in theRedColors where g.Length > 6 select g).Count();
            // Antal over 6 tegn??
            Console.WriteLine("{0} stk i LINQ query.", numb);


            return theRedColors.ToArray();
        }

...
```

Reverse

```
...
foreach (string item in GetStringSubsetAsArray().Reverse())
  {
                Console.WriteLine(item);
  }

...
```

- Other:
  - Differences : Except()
  - common data items : Intersect()
  - Concat()
  - Removing duplicates: Distinct()
  - Max(), Min(), Average()

Max(), Min(),Average(),Sum() examples

```
...

static void AggregateOps()
{
double[] winterTemps = { 2.0, -21.3, 8, -4, 0, 8.2 };
// Various aggregation examples.
Console.WriteLine("Max temp: {0}",
(from t in winterTemps select t).Max());
Console.WriteLine("Min temp: {0}",
(from t in winterTemps select t).Min());
Console.WriteLine("Avarage temp: {0}",
(from t in winterTemps select t).Average());
Console.WriteLine("Sum of all temps: {0}",
(from t in winterTemps select t).Sum());
}
...
```

Use of delegates as search filter

Example:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LINQ1
{
    class Program
    {
        public static bool etFilter(String s) { return s.EndsWith("2"); }
        public static String items(String s) { return s; }

        static void Main(string[] args)
        {
            String[] ar = { "a", "b", "ca", "d","ea","f" };
            String[] ar2 = { "p", "n", "m" };

            IEnumerable<String> sæt = from c in ar where c.Contains("a")
                                      select c;
            foreach( String s in sæt)
            {
                Console.WriteLine("{0}",s);

            }
            Console.WriteLine("Concat sæt:");
            IEnumerable<String> etAndetSæt = from c in ar2
                                             where c.Contains("p") select c;
            IEnumerable<String> resultatSæt = sæt.Concat<String>(etAndetSæt);

            foreach (String s in resultatSæt)
            {
                Console.WriteLine("{0}", s);

            }
            Console.WriteLine("List<>:");
            List<String> enListe = (from c in ar where c.Contains("f")
                                    select c).ToList<String>();

            foreach (String s in enListe)
            {
                Console.WriteLine("{0}", s);

            }

            UseDelegate();
        }



        public static void UseDelegate()
        {
            String[] versioner = { "1.1", "1.2", "2.1",
                                   "2.2","3.0","3.2","3.5" };
            Func<String, bool> søgeFilter = new Func<String, bool>(etFilter);
            Func<String, String> udfør = new Func<String, String>(items);

            var resultat =
                     versioner.Where(søgeFilter).OrderBy(udfør).Select(udfør);

            Console.WriteLine("Delegater:");

            foreach( var version in resultat)
            {
                Console.WriteLine("{0}", version);
            }

        }
    }
}
```

LINQ and delegates

- The Delegate Func<> can
  - Use a method as an argument (also ***anonymous*** methods)
  - or lambda expressions