* Accessors - Property that contains the executable statements associated with getting and setting the value of the property
* Abstract Class - Classes that cannot be instantiated but can be implemented or not implemented. Can implement an unlimited number of interfaces but can only inherit from only one abstract class
* Constructor - Called whenever a class or struct is called. They enable programmer to set default values, limit instantiation, and write code that is flexible and easy to use
* Delegate - Type that holds references to methods with a particular list and return type. Essentially an easier way to call multiple methods at one time
* Encapsulation - The process of enclosing one or more items within a physical or logical package. Prevents access to implementation details.
* Event - Enable a class or object to notify other classes or objects when something interesting occurs. They are typically used to signal user actions. Ex: Pressing a button on a windows form is an event.
* Inheritance - Enables you to create new classes that reuse, extend, and modify the behavior that is defined in other classes. Classes that inherit from another are called derived class and the class it inherited is called the base
* Interface - Contains only signatures of methods, properties, events, or indexers. If a class implements the interface then it must implement the members of the interface. They can inherit from one or more base interfaces
* Polymorphism - Taking a property or function and changing it in some way, like overloading a function
* Property - A member that provides a flexible mechanism to read, write, or compute the value of a private field. Allows the ability to access members in a class while protecting them from corruption at the same time
* Reference Types - A pointer that that points to another memory location that holds data.
  + Delegates, string, arrays, and class types are all reference types
* Sealed Classes - The sealed modifier prevents other classes from inheriting from it. Much like the private modifier does not allow anything outside the scope to access it
  + <https://msdn.microsoft.com/en-us/library/88c54tsw.aspx>
* Signature - Consists of the name, type, and kind(value, reference, or output) of each of its formal parameters, for method.
  + <https://msdn.microsoft.com/en-us/library/aa691131(v=vs.71).aspx>
* Auto-implemented properties - Make property-declaration more concise when no additional logic is required in the property accessors
* Structs - Value type that is used mainly to encapsulate small groups of related variables.
  + Ex: A struct for a student student’s name, grade level, and email address
* Stack vs. Heap -
  + Stack is used to keep track of the running memory needed in your application, stacks memory allocation on top of another memory allocation. Stack is cleared using Last in, first out meaning that the last memory allocation in is the first one out.
  + Heap is used for dynamic memory allocation. When creating an object, for example, a pointer to that object is created in the stack while the actual object is stored in the heap. The heap does not track running memory, it is just a pile of objects that can be reached whenever necessary.
  + .<http://www.codeproject.com/Articles/76153/Six-important-NET-concepts-Stack-heap-value-types>
* Interfaces (OOP way) - An interface contains definitions for a group of related functionalities that a class or struct can implement
  + By using interfaces you are able to include behavior from multiple sources in a class, important because C# does not allow for multiple inheritance of classes
  + <https://msdn.microsoft.com/en-us/library/ms173156.aspx>
* Derived Classes using the as and is operator - The ‘as’ operator can be used to perform certain types of conversions between compatible reference types or nullable types
  + as is like a cast operation but if the conversion is not possible then as returns null
  + <https://msdn.microsoft.com/en-us/library/cscsdfbt.aspx>
* Value types - Variables that are based on the value types directly contain values.
  + Value types are different types that store data, they are like reference types but value types hold value instead of a reference to a memory location