C# and ASP.NET for Experienced Developers

https://www.accelebrate.com
(877) 849-1850 ❖ info@accelebrate.com

Customized Technical Training

Accelebrate
ACCELERATED LEARNING, CELEBRATED RESULTS®

On-Site, Customized Private Training

Don't settle for a one-size-fits-all class! Let Accelebrate tailor a private class to your group's goals and experience. Classes can be delivered at your site or online (or a combination of both) worldwide. Visit our web site at **https://www.accelebrate.com** and contact us at **sales@accelebrate.com** for details.

Public Online Training

Need to train just 1-3 people? Attend one of our regularly scheduled, live, instructor-led public online classes. Class sizes are small (typically 3-6 attendees) and you receive just as much hands-on time and individual attention from your instructor as our private classes. For course dates, times, outlines, pricing, and registration, visit **https://www.accelebrate.com/public-training-schedule.**

Newsletter

Want to find out about our latest class offerings? Subscribe to our newsletter **https://www.accelebrate.com/newsletter**.

Blog

Get insights and tutorials from our instructors and staff! Visit our blog, **https://www.accelebrate.com/blog** and join the discussion threads and get feedback from our instructors!

Learning Resources

Get access to learning guides, tutorials, and past issues of our newsletter at the Accelebrate library, **https://www.accelebrate.com/library**.

Call us for a training quote!

877 849 1850

Accelebrate, Inc. was founded in 2002 with the goal of delivering **private training** that rapidly achieves participants' goals. Each year, our experienced instructors deliver hundreds of classes online and at client sites all over the US, Canada, and abroad. We pride ourselves on our **instructors' real-world experience** and ability to **adapt the training** to your team and their objectives. We offer a wide range of topics, including:

- Angular, React, and Vue
- JavaScript
- Data Science using R, Python, & Julia
- Excel Power Query
- Power BI & Tableau
- .NET & VBA programming
- SharePoint & Microsoft 365
- DevOps & CI/CD
- iOS & Android Development
- PostgreSQL, Oracle, and SQL Server
- Java, Spring, and Groovy
- Agile & IT Leadership
- Web/Application Server Admin
- HTML5 & Mobile Web Development
- AWS, Azure, and Cloud Computing
- Adobe & Articulate
- Docker, Kubernetes, Ansible, & Git
- Software Design and Testing
- AND MORE *(see back)*

"World class training with experts who know their field and have a passion for teaching. Our team is walking away with a wealth of knowledge that we look forward to bringing back to our daily work."

— Keith, Silicon Valley Bank

Like us on Facebook + Follow us on Twitter + Watch us on YouTube

Visit our website for a complete list of courses!

Adobe & Articulate
Adobe Captivate
Adobe Presenter
Articulate Storyline / Studio
Camtasia
RoboHelp

AWS, Azure, & Cloud
AWS
Azure
Cloud Computing
Google Cloud
OpenStack
VMware

Big Data
Alteryx
Apache Spark
Teradata
Snowflake SQL

Data Science and RPA
Blue Prism
Data Literacy
Django
Julia
Machine Learning
MATLAB
Minitab
Python
R Programming
SPSS
Tableau
UiPath

Database & Reporting
BusinessObjects
Crystal Reports
Excel Power Query
MongoDB
MySQL
NoSQL Databases
Oracle
Oracle APEX
Power BI
PivotTable and PowerPivot
PostgreSQL
SQL Server
Vertica Architecture & SQL

DevOps, CI/CD & Agile
Agile
Ansible
Chef
DEI
Docker
Git
Gradle Build System
IT Leadership
ITIL
Jenkins
Jira & Confluence
Kubernetes
Linux
Microservices
OpenShift
Red Hat
Software Design

Java
Apache Maven
Apache Tomcat
Groovy and Grails
Hibernate
Java & Web App Security
JavaFX
JBoss
Oracle WebLogic
Scala
Selenium & Cucumber
Spring Boot
Spring Framework

JS, HTML5, & Mobile
Angular
Apache Cordova
CSS
D3.js
Flutter
HTML5
iOS/Swift Development
JavaScript
MEAN Stack
Mobile Web Development
Node.js & Express
React & Redux
Svelte
Swift
Xamarin
Vue

Microsoft & .NET
.NET Core
ASP.NET
Azure DevOps
C#
Design Patterns
Entity Framework Core
IIS
Microsoft Dynamics CRM
Microsoft Exchange Server
Microsoft 365
Microsoft Power Platform
Microsoft Project
Microsoft SQL Server
Microsoft System Center
Microsoft Windows Server
PowerPivot
PowerShell
VBA
Visual C++/CLI
Visual Studio
Web API

Security
.NET Web App Security
C and C++ Secure Coding
C# & Web App Security
Linux Security Admin
Python Security
Secure Coding for Web Dev
Spring Security

SharePoint
Power Automate & Flow
SharePoint Administrator
SharePoint Developer
SharePoint End User
SharePoint Online
SharePoint Site Owner

SQL Server
Azure SQL Data Warehouse
Business Intelligence
Performance Tuning
SQL Server Administration
SQL Server Development
SSAS, SSIS, SSRS
Transact-SQL

Teleconferencing Tools
Adobe Connect
GoToMeeting
Microsoft Teams
WebEx
Zoom

Web/Application Server
Apache httpd
Apache Tomcat
IIS
JBoss
Nginx
Oracle WebLogic

Other
C++
Go Programming
Project Management
Regular Expressions
Ruby on Rails
Rust
Salesforce
XML

*Visit **www.accelebrate.com/newsletter** to sign up and receive our **newsletters** with information about **new courses**, **free webinars**, **tutorials**, and **blog articles**.*

Call us for a training quote! 877 849 1850 (US/Canada) or +1 678 648 3133

C# and ASP.NET for Experienced Developers

Agenda

- Introduction to .NET
- C# Language
- Database Access
- ASP.NET Web Applications
- ASP.NET Core
- Web APIs
- Security
- Application Migration
- Additional Topics (Time Permitting)

© Treeloop, Inc. - All rights reserved (22-135)

C# and ASP.NET for Experienced Developers

Introduction to .NET

- What is .NET?
- Evolution of the .NET Platform
- .NET SDK and Runtimes
- Application Models
- Managed Code
- Visual Studio
- C# Console and GUI Programs

© Treeloop, Inc. - All rights reserved (22-135)

Introduction to .NET

What is .NET?

- Microsoft's definition...
  - .NET is a free, cross-platform, open source developer platform for building many different types of applications
  - You can use multiple languages, editors, and libraries to build for web, mobile, desktop, games, and IoT

© Treeloop, Inc. - All rights reserved (22-135)

2002 .NET 1.0, C# 1.0, Visual Studio .NET

2003 .NET 1.1, Visual Studio 2003

2004 Mono 1.0

2005 .NET 2.0, C# 2.0, Visual Studio 2005
Generics, Nullable Value Types

2006 .NET 3.0, Mono 1.2
WPF, WCF, WF

2007 .NET 3.5, C# 3.0, Visual Studio 2008
LINQ, Anonymous Types, Lambda Expressions, Extension Methods, Implicit Typing

2008 Entity Framework 1.0

2009 ASP.NET MVC 1.0

2010 .NET 4.0, C# 4.0, ASP.NET MVC 2, Visual Studio 2010
Named / Optional Arguments, Dynamic Binding

2012 .NET 4.5, C# 5.0, Mono 3.0, ASP.NET MVC 4, Visual Studio 2012
Asynchronous Members (async / await)

2013 .NET 4.5.1, ASP.NET MVC 5, Visual Studio 2013
SignalR 1.0

© Treeloop, Inc. - All rights reserved (22-135)

2015 .NET 4.6, C# 6.0, Mono 4.0, Visual Studio 2015, Visual Studio Code 1.0
Expression Bodied Members, Null Propagator, String Interpolation

2016 Xamarin Acquisition, .NET Core 1.0, .NET Standard 1.0
Entity Framework Core 1.0

2017 .NET 4.7, .NET Core 2.0, C# 7.0, Visual Studio 2017
ASP.NET Razor Pages, Out Variables, Tuples, Ref Locals and Returns

2018 GitHub Acquisition, .NET Standard 2.0
Blazor Server

2019 .NET 4.8, .NET Core 3.0, C# 8.0, Visual Studio 2019
gRPC, Default Interface Methods, Using Declarations, Nullable Reference Types

2020 .NET 5, C# 9.0
Blazor WebAssembly, Records, Init Only Setters, Top-Level Statements

2021 .NET 6, C# 10.0, Visual Studio 2022
.NET MAUI

© Treeloop, Inc. - All rights reserved (22-135)

Introduction to .NET

Evolution of the .NET Platform

- .NET 1.0 had the potential to be cross-platform but was only officially supported on Windows
- Current version of this variant is 4.8 and now referred to as ".NET Framework"
- Will be supported for many years to come (end of support for .NET 3.5 SP1 is October 2028)

© Treeloop, Inc. - All rights reserved (22-135)

Introduction to .NET

Evolution of the .NET Platform

- In 2016, Microsoft introduced a new variant of .NET called .NET Core
- Many components were completely rewritten
- Fully supported on Windows, macOS, and Linux
- Included a subset of the functionality provided by .NET Framework
  - Focused on web-based workloads (web UIs and services)

© Treeloop, Inc. - All rights reserved (22-135)

Introduction to .NET

Evolution of the .NET Platform

- The version of .NET Core after 3.1 became the "main line" for .NET and was labeled .NET 5.0
- Supports development of Windows Forms and WPF applications that run on Windows
- The ASP.NET framework in .NET still includes the name "Core" to avoid confusion with previous versions of ASP.NET MVC

© Treeloop, Inc. - All rights reserved (22-135)

Introduction to .NET

Evolution of the .NET Platform

- The entire .NET platform is made available as open-source
- Community contributions are encouraged via pull requests
  - Thoroughly reviewed and tightly controlled by Microsoft

`github.com/dotnet`

© Treeloop, Inc. - All rights reserved (22-135)

Introduction to .NET

.NET SDKs and Runtimes

- .NET Runtime
  - Different version for each platform
  - Provides assembly loading, garbage collection, JIT compilation of IL code, and other runtime services
  - Includes the dotnet tool for launching applications
- ASP.NET Core Runtime
  - Includes additional packages for running ASP.NET Core applications
  - Reduces the number of packages that you need to deploy with your application

© Treeloop, Inc. - All rights reserved (22-135)

Introduction to .NET

.NET SDKs and Runtimes

- .NET SDK
  - Includes the .NET runtime for the platform
  - Additional command-line tools for compiling, testing, and publishing applications
  - Contains everything needed to develop .NET applications (with the help of a text editor)

© Treeloop, Inc. - All rights reserved (22-135)

Introduction to .NET

.NET SDKs and Runtimes

- Each version of .NET has a lifecycle status
  - Current – Includes the latest features and bug fixes but will only be supported for a short time after the next release
  - LTS (Long-Term Support) – Has an extended support period
  - Preview – Not supported for production use
  - Out of support – No longer supported

`dotnet.microsoft.com/download`

© Treeloop, Inc. - All rights reserved (22-135)

Introduction to .NET

Application Models

- Web
- Mobile (native mobile apps for iOS and Android)
- Desktop (native apps for Windows and macOS)
- Microservices
- Cloud
- Machine Learning
- Game Development
- Internet of Things
  - Raspberry Pi and other single-board computers

© Treeloop, Inc. - All rights reserved (22-135)

Introduction to .NET

Managed Code

- Managed code is code whose execution is managed by a runtime component
  - For .NET, this is implemented as the Common Language Runtime (CLR)
- Runtime is responsible for compiling code into machine code and executing it
- Provides important services such as automatic memory management, type safety, and enforcing security boundaries

© Treeloop, Inc. - All rights reserved (22-135)

Introduction to .NET

Managed Code

- .NET defines a Common Type System (CTS)
  - Same primitive types defined for all .NET languages
  - All types inherit from a single root `object` type
  - Support for user defined value and reference types

© Treeloop, Inc. - All rights reserved (22-135)

Introduction to .NET

Visual Studio

- Visual Studio is available for Windows and macOS
  - Full-featured IDE
- Visual Studio Code is available for Windows, macOS, and Linux
  - Includes IntelliSense and debugging features
  - Thousands of extensions are available for additional functionality

`visualstudio.microsoft.com`

© Treeloop, Inc. - All rights reserved (22-135)

Introduction to .NET

Visual Studio

- JetBrains also offers an IDE for .NET development called Rider
- Available for Windows, macOS, and Linux
- Includes advanced capabilities in the areas of refactoring, unit testing, and low-level debugging

www.jetbrains.com/rider

© Treeloop, Inc. - All rights reserved (22-135)

Introduction to .NET

C# Console and GUI Programs

- Lab Exercise

© Treeloop, Inc. - All rights reserved (22-135)

C# and ASP.NET for Experienced Developers

C# Language

- Namepaces
- Data Types and Variables
- Enums, Structures and Classes
- Memory Management
- Branching and Flow Control
- Object-Oriented Techniques
- Interfaces
- Generics
- Delegates

© Treeloop, Inc. - All rights reserved (22-135)

C# and ASP.NET for Experienced Developers

C# Language

- Attributes
- Arrays and Collections
- LINQ

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Namespaces

- The "Hello, World" program can include a `using` directive that references the `System` namespace

```
using System;
```

- Namespaces provide a hierarchical means of organizing C# programs and libraries

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Main Method

- The Hello class declared by the "Hello, World" program has a method named `Main`
- By convention, a static method named Main serves as the entry point of a program and can have one of the following signatures:

```
static void Main() {...}

static void Main(string[] args) {...}

static int Main() {...}

static int Main(string[] args) {...}
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Program Structure

- The key organizational concepts in C# are programs, namespaces, types, members, and assemblies
  - C# programs consist of one or more source files
  - Programs declare types, which contain members and can be organized into namespaces
    - Classes and interfaces are examples of types
    - Fields, methods, properties, and events are examples of members
  - When C# programs are compiled, they are physically packaged into assemblies (typically .exe or .dll)

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Program Structure

- Assemblies contain executable code in the form of Intermediate Language (IL) instructions, and symbolic information in the form of metadata
  - Before it is executed, IL code is automatically converted to processor-specific code by the just-in-time (JIT) compiler of the .NET Common Language Runtime
- Because an assembly is a self-describing unit of functionality containing both code and metadata, there is no need for `#include` directives and header files in C#

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Compilation

- C# permits the source text of a program to be stored in several source files
  - When a multi-file C# program is compiled, all of the source files are processed together as if they were in one large file
  - Forward declarations are never needed in C# because, with very few exceptions, declaration order is insignificant
  - C# does not limit a source file to declaring only one public type, nor does it require the name of the source file to match a type declared in the source file

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Values and References

- There are two kinds of types in C#: value types and reference types
  - Variables of value types directly contain their data
    - Each variable has its own copy of the data, and it is not possible for operations on one variable to affect the other
  - Variables of reference types store references to their data
    - It is possible for two variables to reference the same object and, therefore, possible for operations on one variable to affect the object references by the other variable

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Values Types

- Simple types
  - Signed integral: `sbyte`, `short`, `int`, `long` (8, 16, 32, and 64 bits)
  - Unsigned integral: `byte`, `ushort`, `unit`, `ulong` (8, 16, 32, and 64 bits)
  - Unicode characters: `char` (UTF-16)
  - IEEE floating point: `float`, `double` (32 and 64 bits)
  - High-precision decimal: `decimal` (128 bits)
  - Boolean: `bool`

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Values Types

- Enum types

```
enum DayOfWeek { Monday, Tuesday, ... };
```

- Struct types

```
struct Point
{
  float X;
  float Y;
}
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Reference Types

- Class types `class Person { ... };`
- Interface types `interface IGetsPaid { ... };`
- Array types `int[] nums;`
- Delegate types `delegate int Foo(int a);`

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Nullable Types

- For each non-nullable value type T, there is a corresponding nullable type T? which can hold an additional value of null
  - The syntax for a nullable type is short for Nullable<T>
  - An instance of a nullable type T? has two public properties
    - HasValue of type bool
    - Value of type T

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Enums

- An enum type is a distinct type with named constants
  - Every enum type has an underlying type which must be one of the eight integral types
  - Defaults to int with a starting value of zero

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Structs

- A struct type represents a structure with data members and function members
  - Structs are value types and do not require heap allocation
  - Do not support user-specified inheritance

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Classes

- A class defines a data structure that contains data members and function members
  - Classes are reference types
  - Support single inheritance and polymorphism

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Memory Management

- The .NET CLR provides garbage collection for all heap allocated objects
- Runs on a background thread
  - May be deferred to avoid affecting application performance
  - Will run sooner if available memory is low
- Destructor syntax can be used for code that should run when memory is collected

```
class Person
{
  ~Person() { ... }
}
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Memory Management

- For objects that acquire external resources, the dispose pattern should be employed to provide more control

```
class Repository : IDisposable
{
  void Dispose()
  {
    // free resources
  }
}
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Memory Management

- The using statement can be used to call Dispose( ) in a guaranteed fashion for a specified object

```
using (Repository r = new Repository)
{
  r.DoStuff();
}
```

```
Repository r = new Repository();
try
{
  r.DoStuff();
}
finally
{
  r.Dispose();
}
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Boxing and Unboxing

- The .NET CTS allows any type to be treated as if of type object
  - Values of value types are treated as objects by performing boxing and unboxing operations
  - Boxing incurs the overhead of a heap allocation and produces an object that needs to be garbage collected

```
static void Main()
{
  int i = 123;
  object o = i;    // boxing
  int j = (int)o;  // unboxing
}
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Method Parameters

- A value parameter corresponds to a local variable that gets its value from the argument that was passed
  - Modifications do not affect the argument that was passed
- A reference parameter represents the same storage location as the argument variable
  - Declared with the ref modifier
- An output parameter is similar to a reference parameter but the initial value of the caller-provided argument is unimportant
  - Declared with the out modifier

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Statements

- Selection statements
  - if and switch
- Iteration statements
  - while, do, for, and foreach
- Jump statements
  - break, continue, throw, and return

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Classes and Objects

- Instances of a class are created using the new operator
  - Allocates memory for a new instance
  - Invokes a constructor to initialize the instance
  - Returns a reference to the instance

```
Person p = new Person("Bill", "Gates");
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Members

- Members of a class are either static members (belong to the class) or instance members (belong to the object)

Member	Description
Constants	Constant values associated with the class
Fields	Variables of the class
Methods	Actions that can be performed by the class
Properties	Actions for reading and writing field values
Indexers	Actions invoked via a index-style syntax
Events	Notifications that can be generated by the class
Operators	Conversions and expression operators supported by the class
Constructors	Actions to initialize instance of the class or the class itself
Destructors	Actions to perform before instances of the class are destoryed
Types	Nested types declared by the class

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Accessibility

- Each member of a class has an associated accessibility which controls the the code able to access the member
  - public : Access is not limited
  - protected : Access limited to this class or classes derived from this class
  - internal : Access limited to this assembly
  - protected internal : Access limited to this assembly or class derived from this class
  - private : Access limited to this class

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Class Modifiers

- abstract
  - Can inherit from but can not instantiate
- sealed
  - Can instantiate but cannot inherit from
- partial
  - Allows for a single class to span multiple physical files

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Virtual Methods

- When an instance method includes the virtual modifier, the method is said to be a virtual method
  - When a virtual method is invoked, the runtime type of the instance determines the implementation to invoke
  - When a non-virtual method is invoked, the compile-time type of the instance determines the implementation to invoke
- A virtual method can be overridden in the derived class
  - Requires use of the override modifier

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Abstract Methods

- An abstract method is a virtual method with no implementation
  - Declared with the abstract modifier
  - Requires class to be abstract
  - Must be overridden in every non-abstract derived class

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Properties

- Properties are a natural extension of fields
- A property is declared like a field, except that the declaration ends with a get accessor and / or a set accessor
- A property that has both a get and set accessor is a read-write property
- A property that has only a get accessor is a read-only property
- A property that has only a set accessor is a write-only property

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Interfaces

- An interface defines a contract that can be implemented by classes and structs
  - Can contain methods, properties, events, and indexers
  - An interface can not provide implementations
  - Interfaces may employ multiple inheritance

```
interface IControl { void Parent(); }

interface ITextBox : IControl { void SetText(string text); }

interface IListBox : IControl { void SetItems(string[] items); }

interface IComboBox : ITextBox, IListBox { }
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Generics

- A method or class definition may specify a set of type parameters with angle brackets

```
public void Foo<T>(T obj1, T obj2);
```

```
public class Pair<TFirst, TSecond>
{
  public TFirst First;
  public TSecond Second;
}
```

```
Pair<int, string> pair = new Pair<int, string>();
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Delegates

- A delegate type represents references to methods with a particular parameter list and return type
  - Make it possible to treat methods as entities that can be assigned to variables and passed as parameters

```
delegate void DoSomething(string msg);

public static void Main()
{
  DoSomthing ds = Foo;
  ds.Invoke("Hello");
  ds("Hello");
  ds.BeginInvoke("Hello", ...);
}

void Foo(string thing) { ... }
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Delegates

- .NET includes some generic delegate types

```
delegate bool Predicate<T>(T obj);
```

```
delegate void Action<T>(T obj);
```

```
delegate int Comparison<T>(T x, T y);
```

```
delegate TResult Function<T, TResult>(T arg);
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Delegates

- Some framework methods take a parameter whose type is a generic delegate

```
void DoSomething()
{
  List<int> nums = new List<int> { 1, 2, 3 };
  List<int> results = nums.FindAll(IsEven);
  foreach (int n in results) Console.WriteLine(n);
}

bool IsEven(int i)
{
  return (i % 2) == 0;
}
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Delegates

- An anonymous delegate is a way specify the function for a delegate inline

```
void DoSomething()
{
  List<int> nums = new List<int> { 1, 2, 3 };
  List<int> results = nums.FindAll(
      delegate(int i) { return (i % 2) == 0; });

  foreach (int n in results) Console.WriteLine(n);
}
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Delegates

- A lambda expression is a shorter way to define an anonymous delegate

```
void DoSomething()
{
  List<int> nums = new List<int> { 1, 2, 3 };
  List<int> results = nums.FindAll(d => (i % 2) == 0);

  foreach (int n in results) Console.WriteLine(n);
}
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Delegates

- The left side of the lambda operator (=>) are the function's parameters
  - Types can be excluded as implicit typing can be used
- The right side of the lambda operator (=>) is the implementation of the function
  - The return keyword and brackets are options if only one statement

```
(int i) => { return (i % 2) == 0; }
```

```
i => { return (i % 2) == 0; }
```

```
i => (i % 2) == 0
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Delegates

- Lambda expression can have any number of arguments (including none)

```
i, j => i + j
```

```
() => "Hello world!"
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Delegates

- Lambda expression can also have any number of statements

```
i, j => {
  i = i * 2;
  j = j + 10;
  if (i > j) {
    return i;
  } else {
    throw new ApplicationException();
  }
}
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Delegates

- A lambda expression can be used anywhere an instance of a delegate is required
- Using a lambda expression is always optional
  - A named or anonymous delegate can be used instead

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Attributes

- User-defined types of declarative information can be attached to program entities and retrieved at runtime
  - Specified using attributes
- All attribute classes derive from the System.Attribute base class
- When an attribute is requested via reflection, the constructor for the attribute is invoked and the resulting attribute instance is returned

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Implicitly Typed Local Variables

```
public void Foo()
{
  int i = 5;
  string str = "Hello";
  Person p = new Person("Bill", "Gates");
}
```

```
public void Foo()
{
  var i = 5;
  var str = "Hello";
  var p = new Person("Bill", "Gates");
}
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Implicitly Typed Local Variables

- Variables still strongly-typed
- Compiler determines type at compile-time (not runtime)
- Can only be used for local variables

```
public var Foo() { }
```

```
public void Foo(var x) { }
```

```
public class Person
{
  private var Name = "Joe";
}
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Implicitly Typed Local Variables

- Must be initialized at the time of declaration
- Can be set to null but cannot be initialized to null

```
var s = "Test";
s = null;
```

```
var s = null;
s = "Test";
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Implicitly Typed Local Variables

- Cannot change type after declaration

```
var i = 5;

i = 7;

i = "Ten";
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Implicitly Typed Local Variables

- Can be used with collections if all values are valid for one type

```
var a = new[] { 1, 2, 3 };  // int[]
```

```
var b = new[] { 1.5, 2.2, 3.75 };  // double[]
```

```
var c = new[] { "Hi", null, "Hello" };  // string[]
```

```
var d = new[] { new Person(), new Person() };  // Person[]
```

```
var d = new List<string> { "Bill", "Steve" };
```

```
var o = new[] { 1, "two" };
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Extension Methods

- LINQ uses extension methods to extend existing collection types
- Are only available when extension method's namespace is included with a using directive

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Object Initialization Syntax

- Object initialization syntax allows for the setting of properties as part of the construction statement

```
Person p = new Person();
p.FirstName = "Bill";
p.LastName = "Gates";
```

```
Person p = new Person { FirstName = "Bill", LastName = "Gates" };
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Object Initialization Syntax

- Can be used with a non-parameterless constructor

```
Person p = new Person("Bill") { LastName = "Gates" };
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Object Initialization Syntax

- Setting of properties occurs after the constructor has completed

```
Person p = new Person("Bill") { FirstName = "Steve" };
```

```
Person p = new Person("Bill");
p.FirstName = "Steve";
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Object Initialization Syntax

- Can be nested if desired

```
Rectangle r = new Rectangle();
Point p1 = new Point();
p1.X = 10;
p1.Y = 10;
r.TopLeft = p1;
Point p2 = new Point();
p2.X = 200;
p2.Y = 200;
r.BottomRight = p2;
```

```
Rectangle r = new Rectangle {
    TopLeft = new Point { X = 10, Y = 10 },
    BottomRight = new Point { X = 200, Y = 200 }};
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Object Initialization Syntax

- Can be used with collections

```
List<Person> = new List<Person>
{
  new Person { FirstName = "Bill", LastName = "Gates" },
  new Person { FirstName = "Steve", LastName = "Jobs" }
}
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Anonymous Types

- Representing application data as instances of objects is a good idea
  - This makes many tasks such as data binding easier
- Query results would ideally also be returned as a collection of "ad hoc" objects based on what's being selected
  - Anonymous types provide for this

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

Anonymous Types

- You define an anonymous type by combining implicit typing with object initialization syntax

```
var car = new { Make = "Ford", Model = "Focus", Speed = 55 };
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

LINQ

- Language Integrated Query (LINQ) introduces queries as a first-class concept in the .NET languages
  - Compile-time support
  - Consistent query syntax across data sources

```
var query =
    from   c in Customers
    where  c.Country == "Italy"
    select c.CompanyName;

foreach (string name in query) {
  Console.WriteLine(name);
}
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

LINQ

- LINQ defines a set of extension methods for collections which take delegates as parameters
- The compiler translates a LINQ expression into the appropriate method calls and delegates

```
var query = Customers
    .Where(c => c.Country == "Italy")
    .Select(c => c.CompanyName);

foreach (string name in query) {
  Console.WriteLine(name);
}
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

LINQ

- By default, LINQ will not retrieve the results of a LINQ expression until you attempt to enumerate the results

```
var query =
    from   c in Customers
    where  c.Country == "Italy"
    select c.CompanyName;

foreach (string name in query) {
  Console.WriteLine(name);
}
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

LINQ

- Enumerating over the result of a LINQ expression a second time will cause a reevaluation of the expression
- Deferred execution can be effectively disabled by forcing LINQ to populate a "disconnected" collection

```
var query =
   (from   c in Customers
    where  c.Country == "Italy"
    select c.CompanyName).ToList();

foreach (string name in query) {
  Console.WriteLine(name);
}
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

LINQ

- The results of a LINQ expression can be projected into another type

```
var query =
    from   emp in Employees
    where  emp.Department == "Sales"
    select new Person { FirstName = emp.FirstName,
                        LastName = emp.LastName };

foreach (Person p in query) {
  Console.WriteLine(p.FirstName);
}
```

© Treeloop, Inc. - All rights reserved (22-135)

C# Language

LINQ

- Data projection can be combined with anonymous types
  - Can be especially useful to retrieve "sub results" for use in data binding

```
var query =
    from   emp in Employees
    where  emp.Department == "Sales"
    select new { FirstName = emp.FirstName,
                 LastName = emp.LastName };

foreach (var p in query) {
  Console.WriteLine(p.FirstName);
}
```

© Treeloop, Inc. - All rights reserved (22-135)

C# and ASP.NET for Experienced Developers

Database Access

- Introduction
- Asynchronous Data Access
- ADO.NET
- Object-Relational Mapping
- Entity Framework Core
- Dapper

© Treeloop, Inc. - All rights reserved (22-135)

Database Access

Introduction

- Models represent "real world" objects the user is interacting with
- Entities are the objects used during Object-Relational Mapping and provide a way to obtain and persist model data
- The term Data Transfer Object (DTO) is often used to describe an object that carries data between different processes or subsystems
  - A single DTO may contain multiple different entities, exclude some entity properties, or use different property names
  - In a Web API application, the object that get serialized into JSON is often a DTO

© Treeloop, Inc. - All rights reserved (22-135)

Database Access

Introduction

- The model data typically comes from an external source (database, web service, file, etc.)
- For better maintainability and testability, it is a best practice to use a data access component to encapsulate the details about where the model data comes from
- In ASP.NET, data access should be performed by a service made available via dependency injection
  - Makes it easy to test components independently with hard-coded data (no database)

© Treeloop, Inc. - All rights reserved (22-135)

Database Access

Introduction

- One of the SOLID design principles is Dependency Inversion
- "High-level modules should not depend on low-level models. Both should depend on abstractions."
  - The web application/service should not be built based on how the data access library was designed
  - An interface should be used to define the functionality that the web application requires
  - The data access library should provide a component that implements the required interface

© Treeloop, Inc. - All rights reserved (22-135)

Database Access

Asynchronous Data Access

- When performing IO-bound operations (database access, web service calls, etc.), it is a best practice to perform that work asynchronously
- Allows for the efficient use of thread resources
  - Thread pool threads can be used to handle other incoming requests while the IO-bound operation is in progress
  - Improves the scalability of a web application

```
public async Task<IEnumerable<Product>> GetAllProducts()
{
  return await _repository.GetProductsAsync();
}
```

© Treeloop, Inc. - All rights reserved (22-135)

Database Access

ADO.NET

- ADO.NET is a set of classes that expose data access services for .NET developers
- Includes data providers for connecting to a database, executing commands, and retrieving results
- Provides the most direct method of data access in .NET
- Higher-level data access frameworks typically use ADO.NET behind-the-scenes
  - Entity Framework, Dapper, etc.

© Treeloop, Inc. - All rights reserved (22-135)

Database Access

Object-Relational Mapping

- If a data access component communicates with a relational database, a necessary task will be to convert between relational data and C# objects
- This can be done manually by with ADO.NET, or several frameworks exist that can help with this task
  - Entity Framework Core
  - Dapper (3rd-party micro-ORM)
  - AutoMapper (mapping one object to another)

© Treeloop, Inc. - All rights reserved (22-135)

Database Access

Entity Framework Core

- Modeling based on POCO entities
- Data annotations
- Relationships
- Change tracking
- LINQ support
- Built-in support for SQL Server and Sqlite (3rd-party support for Postgres, MySQL, and Oracle)

© Treeloop, Inc. - All rights reserved (22-135)

Database Access

Entity Framework Core

- By creating a subclass of DbContext, EF Core can populate your entity objects and persist changes

```
public class ECommContext : DbContext
{
  public DbSet<Product> Products { get; set; }
}
```

- The DbContext can be used to create a new database based on the definition of your model objects or it can work with a database that already exists (as we will do)
- The Migrations feature of EF Core can be used to incrementally apply schema changes to a database (beyond the scope of this course)

© Treeloop, Inc. - All rights reserved (22-135)

Database Access

Entity Framework Core

- EF Core will make certain assumptions about your database schema based on your entity objects
- For example, EF Core will assume the database table names will match the name of each DbSet property

```
public class ECommContext : DbContext
{
  public DbSet<Product> Products { get; set; }
}
```

© Treeloop, Inc. - All rights reserved (22-135)

Database Access

Entity Framework Core

- To specify different mappings, you can use data annotations on your entities or use EF's fluent API

```
[Table("Product")]
public class Product
{
  [Column("Name")]
  public string ProductName { get; set; }
}
```

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
  modelBuilder.Entity<Product>().ToTable("Product");

  modelBuilder.Entity<Product>().Property(p => p.ProductName)
                                .HasColumnName("Name");
}
```

© Treeloop, Inc. - All rights reserved (22-135)

Database Access

Entity Framework Core

- Objects retrieved from the context are automatically tracked for changes
- Those changes can be persisted with a call to SaveChanges

```
Product product = _context.Products(p => p.Id == id);
product.ProductName = "Something else";
_context.SaveChanges();
```

© Treeloop, Inc. - All rights reserved (22-135)

Database Access

Entity Framework Core

- EF Core will not automatically load related entities
- The Include method can be used to perform "eager loading" of one or more related entities

```
_dataContext.Products.Include(p => p.Supplier)
  .SingleOrDefault(p => p.Id == id);
```

© Treeloop, Inc. - All rights reserved (22-135)

Database Access

Entity Framework Core

- EF Core sends the SQL it generates to the logging system when executed
- Interception API can also be used to obtain or modify the SQL

```
public class HintCommandInterceptor : DbCommandInterceptor
{
  public override InterceptionResult ReaderExecuting(DbCommand command,
      CommandEventData eventData, InterceptionResult result)
  {
    command.CommandText += " OPTION (OPTIMIZE FOR UNKNOWN)";
    return result;
  }
}
```

```
services.AddDbContext(b => b.UseSqlServer(connStr)
    .AddInterceptors(new HintCommandInterceptor()));
```

© Treeloop, Inc. - All rights reserved (22-135)

Database Access

Entity Framework Core

- EF Core can also be used to execute custom SQL or call a stored procedure

```
var products = context.Products
    .FromSqlRaw("SELECT * FROM dbo.Products")
    .ToList();
```

```
var product = context.Products
    .FromSqlRaw("EXECUTE dbo.GetProduct {0}", id)
    .SingleOrDefault();
```

- In the example above, EF Core uses an ADO.NET parameterized query and SQL injection is not a concern
  - Still an issue if the entire string is constructed first and then passed to FromSqlRaw

© Treeloop, Inc. - All rights reserved (22-135)

Database Access

Entity Framework Core

- The entity classes can be defined manually, or the code for them can be automatically generated
  - CLI tools
  - Package Manager Console tools in Visual Studio

© Treeloop, Inc. - All rights reserved (22-135)

Database Access

Entity Framework Core

- EF Core is a large topic and in-depth coverage is beyond the scope of this course
  - Inheritance
  - Shadow Properties
  - Cascading Updates and Deletes
  - Transactions
  - Concurrency Conflicts
  - Migrations

`docs.microsoft.com/en-us/ef/core/`

© Treeloop, Inc. - All rights reserved (22-135)

Database Access

Dapper

- Dapper is an open-source ORM framework that has become a very popular alternative to Entity Framework

```
github.com/StackExchange/Dapper
```

- Has less features than EF but provides a good high-performance "middle-ground" between ADO.NET and EF
- Dapper is not specifically covered in this course

© Treeloop, Inc. - All rights reserved (22-135)