C# Training Workshop

Contents

[Introduction 4](#_Toc483868255)

[Last Updated: 05/20/2017 4](#_Toc483868256)

[Acknowledgements 4](#_Toc483868257)

[Disclaimer 5](#_Toc483868258)

[Requirements 5](#_Toc483868259)

[Instructions 5](#_Toc483868260)

[Reminders 5](#_Toc483868261)

[Setup 6](#_Toc483868262)

[Install Visual Studio Express Community 2017 (free) 6](#_Toc483868263)

[Visual Studio Settings 6](#_Toc483868264)

[Visual Studio Extensions 6](#_Toc483868265)

[Create a Console Application 7](#_Toc483868266)

[Action 7](#_Toc483868267)

[Solution Explorer 8](#_Toc483868268)

[Overview 8](#_Toc483868269)

[Debugging 9](#_Toc483868270)

[Action 9](#_Toc483868271)

[Comments 10](#_Toc483868272)

[Overview 10](#_Toc483868273)

[Variable Declarations 10](#_Toc483868274)

[Overview 10](#_Toc483868275)

[Methods 11](#_Toc483868276)

[Action 11](#_Toc483868277)

[Overview 12](#_Toc483868278)

[Unit Tests 13](#_Toc483868279)

[Action 13](#_Toc483868280)

[Overview 15](#_Toc483868281)

[Create Sandbox class 15](#_Toc483868282)

[Action 15](#_Toc483868283)

[Operators 17](#_Toc483868284)

[Action 17](#_Toc483868285)

[Conditions 21](#_Toc483868286)

[Action 21](#_Toc483868287)

[Update Console Program with a Condition 24](#_Toc483868288)

[Action 24](#_Toc483868289)

[Overview 24](#_Toc483868290)

[Exceptions 25](#_Toc483868291)

[Action 25](#_Toc483868292)

[While Loops 29](#_Toc483868293)

[Action 29](#_Toc483868294)

[For Loops 32](#_Toc483868295)

[Action 32](#_Toc483868296)

[Objects 35](#_Toc483868297)

[Action 35](#_Toc483868298)

[Overview 36](#_Toc483868299)

[Objects – Repository 36](#_Toc483868300)

[Action 36](#_Toc483868301)

[Overview 38](#_Toc483868302)

[Objects – Repository Unit Tests 39](#_Toc483868303)

[Action 39](#_Toc483868304)

[Generics and LINQ 40](#_Toc483868305)

[Action 40](#_Toc483868306)

[Overview 40](#_Toc483868307)

[Generics and LINQ – Find 42](#_Toc483868308)

[Action 42](#_Toc483868309)

[Generics and LINQ – Insert 43](#_Toc483868310)

[Action 43](#_Toc483868311)

[Generics and LINQ – Delete 44](#_Toc483868312)

[Action 44](#_Toc483868313)

[Enum 45](#_Toc483868314)

[Action 45](#_Toc483868315)

[Enum – Update User 46](#_Toc483868316)

[Action 46](#_Toc483868317)

[Enum – Update Repository 47](#_Toc483868318)

[Action 47](#_Toc483868319)

[Enum – Unit Tests 48](#_Toc483868320)

[Action 48](#_Toc483868321)

[Polymorphism 49](#_Toc483868322)

[Action 49](#_Toc483868323)

[Overview 49](#_Toc483868324)

[Polymorphism – Add AdminUser 50](#_Toc483868325)

[Action 50](#_Toc483868326)

[Polymorphism – Add GuestUser 50](#_Toc483868327)

[Action 50](#_Toc483868328)

[Polymorphism – Update EnumTest 51](#_Toc483868329)

[Action 51](#_Toc483868330)

[Polymorphism – Inheritance 51](#_Toc483868331)

[Overview 51](#_Toc483868332)

[Polymorphism – Update Repository 52](#_Toc483868333)

[Action 52](#_Toc483868334)

[Polymorphism – Update GenericsTest 53](#_Toc483868335)

[Action 53](#_Toc483868336)

[Action 53](#_Toc483868337)

[Polymorphism – Type Casting 55](#_Toc483868338)

[Action 55](#_Toc483868339)

[Overview 55](#_Toc483868340)

[Interfaces 56](#_Toc483868341)

[Action 56](#_Toc483868342)

[Overview 58](#_Toc483868343)

[Update Console Program with the Repository call 58](#_Toc483868344)

[Action 58](#_Toc483868345)

[Overview 59](#_Toc483868346)

[ReSharper 59](#_Toc483868347)

[Overview 59](#_Toc483868348)

[References 60](#_Toc483868349)

[Epilogue 60](#_Toc483868350)

# Introduction

This workshop is intended to be used as a tool by an instructor working with those interested in learning to program in C#. As such there are some gaps where there is assumed knowledge or a mentor ready to assist. For those going solo, Google is your friend.

You'll be introduced to Visual Studio, an integrated development environment (IDE), which is where you type and compile your code. You could use notepad and other tools, but I prefer Visual Studio. This workshop heavily promotes unit tests. Introductory discussions on design patterns including repositories are brought up, though nothing in-depth.

Once you reach the object-oriented programming (OOP) descriptions, don't worry if you don't get it right away. Continue with the workshop and slowly you'll come to put the pieces together. This section is by far the hardest concept for new developers to grasp, so don't despair. There are plenty of WIKI articles on the web that can further assist you. The key point for this workshop is to keep doing the workshop as doing it helps you get to the AHA moment where everything comes together.

If for any reason the installation instruction or extensions don't work as expected, please realize that these tools change very frequently. A quick search on the web will usually help you find what you need. Always try the StackOverflow articles first as they are usually more relevant and on point.

## Last Updated: 05/20/2017

## Acknowledgements

* This workshop was developed by Andrew Hinkle and published by <http://www.danylkoweb.com>
* Social Media
  + Website: <http://www.penblade.com>
  + LinkedIn: <https://www.linkedin.com/in/penblade/>
  + Twitter @AndrewHinkle5
* GitHub
  + This workshop is available on GitHub for your review. If your code fails to compile or you don't get your expected results, then compare your code to the repository.
  + <https://github.com/penblade/Training.CSharpWorkshop>

## Disclaimer

* Reference the License.md in the GitHub repository regarding the license. The author and publisher of this workshop hold no liability to any issues that may occur during your use of this material or recommended resources. As an example if you happen to install Visual Studio and your PC catches fire and burns your house down. Not liable. You decide to hack the CIA with the great font of information learned within this resource. Impressive. This is a beginner's class, so I didn't teach you that, but that's pretty cool. Also, you better be a white hat hacker or you're going to jail. Not liable for that either in case you were wondering.

## Requirements

* Estimated length of workshop: 20 hours
* Visual Studio 2017 Community
  + Given this workshop is targeted to Beginners the course is based on the free version accessible to everyone
  + Basic installation instructions are included in the workshop

## Instructions

* Most of this workshop will use the first level of bullets as a title or description of what we are doing followed by the steps to perform the action with notes in italics
* Sections after setup are broken into one or two section:
  + Overview: Discussion about the topic
  + Action: Steps that you are expected to perform
* Actual code is provided to check your work against and you can compare your code against the GitHub repository as well

## Reminders

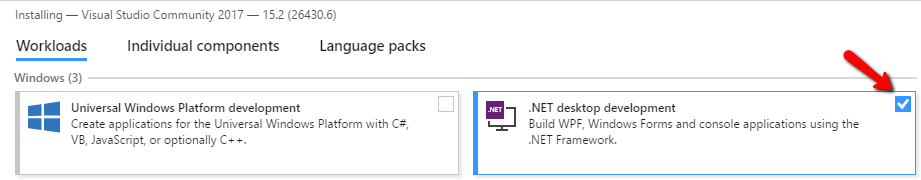
* Remember to KISS (Keep It Simple Stupid) your apps and don't over architect your solutions
* Run your tests after every change and verify that the unit tests succeed or fail as expected
* C# is Case Sensitive
* Code should be typed as is to avoid errors
  + Example: If the namespaces do not match, then additional using statements to the other namespaces need to be added, or better type as is
* I may have used terms you aren't familiar with. If that is the case, then please perform a Google search for it and let me know if adding clarification to the workshop would help.

# Setup

## Install Visual Studio Express Community 2017 (free)

*Troubleshooting installation issues is outside the scope of this workshop. The following are basic instructions subject to change.*

1. <https://www.visualstudio.com/vs/visual-studio-express/>
2. Click the Download Community 2017 link
3. Sign up for notifications or just exit
4. Run the downloaded executable
5. Follow the on-screen instructions until the Workloads tab appears
6. On the Workloads tab check ".NET desktop development"



1. Click Install
2. If prompted, click the Launch button

## Visual Studio Settings

*If one of these options is already checked, then you are all good and should move on to the next step.*

1. Enable line numbers
   1. Click Tools > Options > Text Editor > All Languages > Check Line numbers > Click OK
2. Track Active Item in Solution Explorer
   1. Click Tools > Options > Projects and Solutions > Check Track Active Item in Solution Explorer > Click OK

## Visual Studio Extensions

*Common extensions include Unit Test Generator, Web Essentials, GhostDoc, and Power Tools.*

1. Install Unit Test Generator
2. Click Tools > Extensions and Updates > Online (left sidebar) > Search (top right) Unit Test Generator > Click Download > Install > Click restart now (bottom)

# Create a Console Application

*Goal: Create a simple console app and add some simple logic to the app.*

## Action

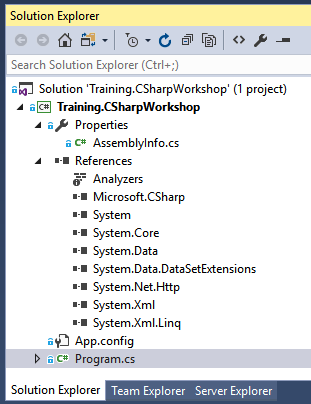
1. Create a new C# Console application
   1. Click File > New Project
   2. Select Templates (left sidebar) > Visual C# > Windows Classic Desktop > Console App (.NET Framework)
      1. Name: Training.CSharpWorkshop
      2. Location: C:\Projects\Training
      3. Solution name: Training.CSharpWorkshop
      4. Check Create directory for solution
   3. Click OK
2. Read input and write Hello {name} to the console
   1. Open Program.cs (right sidebar)
   2. Update the Main method with the following code (leave the usings alone):



# Solution Explorer

## Overview

1. A Solution is a container that references projects and files associated with an application
2. A Project references all of the files required to run your application
   1. Applications may consist of multiple projects
3. A Unit Test Project (added later) tests a Project
4. The Properties > AssemblyInfo file describes your Project such as title, name, and version
5. References are a list of Projects that are part of the .NET Framework, NuGet packages, and projects in your solution that your Project requires in order to perform its function
6. App.config stores your configuration settings
   1. The Project that serves as the running application, in this case the Console Project "Training.CSharpWorkshop" must have all configuration settings required by any projects it references in order to run
   2. In the same regard the Unit Test project must also have any configuration settings required to properly test a project
7. Program.cs and any other files you add contain the logic to run your application



# Debugging

*Learn to add a breakpoint and debug. Add a breakpoint to the console application and practice the basic debug steps.*

## Action

1. Adding Breakpoints (shortcut: F9)
   1. Click the gray bar to the left of the command you would like to stop at while debugging so you may investigate the variable values and how commands are processed
      1. The breakpoint is noted by the red circle and the line highlighted red
2. Debug
   1. Add breakpoints
   2. Run the application in Debug mode by clicking Debug > Start Debugging (shortcut: F5)
   3. Process the current line and move to the next
      1. Click Debug > Step Over (shortcut: F10)
   4. Go to the method called on the current line and if you can't Step Over
      1. Click Debug > Step Into (shortcut: F11)
      2. If you call a method you didn't write you can't step into the method without proper debug files
   5. Continue processing until the next breakpoint is reached or to continue until the application is finished
      1. Click Debug > Continue (shortcut: F5)
3. Practice
   1. Add a breakpoint on the first Console.WriteLine and debug to that step.
      1. Step Over and Step Into will perform the same action for now, but later you'll see Step Into in action
   2. While debugging you can hover over parts of your code to get information such as the variables stored in variables
4. There are plenty of other debugging features available to use such as Add Watch that you can further research on your own

# Comments

## Overview

*Single-line comments are by far the common. You'll see them through-out this course. In most cases if you need to add a comment to clarify your code, consider renaming your variables and methods. If the code is still complex consider breaking your code into more methods with better descriptions. In the end you'll need to decide when these steps aren't enough and you need to add a comment. As an example, the unit tests have three parts broken up by "Arrange", "Act", and "Assert".*

1. Single-line



1. Multi-line



# Variable Declarations

## Overview

*The following variable declarations are very common. Variables store information so you may use that data later. C# is a strongly-typed language. Even though "var" looks like it could be anything, if you hover your cursor over the variable name the actual type is displayed. The "implicit types" line up with the "primitive types" so you know what are the types. "var" comes in handy for simplifying work with objects later in this course, so get used to them now.*

1. Primitive types



1. Implicit types
   1. var (strongly typed, but the compiler determines the type)



# Methods

*Methods perform a specific task. Given certain parameters your code processes the information and returns the results. Methods don't have to use parameters and don't have to return results, but they should do something useful. There are additional features available for methods such as parameters can use "out" and "ref" and other interesting and useful features that are not covered by this workshop.*

*Methods can call other methods. Keep methods simple and only perform a specific task. Some methods may simply be a checklist that calls other methods to separate concerns. If a task has multiple steps, then break up the steps into additional methods and have your task method call the other step methods.*

## Action

1. Refactor the code in Main to a new public method named Process  
   *Main should be simple and clean*
   1. Create a new empty public method call Process
   2. Cut the contents of the Main method
   3. Paste into the Process method
   4. Call the Process method in the Main method
   5. Run Debug to verify the console app runs the same



1. Refactor the Role line to a method because we will be adding logic later
   1. Create a new public static method called GetRoleMessage that returns the string "Role: Admin"
   2. Replace the string in the Process method with a call to the GetRoleMessage method



## Overview

1. Method signatures
   1. Parameters
      1. The values passed into a method
   2. Return Types
      1. The values returned from a method
      2. Use void when there is no value to return
   3. Scope
      1. Pass by reference
         1. If a value is passed in by reference the value is also changed outside the method
         2. Objects are passed by reference
      2. Pass by value
         1. If a value is passed in by value the value is not changed outside the method
         2. Primitive types such as string, int, float, etc are passed by value

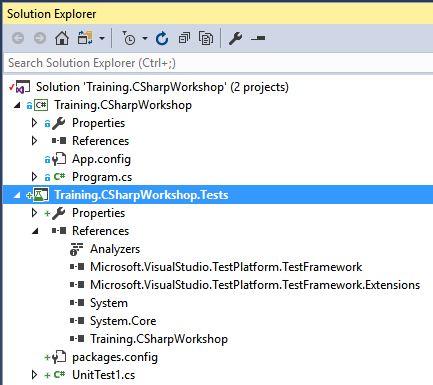
# Unit Tests

*Unit tests verify your Project performs as expected. If you run your suite of tests after any change, you should find that your tests all pass. If they don't, then you know you introduced a bug and should resolve it. Better to find out sooner than later.*

*Important: From here on until you reach the Polymorphism section after every change is performed you should run all of your unit tests and they should pass except for the one test we'll set to ignore.*

## Action

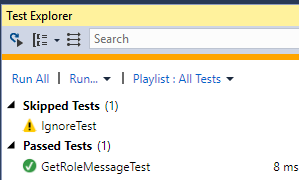
1. Create a test project with all required references to the project we are testing
   1. Right-click the solution > Add > New Project > Expand Visual C# and select Test > Click Unit Test Project
   2. Name: Training.CSharpWorkshop.Tests
   3. Location: C:\Projects\Training.CSharpWorkshop\Training.CSharpWorkshop\Training.CSharpWorkshop.Tests
   4. Click OK
   5. Right-Click the test project References folder > Add Reference > Select Projects > Solution > Check the project "Training.CSharpWorkshop > Click OK



1. Rename the test class and add a couple tests
   1. Rename UnitTest1.cs to ProgramTests.cs
   2. Click Yes to rename the class
   3. Double-click ProgramTests.cs to open the file
   4. Rename TestMethod1 to IgnoreTest
   5. Update IgnoreTest to call Assert.Fail();
   6. Add the [Ignore] attribute to ProcessTest to show how tests can be ignored
   7. Add our first unit test method GetRoleMessageTest



1. Running Tests
2. Run all tests
3. Click Test > Run > All Tests



1. Debug all tests
2. Add breakpoints as necessary
3. Click Test > Debug > All Tests
4. Debug all test methods in a test class
5. Right-click a test class > Run Tests (shortcut: Ctrl-R, Ctrl-T)
6. Debug a test method
7. Right-click a test class > Run Tests (shortcut: Ctrl-R, Ctrl-T)

## Overview

1. You should only test public methods
2. Private and protected methods can only be tested with a few unit test frameworks
3. Test classes will have the [TestClass] attribute
4. Test methods will have the [TestMethod] attribute
5. Test methods with the [Ignore] attribute will not be run
   1. While convenient at times until a reason for failure is determined, unit tests should always pass before checking in with few exceptions and even then should be resolved ASAP
6. Understanding the triple A's of unit tests
7. Arrange
   * 1. Create variables required to test a method
8. Act
9. Call the method we are testing saving the results to a variable named actual
10. Assert
11. Create a variable named "expected" representing the expected results
12. Call an Assert method for validating the expected result against the actual

# Create Sandbox class

*While running this workshop you may find that you are inspired to try something out. Let's create a Sandbox class so you can do just that without worrying about messing up the workshop.*

## Action

1. Create a test file for learning techniques on your own
   1. Right-Click Training.CSharpWorkshop.Tests > Add > Unit Test
   2. Rename UnitTest#.cs to Sandbox.cs
   3. Click Yes to rename the class
   4. Delete TestMethod1
   5. Add the an AdditionTest method as follows



# Operators

*In order to make decisions you use operators such as "equals", "greater than", "and", "or", etc. to compare a value against another value.*

## Action

1. Create Operators test class
   1. Right-Click Training.CSharpWorkshop.Tests > Add > Unit Test
   2. Rename UnitTest#.cs to Operators.cs
   3. Click Yes to rename the class
   4. Delete TestMethod1



1. Comparers



1. And



1. Or



1. Math



1. Math with Assignments



1. Incrementers/Decrementers



1. Conditional



1. Null Coalescing



# Conditions

*Conditions are logical forks in the road where you need to make a decision based on some information.*

## Action

1. Create Conditions test class
   1. Right-Click Training.CSharpWorkshop.Tests > Add > Unit Test
   2. Rename UnitTest#.cs to Conditions.cs
   3. Click Yes to rename the class
   4. Delete TestMethod1



1. If



1. If/else



1. If/else if/else



1. Switch/case, break



1. Switch - Combined Cases



# Update Console Program with a Condition

*Update the application to implement a standard "if" condition and create unit tests to verify the logic works as expected.*

## Action

1. Open Training.CSharpWorkshop.Program.cs
2. Update GetRoleMessage



1. Open Training.CSharpWorkshop.Tests.ProgramTests
2. Rename GetRoleMessageTest to GetRoleMessageForAdminTest so it is more specific
3. Add unit test for Guest role



## Overview

1. String.Format is a common .NET method for concatenating strings
   1. {#} represents the elements in the parameters listed to the right of the string
   2. {#:\*} the star represents the format of the data such as currency and zero-padding



# Exceptions

*Exceptions are errors that are "exceptional", because you do not expect them. If you expect a string to not be null or empty because your code is supposed to populate it then throw an exception. If you have an expected error, then handle it properly by logging it and/or returning a message, but do not throw an exception in that case.*

*If your debugger breaks where the exception is thrown, then press F5 to continue and verify in the Test Explorer that the tests pass.*

## Action

1. Create Exceptions test class
   1. Right-Click Training.CSharpWorkshop.Tests > Add > Unit Test
   2. Rename UnitTest#.cs to Exceptions.cs
   3. Click Yes to rename the class
   4. Delete TestMethod1



1. Throw Exception



1. Try/catch



1. Try/catch/finally



# While Loops

*When you have a list of values that need processed, you perform that action on each item in a list via a loop. You commonly run loops while a specific condition is true such as reaching the last item in a list.*

*Be aware that we are now "using System.Collections.Generic".*

## Action

1. Create WhileLoops test class
   1. Right-Click Training.CSharpWorkshop.Tests > Add > Unit Test
   2. Rename UnitTest#.cs to WhileLoops.cs
   3. Click Yes to rename the class
   4. Delete TestMethod1



1. While



1. While, continue/break



# For Loops

*While loops could be any condition, however, For loops are specifically for cycling through and processing a list of items. The common steps to initialize a counter, checking a condition, and incrementing the counter is simplified into a single line. The process became so common the Foreach loop was created hiding the counter maintenance behind the scenes.*

*Be aware that we are now "using System.Collections.Generic".*

## Action

1. Create ForLoops test class
   1. Right-Click Training.CSharpWorkshop.Tests > Add > Unit Test
   2. Rename UnitTest#.cs to ForLoops.cs
   3. Click Yes to rename the class
   4. Delete TestMethod1



1. For, continue/break



1. Foreach, continue/break



# Objects

*Objects are the biggest hurdle for most people to overcome. While this workshop tries to explain the how objects work, don't worry if you don't get it right away. Continue with the workshop and if you still don't quite understand what they are, there are plenty of articles on the web that may assist you getting over that hurdle.*

*All of the classes for this workshop are public to help minimize the number of potential issues with trying to deal with scope issues.*

*Goals: Create a RepositoryTest test class for unit tests. Create a User class to store an Id and Name as properties.*

*Note: I'm having you create the unit test file now even though we don't populate right away, because I want you to start thinking what your unit tests should be before you start programming. Unit tests have a purpose to test the behavior of your class, so know that purpose before you start banging away on the keyboard.*

## Action

1. Create RepositoryTest test class
   1. Right-Click Training.CSharpWorkshop.Tests > Add > Unit Test
   2. Rename UnitTest#.cs to RepositoryTest.cs
   3. Click Yes to rename the class
   4. Delete TestMethod1



1. Create a new class named User
   1. Right-click Training.CSharpWorkshop > Add > Class > Change name to User.cs
   2. Open User.cs and add public before "class User"



## Overview

1. An object is a container of properties and/or methods
2. Example: Every product in a grocery store has properties such as price and weight
3. Example: Every cash register has properties to track the list of products scanned and methods for totaling the price of each product in that list
4. Properties
   1. Properties act as if they are just public variables



* 1. With the benefit of maintaining a private variable that can be changed as needed without changing actual usage of the property



# **Objects – Repository**

*Goal: Create a Repository to return an instance of a User object given a user name.*

## Action

1. Create a new class named Repository
2. Right-click Training.CSharpWorkshop > Add > Class > Change name to Repository.cs
3. Open Repository.cs and add public before "class Repository"



1. Create a new instance of a list of users in the constructor



1. Add a method to GetUserByName



## Overview

1. Repository is a design pattern used to manage data retrieval whether it comes from a database, files, or other means
2. Constructors
3. Constructors are used to setup an object and can take in parameters like a method
4. Class Security
5. Public (Common)
   * 1. Can be accessed by all
6. Private (Common)
7. Can only be accessed within this class
8. Protected
9. Can only be accessed within this class and any class that inherits it
10. Internal
11. Can be accessed by all as long as the class is in the same project (assembly)
12. New is used to create a separate instance of an object that follows a specific template
    1. In the case of User, I can create a new instance of a User object for Andrew and another separate instance for Dave; They are of the same type of object, but do not use the same instance
13. Static is used when you want to always use the same version every time without having to new up an instance at all
    1. String.Format is an example where you don't want to create a new instance of a String object just to format a string
14. Constants are set at declaration and never changed
15. Readonly are set by the constructor only

# Objects – Repository Unit Tests

*Goal: Create a couple unit tests to verify specific users are in the list.*

## Action

1. Create unit tests for the method GetUserByName in the RepositoryTest test class covering a user that should and shouldn't be in the list



# Generics and LINQ

*Developers inevitably want to find quicker and easier ways to accomplish a goal. Generics and LINQ are an example of some features that help us perform tasks in a universal way that can be reused. The following is a short overview of some of the capabilities these two features offer.*

## Action

1. Create GenericsTest test class
   1. Right-Click Training.CSharpWorkshop.Tests > Add > Unit Test
   2. Rename UnitTest#.cs to GenericsTest.cs
   3. Click Yes to rename the class
   4. Delete TestMethod1



## Overview

1. Object Initializers
   1. Object Initializers allow you to create a new instance of an object or collection as if you defined the object and set the variables immediately after creating the new instance.



1. You can take this example one step further by doing a collection initializer



1. Generic Lists
   1. List is a generic class in the .NET framework that lets you define the type of objects the list will contain with standard CRUD (Create, Read, Update, Delete)
      1. Common methods of List:
         1. Add
         2. Insert
         3. Remove
         4. RemoveAt
         5. Count
         6. Clear
         7. Index
         8. Sort
2. LINQ (Language-Integrated Query)
   1. Lists can be filtered with LINQ and can follow syntax that use object chaining or a more SQL like format that starts with the From and ends with the Select
      1. Common methods of LINQ on lists
         1. Any
         2. Contains
         3. Exists
         4. Where
         5. Single
         6. SingleOrDefault
         7. First
         8. FirstOrDefault
         9. Find

# Generics and LINQ – Find

*Goal: Call some common methods to find items in a list.*

## Action

1. In Repository, create a Find method to illustrate LINQ



1. Add Find Unit Tests which are the same as the GetUserByName Unit Tests except the different method call



# Generics and LINQ – Insert

*Goal: Call some common methods to insert items in a list.*

## Action

1. Create an Insert method in the repository



1. Create an Insert Unit Test



# Generics and LINQ – Delete

*Goal: Call some common methods to insert items in a list.*

## Action

1. Create a Delete method in the repository



1. Create a Delete Unit Test



# Enum

*Enumerations (Enum) are logical representation of specific values so you don't have hard-coded strings all over the place. You don't want hard-coded strings also known as magic strings, because they are difficult to maintain and their meaning could change between your files or even methods preventing you from safely changing them in mass and definitely not safely.*

*Goal: Create an enum to represent your roles.*

## Action

1. Create EnumTest test class
   1. Right-Click Training.CSharpWorkshop.Tests > Add > Unit Test
   2. Rename UnitTest#.cs to EnumTest.cs
   3. Click Yes to rename the class
   4. Delete TestMethod1



1. Create a new class named RoleEnum
   1. Right-click Training.CSharpWorkshop > Add > Class > Change name to RoleEnum.cs



# Enum – Update User

*Goal: Implement the enum in the User class along with standard CRUD (Create, Read, Update, Delete). Create = Insert. Read = Find. Update is performed by Find and then change the value. Delete = Delete.*

## Action

1. Add RoleEnum as Role to the User class



1. Add methods to verify against the user's role if the user can add, delete, or find users



# Enum – Update Repository

*Goal: Update the Repository to add the enum when initializing each User.*

## Action

1. In the Repository, set the role in GetUsers



# Enum – Unit Tests

*Add unit tests to verify the enum role is applied properly.*

## Action

1. Add Unit Tests for Admin and Guest user permissions



# Polymorphism

*Object-Oriented Programming (OOP) at its best is the capability to reuse code and keep it related. In the following steps we have an AdminUser and a GuestUser. Both are a type of User and share those properties and methods. Both also have their own unique functionality, but they are still a User giving you more flexibility.*

*Warning: Up to now after every change you could run your tests and they should all pass. For the next few changes, some of the tests will fail until the refactoring is finished. I understand this feels dirty, trust me you'll understand soon.*

## Action

1. Create PolymorphismTest test class
   1. Right-Click Training.CSharpWorkshop.Tests > Add > Unit Test
   2. Rename UnitTest#.cs to PolymorphismTest.cs
   3. Click Yes to rename the class
   4. Delete TestMethod1



1. Change User class to be an abstract class



## Overview

1. Abstract Classes
   1. Cannot be instantiated
   2. Child classes can use methods and properties of parent (grand-parent, etc) classes as long as they are public or protected
      1. Example: Can be used to define the roles of the User so there is never an opportunity where the role is never defined

# Polymorphism – Add AdminUser

*Goal: Add an AdminUser with a method that only it can use. Set the Role to RoleEnum.Admin in the constructor since the role is now described by the class. AdminUser can still use the methods defined in the User class as we'll see later.*

## Action

1. Create a new class named AdminUser that inherits User
   1. Right-click Training.CSharpWorkshop > Add > Class > Change name to AdminUser.cs



# Polymorphism – Add GuestUser

*Goal: Add a GuestUser with a method that only it can use. Set the Role to RoleEnum.Guest in the constructor since the role is now described by the class. Demonstrate how to override the base class User's methods by changing the CanInsert method to return true. GuestUser can still use the methods defined in the User class as we'll see later.*

## Action

1. Create a new class named GuestUser that inherits User
   1. Right-click Training.CSharpWorkshop > Add > Class > Change name to GuestUser.cs



# Polymorphism – Update EnumTest

*Goal: Update the GuestUser expected permissions made to demonstrate how to override the CanInsert method defined in the base class User.*

## Action

1. Update EnumTest test class method GuestUserPermissions CanInsert assertion to true



# Polymorphism – Inheritance

## Overview

1. A class that inherits another class is considered a child class of that parent class
2. Two classes that share the same parent class are considered siblings
3. Defining a method as virtual will allow a child class to override the method to perform different logic
4. A sealed class cannot be extended (polymorphed)

# Polymorphism – Update Repository

*Goal: Update the Repository to initiate the AdminUser and GuestUser appropriately.*

## Action

1. Update Repository to use the new polymorphed objects AdminUser and GuestUser appropriately
   1. Refactor the declaration of user away from the first user and define it as User user;
   2. Change the first user to an AdminUser
   3. Change the second user to a GuestUser
   4. Remove the role definitions since they are now handled in the abstract User class



# Polymorphism – Update GenericsTest

*Goal: Update the GenericsTest to create a GuestUser for the test.*

## Action

1. Update GenericsTest to create the user as a GuestUser since User is now an abstract and can no longer be instantiated

Polymorphism – Unit Tests

*Sometimes you are working with an object of a base class such as User; however, you know you are working with a specific class type such as AdminUser. You can type cast the object to the expected class type so you may access the additional functionality provided by the child class.*

*Goal: Add unit tests for the AdminUser and GuestUser specific methods by type casting. Type Casting is explained in the next section.*

*Important: All unit tests should be passing now. If any tests are failing at this point, compare your code to the previous examples or compare your code to the GitHub repository (link is at the beginning of this document).*

*Once you have all of your unit tests passing again you'll feel better. Did you miss them? Unit tests are a great safety net that makes you feel good that after every change you have total confidence that you didn't break anything. As with anything, moderation is key. It's best to test only public methods that represent the behavior of you classes. Keep that in mind when you move on from this workshop.*

## Action

1. In PolymorphismTest, add unit tests for new User methods



# Polymorphism – Type Casting

*Sometimes you are working with an object of a base class such as User; however, you know you are working with a specific class type such as AdminUser. You can type cast the object to the expected class type so you may access the additional functionality provided by the child class.*

*Goal: The previous section showed you how type casting works when the object is properly type casted. Now let's demonstrate how type casting works when can't perform the type cast.*

## Action

1. Add Type Casting examples where the type cast fails



## Overview

1. Type casting
   1. Typeof
      1. Method will return the class type of the class specificed
   2. GetType
      1. Method will return the type of the object
   3. ({class name}) actual
      1. Attempts to cast the variable to that type and will throw an exception if it fails
   4. actual as {class name}
      1. Attempts to cast the variable to that type and will return null if it fails
   5. Example: Gain access to methods only available in a child object
   6. Example: Perform logic if the object is of a specific type

# Interfaces

*Interfaces are contracts that specify what an object can do. Interfaces should be very specific to the expected properties or methods you should expect. When you program against an interface or a base class you simplify some maintenance at the cost of increasing complexity and number of files. Interfaces make the most sense when exposing the public methods on your public classes.*

## Action

1. Create a new class named IUser and change it to an interface
   1. Right-click Training.CSharpWorkshop > Add > Class > Change name to IUser.cs



1. Change User class to implement the IUser interface



1. Change Repository to use IUser instead of User in all cases.



## Overview

1. Interfaces
2. Cannot be instantiated
3. Interfaces are contracts that are implemented by a class so that every property and method (not variables) defined in the interface are also defined in the class
   1. Every property and method defined in the class DO NOT have to be defined in the interface
4. Only need applied to the lowest common class
   1. Typically that is the root parent object, but not always
5. Interfaces are crucial to properly unit test objects in isolation to avoid "Integration Tests" that are reliant upon databases, connections, and files
   1. Example: Dave is a specific user in the database that may be deleted when he leaves the company or his role changes from a Guest to an Admin

# Update Console Program with the Repository call

*Goal: Implement the repository in the console program.*

## Action

1. Open Training.CSharpWorkshop.Program.cs
2. Update GetRoleMessage



1. Run the program F5 and verify that Andrew is an Admin and Dave is a Guest
2. Open Training.CSharpWorkshop.ProgramTest.cs
3. Add the GetRoleMessageForNoneTest test method to test the None case
   1. The existing tests are fine, however, we now have a None case we need to test against



## Overview

We've now implemented the Repository in the console program. Seems like a lot of effort for little benefit? Sure is for this simple application. Please remember that this is just an introduction. Real world applications are much more involved and complicated with loads of data to consume.

Design Patterns like the Repository pattern simplified for this workshop are excellent for isolated data access so if the underlying way you access the data changes such as from an XML file to a database, you can rest assured it is all done in those repository objects typically isolated away in a separate project.

# ReSharper

## Overview

ReSharper is an add-on that simplifies some complex tasks. The following is the short list of ReSharper features I use frequently.  I’ll demonstrate a few in the next lesson.

* Enhanced Intellisense
  + Better description of what is wrong with the syntax
  + Offers to refactor code to use other techniques such as for loops to LINQ
    - This feature is hit or miss as to whether you like, can read/maintain, however, it is a great learning tool to learn new techniques
      * I learned much of what I know about LINQ with this feature
* If an object is not referenced in a the project, but it is included in another project in the same solution, ReSharper will ask if you want to import it
  + When most students had the issue with the Program class not being referenced, ReSharper would have fixed that with a quick shortcut Alt-Enter
* Extract Method
  + Move highlighted code to a new method and add a call to that method
* Extract Interface
  + Interfaces are contracts that are very useful for unit tests
* Move files
  + Moves the files to the new location
  + Updates all namespaces
  + Updates all references to the file and objects therein throughout the entire solution
* Safe Delete
  + When you deprecate an object and you are ready to delete, this feature will review your solution to ensure there are no other references
    - Still be careful as it won’t detect references from other solutions that may use the DLL such as from a common library file used by multiple solutions
* Adjust Namespaces
  + Fixes namespaces to match your folder structure, much easier than doing it manually
* Menu option: Refactor
  + Gives you access to most of the features above
* Menu option: Run Unit Tests
  + Run Unit Tests option is available at the solution level, test project, test class, test method
* Much, much more!
  + These are just the most commonly used features though it supports most Refactoring techniques.

# References

Here’s a short list of recommended books that transformed me as a developer that I highly recommend any developer to read.

* [Top 10 books every .NET developer should own](http://www.danylkoweb.com/Blog/top-10-books-every-net-developer-should-own-QG) - This is a great list of books. I've personally read and recommend reading the following three books on that list in the following order.
  + [Refactoring: Improving the Design of Existing Code](http://www.amazon.com/gp/product/0201485672/ref=as_li_tl?ie=UTF8&camp=1789&creative=390957&creativeASIN=0201485672&linkCode=as2&tag=danyl-20&linkId=JWATUXHVFPXJM5LU) By Martin Fowler, Kent Beck, John Brant, Williams Opdyke, and Don Roberts
  + [Design Patterns in C#](http://www.amazon.com/gp/product/0321126971/ref=as_li_tl?ie=UTF8&camp=1789&creative=390957&creativeASIN=0321126971&linkCode=as2&tag=danyl-20&linkId=GLRAEO3PLQISPDUA) By Steven John Metsker
  + [Patterns of Enterprise Application Architecture](http://www.amazon.com/gp/product/0321127420/ref=as_li_tl?ie=UTF8&camp=1789&creative=390957&creativeASIN=0321127420&linkCode=as2&tag=danyl-20&linkId=5K6QOZY7V4BE5UMK) By Martin Fowler

# Epilogue

Thank you for using this workshop. If you gained anything from this workshop then give it a shout out. If you didn't, then give me your feedback so I may improve it. Thanks!