MCSD Certification Toolkit 70-483

1. **Objectives of MCSD Toolkit Notes**

**Manage Program Flow 25%**

*Implement Multithreading and Asynchronous Processing*

The objective may use Task Parallel Library, create continuation tasks spawn thread using Thread Pool, unblock the UI, use ‘Async’ and ‘Await’ keywords, and manage data by using concurrent collections.

*Manage Multithreading*

Resource Synchronization, locking, cancel running tasks, safe methods for race conditions

*Implement Program Flow*

Iterations across collection and array items, program decisions using switch statements, if/then, operators, expression evaluation.

*Create and Implement Events and Callbacks*

Create Event Handlers, Delegates to create events, lambda expressions, and anonymous methods

*Implements Exceptions Handling*

Handle exception types (SQL, network, communication, timeout) try-catch-finally blocks, throw exceptions, determine throw v rethrow and create custom exceptions.

**Create & Use Types 24%**

*Create Types*

Create value types (structs, enum), references, generics, constructors, static variables, methods, classes, extension methods, optional and named parameters, indexed properties, overloaded v overridden methods

*Consume Types*

Convert between value types, cast types, handle dynamic types, ensure interoperability of unmanaged code

*Enforce Encapsulation*

Using accessors (private, public, protected), use explicit implementation

*Create and Implement Class Hierarchy*

Design and implement an interface, inherit from a base class, implement classes based on IComparable, IEnumerable, IDisposable, IUnknown interfaces

*Find, Execute, Create types at Runtime using Reflection*

Create and apply attributes, read attributes, generate code at runtime using CodeDom and lambda expressions, use types from the system, reflection namespace (Assembly, Property Info, MethodInfo and Type)

*Manage the Object Life Cycle*

Manage resources, implement IDisposable, interaction with finalization, manage IDisposable using ‘Using’ statement, manage finalisation and garbage collection

*Manipulate Strings*

Manipulate strings with String Builder, String Writer, StringReader, search strings, and enumerate string methods, format strings

**Debug Applications and Implement Security**

*Validate Application Input*

Validate JSON data, data collection types, manage data integrity, evaluate a regular expression to validate the input format, use built-in functions to validate type and content out of scope, writing regular expressions.

*Perform Symmetric and Asymmetric Encryption*

Encryption algorithms, manage and create certificates, implement keys and system namespaces, hashing data, encrypt streams

*Manage Assemblies*

Version Assemblies, sign assemblies using strong names, side-by-side hosting, put an assembly in global cache, create Win-MD assembly

*Debug an Application*

Create and manage compiler directives, choose appropriate build types, manage programming database files and symbols

*Implement Diagnostics in an Application*

Implement logging and tracing, profiling applications, create and monitor performance counters, write to event log

**Implement Data Access 26%**

*Perform I/O Operations*

Read-and-write file streams, read and write from network using classes in the System, net namespace, implement asynchronous operations

*Consume Data*

Retrieve data from a database, update data in a database, consume JSON and XML, retrieve data using web services

*Query and Manipulate Data and Objects Using LINQ*

Query data using operators, create method based LINQ queries, query data using comprehension syntax, select data using anonymous types, force query execution, read/filter/create/modify data structures using LINQ to XML

*Serialize and Deserialize Data*

Serialize and Deserialize data using binary, custom, XML, JSON, Data Contract Serializers

*Store Data in and retrieve data from collections*

Store and retrieve data using dictionaries, arrays, lists, sets , queues, collection types, initialize a collection, add/remove items from collections, typed versus not-typed collections, custom collections, collection interfaces

1. **Key Terms Review**

*Assignment* providing a value for a variable

*Boolean* a value represented as either true or false

*Branching* changing code execution to a different path

*Condition* evaluation of operands using logical operators

*Conditional instructions* instructions that evaluate

*Comment* A code line that starts wth // characters for inline documentation

*Complex statement* Enclose simple statements into a code block surrounded curly braces. For repetition and decision structures

*Constant* Named value assigned at time of declaration

*Declaration* used to create variable in code

*Decrement* decrease by a certain value

*Expression* activity or code statement which returns a result

*IEnumerable* Code component in C# that supports iteration

*Increment* increase by a certain value

*Initialize* set a starting value

*Iterator* portion of loop that changes value

*Literal* notation used to indicate fixed values in code. Not same to constant and cannot assign a value to a literal

*Loop* repetition structure that repeats instruction

*Operator* performs an operation on values

*Program flow* logical execution of code

*Sentinel* value used to signal the end of execution on a loop

*Simple statement* statement ending in semicolon, typically used for actions such as declaring variables, assigning values to variables, method calls, code branching

*Spaghetti code* code that is complicated to follow to due to branching

*Statement* actions are performed by statements

*Ternary operator* takes 3 arguments, condition, and value for true, value for false

*Variables* name values that can be changed in code

1. **Working with the Type System**
   1. Value Types
      1. Store the value directly
      2. An alias for System types such as int for System.Int32
      3. Passed as copy to methods
      4. Standard data types in framework
      5. Legal values based on number of bits used to store the type
   2. Data Structures
      1. Structs, Enumerations, Classes
      2. Lightweight data structures
      3. Contain member variables and methods
      4. Passed by value unlike reference types
   3. Enumerations
      1. Contain list of constants
      2. Make code more readable
      3. Underlying value for named constant
      4. Underlying value for type int start 0 and increment by one unless otherwise indicated in declaration
   4. Reference types
      1. Reference types commonly referred to as classes
      2. Contain member variables to store characteristics
      3. Contain member functions to provide functionality
      4. Encompass data and functionality in one package
   5. Modifiers
      1. Determine access for classes and members
      2. Modifiers are listed first in declaration

|  |  |  |
| --- | --- | --- |
| Public | Private | Internal |
| Protected | Abstract | Async |
| Const | Event | Extern |
| New | Override | Partial |
| Readonly | sealed | Static |
| Unsafe | Virtual | volatile |

* 1. Fields
     1. Fields contain data for classes
     2. Fields are also known as member variables
     3. Describe characteristics of the class
     4. Marked as private to avoid unwanted modification
  2. Constructors
     1. Initialize classes
     2. Do not include return type
     3. Same name as class
     4. May contain no parameters(no-args/ default constructor)
     5. If no constructor defined compiler generates default constructor
  3. Methods cats!
     1. Provide class functionality
     2. Can be used with modifiers
     3. Can return values or void type
     4. Accept arguments through parameters in the signature
     5. Optional and named parameters
  4. Overloaded Methods
     1. Same method name with multiple instances for different functionality
     2. Defined by signature
  5. Abstract methods
     1. Do not define implementation
     2. Declared in abstract classes only
     3. End with semicolon
  6. Overridden methods
     1. Hide the implementation of method of same name in base class
     2. Provide means to change method behaviour in derived class
     3. Used for virtual and abstract methods in base class
  7. Extension Methods
     1. Can be applied to your own types or even existing types in .NET
     2. Extend existing classes by adding methods without recompiling
  8. Optional Parameters
     1. Enable choice which parameters are required in a method
     2. Defined as optional but including a default value
     3. The default value is used if none is passed by caller
     4. Must exist after required parameters
     5. If multiple optional parameters exist and a value is specified for one, all optional parameters must also be supplied values
  9. Named Parameters
     1. Allow for giving parameters in a method a name
     2. Increase code readability
     3. Enable Argument passing to a method order
  10. Encapsulation
      1. Also known as data hiding
      2. Invokes making member variable private
      3. Data exposed through properties
      4. Functionality and date are enclosed in the class
      5. Create the black box concept
  11. Properties
      1. Present public interface to your class
      2. Enforce Encapsulation
      3. Read/Write, read-only, write-only
      4. Perform data validation on incoming and outgoing data values
  12. Indexed Properties
      1. Allow array-like access to groups of items
      2. Must be accessible using an index in similar manner to arrays
  13. Generic Types
      1. Design classes without specifying types at definition stage
      2. Design methods without specifying types for parameters
      3. Use a placeholder at definition stage that will be replaced at instantiation stage
      4. Enable type-safe coding
      5. Increase performance due to reduction in conversions, boxing/ unboxing

1. **Abstract Classes v Interfaces**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Interface** | **Abstract Class** |
| *Multiple inheritance* | A class inherits several interfaces | A class may inherit only one abstract class |
| *Default implementation* | Interface cannot provide code, only signature | Abstract Class can provide complete default code, details to be overridden |
| *Access Modifiers* | Cannot have access modifiers for the methods, everything is assumed public | Can contain access modifiers for its methods |
| *Core vs Peripheral* | Interfaces are used to define the peripheral abilities of a class. | Abstract class defines the core identity of a class and used for object of the same type |
| *Homogeneity* | If various implementation share method signatures it is better to use interfaces | If various implementation are of the same kind and use, common behaviour or status then abstract class is better |
| *Speed* | Requires more time to find the actual method in corresponding classes | Fast |
| *Adding Functionality* | If we add a new method to an Interface then we have to track down all implementations of the interface and define implementation for the new method | If we add a new method to an abstract class then we have the option of providing default implementation and therefore all existing code might work properly |
| *Fields And Constants* | No fields can be defined | Fields and constants can be defined |

1. **Using Types**
   1. Conversion Basics
      1. Implicit conversion does not use a cast operator
      2. Explicit conversion uses a cast operator
      3. Widening conversions have optional casting and magnitude is maintained while precision is may be.
      4. Narrowing conversions do not always perform and a cast conversion is required.
      5. Integer operations that result in overflow or underflow are ignored unless you use a checked block or advanced builds
   2. The is and as Operators
      1. The is operator determines If a variable is compatible with a certain type
      2. The as operator converts an object into a compatible type or null
      3. The as is useful in the case of knowing an objects type.
   3. Parsing
      1. The parse method reads text in a value. Protect parse methods with try-catch blocks
      2. TryParse attempts to parse text and see if there is an error. It returns true if successful and false otherwise
      3. System.Globalization.NumberStyles enumeration allows parse and tryparse to understand special symbols such as decimal points, currency symbols and thousand separators.
      4. Some useful NumberStyles include Integer, HexNumber, number, Float, Currency and Any
   4. Specialized Conversions
      1. System.convert class provides methods that convert from one data type to another
      2. System.Convert includes ToBoolean, ToDouble, toSingle, ToByte, ToInt16, ToString, ToChar, ToInt32, ToUInt32, ToUInt16, ToDateTime, ToInt64, ToUInt32, ToDecimal, ToSByte, ToUInt64.
      3. System.BitConverter converts data to and from arrays of bytes
      4. Boxing occurs when you convert a value type into a reference type, to be avoided if possible
      5. Unboxing occurs when you convert a reference type back into a value type
      6. Dynamic type is a static type, but its value isn’t evaluated until run time.
   5. Strings
      1. Strings are immutable
      2. Internal pool holds an instance of every unique string
      3. Stringbuilders are mutable and can be more efficient for long series of concatenations.
      4. StringWriter and StringReader classes provide methods for writing and reading characters and lines with an underlying Stringbuilder object.
   6. Formatting
      1. The ToString and string.format methods convert values to strings.
      2. String.format uses composite format strings that specify argument numbers, field widths, alignments, format strings, indexes start at 0.
      3. Standard format strings are locale-aware so you should use them whenever possible
      4. Useful standard numeric formatting strings include

* C/c – Currency
* D/d – Decimal
* E/e – Exponential
* F/f – fixed Point
* G/g – Short of E or F
* N/n – number with commas
* P/p – Percent
* X/x – Hexadecimal
  + 1. Useful standard Datetime formats include
* D/d – Long/ short date
* F/f – full
* G – general
* M/m – month/day
* T/t – long/short time
* Y/y – year/month

1. **Key Terms continued**

*Boxing v Unboxing*

Boxing is the process of converting a value type to a reference type. Generally a value will be converted to an Object type and placed on the heap. This will have to be unboxed

*Intern Pool*

The CLR (Common Language Runtime) maintains a table called intern pool which contains a single reference to every unique string used by the program.

*Widening Conversion*

Type conversion where the destination data type can hold any value of the source although with some loss of data e.g. int to long.

*Narrowing Conversion*

Type conversion where the destination type cannot hold every value provided by the source value type e.g. long to int.

*Enumerations*

A complete ordered listing of all items in a collection. The elements of a set. The underlying type of each element is int. Client code is specified which values are valid for a variable.

*Boilerplate code*

Boilerplate code is a term for repetitive code that is subjective for getting some result which ought to be quite simpler.

*Syntactic sugar*

This term describes the syntax in a programming language which is designed to make things easier to read or to express. It is attributed to code that is concise and human readable.

*Deep Clone*

Copy of an object where reference fields refer to new instances of objects not to the same objects referred from the original object

*Shallow Clone*

A copy of an object that is directly referring to the object

*Finalization*

The Garbage collection process calling the process of destructors

*Nondeterministic Finalization*

The programmer does not know when a destructor is being called.

1. **Creating and Implementing Class Hierarchies**

Inheritance

* C Sharp does not support multiple inheritance
* The base keyword provides constructors to invoke a parent class method
* The this keyword provides constructors to invoke from the same class
* A constructor can invoke at most one base class constructor or same class constructor#
* If a parent class has constructors, a child class constructor must invoke them directly or indirectly

Interfaces

* Convention dictates that interface names must being with I i.e. IComparable
* A class can inherit from at most one parent class but can implement any number of interfaces
* If a class implements an interface explicitly then the code cannot use an object reference to access interface members. Use an interface instance.
* If a class implements an interface implicitly the code can use a class instance or interface instance to access the interface members
* IComparable interface provides CompareTo which determines object order
* IEquatable provides Equals method that determines whether an object is equal to another object
* IClonable procdies a Clone method which returns a copy of an object
* IEnumerable provides a GetEnumerator method that returns an IEnumerator object with MoveNext and Reset for list parsing
* A method can use yield return to add objects to an IEnumerator result

Destructors

* Destructors can only be defined in classes
* Classes can only have one destructor
* Destructors cannot be inherited or overloaded or called directly or have modifiers and parameters
* Destructors are converted into an override version of the Finalize method. You cannot override Finalize or call it directly.

Resource Management

* If a class contains no managed resources or unmanaged resources, it doesn’t need to implement IDisposable or have a destructor.
* If a class has only managed sources, it needs to implement IDisposable but it doesn’t need a destructor.
* If a class has only unmanaged resources it should implement IDisposable and needs a destructor aswell
* The dispose method should be safe to run more than once and free managed/unmanaged resources.

1. **Working with Delegates, Events and Exceptions**

***Delegates***

* Delegates are types that represent a method. It defines the method parameters and return type
* The delegate type name ends with Delegate or Callback
* + and – are usable for combining delegating variables
* Covariance lets a method return a value from some subclass of result expected by a delegate
* Contravariance lets a method take parameters from a superclass expected by a delegate
* The .NET framework defines two built-in delegate types, Action and Func.
* An anonymous method is a method with no name.
* Lambda expressions use concise syntax to create these anonymous methods.
* Expression lambda evaluates a single expression whose value is returned by the anonymous method
* A statement lambda executes a series of functions. It must use return keyword to produce a value
* An async lambda includes the async keyword

***Events***

* Events have publishers and subscribers. Some event may have many subscribers or none at all.
* The delegate type defines an event.
* The best practice is to make the events first parameter a sender object and second object that gives more information about the event. Derive the type of the object ifrom the EventArgs class and end its name with Args as in OverdrawnEventArgs
* Classes cannot inherit events. To make it possible for derived classes to raise base class events, give the base class an OnEventNme method which raises the Event

***Exception Handling***

* Error checking is the process of proactively anticipating errors and looking for them. Exception handling protects a programs from unexpected errors. Error checking is usually more efficient than exception handling.
* Try-catch-finally blocks must have at least one catch section or a finally section
* The finally section always executes no matter the content of the try block
* A using statement is equivalent to a try-catch-finally block with a finally section that disposes of the object.
* If an exception is noth handled, control moves up the call stack until a catch block handles it or the program crashes
* Integer operations do not throw OverflowExceptions. Use a checked block to make overflows throw this exception
* Floating point operations do not cause overflow. Instead resulting in PositiveInfinity, NegativeInfinity or NaN
* Use Boolean methods like IsInfinity to determine whether a result is one of these values.
* To rethrow the current exception, use the throw statement without passing it an exception object.
* To create a custom exception drive it from the System.Exception class and end its name with Exception. Then make it serializable and give it constructors that math the defined Exception class
* You can use Debug.Assert to throw an exception in a debug build to find suspicious data.

***Key Terms***

*Checked* Use for throwing exceptions related to OverFlowExceptions

*Contravariance* Enables a method to take parameters from a superclass of the type expected by a delegate.

*Covariance* Enables a method to return a value from a subclass of a result expected by a delegate

*Publisher An object which raises an event*

*Subscriber An object that receives some event*

1. **Multithreading and Asynchronous Processing**

Threads

* Create a thread object by sending to the constructor a delegate that wll be the threads main method.
* Start the thread by calling explicitly the start method within Thread object

Tasks

* Create a task object by sending to the constructor a delegate that will be the tasks main method.
* Call a Task.Run function with parameter as a delegate that will be the mina method.
* Call one of the Task.Factory.StartNew methods with delegate as parameter that will be the tasks main method.

Locks

* Use the lock statement in C# and Monitor.Enter to acquire or Monitoer.Exit to release.

Cancellations

* Use CancellationTokenSource objects to control cancellable operations
* Call Cancel on the token source to start the operation.
* Inside the cancellable operation stop what you were doing to cancel operation.

Await/async

* Two keywords that are new
* Async keyword makes a method asynchronous and tells the compiler to have an await instruction in the body.If there is no await keyword in the body compiler will throw a warning

Background Worker and Task continuation

* Wire a method to be executed usin event DoWork
* Start the operation by calling RunWorkerAsync
* To find when an operation is completed, subscribe to RunWorkerCompleted event
* To get info on progress, subscribe to ProgressChanged event.

***Key Terms***

*asynchrony* Operations that run in a non-blocking fashion

*Asynchronous Pattern Model* When using this pattern, a method splits into two parts a Begin and End part.

*Atomic operation* An operation that will run without interruption from a scheduler

*deadlock* Occurs when a thread tries to acquire a lock on a resource that is already locked by another. Both thread do no make any progress in such an instance

*Event-based Async Pattern* Provides events and delegates to signal when a method is finished or fails.

*Fork-Join Pattern* The process of spawning a thread from the main thread while the current thread continues to do work.

*Multithreading* The capability of an operating system, or hardware platform to have several threads of execution simultaneously

*Mutual Exclusion* A problem where two threads cant be in the same critical section at the same time

*Race condition* Occurs where two or more thread access the same data and write at the same time. If it is read-only then its fine but if write occurs then there may be overwritten data

*Task Parallel library* A .NET library created by Microsoft that tries to abstract away and simplify code that deals with threads

*Task based Async Pattern* A pattern based on a single method that returns Task or Task<Result> object that represents asynchronous WIP

*Thread pool* Represents a pool of threads to queue work items and run asynchronous operations

1. **Creating and Using Types with Reflection, Custom Attributes, the CodeDOM and Lambda Expressions**

***Reflection***

* Two ways to get reference for a Type object using typeof() or .GetType() method of an object
* Assembly.Load methods loafs a class to examine types within an EXE
* Assembly.ReflectionOnlyLoad method loads assembly into memory but without code launching abilities.
* Assembly.CreateInstance method creates an instance of a type
* System.Type represents a class, interface, array, value type, enumeration, parameter, generic type definitions and open/closed generic types.
* Type.GetProperty method returns a PropertyInfo object and enables you to set a propertys values.

***Attributes***

* Attributes enable creation of metadata for a class, property or method
* Attributes are contained in square brackets [] just above target
* Custom attributes must inherit from System.Attribute class.

***CodeDOM***

* CodeDOM is a set of classes that enables creation of code generators
* System.CodeDom.CodeCompileUnit is top-level class; container for all other object within class you want to generate
* CodeDOM provider class generates the class file in either C#, VB or Jscript

***Lambda Expressions***

* Shorthand syntax for anonymous functions
* Delegates are lambda expressions
* Covariance enables methods with more derived return type than delegates return type
* Contravariance permits parameter types less derived than delegate type
* The => in lambda expression is ‘’goes to’’.

***Key terms***

*Assembly* A compiled piece of code in DLL or EXE file

*Code Document Object Model* Enables the developer to generate code in multiple languages at run time based on a single code set

*Expression lambda* Lambda expression with only one statement for the body

*Expression* *tree* Tree-like structure where each node is an expression

*Lambda expression* Shorthand syntax for anonymous method associated with a delegate or expressions tree

*Probing* Process of looking into Global Assembly Cache, the host assembly store, the folder of an executed assembly or a private bin folder to find an assembly

*Statement lambda* Lamdda expression with one or more statement in body of expression

1. **EV Terms**

**Puppet**

Puppet is a software configuration management tool. It includes a declarative language to describe its configuration. A user describes the system resources and their state, either with Puppets language or RubyDSL. Info is stored in Puppet manifests. Puppet Server is installed on one or more servers and Puppet Agent is installed on the machines that the user wants to manage. Agents communicate with the server and fetch config instructions. The Agent then applies the config on the system and sends a status report to the server. Devices can run Puppet Agent as a daemon which is a program that runs as a background process not under user control.

**Terraform**

Terraform is an infrastructure as code (process of managing and provisioning computer data centers through machine-readable definition files rather than physical hardware config tools). It allows users to define an infrastructure in high-level config language. It can create a plan to build examples like IBM Cloud, AWS, Azure and Google Cloud. Infrastructure is defined in HCL Terraform or JSON format.

**Selenium**

Selenium is a portable testing framework for web applications. It provides a playback tool for authoring tests without the need to learn a test scripting language (Selenium IDE). It provides a test domain-specific language (Selenese) to write tests in a number of programming languages including C#, groovy, Java, PHP, Python and Ruby.

**Fitnesse**

Fitnesse is a web server, wiki and automated testing tool for software. It is designed for support of acceptance testing rather than unit testing in facilitating readable descriptions of system function. It supports an agile style of black-box testing with acceptance and regression.

Based around the examination of outputs based on inputted data. They are expressed in a decision table. Fitnesse supports these in its interface with creating tables easily, translating tables into calls to the system under test and allowing for easy documentation. Fitnesse is developed in Java. Its package includes a wiki, web server, testing engine and resources for documentation.

**Validation**

Validation is an automated computer check to ensure the data entered is sensible and reasonable. It doesn’t check data accuracy. Process for confirming values being entered into data conforms rules and constraints of the objects schema, the back-end data stores and database schema as well as bussiness/application rules.

**SSO**

Stands for Single sign-on which permits users to ues one set of login credentials to access multiple different applications.

**JSON**



Javascript Object Notation is a syntax for storing and exchanging data. It is written in text and can be converted to and from an object through Javascript. It uses human-readable text to transmit objects consisting of value-pairs and array data types. It is used for asynchronous browser-server communication, including XML translations. Many languages can parse JSON data.

1. **Working With Data**

***Arrays and Collections***

* Arrays inherit from System.Array type
* There are numerous collection types similar to arrays, but they offer more flexibility for manipulating data in the collection.
* ArrayList,Hashtable,Queue,SortedList,Stack all in System.Collections namespace
* Dictionary<Tkey, TValue>, List<T>, Queue<T>, SortedList<Tkey,TValue> and Stack<T> are all in the Systems.Collections.Generic namespace
* Generic collections classes are used for objects of the same type
* Queues are first in first out
* Stacks are last in first out
* Implement the Icomparable interface to control object comparison
* Dictionary object stores a key,value pair
* Custom collection inherit from CollectionBase class

***ADO.NET***

* A set of classes used to execute commands on a database
* Command object Is used to call a store procedure or execute dynamic SQL statement
* The commands ExecuteNonQuery method is used to execute nonresult- returning queries such as INSERT ot UPDATE
* DBDataReader object is read-only, forward-only cursor connected to database
* DataSet is disconnected resultset and can contain one or more DataTables.
* DataAdapter is used to fill DataSet
* DataAdapter can be used with DataSet to add, update or delete records in a database

***ADO.NET Entity Framework***

* Entity Framework is ORM tool that masks the syntax for using ADO.NET to communicate with a database
* An EFM contains classes that represent objects in a database
* Stored procedures are methods on an EFM

***WCF Data Services***

* WCF Data Services enables access to database over web or intranet
* WCF uses Odata protocol
* WCF Data Services returns data in Odata ATOM format but alco can return data in JSON format
* You can query data in a database by passing parameters in the URL query string

***File I/O***

* File and FileInfo objects determine properties about a file and also perform operations on a file
* A Stream is used to represent the file contents in memory and be used read/write data in a file
* BinaryReader and BinaryWriter work with binary values
* StreamReader and StreamWriter are used for read/write characters by using an encoded value to convert characters to and from bytes
* Default encoding is UTF-8
* Use a StreamReader to read by line or in entirety
* StringReader and StringWriter is used to read and write String data
* Async and await keywords are used to perform asynchronous operations
* They async keyword modifies a method signature for it to use await keyword
* The await command kicks off the method but returns processing back to the calling method.

***Serialization***

* Serialization is the process of transforming an objects data to persisted storage or to transfer the object from one domain to another
* BinaryFormatter is used to perform binary serialization
* XMLSerializer is used to perform XML serialization
* DataContractJsonSerializer is used to perform JSON serialization
* There are two ways to customize serialization by Iserializable & use attributes

**Key Terms**

*ADO.NET* A set of classes in .NET framework that enables you to connect to database, retrieve data, execute stored procedures, add, update, or delete records in a table

*ADO.NET Entity Framework* Object relational mapping tool that provides a GUI that generates to code and perform operations against database using ADO.NET

*Collection* Generic term that encompasses lists, dictionaries, queues, stacks, hash tables and other objects that contains sets of data.

*Connection object* An object in ADO.NET that allows you to open and execute commands against a database

*Object Relational Mapping* Term for tools that convert data between type systems using an OO programming language

*Odata ATOM* The XML representation of data returned from Odata query

*Open Data Protocol* A web protocol for querying, updating data through Internet

*Shallow copy* Creating a new copy of an object that copies all value types and copies object reference types

*Text Transformation Template Toolkit* A file that contains text blocks and control statements tha enable code generation in a file

*WCF Data Services* Enables you to use Odata to expose and consume data over the web and intranet

1. **Working with Language Integrated Query**

***Language Integrated Query***

* Any object that implements Ienumerable<T> or Iqueryable<T> interface can be queries using LINQ
* The result of aLINQ query are normally returned to a variable of type var, which is an implicit typed variable

***Query Expression***

* A query expression contains a fom clause and can contain a select, groupby, orderby, where and join clause.
* Joins are always equivalence based for LINQ queries.
* The query execution does not occur until a result is enumerated. You can force query execution by using an aggregate function.
* The code in where clause use the and operator
* The orderby clause is used in query expressions to sort results on one or more properties.
* You create a new type on the fly in the select clause of an expression which is known as projection
* You use the keyword equals in a join clause
* To create an outer join, include an into clause in your join, also call DefaultIfEmpty method to set the properties on an object when no match was found between two sequences
* A join cluase contains an anonymous type to create a composite key
* The groupby clause returns an Igrouping<Tkey, TElement> collection.

***Method based Queries***

* Method-based queries and query expressions are interchangeable and produce the same results. Only difference is syntax.
* Method-based query use lambda expressions as parameters to the methods
* Use SelectMany method to flatten two sequences into one sequence similar to how join works
* Use GroupJoin to create outer joins when using method-based queries
* Concatenate two sequences using concat
* Use skip method to skip a specific number of elements in sequence
* Use take method to return a limited number of elements in a sequence
* Use distinct method to return the distinct list of elements from a sequence

***LINQ to XML***

* You can use Xelement class in aLINQ to XML query to return result of a query in XML.

**Key Terms**

*Composite keys* Contains multiple properties for purpose of join

*Deferred execution* Execution of a LINQ query is deferred until the result is enumerated or by calling a function as result

*Goes* *To* *operator* Goes To is lambda expression as ‘=>’

*Implicitly* *typed* *variable* A variable that has its type determined by the expression on the right side of initializataion statement. Use keyword var to declare it

*LINQ*  A set of features that extends query capabilities for C#

*Method based query* A feature of LINQ that uses extension methods on types that implement Ienumerable<T> or Iquerable<T> interface to query data.

*Outer join* Selects all elements from one sequence when joined to another sequence even if there is not a match

*Outer sequence* When using Join function, this refers to the sequence calling Join

*ParamArray* A parameter to a method that enables you to pass unknown number of parameter to the method.

*Predicate* The code executed in a where clause for a query expression

*Projection* Selecting a subset of properties from a type to create an anonymous type

*Query expression* A feature of LINQ that enables to query any type that implements Ienumerable<T> or Iqueryable<T> by using syntax that’s easy to comprehend

1. **Input Validation, Debugging and Instrumentation**

***Input Validation***

* Use TrackBar, ComboBox, ListBox, DateTimePicker, FolderBrowserDialog to avoid validation if possible
* Make frequent validations provide nonintrusive feedback as changing the field background color
* Do not trap the user in a field until its avalue is entered correctly
* Some values may be invalid but be part of a valid value such as -.0
* When the user tries to accept a form, validate all fields, Refuse to accept the form if there are invalid calues. Warn the user incase of unusual values.

***Validating Data – Built in Validation Functions***

* Use string length to check for missing values.
* Initialize a ComboBox or ListBox so that it always had a valid selection
* Use TryParse to validate data types such as int or decimal.
* String methods that can help with validation include Contains, EndsWith, IndexOf, IndexOfAny, IsNullOrEmpty, IsNullOrWhitespace, LastIndexOf, LastIndexOfAny, Remove, Replace, Split, StartsWith, Substring, ToLower, ToUpper, Trim, TrimEnd, and TrimStart.

***Validating Data – Regular Expressions***

* Useful regex methods are IsMatch, Match, Matches, Replace, Split.
* Use string literals to make it easier to use regular expressions that contain escape characters.
* Use sanity checks to look for unusual values

Regex components table

|  |  |
| --- | --- |
| Item | Purpose |
| \ | Special symbol followed by characters |
| ^ | Beginning of string |
| $ | Matches the end of string |
| \A | Beginning of string |
| \Z | End of string |
| \* | Preceding 0 or more times |
| + | Preceding 1 or more times |
| ? | Preceding 0 or 1 times |
| . | Match any character |
| [abc] | Any characters inside the brackets |
| [^abc] | Any characters not inside the bracket |
| [a-z] | Any characters in the range |
| [^a-z] | Any characters not in the range |
| X|Y | Matches X or Y |
| (pattern) | Matches a numbered match group |
| (?<name>expr) | Match named match group |
| \2 | Refers to previously defined value |
| \k<name> | Refers to defined group named name |
| {n} | Matches n occurences |
| {n,} | Matches n or more occurences |
| {n,m} | Matches between n and m occurences |
| \b | Word boundary |
| \B | Nonword boundary |
| \d | Digit |
| \D | Nondigit |
| \f | Form-feed |
| \n | Newline |
| \r | Carriage return |
| \s | Whitespace |
| \S | Non white space |
| \t | tab |
| \v | Vertical tab |
| \w | Word character |
| \W | Non word character |

***Managing Data Integrity***

* After you validate user input, the code must still protect data as it is processed
* Use Debug.Assert statements to validate data as it moves through the program

***Debugging***

* Use #define and #undef directives to define and undefine preprocessor symbols
* Use the #if,#elif,#else,#endif directives to determine what code is included in the program dependind on which preprocessor symbols are defined
* Use #warning and #error to add warnings and errors to Error List
* Use #line to change a line number and name of the file
* Use #region and #endregion to make collapsible code regions
* Use #pragma warning dissable number and #pragma warning restore to disable and restore warnings
* The DEBUG and TRACE compiler constants are predefined. Normally, DEBUG is defined in debug builds, and TRACE is defined in debug and release builds
* Useful Debug and Trace methods include Assert, Fail, Fluch, Indent, Unindent, Write, WriteIf, WriteLine and WriteLineIf
* You can add listeners to Debug and Trace objects. Standard listeneres write messages to the Output window, event logs and text files.

***Program Database Files***

* You need a PDB file to debug a compiled executable

***Instrumenting applications***

* Logging means recording key events.
* Gather information about a program to study characteristics such as speed and memory usage. Methods for profiling include using a profiler and using performance counters.

***Key Terms***

*Assertion* Makes claim about data and throws an exception if that claim is false. Use System.Diagnostics.Debug.Assert to make assertions

*Character Class* Regular expression construction that represents a set of characters

*Conditional Compilation Constant* Predefined symbol to use with directives

*Data Validation* Code that verifies a data value such as string value

*Escape Sequence* Sequence of characters that have special meaning

*Inline Options* Options set in a regex by using syntax

*Sanity Check* A test on data to see if data makes sense.

1. **Using Encryption and Managing Assemblies**

***Choosing an Encryption algorithm***

* If you need to encrypt data that is local, or secure a way to distribute the encryption key use symmetric encryption.
* If there is no way to send secure encryption key between parties, symmetric encryption is recommended.
* To ensure data intergrity, use the hashing algorithm
* To ensure both integrity and authenticity, choose a MAC algorithm

***Symmetric Encryption***

* A shared secret or common key
* Reads an initialization vector that is not secret but encrypts the first block of data
* Instantiating an algorithm object and then calling a Encryptor or Decryptor
* The encryptor/decryptor is used with either by calling the TransformFinalBlock or by sending it to CryptoStream

***Asymmetric Encryption***

* Based on a pair of complementary keys. Encryptd data with one key can be decrypted with the other key
* One key is secret and called the private key, the other one is made available to anyone who wants to encrypt data or verify encrypted data and is called public key

***Hashing***

* Mapping binary data of a variable length to a fixed size binary data called hash
* Calculate the cryptographic hash and send it together with data to be verified to ensure that data is not modified while being transferred
* Two common algorithms are SHA256 and SHA512 with hashes os 256 and 512 bite respectively

***Key Management***

* Symmetric keys can be exchanged using asymmetric algorithms
* Asymmetric private keys can be secured either by using certificates or by using Crypto Service Providers containers

***Assembly Version***

* An assembly version is specified by four parts: Major, Minor, Build, Revision.

***Strong name***

* An assembly that is digitally signed is called a strongly named assembly
* A strong name has five parts: Friendly Name, Version, Culture, Public Key Token and Processor Architecture

***GAC (Global Assembly Cache)***

* A repository for .NET assemblies
* Only strongly named assemblies are deployed on GAC
* Several version of same assembly are available at same time.

***Key Terms***

*Assembly* A unit of reuse, deployment, versioning and security

*Certificate Authority* An entity which issues digital certificates

*Certificate Revocation List* A list of certificates that have been revoked

*Hash bucket* Data structure that holds items with same hash value

*Intermediate Language* Result of compiling a .NET application from source code

*Just-In-Time compiler* A component of .NET that transforms IL into binary code that can be run on target platform

*Message authentication code* Family of cryptographic algorithms used to provide integrity and authenticity

*Public key Infrastructure* Infrastructure which handles digital certificates

*Secured Hash Algorithm* Family of cryptographic algorithms used to calculate hashes published by NIST

*Secure Socket Layer*  Cryptographic protocol used for secure communications

*Transport Layer Security* Cryptographic protocol used for secure communications