C#

By

Mazin Hakeem

CSCI 5448 Object Oriented Analysis & Design

Grad Student Presentation

Contents (1)

Contents (2)

Contents (3)

About C#

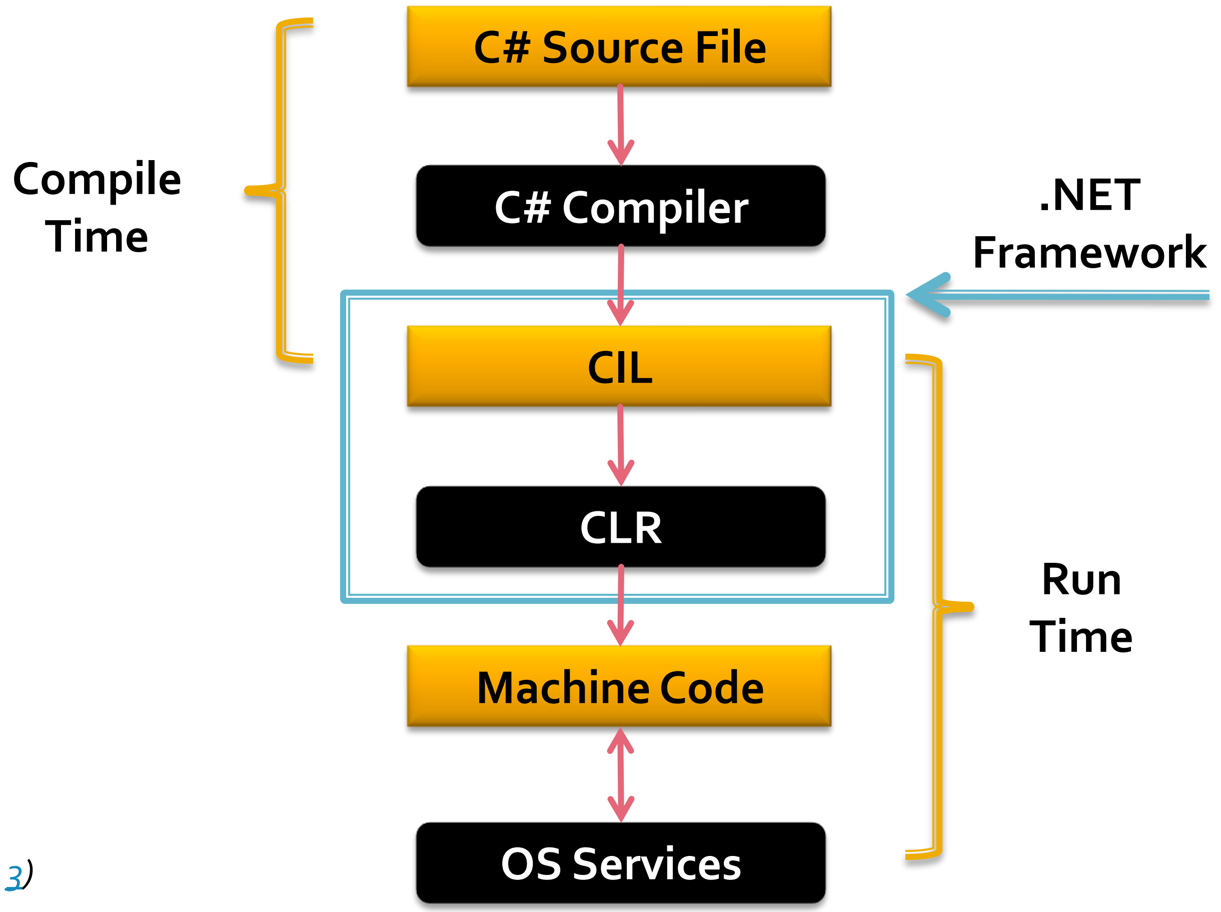
What is C#?

- Pronounced: **C Sharp**
- Called **Visual C#**, or just **C#**
- Developed by Microsoft for the .NET framework initiative
- Is a pure object-oriented programming language
- Also, a multi-paradigm programming language (*imperative, declarative, functional, generic, & component oriented*) [1]

More C#

- Is safer than C++
  - Is type-safe
  - No misuse of pointers; must use the “unsafe” mode to explicitly deal with pointers
  - Has a Garbage Collector (GC); Memory management is implicit
- In the .NET framework, C# is complied into a binary-based intermediate language, Common Intermediate Language (CIL), then the framework converts it to machine code using Common Language Runtime (CLR) (2 & 3)

C# in the .NET Framework



(Source: 2 & 3)

Where C# is used?

- Desktop apps
- Websites (w/ ASP .NET)
- Web services
- Mobile phones (WM & WP7)
- DB apps (w/ ADO .NET)
- Distributed components
- UI design [Desktop/Web] (w/ Silverlight)
- ... and many more

Version History (1)

- 1.0 with .NET 1.0 w/ VSDN 2002 (2002)
- 1.2 with .NET 1.1 w/ VSDN 2003 (2003)
- 2.0 with .NET 2.0 w/ VSDN 2005 (2005)
- 3.0 with .NET 3.5 w/ VSDN 2008 (2007)
- 4.0 with .NET 4.0 w/ VSDN 2010 (2010)

- VSDN → Visual Studio .NET
- In each version after 1.2, a lot of new features were added to the language

Version History (2)

- C# 2.0 *(1 & 4)*
  - Generics
  - Partial types
  - Anonymous methods
  - Iterators
  - Nullable types

Version History (3)

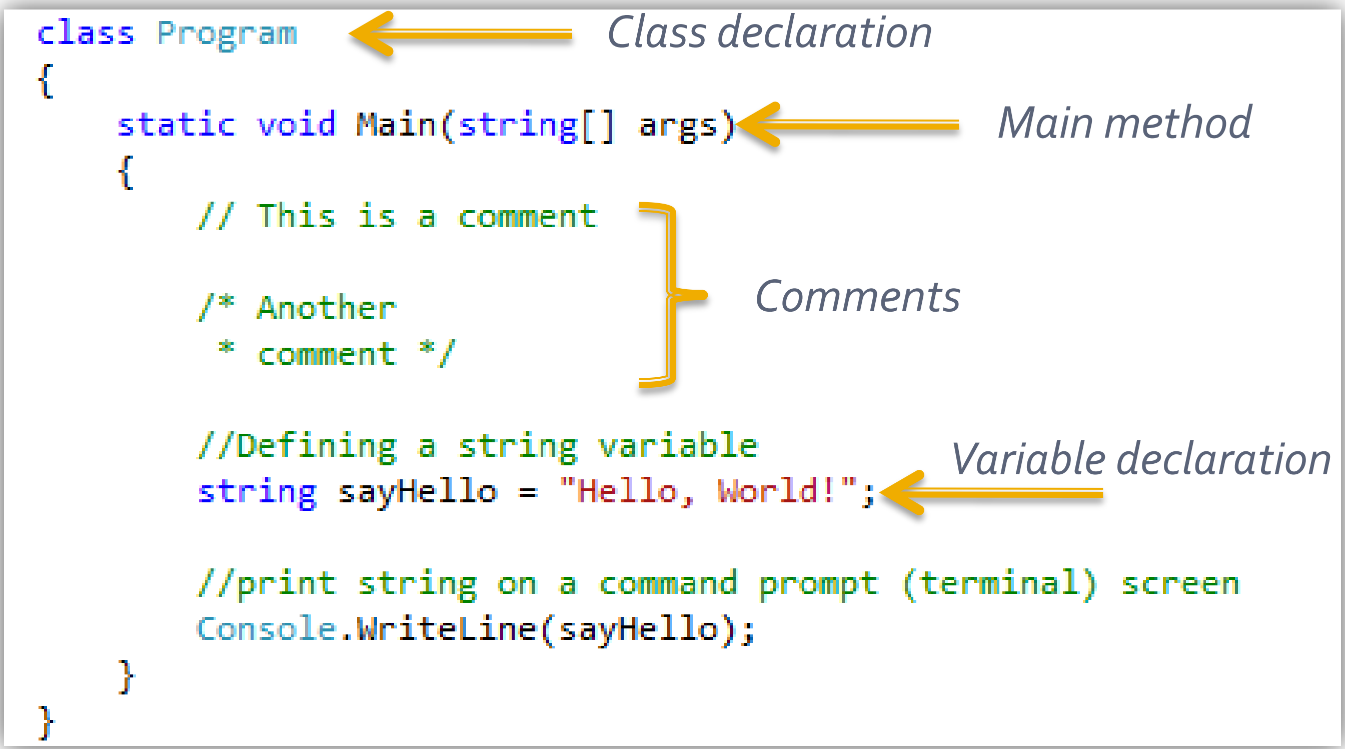
- C# 3.0 *(1 & 4)*
  - Implicitly typed local variables
  - Object and collection initializers
  - Auto-Implemented properties
  - Anonymous types
  - Extension methods
  - Query expressions
  - Lambda expressions
  - Expression trees

Version History (4)

- C# 4.0 *(1 & 5 & 6)*
  - Dynamic binding
  - Named and optional arguments
  - Generic co- and contravariance
- On the next coming slides, a number of features introduced in these versions will be covered

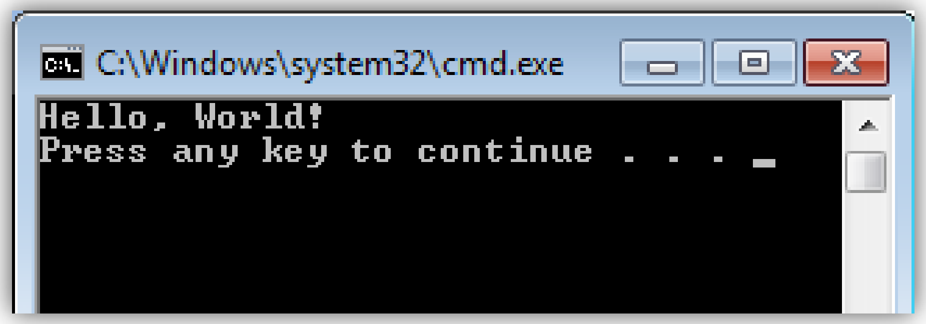
The Syntax (1)

- Very similar to C++ & Java



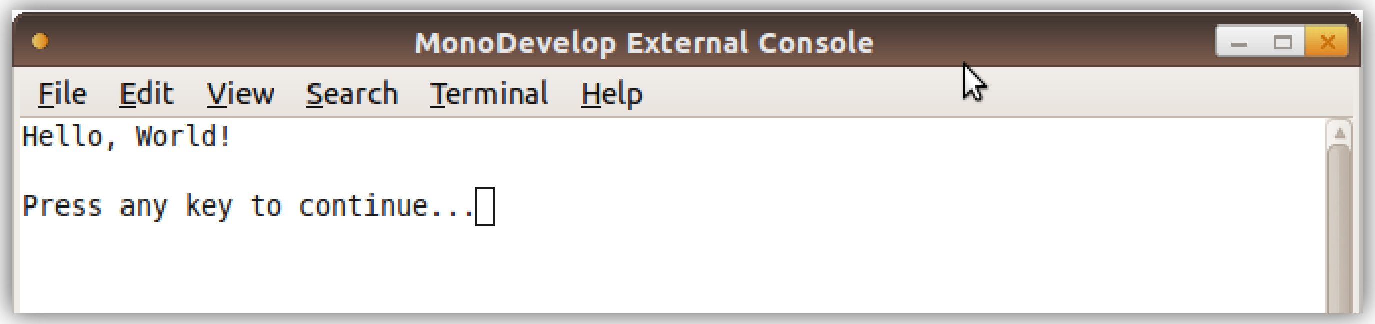
```
class Program
{
    static void Main(string[] args)
    {
        // This is a comment

        /* Another
         * comment */

        //Defining a string variable
        string sayHello = "Hello, World!";

        //print string on a command prompt (terminal) screen
        Console.WriteLine(sayHello);
    }
}
```

The Syntax (2)



```
Hello, World!
Press any key to continue . . . _
```

The result using VSDN 2010 Professional on Windows 7



```
Hello, World!

Press any key to continue...
```

The same result using Mono on Ubuntu 10.10 (Linux)

“Popular” IDEs

- C# is mainly used to develop under the .NET framework environment for MS Windows®
- Mono allowed cross-platform development
- The “popular” IDEs:
  - Visual Studio .NET
    - For Windows XP to 7
    - Free (limited) version (Express Edition) {since 2005}
    - Various paid versions (Standard, Pro, Team, etc.)
  - Mono
    - Is open source and free
    - Cross-platform (Win, Mac, and various Linux distros)

Some C# Features

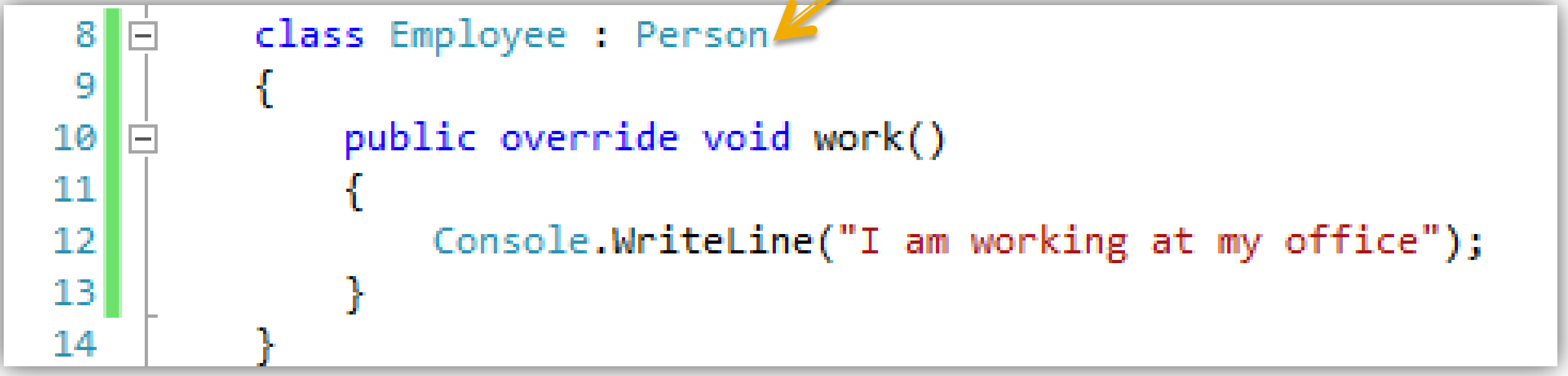
Object-Orientation

- Since C# is an object-oriented language, then all object-oriented concepts are supported
  - Abstraction
  - Encapsulation
  - Inheritance
  - Polymorphism

Inheritance

- C# allows single class inheritance only
- Use colon “ : ”

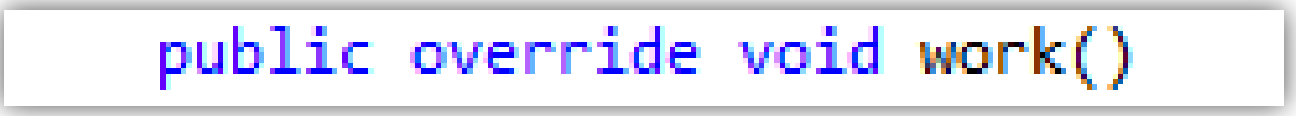
Class **Employee** inherits from class **Person**



```
class Employee : Person
{
    public override void work()
    {
        Console.WriteLine("I am working at my office");
    }
}
```

Polymorphism (1)

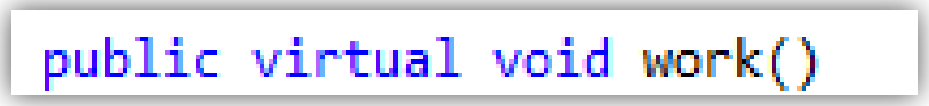
- To override an inherited method for the polymorphic behavior, the "`override`" keyword must be written within the method declaration in the inherited class



```
public override void work()
```

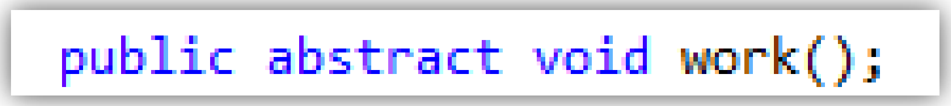
Polymorphism (2)

- Must declare the function to be overridden in the base class first
  - by using "`virtual`" keyword for a regular class



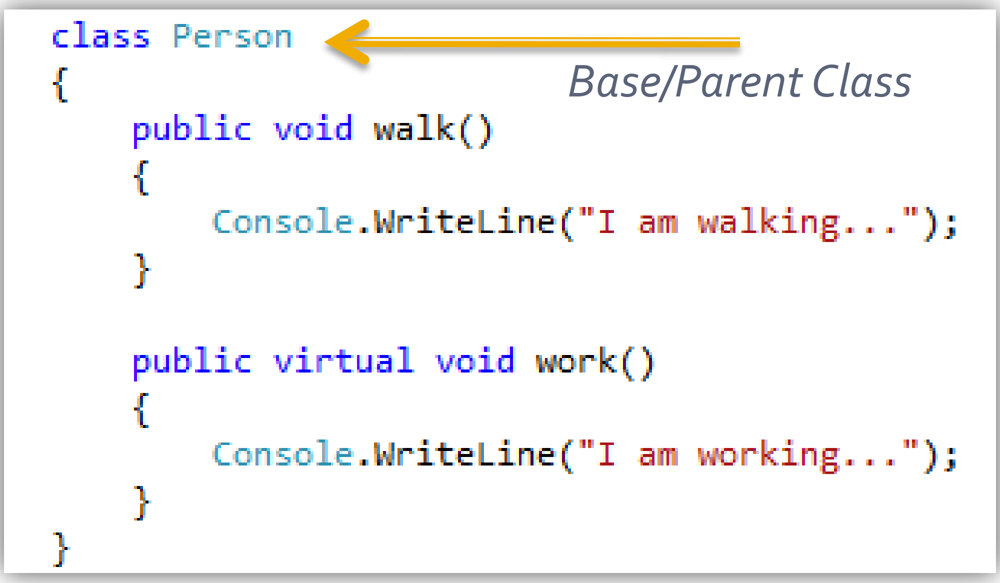
```
public virtual void work()
```

  - or, by defining an abstract method in an abstract class



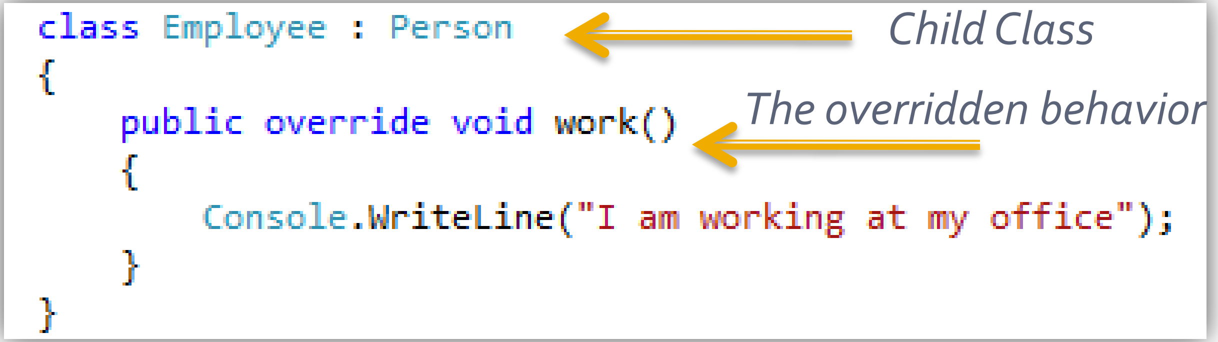
```
public abstract void work();
```

Polymorphism (3)



```
class Person
{
    public void walk()
    {
        Console.WriteLine("I am walking...");
    }

    public virtual void work()
    {
        Console.WriteLine("I am working...");
    }
}
```

Base/Parent Class

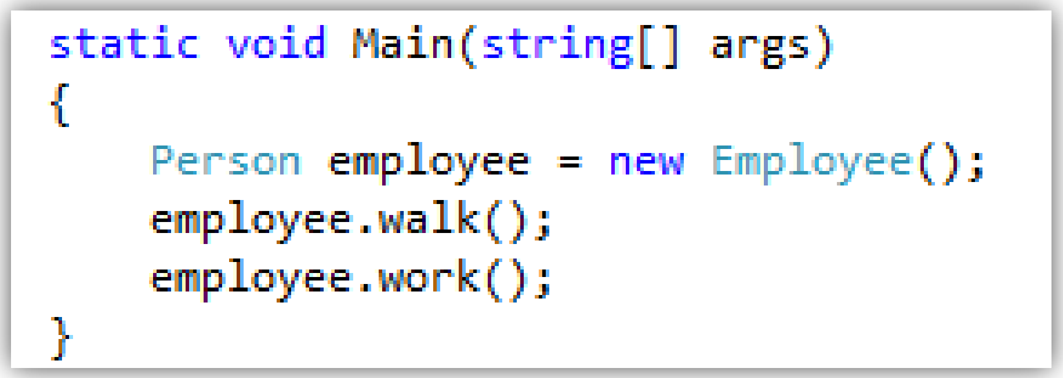


```
class Employee : Person
{
    public override void work()
    {
        Console.WriteLine("I am working at my office");
    }
}
```

Child Class

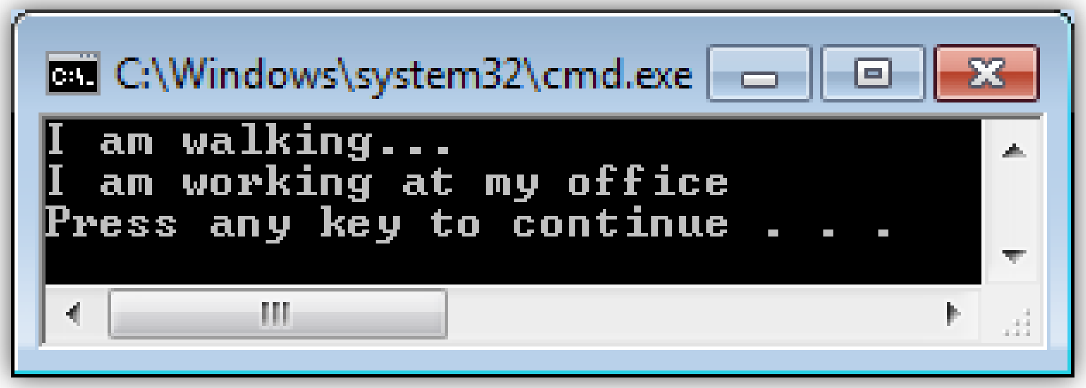
The overridden behavior

Polymorphism (4)



```
static void Main(string[] args)
{
    Person employee = new Employee();
    employee.walk();
    employee.work();
}
```

Main method



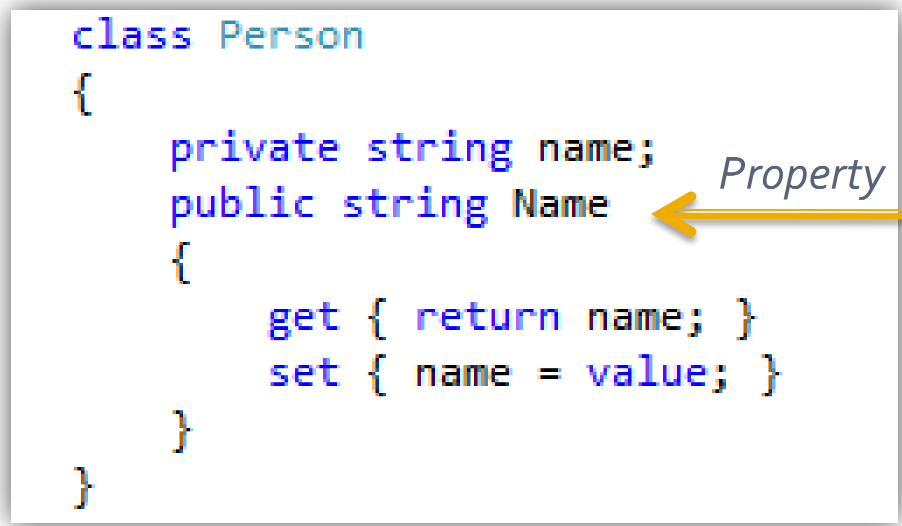
```
I am walking...
I am working at my office
Press any key to continue . . .
```

Result

Properties (1)

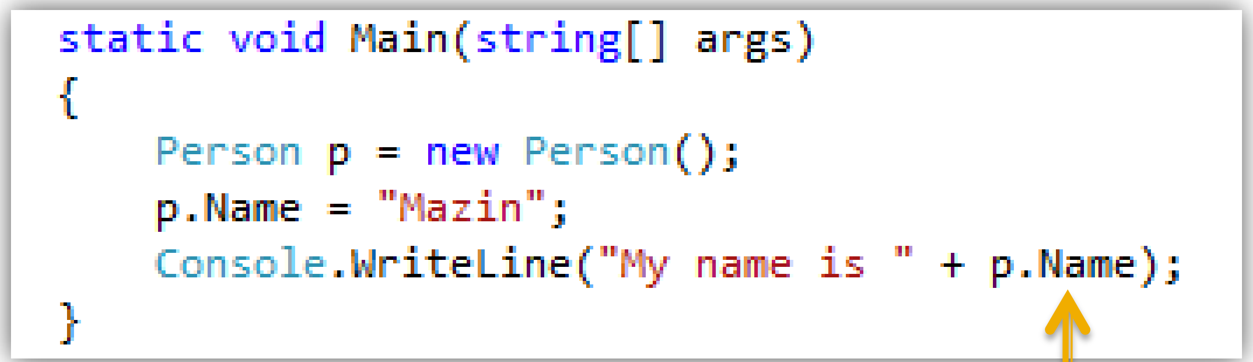
- "A property is a member that provides a flexible mechanism to read, write, or compute the value of a private field" (7)
- Properties act as public data members, but are methods called "accessors" (7)
- They represent getters and setters
- The private data is not exposed, but protected
- Provides a layer of abstraction & encapsulation (2 & 3)

Properties (2)



```
class Person
{
    private string name;
    public string Name
    {
        get { return name; }
        set { name = value; }
    }
}
```

Property

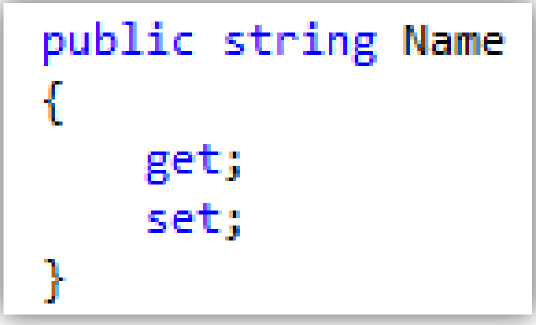


```
static void Main(string[] args)
{
    Person p = new Person();
    p.Name = "Mazin";
    Console.WriteLine("My name is " + p.Name);
}
```

Access to a property

Properties (3)

- Auto-Implemented Properties [(8)]
  - Introduced in C# 3.0
  - Used if there is not much code logic
  - No need to define private data members

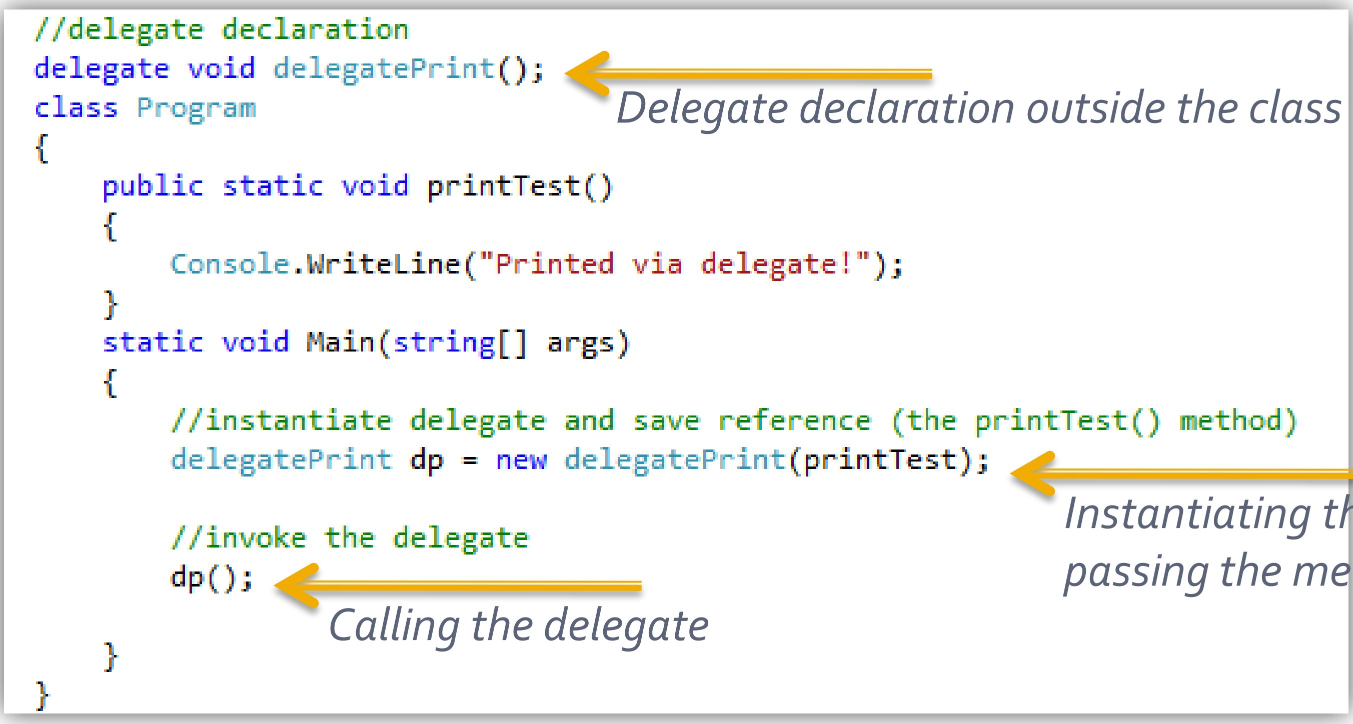


```
public string Name
{
      get;
      set;
}
```

Delegates (1)

- "A delegate can be thought of as an object containing an ordered list of methods with the same signature and return type" (2)
- Like C/C++ function pointers, but type-safe
- Declared outside the class structure w/ "`delegate`" keyword
- No method body
- Methods are passed as parameters; encapsulated inside the delegate object (9 & 10)
- Mostly used for UI control event handlers (e.g. Button, Text box, etc.) (similar to Listeners in Java)

Delegates (2)

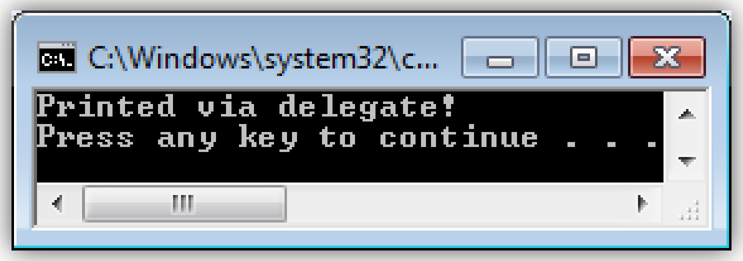


```
//delegate declaration
delegate void delegatePrint();
class Program
{
    public static void printTest()
    {
        Console.WriteLine("Printed via delegate!");
    }
    static void Main(string[] args)
    {
        //instantiate delegate and save reference (the printTest() method)
        delegatePrint dp = new delegatePrint(printTest);

        //invoke the delegate
        dp();

    }
}
```

Delegate declaration outside the class

Instantiating the delegate and passing the method

Calling the delegate



```
Printed via delegate!
Press any key to continue . . .
```

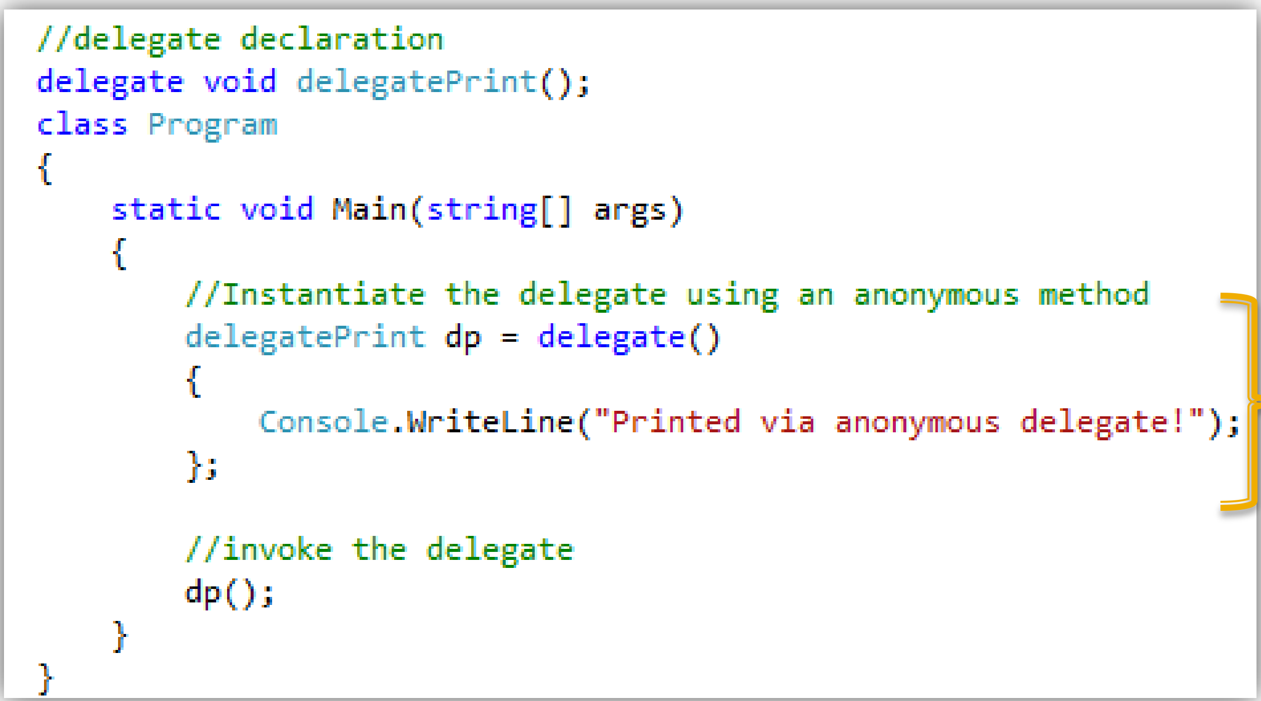
The result

(Source: 2 & 11)

Anonymous Methods (1)

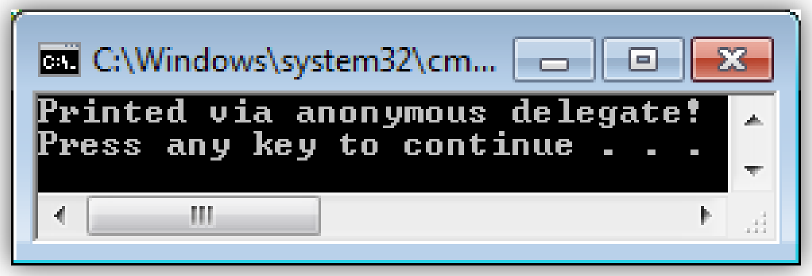
- The concept introduced in C# 2.0
- Also called "**Anonymous Delegates**" *(3 & 12)*
- We Declare a method when instantiating a delegate; "passing a code block as a delegate parameter" *(2 & 13)*
- Reduces the creation of a separate method
- Mostly used for a "one time" use of a method
- A bit similar to the "**Anonymous Classes**" concept in Java

Anonymous Methods (2)



```
//delegate declaration
delegate void delegatePrint();
class Program
{
    static void Main(string[] args)
    {
        //Instantiate the delegate using an anonymous method
        delegatePrint dp = delegate()
        {
            Console.WriteLine("Printed via anonymous delegate!");
        };

        //invoke the delegate
        dp();
    }
}
```

The structure of an Anonymous Method



```
Printed via anonymous delegate!
Press any key to continue . . .
```

The result

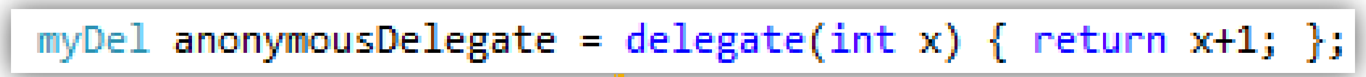
(Source: 14)

Lambda Expressions (1)

- Introduced in C# 3.0
- Another kind of "Anonymous Methods"
- Less verbose
- No need to mention the "`delegate`" keyword like in the regular "Anonymous Methods"
- Use the lambda operator "`=>`"; Is read "*goes to*" *(2)*

Lambda Expressions (2)

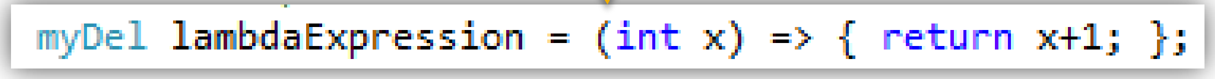
- Anonymous Method vs. Lambda Expression



```
myDel anonymousDelegate = delegate(int x) { return x+1; };
```

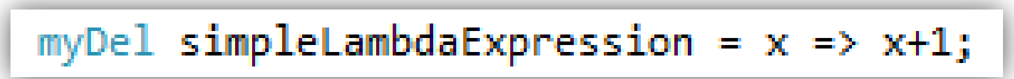
Anonymous Method

Simplified to



```
myDel lambdaExpression = (int x) => { return x+1; };
```

Lambda Expression



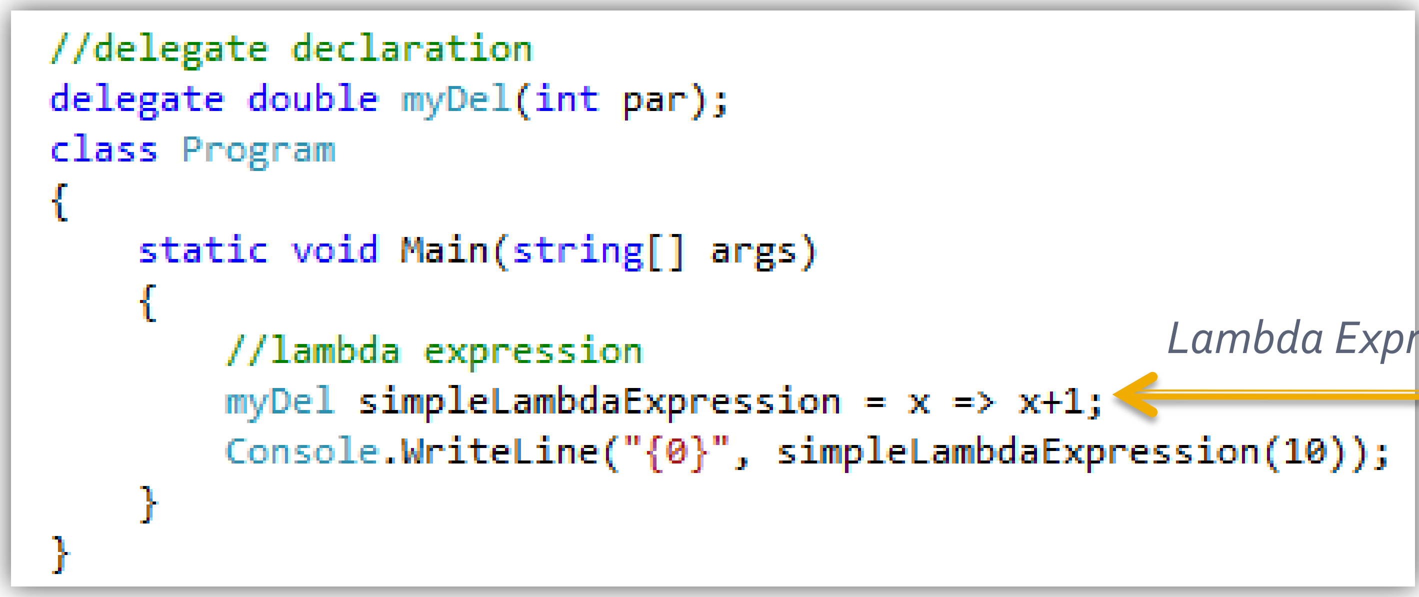
```
myDel simpleLambdaExpression = x => x+1;
```

A clean version of the Lambda Expression

- All of them produce the same result
- The last one is more clean, short and readable

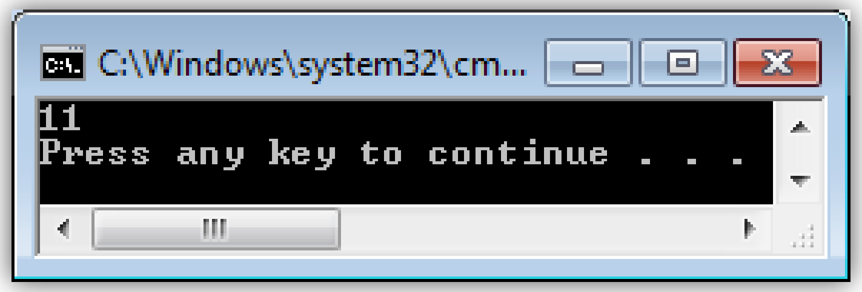
(Examples from: 2)

Lambda Expressions (3)



```
//delegate declaration
delegate double myDel(int par);
class Program
{
    static void Main(string[] args)
    {
        //lambda expression
        myDel simpleLambdaExpression = x => x+1;
        Console.WriteLine("{0}", simpleLambdaExpression(10));
    }
}
```

Lambda Expression

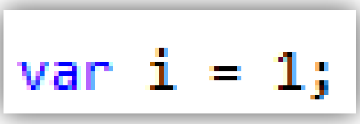


```
11
Press any key to continue . . .
```

The result

Implicitly Typed Local Variables

- Introduced in C# 3.0
- Variable types are not declared explicitly
- The "`var`" keyword is used to define variables
- The compiler infers the type from the initialized statement
- Similar to JavaScript's "`var`" variable declaration
- Variable must be initialized & can't be "`null`"
- Can't have more than one type defined



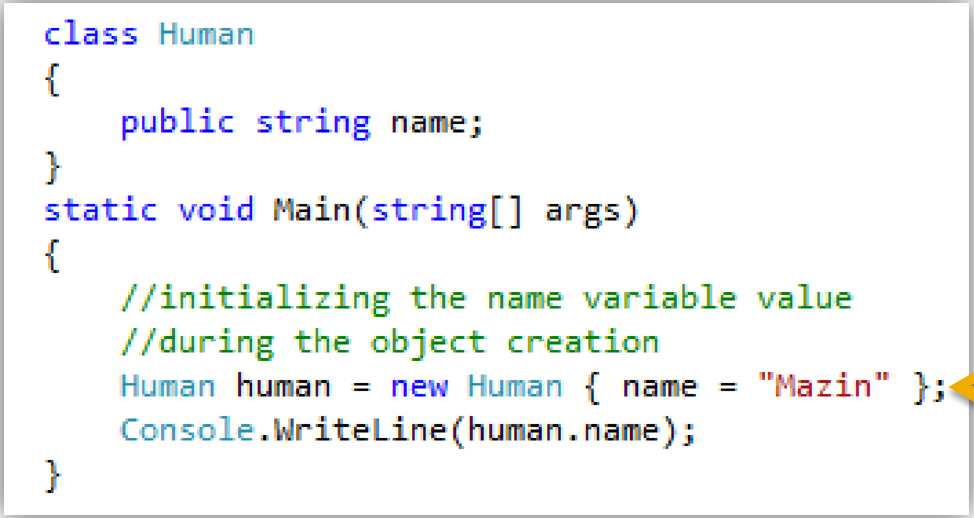
```
var i = 1;
```

*Variable "**`i`**" is compiled as type "`int`"*

(Source: 14 & 15)

Object Initializers

- Introduced in C# 3.0
- Used when there is no class constructor
- The idea is to assign values to any accessible property or field at the object's creation time



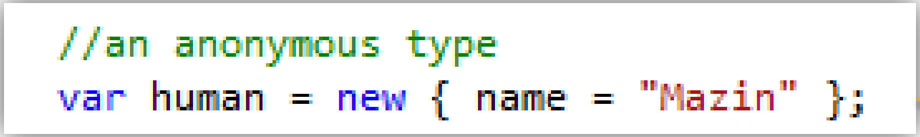
```
class Human
{
    public string name;
}
static void Main(string[] args)
{
    //initializing the name variable value
    //during the object creation
    Human human = new Human { name = "Mazin" };
    Console.WriteLine(human.name);
}
```

Initializing the variable at runtime

(Source: 2 & 16)

Anonymous Types

- Introduced in C# 3.0
- The concepts is to create unnamed class types
- Combines the "**Object Initializer**" concept to assign values to fields on creation time, & the "**Implicitly Typed Local Variable**" concept to let the compiler infer the variable type
- Anonymous Types are common in **LINQ**



```
//an anonymous type
var human = new { name = "Mazin" };
```

Anonymous Type which is inferred as a class by the compiler

(Source: 17)

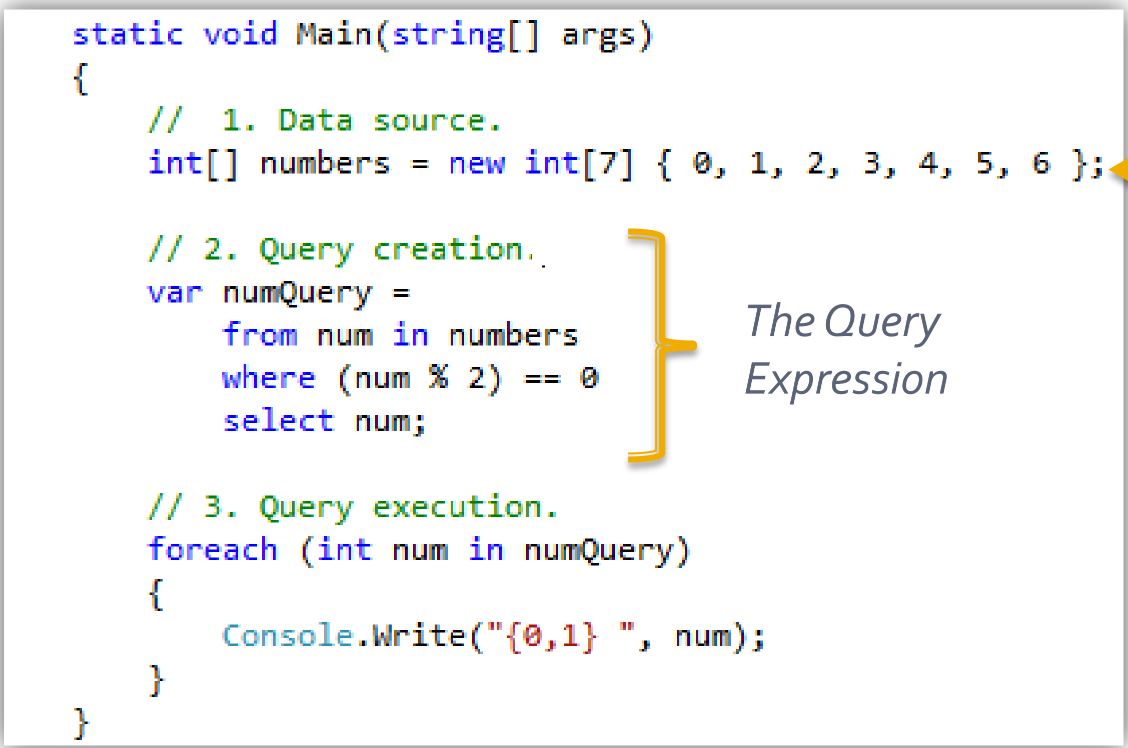
LINQ (1)

- “**Language Integrated Query**”
- Pronounced “**Link**”
- An extension for the .NET 3.5 framework
- Introduced in C# 3.0 in VSDN 2008
- Used to query data like DB queries [(2)]
- Similar to SQL (a.k.a. Query Expression) [(18)]

LINQ (2)

- Data could be represented in any object types (e.g. arrays, class objects), relational DBs, & XML
- Also, to manipulate any data source [3]
- Can perform filtering, ordering, grouping, & joining operations in a few lines of code [19]
- "**Anonymous Types**" & "**Implicitly Typed Local Variables**" concepts are used for the querying part of the code (Query Expression)

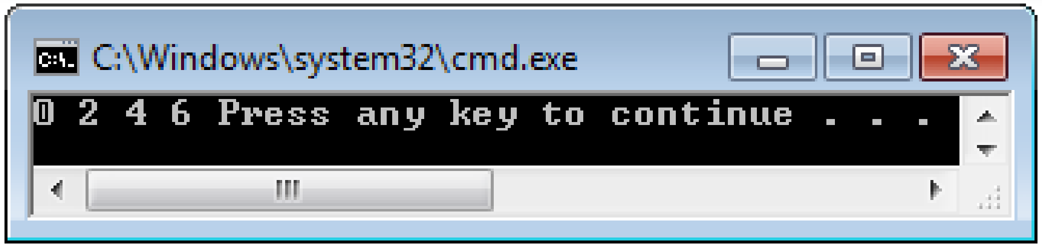
LINQ (3)



```
static void Main(string[] args)
{
    //  1. Data source.
    int[] numbers = new int[7] { 0, 1, 2, 3, 4, 5, 6 };

    // 2. Query creation.
    var numQuery =
        from num in numbers
        where (num % 2) == 0
        select num;

    // 3. Query execution.
    foreach (int num in numQuery)
    {
        Console.Write("{0,1} ", num);
    }
}
```

The data source (An array of integers)

The Query Expression



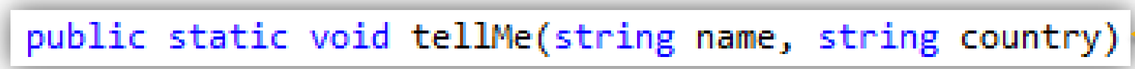
```
0 2 4 6 Press any key to continue . . .
```

The result

(Source: 20)

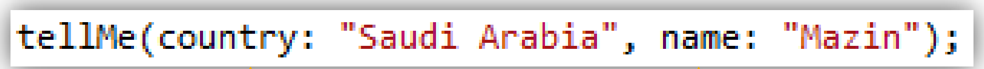
Named Arguments & Optional Parameters

- Introduced in C# 4.0
- Each is distinct, but useful together
- Used to reduce code & make it easy to code
- Named Arguments
  - No need to remember parameters' positions
  - Name the argument with its value using colon ":"



```
public static void tellMe(string name, string country)
```

A method w/ arguments



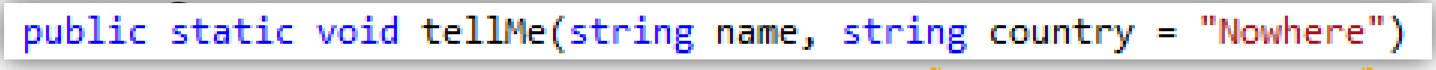
```
tellMe(country: "Saudi Arabia", name: "Mazin");
```

Passing arguments values by explicitly mentioning their names not in the original order of the actual method

(Source: 21)

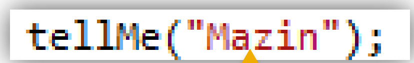
Named Arguments & Optional Parameters

- Optional Parameters
  - Can omit some arguments when passing to a method
  - No need for method overloads (defining the same method more than once but w/ different parameters)
  - Default values must be assigned last in the method



```
public static void tellMe(string name, string country = "Nowhere")
```

Declaring the optional argument in the method by assigning a default value



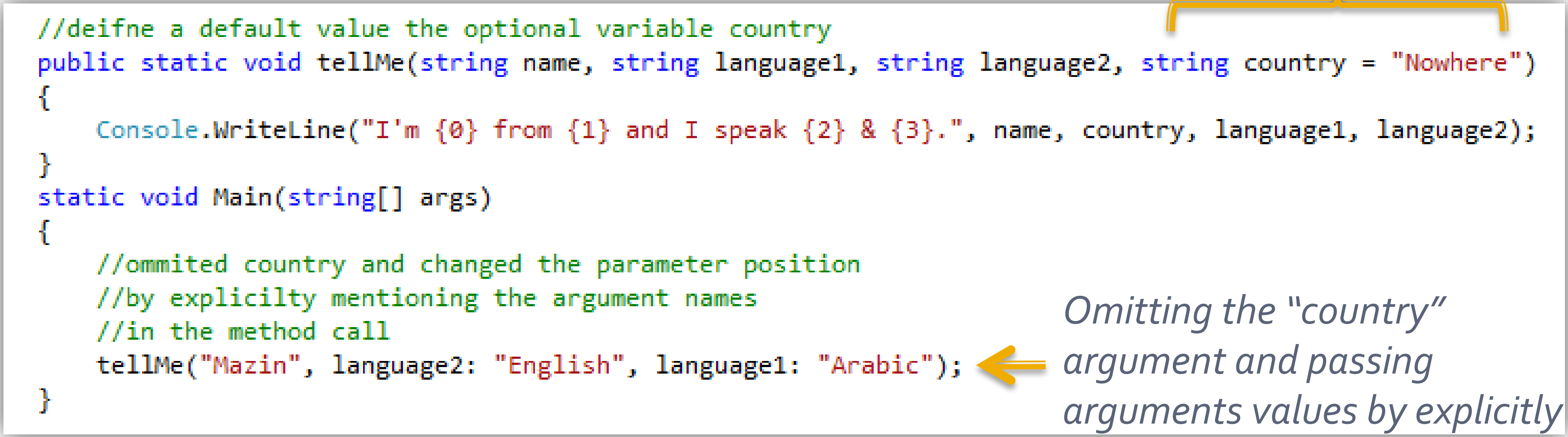
```
tellMe("Mazin");
```

Omitted an argument (country)

(Source: 22)

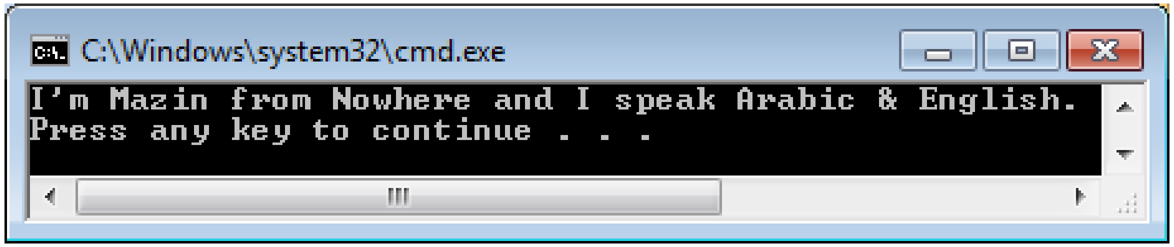
Named Arguments & Optional Parameters

Declaring the optional argument in the method by assigning a default value



```
//deifne a default value the optional variable country
public static void tellMe(string name, string language1, string language2, string country = "Nowhere")
{
    Console.WriteLine("I'm {0} from {1} and I speak {2} & {3}.", name, country, language1, language2);
}
static void Main(string[] args)
{
    //ommited country and changed the parameter position
    //by explicilty mentioning the argument names
    //in the method call
    tellMe("Mazin", language2: "English", language1: "Arabic");
}
```

Omitting the "country" argument and passing arguments values by explicitly mentioning their names not in original order



C:\Windows\system32\cmd.exe

```
I'm Mazin from Nowhere and I speak Arabic & English.
Press any key to continue . . .
```

The result

Conclusion

- C# is an Object-Oriented language
- Is now a cross-platform language
- Lots of features have evolved and added since the 1st version in 2002
- The programmer can write readable, few lines of code
- Getters & setters are defined in a single "accessor" method called "**Property**"
- Provides on-the-fly variable, method, & class creation
- No more method overloads or remembering arguments positions in a method w/ Named & Optional Arguments

References (1)

1. http://en.wikipedia.org/wiki/C_Sharp_(programming_language)
2. Illustrated C# 2008, *Apress, ISBN: 978-1590599549*
3. Beginning C# 2008 from Novice to Professional, *Apress, ISBN: 978-1590598696*
4. http://en.csharp-online.net/CSharp_Overview
5. http://msdn.microsoft.com/en-us/magazine/ff796223.aspx
6. http://msdn.microsoft.com/en-us/vcsharp/ff628440.aspx
7. http://msdn.microsoft.com/en-us/library/x9fsa0sw.aspx
8. http://www.csharp-station.com/Tutorials/Lesson10.aspx
9. http://msdn.microsoft.com/en-us/library/ms173171(v=vs.80).aspx
10. http://msdn.microsoft.com/en-us/library/aa288459(v=vs.71).aspx
11. http://www.akadia.com/services/dotnet_delegates_and_events.html

References (2)

12. http://www.switchonthecode.com/tutorials/csharp-tutorial-anonymous-delegates-and-scoping
13. http://msdn.microsoft.com/en-us/library/0yw3tz5k(v=vs.80).aspx
14. http://msdn.microsoft.com/en-us/library/bb384061.aspx
15. http://www.dotnetfunda.com/articles/article631-new-features-in-csharp-2008-.aspx
16. http://msdn.microsoft.com/en-us/library/bb384062.aspx
17. http://msdn.microsoft.com/en-us/library/bb397696.aspx
18. Beginning C# 3.0, Wrox, ISBN: 978-0470261293
19. C# in Depth, Manning, ISBN: 1933988363
20. http://msdn.microsoft.com/en-us/library/bb397676.aspx
21. http://msdn.microsoft.com/en-us/library/dd264739.aspx
22. http://msdn.microsoft.com/en-us/vcsharp/ff628440.aspx#argsparams