Week 1 Study Guide  
  
Key Concepts:

|  |  |
| --- | --- |
| * .NET | * SOLID |
| * CLR | * Access/extended modifiers |
| * CIL | * Class interface, structure, and Enum |
| * C# | * Reference/value data types |
| * Git | * Casting, boxing, ref, out |
| * Visual Studio | * Garbage Collection |
| * VS Code | * Generics |
| * OOP pillars | * Serialization |
| * Unit Testing | * Debugging |
| * Delegates | * Events |
| * Lambda | * LINQ |
| * Func/Action | * Task await/async |
| * Logging | * Extension methods |
| * Singleton |  |

.NET

Based on the .NET Framework, it started out as a Windows specific platform. Now we use the .NET Core because it is portable (your code will run natively on any compatible OS), open-source, and is friendly to build automation. Mono is a .NET implementation for running apps on all major mobile operating systems.

.NET Standard - common interface from Framework and Core that allows both to use identical libraries. Implemented by .NET Core

.NET Core is compatible with many languages. C#, C++, F#, and more.

.NET Core is the libraries we use, the compiler we use, and the run-time environment of the code being ran.

.NET complies code to MSIL (Microsoft Intermediate Language).

Common Language Runtime uses JiT compilation on MSIL to convert to machine code

MSIL is deployed so it can run on any computer. MSIL still includes things like classes, inheritance, exception handling, and access modifiers

Diagram: <https://csharpcorner-mindcrackerinc.netdna-ssl.com/UploadFile/9582c9/what-is-common-language-runtime-in-C-Sharp/Images/CLR.jpg>

CLR (Common Language Runtime) provides:

* JiT compilation (Just in time compilation) – A way of executing computer code that involves compilation during the execution of a program at runtime
* BCL (Base class library) - Any .NET language can use these libraries in BCL
* CTS (Common Type System) - Fixed types that .NET can use, common across all languages that have been converted to MSIL
* VES (Virtual Execution System) - The final runtime for .NET Core compiled code

Advantages of this system:

* Interoperability between languages
* Portability across architectures and operating systems (Core is different based on OS, but produces CLR still)

What is in .NET?

* Web: ASP.NET
* Data Access: Entity Framework
* Mono: Partial port to Mac/Linux

C# Hello World Sample

using System; // “using” creates an alias for a namespace or to import types defined in other namespaces

namespace HelloWorld //Namespaces are used to provide a "named space" in which your application resides

{

class Program

{

static void Main(string[] args) //how to define main in C#

{

System.Console.WriteLine("Hello World!");

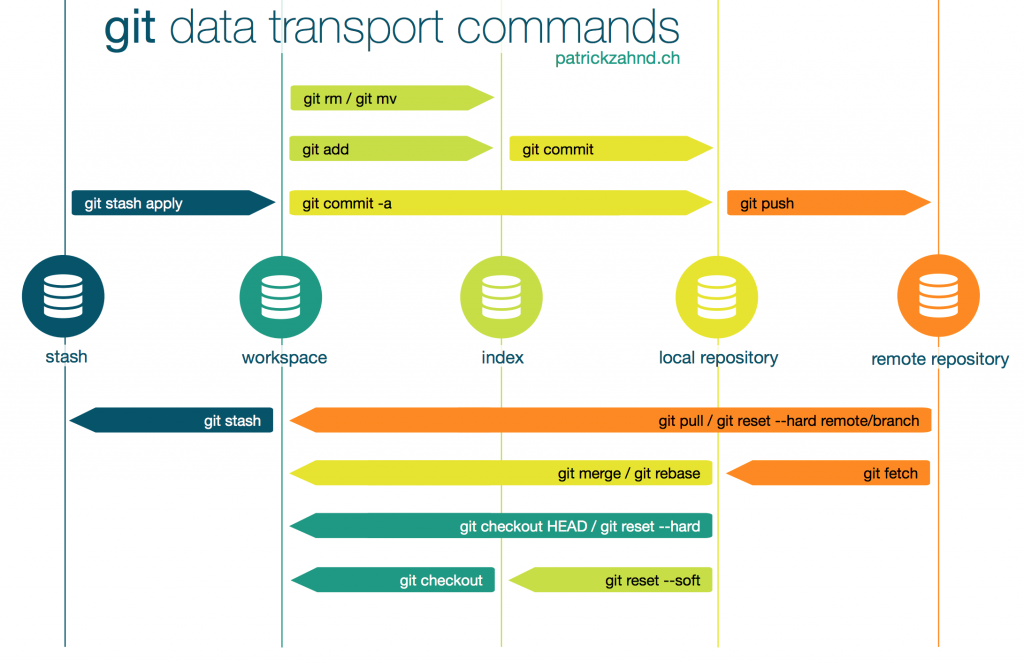
}

}

}

Git and GitHub

* git clone "git hub link" - clones repository from git hub
* git status - shows you differences from original repository
* git ignore - use ignore to ignore certain files that dont matter for compilation
* Helpful resource gitignore.io - creates git ignore commands for technologies used (file name must be .gitignore)
* git add . - adds every change made to repository
* git commit -m "description of change" - commits all added files to the repository
* git push - pushes all committed files to the online repository
* git pull - pulls all updated files to your local repository



Branches

* A branch is chain of commits
* By default github names its remote repository branch "origin"
* You can only work on branches one at a time
* You can switch branches by using git checkout "branch". It will change your directory to match the switch to branch
* Follow git checkout "branch" with git pull to update your local repository
* git merge "branch" (of other branch) - merges 2 branching paths

**Development Branch Diagram**: <https://raw.githubusercontent.com/arslanbilal/git-cheat-sheet/master/Img/git-flow-commands-without-flow.png>

OOP Pillars

*What are the 4 pillars of OOP?*

Abstraction, Encapsulation, Polymorphism, and Inheritance.  
  
*What is inheritance?*

The ability of a new class to be created, from an existing class by extending it, is called inheritance. In object-oriented programming, inheritance allows a subclass to inherit some or all of the data members and functions of its superclass.

Additional Notes: Ability to derive a class from an existing class, gaining its data and behavior.

*What is Abstraction?*

An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects, and provide conceptual boundaries for the viewer. It is an emphasis on ideas and properties rather than particulars. Helps manage complexity in large problems. The objective of **hiding the implementation complexity** of how the the features offered by an API / design / system were implemented, in a sense simplifying the 'interface' to access the underlying implementation.  
  
Additional notes: Separation between needed functionality and implementation details. Leaky abstraction is when the walls of abstraction are not perfectly separating functionality and implementation. In C#, our number one tool in implementing abstraction is interfaces.  
  
*What is Encapsulation*?  
It refers to the bundling of data with the methods that operate on that data. Encapsulation is *used to hide the values or state of a structured data object inside a class*, preventing unauthorized parties' direct access to them. Publicly accessible methods are generally provided in the class (so-called *getters* and *setters*) to access the values, and other client classes call these methods to retrieve and modify the values within the object.

Additional notes: Bundle data and behavior into objects. Hide implementation specific data/behavior.

*What is Polymorphism?*  
**Polymorphism describes a pattern in object oriented programming in which classes have different functionality while sharing a common interface.** Polymorphism means the ability to request that the same operations be performed by a wide range of different types of things. The ability of different objects to respond, each in its own way, to identical messages is called polymorphism.

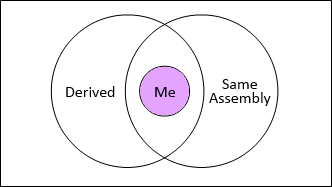
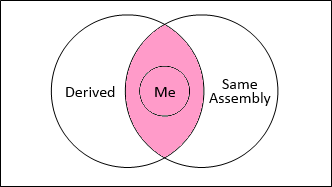
Additional notes: Ability to treat a derived class object as though it were a base class object. Examples are method overloading, generic types

Access Modifiers

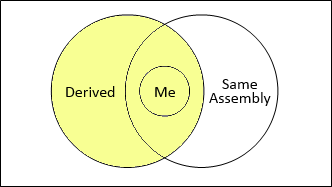
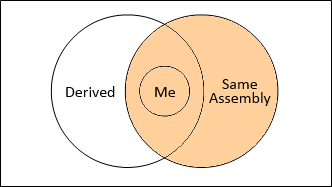
Access modifiers are keywords used to specify the declared accessibility of a member or a type.

Interfaces and classes can only be public or internal.

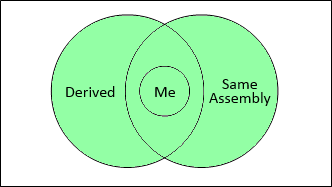
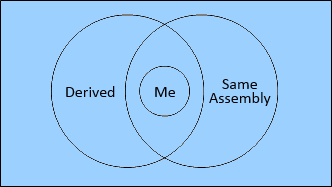
Private: Access is limited to the containing type. Private protected: Access is limited to the containing class or types

derived from the containing class within the current assembly.  
 

Internal: Access is limited to the current assembly. Protected: Access is limited to the containing class or types derived from

the containing class.  


Public: Access is not restricted. Protected internal: Access is limited to the current assembly or types

derived from the containing class.   


Extended Modifiers

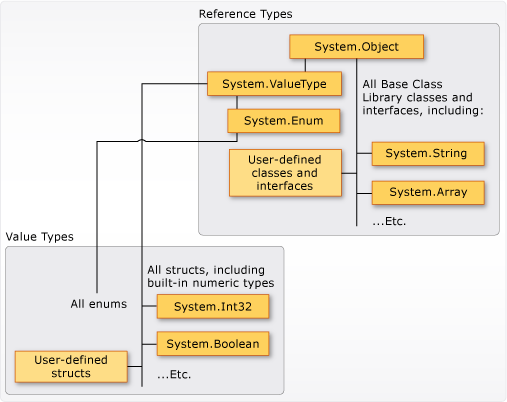
|  |  |  |  |
| --- | --- | --- | --- |
| Modifier | Class | Member | Description |
| static | Yes | Yes | A static method or field will be fixed for all derived objects. A static class is just a container for static methods. You cannot make an instance of this class. Cannot be derived from. |
| const | No | Yes | Specifies that the value of the field or the local variable cannot be modified. |
| readonly | No | Yes | Declares a field that can only be assigned values as part of the declaration or in a constructor in the same class. |
| sealed | Yes | No | Specifies that a class cannot be inherited. |
| abstract | Yes | Yes | Indicates a class or member that is intended to only be a base class of other classes. Classes or members with abstract modifiers do not provide implementation. |
| virtual | No | Yes | Declares a method or an accessor whose implementation can be changed by an overriding member in a derived class. |
| new | No | Yes | Explicitly hides a member inherited from a base class. Used to "override" existing members that are not virtual. |
| partial | Yes | No | Allowed to break definition of a class into multiple namespaces within an assembly. |

new vs virtual:

Parent p = new Child(); //upcasting

virtual: P.method() -> calls the child's implementation

new: P.method() -> calls the parent's implementation

Types  
  


Value types are defined by using the struct keyword/or are built-in numeric types. Structs cannot be used to derive a new struct. Enums are value types. This refers to a distinct type that consists of a set of named constants called the enumerator list. Methods, constructors, and other properties can be defined in a struct.

As soon as value types are out of scope (the method they are in ends), their memory is freed.

Reference types are defined by using the class keyword. Reference type variables only contain a reference to the location of the object. These 2 types have different compile-time rules, and different run-time behavior.

Reference types stay in memory, even outside of their own scope. CLR will try to figure out when to free up the memory of the reference types.

Garbage Collection

Garbage Collection refers to the CLR's attempt to free up memory. It will try to figure out what objects could never be referenced again

Automatic Garbage Collection takes up resources to find these objects.

In the CLR, "managed" code is what is referred to as the code that has GC.

CLR managed code hierarchy

Gen 0: when the object is created. Short-lived.

Garbage collection in CLR

Gen 1: object survived a number of passes. Checked less

Garbage collection in CLR

Gen 2: object survived and is now checked very rarely.

.NET can force garbage collection by using method GCCollect();

Interfaces

An interface contains definitions for a group of related functionalities that a [class](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/class) or a [struct](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/struct) can implement. A class or struct that implements an interface must implement all the members of that interface, because the interface has no implementation of its own. They allow you to include behavior from multiple sources in a class. This is important because C# has no native implementation of multiple inheritance.

Syntax:

interface ISampleInterface

{

void SampleMethod();

}

class ImplementationClass : ISampleInterface

{

// Explicit interface member implementation:

void ISampleInterface.SampleMethod()

{

// Method implementation.

}

}

Generics

Generics make it possible to design classes and methods that defer the specification of one or more types until the class or method is declared and instantiated by client code. They introduce the concept of type parameters to C#. Use generics to limit code reuse, increase type safety, and increase performance. The information on the types that are used is obtained at run-time. Generics are most frequently used with collections and the methods that operate on them. 

Example:

public class GenericList<T>

{

public void Add(T input) { }

}

class TestGenericList

{

private class ExampleClass { }

static void Main()

{

// Declare a list of type int.

GenericList<int> list1 = new GenericList<int>();

list1.Add(1);

// Declare a list of type string.

GenericList<string> list2 = new GenericList<string>();

list2.Add("");

// Declare a list of type ExampleClass.

GenericList<ExampleClass> list3 = new GenericList<ExampleClass>();

list3.Add(new ExampleClass());

}

}

SOLID

Single Responsibility Principle – A class/method should only have one responsibility. In other words, a class/method should have one, and only one, reason to change.

Open-Closed Principle - You should be able to extend a class's behavior, without modifying it. Class should be open for extension, closed for modification. Achieved through abstraction. Delegates are a good example.

Liskov Substitution Principle - Derived classes must be substitutable for their base classes/ Subclasses should not be made in a way that breaks the functionality of the superclass. Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program

Interface Segregation Principle - Make fine grained interfaces that are client specific. Specific interfaces > larger general purpose interfaces. Reduces redundant coupling

Dependency Inversion Principle - Depend on abstractions, not on concretions. Prevents redundancy in code and hard dependencies between classes. Provides extensibility to your code and reduces coupling.

Casting, Boxing, ref, out (fix)

Boxing is where you take a value type and you give it reference type semantics.

Boxing requires upcasting from a value type.

Ex.

int a = 10; //value type

object o = a; //boxing (o has reference type semantics)

int b = (int)o //unboxing (b has value type semantics)

downcasting can fail, if what is being downcasted is not actually above the source class.

ref - modifier for parameters. It indicates a value that is passed by reference.

Used to pass an argument to a method by ref. Used to return a value to the called by reference.

out - is a modifier for parameters. It allows arguments to be passed by reference, but unlike ref, the variable doesn't need to be initialized

Delegates and Event Handling

A [delegate](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/delegate) is a type that represents references to methods with a particular parameter list and return type. When you instantiate a delegate, you can associate its instance with any method with a compatible signature and return type. You can invoke (or call) the method through the delegate instance.

Delegates are used to pass methods as arguments to other methods. Event handlers are nothing more than methods that are invoked through delegates. You create a custom method, and a class such as a windows control can call your method when a certain event occurs.

Example:

delegate bool comparer(string a, string b); //string comparer, which defines a type

void sort(comparer comparer)

{

\*code

}

Func/Action - Generic delegates

Sort(Func<string,string,bool> comparer) //dont have to define the delegate type with func

{

\*code

}

func syntax: func(n-1 parameters, return type)

action syntax: action(n parameters, return void)

events - provided by a class. represents something happening. consumers of that class can subscribe andd run code based on event activation.

+= to subscribe

-= to unsubscribe

Unit Testing

Intuitively, one can view a unit as the smallest testable part of an application. In [procedural programming](https://en.wikipedia.org/wiki/Procedural_programming), a unit could be an entire module, but it is more commonly an individual function or procedure. In [object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming), a unit is often an entire interface, such as a class, but could be an individual method. Unit tests are short code fragments created by programmers or occasionally by [white box testers](https://en.wikipedia.org/wiki/White-box_testing) during the development process. It forms the basis for component testing. Ideally, each [test case](https://en.wikipedia.org/wiki/Test_case) is independent from the others.

xUnit is the tool we will be using in Visual Studio

***Facts*** are tests which are always true. They test invariant conditions.

***Theories*** are tests which are only true for a particular set of data.

Inline data is forced data that is passed to the unit test method

LINQ and Lambda expressions

Language-Integrated Query (LINQ) is the name for a set of technologies based on the integration of query capabilities directly into the C# language

LINQ - provides extension methods. Goes along with SOLID design

An assembly that provides extension methods to any IEnumerable class

Operable on many types of organized data structures.

Use a static method with a "this" keyword to create extension methods

LINQ Select - takes every element in a list and maps them in a new form

LINQ Query Syntax - Looks like SQL/JQuery. Query Syntax has less capablities than method syntax

A lambda expression is an [anonymous function](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/statements-expressions-operators/anonymous-methods) that you can use to create [delegates](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/delegates/using-delegates) or [expression tree](http://msdn.microsoft.com/library/fb1d3ed8-d5b0-4211-a71f-dd271529294b) types. By using lambda expressions, you can write local functions that can be passed as arguments or returned as the value of function calls. Lambda expressions are particularly helpful for writing LINQ query expressions.

To create a lambda expression, you specify input parameters (if any) on the left side of the lambda operator [=>](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/lambda-operator), and you put the expression or statement block on the other side. For example, the lambda expression x => x \* xspecifies a parameter that’s named x and returns the value of x squared. You can assign this expression to a delegate type, as the following example shows:

delegate int del(int i);

static void Main(string[] args)

{

del myDelegate = x => x \* x;

int j = myDelegate(5); //j = 25

}

Extension methods

Extension methods enable you to "add" methods to existing types without creating a new derived type, recompiling, or otherwise modifying the original type. Extension methods are a special kind of static method, but they are called as if they were instance methods on the extended type.

Debugging  
  
Serialization  


The object is serialized to a byte stream, from that stream, it can be stored in a database, file, or memory

XML (Extensible Markup Language) serialization

name-obj.xml (XML example)

<Name>

<FirstName>

Lance

</FirstName>

<LastName>

Von Ah

</LastName>

</Name>

comparable to C# code

public class Name{

public string FirstName {get; set;}

public string LastName {get;set;}

}

Serialization can also be done in reverse. Deserialization will take the byte stream and turn it into objects

Won't totally fail if XML is not formatted correctly.

Ways to serialize: XML, JSON, etc.

When an object is serialized, you cannot deserialize without the definition of the object.

As long as the information can be accepted by the type value, type is not retained after serialization

XML Syntax: <Age /> - empty element/null element

Task, async and await (Asynchronous programming)

The await operator is applied to a task in an asynchronous method to insert a suspension point in the execution of the method until the awaited task completes. The task represents ongoing work.

await can only be used in an asynchronous method modified by the [async](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/async) keyword. Such a method, defined by using the async modifier and usually containing one or more await expressions, is referred to as an async method

Singleton Class

A singleton is a class which only allows one instance of itself to be created - and gives simple, easy access to said instance. The singleton premise is a pattern across software development.

There is a C# implementation ["Implementing the Singleton Pattern in C#"](http://csharpindepth.com/Articles/General/Singleton.aspx) covering most of what you need to know - including some good advice regarding **thread safety**.

To be honest, It's very rare that you need to implement a singleton - in my opinion it should be one of those things you should be aware of, even if it's not used too often.

public class Singleton

{

private static Singleton instance;

private Singleton() { }

public static Singleton Instance

{

get

{

if (instance == null)

instance = new Singleton();

return instance;

}

}

//instance methods

}