Lecture notes/Miscellaneous

SDK -> .Net Core

- Assembly vs. Namespace
  - Namespace: logically arranges types (classes) to avoid naming conflict
  - Assembly: physical separation of types (classes); ends with .exe or .dll; Deployable units
- Properties like getters and setters in Java called smart fields by Microsoft
  - encapsulate value of private variables

C# is typesafe language and case sensitive

Access Modifiers

- Class Accessibility
  - classes and structs declared directly within a namespace (not nested within other classes or structs) can only be either public or internal
  - internal is default
  - derived classes can't have greater accessibility than base classes
  - control which regions of program text can access the member:
    - public
      - type or member can be accessed by any other code in same assembly or another assembly referencing it
      - can be used for types and type members
    - private
      - type or member can be accessed only by code in same class or struct
    - protected
      - type or member that can be accesed only by code in same class or class derived from that class
    - internal
      - type or member can be accessed only by code in same class or in class derived from that class (same assembly)
    - protected internal
      - type or member can be accessed by any code in assembly where it's declared or within derived class in another assembly
      - struct members cannot be this as structs cannot be inherited
    - private protected
      - type or member can only be accessed by types derived from class but only if in same assembly

---

Data Types

- Primitive Types
  - C# classifies what other languages call primitive types as objects
  - makes C# strongly typed
- Structure
  - all data types inherit from bas Class Object
    - when int declared actually declaring instance of struct (object) of type int which inherits from

System.ValueType which inherits from System.Object

- reference and value are the 2 categories of variable type available in C#
  - reference: store reference to data not instance of it (objects)
    - value stored in heap memory
    - expensive retrieval process
    - examples include:
      - classes, interface, and delegates
      - also predefined types: string, arrays, collections, etc.
    - class
      - keyword class
      - can declare class fields, constructors, and methods
    - interface
      - contains definitions for group of related functionalities a non-abstract class or struct must implement
      - defines contract
    - delegate
      - represents references to methods
      - make it possible to treat methods as entities that can be assigned to variables and passed as parameters
      - object-oriented and type-safe
    - object
      - all types inherit directly or indirectly from System.Object
      - can be assigned values of any type to variables of type object
      - can be assigned to default value by using null
      - when variable of a value type is converted to object is boxed
      - when variable of type object is converted to value type is unboxed
    - string
      - represent sequence of 0+ unicode characters
      - alias for System.String
      - operators (+ == !=) defined to concatenate and compare value of string objects not references
      - immutable
      - can find character by index same as in Java using []
  - value: directly contain data
    - stored directly in memory on the Stack
    - faster access
    - built in types: int, long, short, byte, DateTime, char
    - also user-defined types declared with struct
    - struct
      - can encapsulate data and related functionality
      - defined with struct keyword
      - used to design small data-centric types providing little or no behavior

- enum
  - use enum keyword and specify names of members to define enumeration type
  - value type defined by set of named constants of underlying integral numeric type
  - immutable
- integral:
  - numeric types representing integers
  - simple types and can be initialized with literals

Extended Modifiers

- abstract
  - the thing being modified has missing or incomplete implementation
  - only meant to be base class of other classes NOT instantiated alone
  - can apply to classes, methods, properties, indexers, and events
  - members must be implemented by non-abstract classes deriving abstract class
  - Classes
    - cannot be instantiated
    - can contain abstract and non-abstract methods and accessors
    - must provide implementation for interface members
    - cannot include sealed modifier
  - Methods
    - implicitly virtual method
    - only permitted in abstract classes
    - do not have {} or method body
    - only have implementation in derived class methods with overriding
    - cannot have static or virtual modifiers
  - Properties
    - same as methods
    - properties are written with {} but still do NOT have implementation
- Virtual
  - used to modify a method, property, indexer, or event declaration allowing for it to be overridden in derived class
  - implementation can be changed by overriding member in derived class
  - by default methods are non-virtual
  - cannot override non-virtual method
  - cannot be used with static, abstract, private, or override modifiers
  - virtual inherited property can be overridden
- Sealed
  - prevents inheritance from class
  - prevents overriding method from being overriden by a further method
- Static
  - Classes
    - cannot be instantiated or extended

        - all members must be static
        - container for static members
    - All Members
        - cannot use "this" to reference static methods or property accessors
        - belongs to class type itself not specific object instance
        - referenced through the type name
- Const
    - fields and locals aren't variables and may not be modified
    - fields can be numbers, boolean, strings, or null
    - only reference types that can be const are string and a null reference
- readonly
    - initialization can only occur as part of declaration or in constructor in same class
    - like const, but initialization can be deferred until constructor finishes
- Override
    - Classes
        - modifier required to extend or modify abstract or virtual implementation of inherited method, property, indexer, or event
    - all members
        - provide new implementation of inherited methods
        - must have same signiature as inherited method
        - both methods must be virtual, abstract, or override
        - cannot use static or virtual modifiers to modify an override method
- Partial Classes, Structs, Interfaces
    - can split definition of class, struct, interface or method over 2+ source files; each file contains a section; all parts are combined on compilation
    - would be used when working with automatically generated source code that can be added to the class without having to recreate the source file
        - attributes, inherited classes, etc. are merged at compile-time

---

Class, Struct, Interface, Enum

- state = fields
- actions = methods
- Class
    - class declaration start with header specifying attributes and modifiers, name of class, base class (if one is present), and interfaces implemented by class
    - body consisting of a list of member declarations written between {}
    - Accessibility
        - classes and structs are declared within a namespace not nested within other classes or structs
        - classes and structs can be public or internal
            - internal: default
            - derived classes: can't have greater accessibility than base types

- Instance Instantiation
  - instances created with new operator which:
    - allocates memory for new instance
    - invokes constructor to initialize instance
    - returns reference to instance
  - memory occupied by object reclaimed by GC when object is no longer reachable
- Members
  - 2 categories of class members
    - static: belong to classes
    - instance: belong to objects (instance of classes)
  - categories of members in class
    - constructors: initialize instnces of class
    - constants
    - fields: variables
    - methods: computations/actions that can be performed
    - properties: fields combined with actions associated with reading/writing them
    - types: nested types declared by class
  - Accessibility
    - private: class only
    - protected: derived classes
    - private protected: class or derived classes only
    - internal: current assembly (.exe, .dll)
    - protected internal: this class, child classes, or classes within assembly
    - public: unlimited
- Local Variables
  - declared inside body of the method
  - must have type name and variable name
  - all get default value (0, "")
- Methods
  - 2 categories:
    - static: accessed directly through class
      - use static modifier
      - does not operate on specific class instance
      - can only directly access static members
      - cannot use "this"
    - instance: accessed through instances of a class
      - declared without static modifier
      - operates on specific class instance
      - can access static and instance members
  - must have method signiature:
    - name of method

        - optional type parameters
        - parameters
        - does not include return type
    - value and reference parameters
        - used to receive values or variable references from method calls
        - 4 types:
            - value
                - copy of argument passed
                - changes don't effect original argument
            - reference
                - declared with 'ref' modifier
                - used to pass arguments by reference
                - argument must be variable with definite value
                - changes take place on original value
            - output parameter
                - declared with "out"
                - used to pass arguments by reference
                - explicitly assigned value is not required before method call
            - parameter array
                - permits variable number of arguments to be passed to a method
                - declared with "params"
                - must be last parameter and be 1-D array
                - write() and writeline() use parameter arrays
    - Overloading
        - permits multiple methods in same class to have same name
        - must have unique parameter lists
        - compiler uses 'overload resolution' to determine which method to invoke
        - 'overload resolution' finds the method that **best** matches arguments or reports error
        - can be selected by explicitly casting arguments to exact parameter types
- Interface
    - contract, implementing classes must implement all methods defined in interface
    - can contain methods, properties, and events
    - does not provide implementations
    - implementation is not inheritance
    - intended to express a "can do" relationship between interface and implementing type
    - can employ multiple
    - classes and structs can implement multiple
- Class Type Parameters
    - used to define generic class type
    - follow class name and are inside <>
    - used to define members of class

- Base Classes
  - specifies inherited base class by following class name and type parameters

---

Semantic code (DRY, Comments-inline, Comments-XML, KISS)

Common Language Runtime (BCL, CIL, CLI, CLR, CTS, JIT, VES)

- Common Language Runtime (CLR)
  - .NET framework consists of CLR and .NET framework class library
  - foundation for .NET Framework; manages and runs code and provides services like memory management, remoting, type enforcement (through the CTS) and security
  - benefits
    - cross-language integration
    - cross-language exception handling
    - enhanced security
    - versioning and deployment support
    - simplified model for component interaction
    - debugging and profiling services
- Class Libraries
  - Base Class Library (BCL)
    - foundation for C# runtime library and one of the Common Language Infrastructure (CLI)
    - provides types representing built-in CLI data types, basic file access, collections, custom attributes, formatting, security attributes, I/O streams, string manipulation, etc.
  - object-oriented collection of reusable types that can be used to develop apps from command-line or graphical user interface to other apps based on lates inovations
  - .NET CLR and Class Library relationships
  - .NET CoreFX
    - FX fork of .NET framework BCL
    - foundational class library for .NET Core
    - defines types for primitives, collections, file systems, console, JSON, XML, async, and others
    - makes up .NET Core BCL
  - Managed Code
    - managed by CLR at runtime
    - CLR knows what code is doing and can manage it
    - CLR provides memory management (GC), security boundaries, type safety, etc.
    - managed code is written in high-level language that can be run on top of .NET
    - code compiled into Intermediate Language code the CLR compiles and executes
    - manages Just-In-Time compiling of code from IL to machine code that can be run on a CPU
  - Unmanaged Code
    - runs outside CLR
    - .NET Framework promoted interaction with COM components, COM+ services, external type libraries, and many operating system services

- examples:
  - COM components
  - ActiveX interfaces
  - Windows API functions
- Idisposable Interface
  - GC has no knowledge of unmanaged resourses
  - provides method for releasing unmanaged resources
  - must call IDisposable.Dispose implementation when finished using it
    - Using Block
      - using statement can be used instead of explicitly calling IDisposable.Dispose yourself
  - Using Block and IDisposable
    - provides convenient syntax ensuring correct use of IDisposable objects
    - if lifetime of IDisposable object is limited to single method should be declared and instantiated in using statement
    - using statement calls Dispose method on object causing object to go out of scope when .Dispose() is called
    - obect is read-only and cannot be modified or reassigned within using block
    - Using Block
      - ensures .Dispose is called even if exception occurs in block
      - can achieve same result by putting object inside try block and calling .Dispose in finally block
      - expanded to try/catch block at compile time

CLI -> Common Language Infrustructure

C# f# VB

CSC fsc VBC -- language specific; all can be combined using dotnet build using CIL

CIL (makes code platform independent) -> Runtime (CLR) also platform independent

use dotnet new console -l VB -o directory name (to create console with different language than c#)

see .NetCLI on Day 1 slides notes