329-Windows Programlama

Bil. Müh. S. Kıvanç EKİCİ

.Net Delegate

- callback (geri bildirim)
  - object tarafından kendisini oluşturan object tarafına yapılan bildirim. (örn : button_click , buton form nesnesine üzerine tıklandığını bildiriyor)
- Delegate
  - Daha sonra çağılabilecek diğer methodları işaret eden bir object.
    - Çağırılacak methodun adresi
    - Methodun parametreleri
    - Geri dönüş (return) türü
  - senkron, asenkron

.Net Delegate 2

```
// This delegate can point to any method,
// taking two integers and returning an integer.
public delegate int BinaryOp(int x, int y);
```

- MulticastDelegate
  - Compiler yukarıdaki gibi bir tanım gördüğü zaman varsayılan olarak MulticastDelegate sınıfını inherit ederek bir sealed class oluşturur. Bu sınıf içinde ise bu delegate ile ilgili methodların listesi tutulur ve gerektiğinde çağırılabilir.
- Delegate
  - Bu sınıf MulticastDelegate den farklı olarak bir method listesi tutmaz, sadece tek bir method için kullanılabilir.

.Net Delegate 3

```
sealed class BinaryOp : System.MulticastDelegate
{
	public int Invoke(int x, int y);
	public IAsyncResult BeginInvoke(int x, int y,
	AsyncCallback cb, object state);
	public int EndInvoke(IAsyncResult result);
}
```

.Net Delegate 4

```
public delegate string MyDelegate(bool a, bool b, bool c);
```

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

MulticastDelegate

```
public abstract class MulticastDelegate : Delegate
{
    // Returns the list of methods "pointed to."
    public sealed override Delegate[] GetInvocationList();

    // Overloaded operators.
    public static bool operator ==(MulticastDelegate d1, MulticastDelegate d2);
    public static bool operator !=(MulticastDelegate d1, MulticastDelegate d2);

    // Used internally to manage the list of methods maintained by the
    delegate.
    private IntPtr _invocationCount;
    private object _invocationList;
}
```

Delegate

```
public abstract class Delegate : ICloneable, ISerializable
{
    // Methods to interact with the list of functions.
    public static Delegate Combine(params Delegate[] delegates);
    public static Delegate Combine(Delegate a, Delegate b);
    public static Delegate Remove(Delegate source, Delegate value);
    public static Delegate RemoveAll(Delegate source, Delegate value);

    // Overloaded operators.
    public static bool operator ==(Delegate d1, Delegate d2);
    public static bool operator !=(Delegate d1, Delegate d2);

    // Properties that expose the delegate target.
    public MethodInfo Method { get; }
    public object Target { get; }
}
```

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

Örnek

```
namespace SimpleDelegate
{
    // This delegate can point to any method,
    // taking two integers and returning an integer.
    public delegate int BinaryOp(int x, int y);


    // This class contains methods BinaryOp will
    // point to.
    public class SimpleMath
    {
        public static int Add(int x, int y)
        { return x + y; }

        public static int Subtract(int x, int y)
        { return x - y; }
    }
```

Örnek 2

```
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("***** Simple Delegate Example *****\n");
            // Create a BinaryOp delegate object that
            // "points to" SimpleMath.Add().
            BinaryOp b = new BinaryOp(SimpleMath.Add);

            // Invoke Add() method indirectly using delegate object.
            Console.WriteLine("10 + 10 is {0}", b(10, 10));
            Console.ReadLine();
        }
    }
}
```

Örnek 3

- Kitap: Sayfa 367-369 örneği uygulayınız.

Enabling Multicasting

```
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;
    }
    ...
}
------------------------------------------------------------------------------------------------
public void RegisterWithCarEngine( CarEngineHandler methodToCall )
{
    if (listOfHandlers == null)
    listOfHandlers = methodToCall;
    else
    Delegate.Combine(listOfHandlers, methodToCall);
}
```

Removing Targets from a Delegate’s Invocation List

```
public class Car
{
    ...
    public void UnRegisterWithCarEngine(CarEngineHandler methodToCall)
    {
        listOfHandlers -= methodToCall;
    }
}
```

Örnek

```
static void Main(string[] args)
{
    Car c1 = new Car("SlugBug", 100, 10);
    c1.RegisterWithCarEngine(new Car.CarEngineHandler(OnCarEngineEvent));

    // This time, hold onto the delegate object,
    // so we can unregister later.
    Car.CarEngineHandler handler2 = new Car.CarEngineHandler
    (OnCarEngineEvent2);
    c1.RegisterWithCarEngine(handler2);
    // Speed up (this will trigger the events).
    Console.WriteLine("***** Speeding up *****");

    for (int i = 0; i < 6; i++)
        c1.Accelerate(20);
    // Unregister from the second handler.
    c1.UnRegisterWithCarEngine(handler2);

    // We won't see the "uppercase" message anymore!
    Console.WriteLine("***** Speeding up *****");
    for (int i = 0; i < 6; i++)
    c1.Accelerate(20);
}
```

Method Group Conversion Syntax

```
Console.WriteLine("***** Method Group Conversion *****\n");
Car c1 = new Car();

// Register the simple method name.
//sadece method ismi ile kayıtlanma gerçekleşiyor
c1.RegisterWithCarEngine(CallMeHere);

Console.WriteLine("***** Speeding up *****");
for (int i = 0; i < 6; i++)
c1.Accelerate(20);


// Unregister the simple method name.
//sadece method ismi ile silinme gerçekleşiyor
c1.UnRegisterWithCarEngine(CallMeHere);
```

Generic Delegate

```
namespace GenericDelegate
{
    // This generic delegate can call any method
    // returning void and taking a single type parameter.
    public delegate void MyGenericDelegate<T>(T arg);
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("***** Generic Delegates *****\n");
            // Register targets.
            MyGenericDelegate<string> strTarget =
            new MyGenericDelegate<string>(StringTarget);
            strTarget("Some string data");
            MyGenericDelegate<int> intTarget =
            new MyGenericDelegate<int>(IntTarget);
            intTarget(9);
            Console.ReadLine();
        }
```

Generic Delegate 2

```
        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<>

```
static void Main(string[] args)
{
    Console.WriteLine("***** Fun with Action and Func *****");

    // Use the Action<> delegate to point to DisplayMessage.
    Action<string, ConsoleColor, int> actionTarget =
    new Action<string, ConsoleColor, int>(DisplayMessage);

    actionTarget("Action Message!", ConsoleColor.Yellow, 5);

    Console.ReadLine();
}
```

- Action<>
  - 16 adete kadar parametre alabilir.
  - return değeri void dir.

Func<>

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

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);
```

- Eğer return değeri istiyor isek Action<> yerine Func<> kullanıyoruz.
- Son parametre return türü oluyor.

Events

- Event oluşturma adımları
  - Parametreleri ile bir delegate tanımlanır
  - event kelimesi ile bir değişken tanımlanır.

```
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.
    public event CarEngineHandler Exploded;
    public event CarEngineHandler AboutToBlow;
    ...
}
```

Events 2

```
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);
        }
}
```

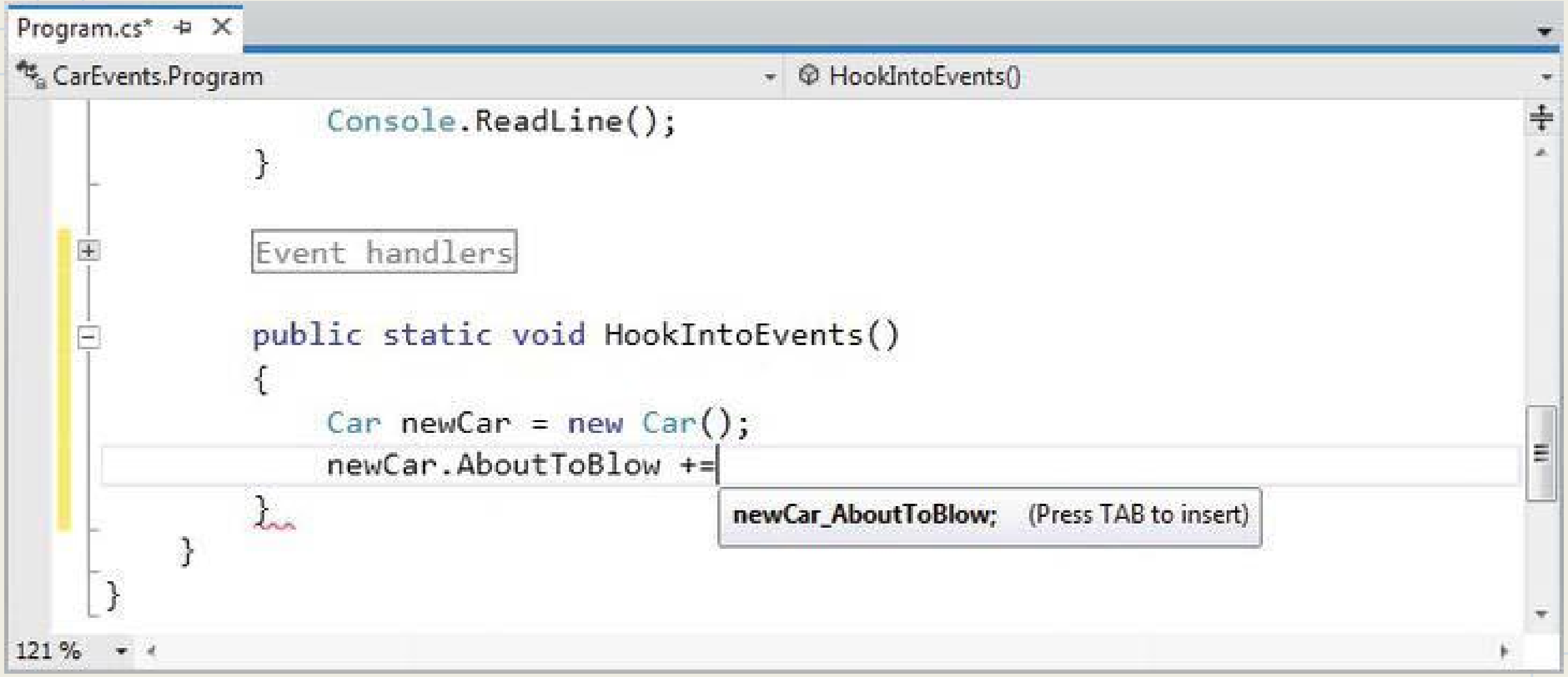
Listening to Incoming Events

```
// NameOfObject.NameOfEvent += new RelatedDelegate(functionToCall);
//
Car.CarEngineHandler d = new Car.CarEngineHandler
(CarExplodedEventHandler);
myCar.Exploded += d;

// NameOfObject.NameOfEvent -= new RelatedDelegate(functionToCall);
//
myCar.Exploded -= d;
```

- Kitap : Sayfa 381-383 Örneği uygulayınız.

Event & Visual Studio



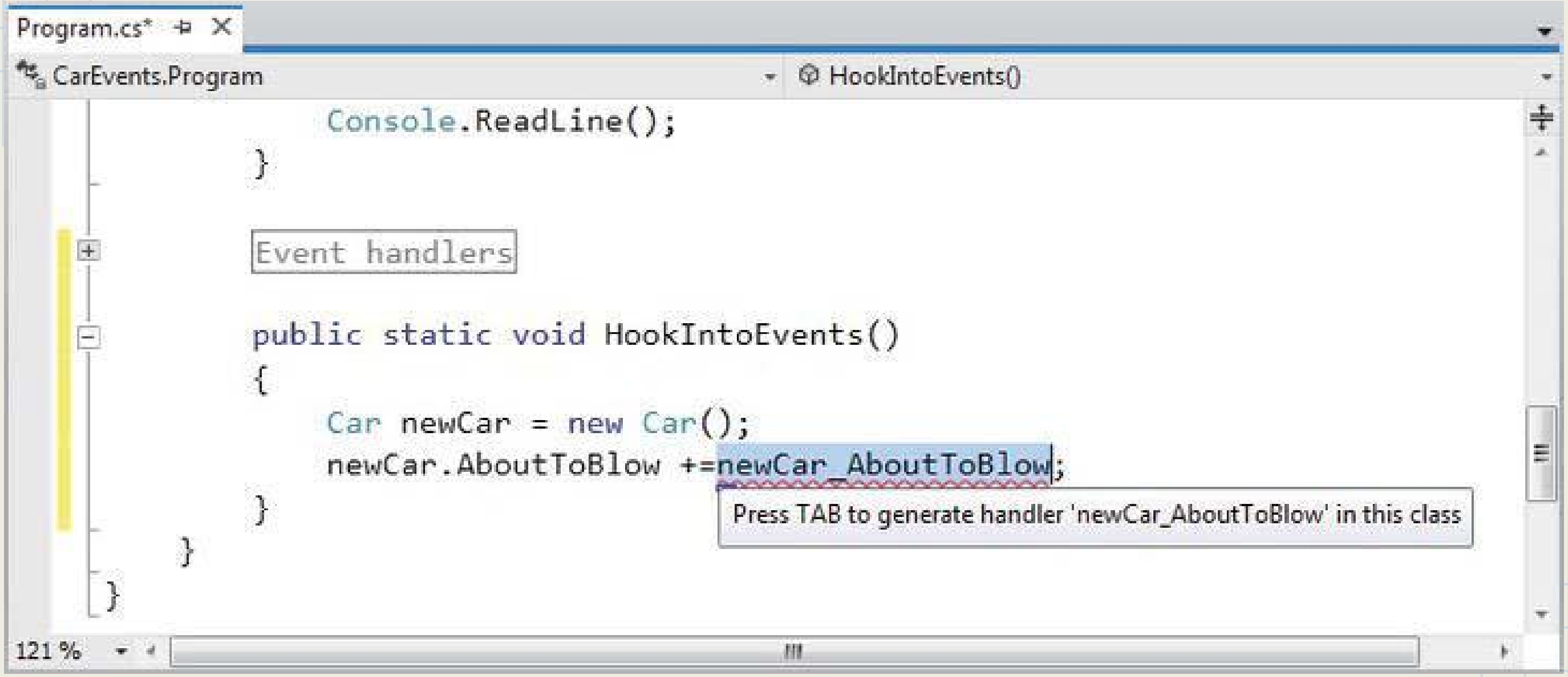
```
                Console.ReadLine();
            }

            Event handlers

            public static void HookIntoEvents()
            {
                Car newCar = new Car();
                newCar.AboutToBlow +=
            }
        }
}
```

newCar_AboutToBlow;   (Press TAB to insert)

Event & Visual Studio 2



```
            Console.ReadLine();
        }

        Event handlers

        public static void HookIntoEvents()
        {
            Car newCar = new Car();
            newCar.AboutToBlow +=newCar_AboutToBlow;
        }
    }
}
```

Press TAB to generate handler 'newCar_AboutToBlow' in this class

EventArgs

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

- Aşağıdaki hali ile sınıfımız Microsoft Event Pattern i ile uyumlu hale gelmiştir.

```
public class Car
{
	public delegate void CarEngineHandler(object sender, CarEventArgs e);
...
}
```

EventArgs 2

```
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..."));
    }...
}

public static void CarAboutToBlow(object sender, CarEventArgs e)
{
    // Just to be safe, perform a runtime check before casting.
    if (sender is Car)
    {
        Car c = (Car)sender;
        Console.WriteLine("Critical Message from {0}: {1}", c.PetName, e.
        msg);
    }
}
```

Generic EventHandler<T>

```
public class Car
{
public event EventHandler<CarEventArgs> Exploded;
public event EventHandler<CarEventArgs> AboutToBlow;
...
}

static void Main(string[] args)
{
    Console.WriteLine("***** Prim and Proper Events *****\n");
    // Make a car as usual.
    Car c1 = new Car("SlugBug", 100, 10);
    // Register event handlers.
    c1.AboutToBlow += CarIsAlmostDoomed;
    c1.AboutToBlow += CarAboutToBlow;
    EventHandler<CarEventArgs> d = new EventHandler<CarEventArgs>
    (CarExploded);
    c1.Exploded += d;
...
```

C# Anonymous Methods

```
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);
        };
        // This will eventually trigger the events.
        for (int i = 0; i < 6; i++)
        c1.Accelerate(20);
        Console.ReadLine();
}
```

Anonymous Methods Scope

- An anonymous method cannot access ref or out parameters of the defining method.
- An anonymous method cannot have a local variable with the same name as a local variable in the outer method.
- An anonymous method can access instance variables (or static variables, as appropriate) in the outer class scope.
- An anonymous method can declare local variables with the same name as outer class member variables (the local variables have a distinct scope and hide the outer class member variables).

Anonymous Methods Scope 2

```
Console.WriteLine("***** Anonymous Methods *****\n");
int aboutToBlowCounter = 0;
// Make a car as usual.
Car c1 = new Car("SlugBug", 100, 10);

// Register event handlers as anonymous methods.
c1.AboutToBlow += delegate
{
    aboutToBlowCounter++;
    Console.WriteLine("Eek! Going too fast!");
};

c1.AboutToBlow += delegate(object sender, CarEventArgs e)
{
    aboutToBlowCounter++;
    Console.WriteLine("Critical Message from Car: {0}", e.msg);
};
```

Lambda Expressions

- Delegate kullanımını daha basit hale getiren bir yazım şeklidir.

```
// Target for the Predicate<> delegate.
static bool IsEvenNumber(int i)
{
    // Is it an even number?
    return (i % 2) == 0;
}

static void TraditionalDelegateSyntax()
{
    // Make a list of integers.
    List<int> list = new List<int>();
    list.AddRange(new int[] { 20, 1, 4, 8, 9, 44 });

    // Call FindAll() using traditional delegate syntax.
    Predicate<int> callback = new Predicate<int>(IsEvenNumber);
    List<int> evenNumbers = list.FindAll(callback);

}
```

Lambda Expressions 2

```
// Now, use an anonymous method.

List<int> evenNumbers = list.FindAll(
        delegate(int i)
        {
                return (i % 2) == 0;
        }
);


// Now, use a C# lambda expression.
List<int> evenNumbers = list.FindAll(i => (i % 2) == 0);
```

Lambda Expressions 3

```
// Now process each argument within a group of
// code statements.

List<int> evenNumbers = list.FindAll(
(i) =>
{
      Console.WriteLine("value of i is currently: {0}", i);
      bool isEven = ((i % 2) == 0);
      return isEven;
}

);
```

Lambda Expressions 4

```
public class SimpleMath
{
	public delegate void MathMessage(string msg, int result);
	private MathMessage mmDelegate;

	public void SetMathHandler(MathMessage target)
	{
		mmDelegate = target;
	 }

	public void Add(int x, int y)
	{
		if (mmDelegate != null)
			mmDelegate.Invoke("Adding has completed!", x + y);
	}
}
```

Lambda Expressions 5

```
SimpleMath m = new SimpleMath();

m.SetMathHandler((msg, result) =>
{Console.WriteLine("Message: {0}, Result: {1}", msg, result);});

// This will execute the lambda expression.
m.Add(10, 10);
Console.ReadLine();
```

```
// Prints "Enjoy your string!" to the console.
VerySimpleDelegate d = new VerySimpleDelegate( () => {return "Enjoy
your string!";} );
Console.WriteLine(d());
```

Lambda Expressions 6

```
static void Main(string[] args)
{
    Console.WriteLine("***** More Fun with Lambdas *****\n");
    // Make a car as usual.
    Car c1 = new Car("SlugBug", 100, 10);

    // Hook into events with lambdas!
    c1.AboutToBlow += (sender, e) => { Console.WriteLine(e.msg);};
    c1.Exploded += (sender, e) => { Console.WriteLine(e.msg); };

    // Speed up (this will generate the events).
    Console.WriteLine("\n***** Speeding up *****");
    for (int i = 0; i < 6; i++)
    c1.Accelerate(20);
    Console.ReadLine();
}
```

Lambda Expressions 7

- ArgumentsToProcess => StatementsToProcessThem

Ödev ve Proje

- Aşağıdaki bölümler 2şer sayfa (toplam 4)özetlenecek.
  - Chapter 9: Collections and Generics 321
  - Chapter 10: Delegates, Events, and Lambda Expressions 359
- Proje
  - ArrayList, Hashtable,Queue kullanan örnek uygulamalar yazınız.
  - Generic bir class oluşturunuz ve kullanan bir uygulama yazınız.