Collections

in memory storage of data

- Arrays: 1-D, 2-D, jagged, multi-dimensional
  - drawbacks = size of array is fixed upon instantiation
  - memory contiguous makes looping through easy
  - elements must have values
  - rank is number of dimensions in array
  - jagged arrays are arrays where array has a set number of row but can have differing lengths
  - type-safe
- Generics (avoid need for boxing by using arrays internally)
  - Stack<T>: linear data structure LiFo
  - Queue<T>: linear data structure FiFo
  - List<T>: dynamic array
  - Dictionary<TValue, TKey>: key value pairs don't need to iterate to get data you need
  - SortedList: array of key value pairs sorted by keys
- Non generics (all items are converted to object when added to collection [boxing])
  - Stack: linear data structure LiFo
  - Queue: linear data structure FiFo
  - List: dynamic array
  - Hash Table(value, key)
  - ArrayList
- Boxing
  - changing value type to object type or interface type
- Unboxing
  - changing object type to value type or from interface type to value type

collections inherit from interfaces that determine functionality

iterator(iterability)

IEnumerable (GetEnumerator func.)

IEnumerable<T>

ICollection ->Add()/Remove() functions

IList<T>	Stack<T>	Queue<T>	IDictionary<TKey, TValue>
List<T>			Dictionary<TKey, TValue>

OOP Pillars

- 4 pillars: (A PIE)
  - abstraction
    - make complex stuff sinpler by hiding implementation details of how data is processed, how logic is executed
    - show end user properties/methods available for use

- implemented using interfaces and abstract classes
- separates needed functionality and implementation details
- interfaces (like a contract)
    - contains definitions for group of related functionalities non-abstract class must implement
    - can't declare instance data
    - all method stubs declared in interfaces are implicitly abstract
    - can define static methods which must have implementation
    - does not have constructors
    - cannot be instantiated
    - may define default implementation for members
    - implemented same way as extending
- abstract classes
    - must use abstract keyword
    - can have constructors to initialize fields
    - typically contain unimplemented method stubs
    - may have abstract and non-abstract methods
    - may have fields
- abstract classes used to impose/create hierarchy
- interfaces used to impose functionality/behavior

- polymorphism
    - many forms
    - present same interface but inputs/data types utilized would be different
    - ability to substitute different implementation details for different needs
    - abilty of object to take on many forms
    - implementations
        - ad hoc
            - common interface for arbitrary set of individually specified types
        - parametric
            - when type can be declared abstractly so aspects of implementation can be declared at runtime through use of parameterized type
        - inclusion (sub-typal)
            - name denotes instances of many different classes related by common superclass
- inheritance
    - mechanism of basing object or class upon another object or class retaining implementation
    - is-a relationship
    - promotes reusability
    - constructors inherited from parent to child
    - can inherit from multiple interfaces but only 1 class
    - denoted by :
    - can inherit from 1 class and multiple interfaces at same time but class must be listed first

and all separated by ,
    - use base to call back to parent class
    - Composition, Aggregation, Association
        - Composition
            - relationship between objects where 1 is instance of the other object(s) [has - a]
            - composite object responsible for lifecycle of all constituaent instances
            - super power needs to have a hero
        - Aggregation
            - describes relationship between objects where object has multiple instances of other objects [has-a]
            - lifecycle of object is independent of aggregate object's
        - Association
            - describes relationsip between 2 objects where one uses a separate instance of another object as part of its functionality (uses-a)
            - lifecycle of objects is entirely separate from one another
- encapsulation
    - treat related data/behavior in single unit/capsule
    - validation and porocessing of data in classes would be done in the class itself
    - implemented using data hiding (by use of access modifiers) and wrapping (grouping logic in classes, assemblies, and namespaces)
        - wrapping focuses on encapsulating complex data to present simpler view for the user
        - hiding focuses on restricting use of the data to assure data security

---

Exception Handling

- Exceptions vs Errors
    - run-time errors can occur for variety of reasons should not be handled as exceptions all the time
    - 3 types of run-time errors:
        - usage errors
            - error in logic that should be addressed by handling faulty code not exception handling
        - program errors
            - run-time error that cannot always be avoided by writing bug-free code
            - may reflect routine error condition
            - avoid using exception handling to deal with program errors
            - better to prevent execution by trying the action first
                - USE => DateTime.TryParseExact (returns Boolean)
                - DO NOT USE => DateTime.ParseExact(throws format exception)
        - system failures
            - run-time error that cannot be handled programatically in a meaningful way
            - should not be handled by using exception handling
            - any method can throw OutOfMemoryException if CLR is unable to allocate additional

memory

- Exception class
  - base class for all exceptions
  - when error occurs system or application throws exception containing information about exception including location
  - idk exception thrown by method far down call stack, CLR will unwind stack looking for method with catch block for specific exception type and execute first catch block that it matches
- Try/Catch/Finally block
  - within try/catch block use catch block for most specific exception first more general later once a catch block is executed it gives the exception and breaks out of block
  - if no catch block matches exception you get unhandled exeption message and stops the program entirely
  - block contains guarded code
  - finally can be put at bottom of block
    - always executed no matter what
    - good idea to put important code that will always run such as closing file
- Exceptions - Throw
  - throw statement can be used to re-throw exception caught by catch statement to next method up the stack also receives it
  - can catch one exception and throw a different one
    - must specify exception that is caught in inner exception
  - throw; keeps track of location where exception occurs
  - thow e; only tracks where the exception is being thrown from most recently (will give where catch block is not problem area of code)
- User-Defined Exceptions
  - create your own exception class
  - derive (inherit) from Exception class
  - end class name with word "Exception"
  - implement the 3 common constructors (?)

Application Monitoring

- highlighting certain events that come up during runtime
- useful in debugging program
- Breakpoints
  - can be added to certain lines of code during development
  - program pauses when it hits this point
  - during pause you can:
    - check values of any variables in scope
    - continue on to next point
    - step over function that is called
    - step into function that is called

    - step out into the original line called the function
- Logging
    - where certain events during program runtime is recorded
    - events that warrant a record:
        - errors
        - business logic events
        - transactions with external systems
        - etc
    - levels
        - verbose: anything and everything you might need to know about running block of code
        - debug: internal system events that aren't observable from the outside
        - information
        - warning: service degreaded or endangered
        - error: functionality unavailable, invariants are broken, or data is lost
        - fatal: very severe events that lead to application stopping
    - where are the logs?
        - files: depends on cofiguration; .txt .xml
        - DBs: can use databases to store logs
        - console: terrible idea
        - etcetera: Marrielle suggests researching logging software to ease sifting through production level logs
- Tracing
    - used to track user's interaction with system
    - shows events that occur during runtime
    - highly involved in debugging and includes information such as functions called and parameters provided while user interacts with program
    - same steps as user to find where bug occurs

Regular Expressions

- used to regulate input
- primarily used in pattern matching
- using certain notation you can build an expression to check if a certain input follows a pattern
- can build and utilize RegEx expressions using RegEx class