Test-driven Development (TDD)

Unit Testing

- Unit Testing
  - testing small units of code
  - helpful to make sure everything works perfectly (without repeating some actions)
  - helpful in pinpointing where code is going wrong
- Parts of unit test
  - arrange
    - any setup necessary to prepare for behavior to test
    - assume what you are testing works
  - act
    - do the thing you want to test
    - usually name the method after what you want to test
  - assert
    - verify behavior was as expected
- Test Driven Development (TDD)
  - write tests that fail
  - implement code to make tests pass

Runtime environment (garbage collection, managed, unmanaged)

- Garbage Collection
  - within CLR
  - manages allocation and release of memory
  - Fundamentals of Memory
    - all programs have 2GB of virutal memory allocated
    - cannot decide where or how memory is allocated
    - GC allocates and frees memory
    - 3 states of virtual memory
      - free
        - unallocated and available
      - reserved

        - available and unusable for other processes
        - must be commited to store data
    - commited
        - assigned to physical storage
    - frequency depends on volume of allocations and amount of survived memory on managed heap
- Benefits
    - no memory leaks
    - efficient memory allocation
    - automatically reclaims unused objects, clears memory, and makes memory available
    - constructors do not have to initialize every data field
    - makes sure that one object cannot use contents of another
- Managed Heap
    - GC allocates segment of memory to store and manage objects
    - one for each managed process
    - calls windows VirtualAlloc() to reserve memory and VirtualFree() to release memory
    - divides objects into small and large objects to send them to proper object heap (Large Object Heap [LOH] and Small Object Heat [SOH])
        - small are instances
        - large are arrays
    - Heap Object Generations
        - GC happens on whole generation at once
        - surviving objects are promoted to next generation
        - if survival rate is high GC allocates more memory to the generation
        - after surviving GC survivors are 'compacted' (defragmentation) to older end of memory

more notes in Day 2 notes on bottom section look at: managed code section to end of bullet points

---

Delegates (Action, Event, Function, lambda, LINQ, Predicate)

- Binding
  - process establishing connection between UI and data it displays
  - when data changes values elements in memory that are bound to the data reflect changes
  - early binding
    - static binding, static dispatch, or compile-time binding
    - compiler (linker) directly associates stack memory address to function
    - replaces function call with machine language instruction telling mainframe to leap to address of function
  - late binding
    - dynamic binding, dynamic dispatch, dynamic linkage, or run-time binding
    - algorithm created and uses caller-supplied method implementing part of algorithm
    - in .NET refers to overriding virtual machine
      - compiler builds virtual tables for every virtual or interface method call
      - method call used at run-time to determine which implementation to execute
- Delegate
  - used to pass methods as arguments to other methods (like parameters)
  - ideal for defining callback methods
  - type representing references to methods with defined parameter list and return type
  - can associate with any method with compatible signature and return type upon instantiation
    - can invoke that method through delegate instance
    - all accessible methods or structs matching delegate type and return type can be assigned to delegate
  - object-oriented, type-safe, and secure
  - step by step creation
    - create method to assign to delegate

- create delegate type (Del) with same return type and argument
- instantiate delegate type variable and assign to function: variable is passed to function
- invoke variable if original method returns value it will be returned through delegate variable

- Action and Action<T>
  - can use action to pass method as parameter withoug declaring custom delegate
  - always return void; encapsulated method must correspond to parameter list and return void as well
- Func<TResult>
  - TResult is tpe of return value of method this delegate encapsulated
  - type of parameter is covariant (specified or more derived type)
  - represents method that can be passed as parameter without explicit declaration of custom delegates; encapsulated method must correspond to method sigiature
- Predicate: returns Boolean type
- Types
  - single-cast
  - multi-cast
    - Multicasting
      - invoking multiple methods when invoked
      - use addition assignment operators to add extr methods
      - reference arguments passed sequentially to each method any changes are visible to the next
      - if a method throughs uncaught exception execution stopes and exception is passed to caller of delegate
      - if delegate has return type and/or out parameters return return value and/or out parameters of last method invoked
      - use subtraction operators to remove method from delegate
  - strongly-typed: predefined
- Events

- enable class or object to notify others when something occurs
- publisher class is the one that sends the event
- subscribers receive the event
- special kind of multicast delegate that can only be invoked from within class or struct where declared (aka the publisher)
- steps to trigger event
    - recipient creates method to handle event
    - creates delegate for method
    - passes delegate to event source
    - source calls delegate when event occurs
    - calls event handling method on recipient delivering event data
        - delegate type for event is defined by event source
- Properties
    - publisher- determines when event raised
    - subscriber- determines what action to take in response; each can handle multiple events from multiple publishers
    - event- can have multiple subscribers; signal user actions like clicks or menu selections
    - event handlers invoked synchronously when an event has multiple subscribers
    - in .NET Framework events based on handler delegate and EventArgs base class
- publishing based on EventHandler pattern
    - standard signature of eventHandler delegate defines method which returns nothing
    - first parameter is type object and refers to instance raising event
    - second parameter derived from EventArgs and holds event data
    - EventHandler delegate is predefined and specifically represents event handler method for event that doesn't enerate data
        - if event generates data must use the generic EventHandler<TEventArgs> class
    - need to add instance of eventHandler delegate to event to associate them to eachother
    - model follows observer design pattern enabling subscriber to register with and receive

notifications from provider

- event sender pushes notification event happened
- events defined by using keyword in signature of event class and specify type of event receiver recerives notification and defines response to it

- step by step pattern to send custom data

1. (skip to 3 for non-customized data) declare class for custom data where it is visible to publisher and subscriber then add required members to hold data
2. (skip if using generic EventHandler<TEventArgs>) declare delegate in publisher with name ending in EventHandler having 2 parameters (object sender, CustomEventArgs a)
3. declare event in publisher using one of these ways:
    a. for no custom EventArgs class: event type is non-generic EventHandler delegate and does not have to be declared (public event EventHandler RaiseCustomEvent;)
    b. for non-generic version of EventHandler **AND** custom class derived from EventArgs: declare event in publishing class and use delegate from step 2 as type (public event CustomEventHandler RaiseCustomEvent;)
    c. for generic version: specify event type (public event EventHandler<[CustomEventArgs> RaiseCustomEvent;)

- Subscribing
    - define event handler method with signature matching delegate signature for event
    - use += to attach event handler to event
- Unsubscribing and Garbage Collection
    - unsubscribe from event using -= to prevent handler being invoked when event is raised
    - should unsubscribe from events before dispose subscriber object to prevent leaks
    - until unsubscribed multicast delegate underlying event in publishing object has reference to delegate encapsulating subscriber's event handler
    - GC will not delete subscriber object as long as publisher holds the reference
    - when all subscribers have unsubscribed from event, instance is null
- Method chaining- creating pipeline and lifecycle of framework
    - make methods run sequentially by combining delegate methods and running del();
- Anonymous Methods- foundation of lambda expression

    - 1 line of code to perform method
- loose coupling

Serialization (FileIO, Regular expressions, Serializer-JSON/text/XML)

- process to convert object into stream of bytes to store or transmit it to memory, database, or file
- save state of object for later recreation
- deserialization
    - process of unpacking serialized objects
- formats
    - custom binary
    - customer text (built into C#)
        - XML or JSON
- File/IO
    - data travels from source to program using stream
    - 2 types of streams
        - byte
            - used to write to external file
            - bridge between code and file
            - send bytes to byte stream so they can be written externally
        - character
            - also used to write external files
            - stream of characters
            - used to write stream of text to file

Variance (As, Boxing, Casting, Contravariant, Covariant,

Invariant, Is, Out, Ref, Typeof)

Variance

- type variance
  - describes type that can be subsititued in place of another type
  - covariance: substitution is covariant with type if any other class with subclass relationship is valid
  - contravariance: substitution is contravariant with type if other classes within superclass relationship is valid
  - invariance: substitution invariant with type if only types exactly the one in question are valid

Types (Boxing, Casting, Typeof)

- type casting
  - boxing
    - upcasting
    - casting to supertype
    - implicit type casting
  - unboxing
    - downcasting
    - casting to subtype
    - explicit type casting
  - note types similar to eachother can be casted to similar type
- Type checking
  - is operator
    - check if run-time type of object compatible with given type
    - returns true if same type otherwise false
  - typeof operator
    - used to get type at compile-time (System.Type object)
    - gives CTS version of type

- takes type as argument and returns marked type of argument
- often confused with GetType() method which returns type does not check type

Passing by Reference (Out, Ref)

- refers to actual object so changes effect the object
- out keyword
  - used to pass arguments to methods as reference type
  - generally used when method returns multiple values
  - parameter does not pass the property
  - variable does not need to be initialized when passing parameter
- ref keyword
  - used to pass arguments by reference
  - changes made in argument method will reflect variable when control returned to calling method
  - does not pass the property
  - needs to be initialized before passing

Multithreading (Async/await, Task, Thread)

Async/Await

- Task<> asynchronous programming model
  - used to avoid performance bottlenecks and enhance responsiveness
  - application can continue running doing something else while blocking task finishes
- Modifiers - Async
  - specify method asynchronous
  - uses await operator to continue working without blocking thread
  - method runs synchronously until reaching first await expression where it stops until awaited task finishes while control returns to async method caller
  - keyword contextual as only keyword when modifies a method otherwise its an identifier

    - can't declare any in, ref, or out parameters or return a reference value but can call methods with those parameters
    - can have only return types Task or Task<TResult>
    - only async methods can call other async methods
    - all methods must have await in them
- both keywords together allows use of .NET Framework and Core to create asynchronous methods
    - signature includes async modifier
    - return type is Task<int> or Task
    - conventionally method name ends in Async
    - GetStringAsync returns Task<string>
    - when you await task, get a string (urlContents)
    - before awaiting task can do work that doesn't rely on string from GetStringAsync