C#

By

Mazin Hakeem

CSCI 5448 Object Oriented Analysis & Design

Grad Student Presentation
Contents (1)

- About C#
  - What is C#?
  - More C#
  - C# in the .NET framework
  - Where C# is used?
  - Version History
  - The Syntax
  - “Popular” IDEs

C# by Mazin Hakeem
Some C# Features

- Object-Oriented Programming
  - Inheritance
  - Polymorphism
- Properties
- Delegates
- Anonymous Methods
- Lambda Expressions
- Implicitly Typed Local Variables
Some C# Features (Continued)
- Object Initializers
- Anonymous Types
- LINQ
- Named Arguments & Optional Parameters

Conclusion

References
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 compiled 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

Compile Time

C# Source File → C# Compiler → CIL → CLR → Machine Code → OS Services

Run Time

(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
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
Very similar to C++ & Java

```csharp
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 result using VSDN 2010 Professional on Windows 7

The same result using Mono on Ubuntu 10.10 (Linux)

C# by Mazin Hakeem
“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
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`
To override an inherited method for the polymorphic behavior, the "override" keyword must be written within the method declaration in the inherited class

```csharp
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
    ```csharp
    public virtual void work();
    ```
  - or, by defining an abstract method in an abstract class
    ```csharp
    public abstract void work();
    ```
Polymorphism (3)

Base/Parent Class

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

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

Child Class

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

The overridden behavior

C# by Mazin Hakeem
Polymorphism (4)

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

Main method

Result

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

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

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

```csharp
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 with "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 outside the class:
```csharp
(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();
    }
}
```

Instantiating the delegate and passing the method:
```
(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();
    }
}
```

Calling the delegate:
```
(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();
    }
}
```

The result:
```
(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();
    }
}
```

(Source: 2 & 11)
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)

The structure of an Anonymous Method

C# by Mazin Hakeem

(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)
Anonymous Method vs. Lambda Expression

Anonymous Method

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

Simplified to

Lambda Expression

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

A clean version of the Lambda Expression

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

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

(Examples from: [2])
Lambda Expressions (3)

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

Lambda Expression

The result

11
Press any key to continue . . .
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

```csharp
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);
}
```

(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

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

(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)
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)
The result

The data source (An array of integers)

The Query Expression

(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)

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

(Source: 21)
Optional Parameters

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

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

tellMe("Mazin");
```

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

(Source: Mazin Hakeem)
Named Arguments & Optional Parameters

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

```csharp
// define 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)
{
    // omitted country and changed the parameter position
    // by explicitly 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

The result

I'm Mazin from Nowhere and I speak Arabic & English. Press any key to continue . . .
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
4. [http://en.csharp-online.net/CSharp_Overview](http://en.csharp-online.net/CSharp_Overview)
11. [http://www.akadia.com/services/dotnet_delegates_and_events.html](http://www.akadia.com/services/dotnet_delegates_and_events.html)