Advance C#

1. Types of Classes

Classes are user-defined data types that represent the **state** and **behavior** of an object. The state represents the properties, and **behavior** is the action that objects can perform.

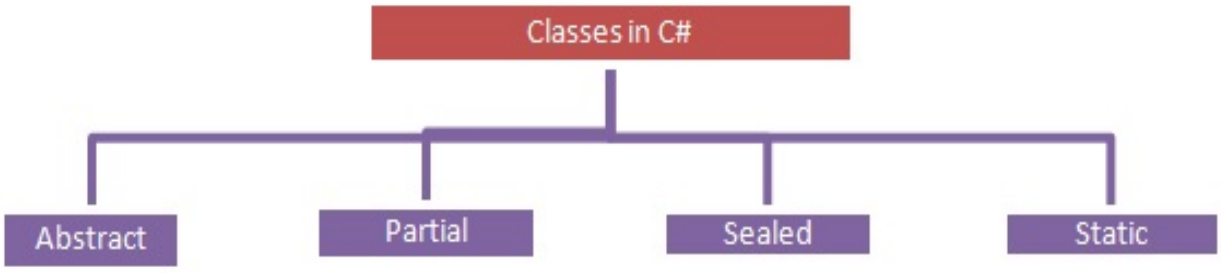
Classes can be declared using the following access specifiers that limit the accessibility of classes to other classes. However, some classes do not require any access modifiers.

1. Public
2. Private
3. Protected
4. Internal
5. Protected internal

Some Key points about classes:

- Classes are reference types that hold the object created dynamically in a heap.
- All classes have a base type of **System.Object**.
- The default access modifier of a class is **Internal**.
- The default access modifier of methods and variables is **Private**.
- Directly inside the namespaces, declarations of private classes are not allowed.

Types of classes in C#



a. Abstract Class

An abstract class is a class that provides a common definition to the subclasses, and this is the type of class whose object is not created.

Some key points of Abstract classes are:

- Abstract classes are declared using the abstract keyword.
- We cannot create an object of an abstract class.
- It must be inherited in a subclass if you want to use it.
- An Abstract class contains both abstract and non-abstract methods.
- The methods inside the abstract class can either have an or no implementation.
- We can inherit two abstract classes; in this case, implementation of the base class method is optional.
- An Abstract class has only one subclass.
- Methods inside the abstract class cannot be private.
- If there is at least one method abstract in a class, then the class must be abstract.

Contents

An abstract class may contain the following,

- Non-Abstract Method
- Abstract method
- Non Abstract Property
- Abstract Property
- Constructor
- Destructor

b. Partial Class

It is a type of class that allows dividing their properties, methods, and events into multiple source files, and at compile time, these files are combined into a single class.

The following are some key points:

- All the parts of the partial class must be prefixed with the partial keyword.
- If you seal a specific part of a partial class, the entire class is sealed, the same as for an abstract class.
- Inheritance cannot be applied to partial classes.
- The classes written in two class files are combined at run time.

Advantages of a partial class

- You can separate UI design code and business logic code so that it is easy to read and understand.
- When working with an automatically generated source, the code can be added to the class without having to recreate the source file.
- More than one developer can simultaneously work the code for the same class.
- You can maintain your application better by compacting large classes. Suppose you have a class that has multiple interfaces so you can create multiple source files depending on interface implementation. It is easy to understand and maintain an interface implemented on which the source file has a partial class.

Points that you should be careful about partial classes

1. You need to use a partial keyword in each part of the partial class.
2. The name of each part of the partial class should be the same but the source file name for each part of the partial class can be different.
3. All parts of a partial class should be in the same namespace.
4. Each part of a partial class should be in the same assembly or DLL, in other words, you can't create a partial class in source files of a different class library project.
5. Each part of a partial class has the same accessibility.
6. If you inherit a class or interface on a partial class then it is inherited on all parts of a partial class.
7. If a part of a partial class is sealed then the entire class will be sealed.
8. If a part of the partial class is abstract then the entire class will be an abstract class.

c. Sealed Class

Sealed classes are used to restrict the inheritance feature of object-oriented programming. Once a class is defined as a **sealed class,** this class cannot be inherited.

In C#, the sealed modifier is used to declare a class as **sealed**. In Visual Basic .NET, the **NotInheritable** keyword serves the purpose of being sealed. The compiler throws an error if a class is derived from a sealed class.

The following are some key points:

- A Sealed class is created using the sealed keyword.
- Access modifiers are not applied to a sealed class.
- To access the sealed members, we must create an object of the class.

Why Sealed Classes?

A sealed class's main purpose is to remove the inheritance feature from the class users so they cannot derive a class from it. One of the best uses of sealed classes is when you have a class with static members.

d. Static Class

It is the type of class that cannot be instantiated. In other words, we cannot create an object of that class using the new keyword, such that class members can be called directly using their name.

The following are some key points:

- It was created using the static keyword.
- Only static members are allowed; in other words, everything inside the class must be static.
- We cannot create an object of the static class.
- A Static class cannot be inherited.
- It allows only a static constructor to be declared.
- The static class methods can be called using the class name without creating the instance.

Advantages of Static Classes

1. You will get an error if you declare any member as a non-static member.
2. When you try to create an instance to the static class, it again generates a compile time error because the static members can be accessed directly with their class name.
3. The static keyword is used before the class keyword in a class definition to declare a static class.
4. Static class members are accessed by the class name followed by the member name.

Static Constructors

A static constructor is used to initialize the static data members, whereas the normal constructor (non-static constructor) is used to initialize the non-static data members.

Rules

1. Static constructors can‘t contain any access modifiers.
2. Static constructors can‘t be defined with arguments.
3. Static constructors can‘t access the non-static data members.

2. Generics

Generics are classes, structures, interfaces, and methods that have placeholders (type parameters) for one or more of the types that they store or use. A generic collection class might use a type parameter as a placeholder for the type of objects that it stores. The type parameters appear as the types of its fields and the parameter types of its methods. A generic method might use its type parameter as the type of its return value or as the type of one of its formal parameters.

When you create an instance of a generic class, you specify the actual types to substitute for the type parameters. This establishes a new generic class, referred to as a constructed generic class, with your chosen types substituted everywhere that the type parameters appear. The result is a type-safe class that is tailored to your choice of types.

Terminology

- A *generic type definition* is a class, structure, or interface declaration that functions as a template, with placeholders for the types that it can contain or use. For example, the System.Collections.Generic.Dictionary<TKey,TValue> class can contain two types: keys and values. Because a generic type definition is only a template, you cannot create instances of a class, structure, or interface that is a generic type definition.
- Generic *type parameters*, or *type parameters*, are the placeholders in a generic type or method definition. The System.Collections.Generic.Dictionary<TKey,TValue> generic type has two type parameters, TKey and TValue, that represent the types of its keys and values.
- A *constructed generic type*, or *constructed type*, is the result of specifying types for the generic type parameters of a generic type definition.
- A *generic type argument* is any type that is substituted for a generic type parameter.
- The general term *generic type* includes both constructed types and generic type definitions.

Advantages of generics

- Type safety. Generics shift the burden of type safety from you to the compiler.
- Less code and code is more easily reused. There is no need to inherit from a base type and override members.

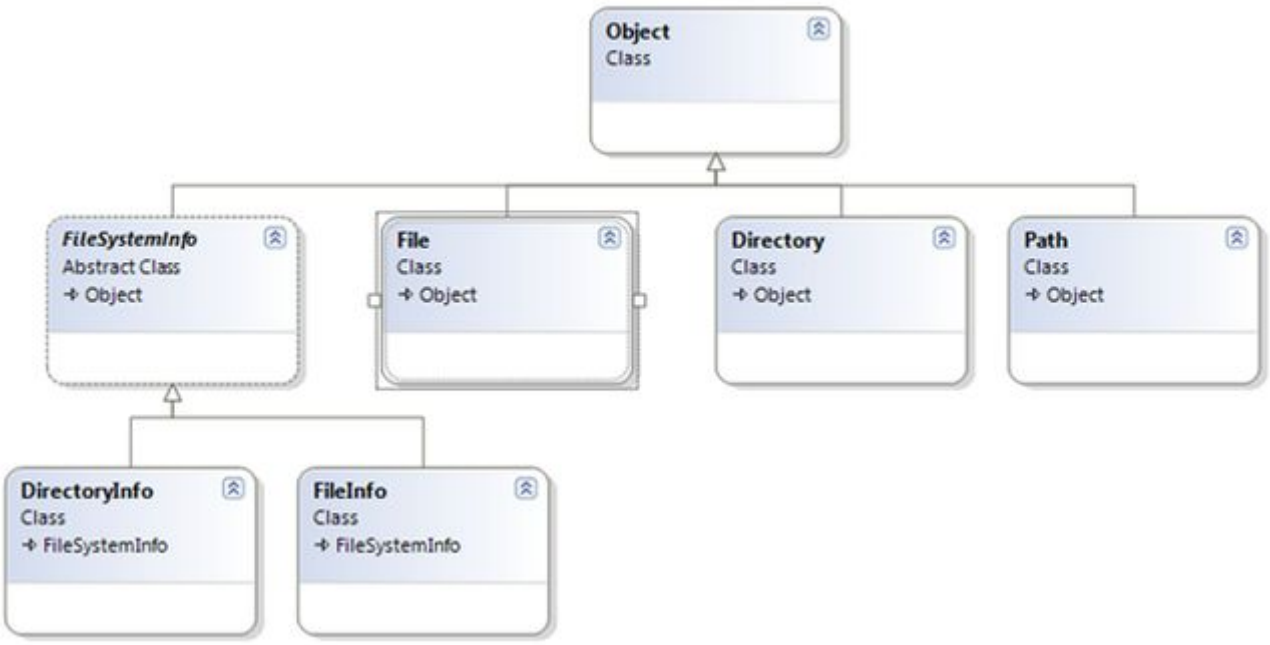
- Better performance. Generic collection types generally perform better for storing and manipulating value types because there is no need to box the value types.
- Generic delegates enable type-safe callbacks without the need to create multiple delegate classes.
- Generics streamline dynamically generated code. When you use generics with dynamically generated code you do not need to generate the type. This increases the number of scenarios in which you can use lightweight dynamic methods instead of generating entire assemblies.

Disadvantages of Generics

- Enumerations cannot have generic type parameters. An enumeration can be generic only incidentally.
- Lightweight dynamic methods cannot be generic.
- In Visual Basic, C#, and C++, a nested type that is enclosed in a generic type cannot be instantiated unless types have been assigned to the type parameters of all enclosing types. Another way of saying this is that in reflection, a nested type that is defined using these languages includes the type parameters of all its enclosing types. This allows the type parameters of enclosing types to be used in the member definitions of a nested type.

3. File system in Depth

- The System.IO namespace provides four classes that allow you to manipulate individual files, as well as interact with a machine directory structure.
- The Directory and File directly extends System.Object and supports the creation, copying, moving and deletion of files using various static methods. They only contain static methods and are never instantiated.
- The FileInfo and DirectoryInfo types are derived from the abstract class FileSystemInfo type and they are typically, employed for obtaining the full details of a file or directory because their members tend to return strongly typed objects. They implement roughly the same public methods as a Directory and a File but they are stateful and the members of these classes are not static.



Here are some commonly used file and directory classes:

- File - provides static methods for creating, copying, deleting, moving, and opening files, and helps create a FileStream object.
- FileInfo - provides instance methods for creating, copying, deleting, moving, and opening files, and helps create a FileStream object.
- Directory - provides static methods for creating, moving, and enumerating through directories and subdirectories.

- DirectoryInfo - provides instance methods for creating, moving, and enumerating through directories and subdirectories.
- Path - provides methods and properties for processing directory strings in a cross-platform manner.

In the .NET framework, the System.IO namespace is the region of the base class libraries devoted to file based input and output services. Like any namespace, the System.IO namespace defines a set of classes, interfaces, enumerations, structures and delegates. The following table outlines the core members of this namespace,

Class Types	Description
Directory/ DirectoryInfo	These classes support the manipulation of the system directory structure.
DriveInfo	<p" style="outline: 0px;">This class provides detailed information regarding the drives that a given machine has. </p">
FileStream	This gets you random file access with data represented as a stream of bytes.
File/FileInfo	These sets of classes manipulate a computer's files.
Path	It performs operations on System.String types that contain file or directory path information in a platform-neutral manner.
BinaryReader/ BinaryWriter	These classes allow you to store and retrieve primitive data types as binary values.
StreamReader/StreamWriter	Used to store textual information to a file.
StringReader/StringWriter	These classes also work with textual information. However, the underlying storage is a string buffer rather than a physical file.
BufferedStream	This class provides temp storage for a stream of bytes that you can commit to storage at a later time.

Stream

The .NET provides many objects such as FileStream, StreamReader/Writer, BinaryReader/Writer to read from and write data to a file. A stream basically represents a chunk of data flowing between a source and a destination. Stream provides a common way to interact with a sequence of bytes regardless of what kind of devices

store or display the bytes. The following table provides common stream member functions:

Methods	Description
Read()/ ReadByte()	Read a sequence of bytes from the current stream.
Write()/WriteByte()	Write a sequence of bytes to the current stream.
Seek()	Sets the position in the current stream.
Position()	Determine the current position in the current stream.
Length()	Return the length of the stream in bytes.
Flush()	Updates the underlying data source with the current state of the buffer and then clears the buffer.
Close()	Closes the current stream and releases any associated stream resources.

FileStream

A FileStream instance is used to read or write data to or from a file. In order to construct a FileStream, first we need a file that we want to access. Second, the mode that indicates how we want to open the file. Third, the access that indicates how we want to access a file. And finally, the share access that specifies whether you want exclusive access to the file.

Enumeration	Values
FileMode	Create, Append, Open, CreateNew, Truncate, OpenOrCreate
FileAccess	Read, Write, ReadWrite
FileShare	Inheritable, Read, None, Write, ReadWrite

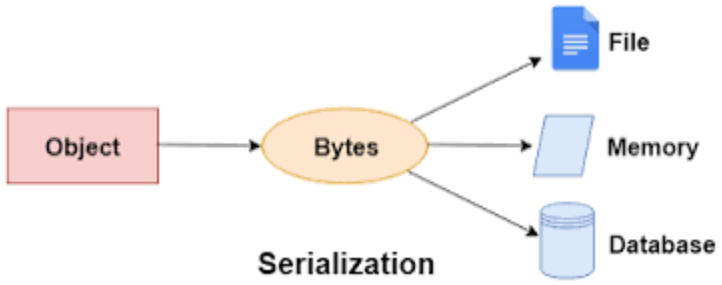
4. Data serialization (JSON,XML)

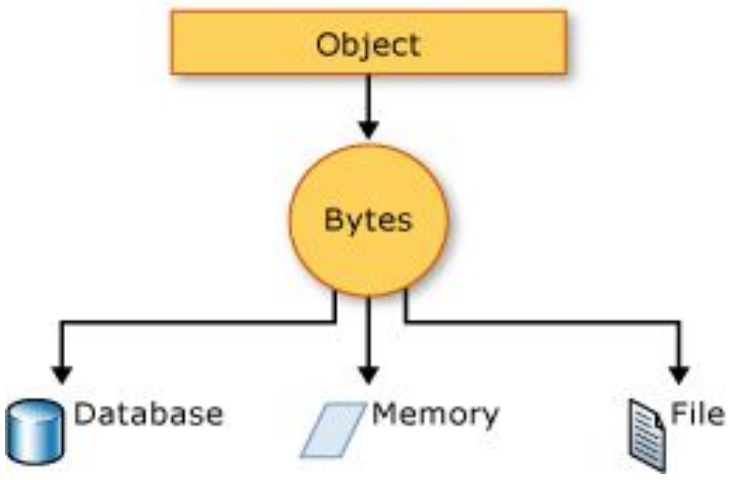
Serialization in C# is the process of converting an object into a stream of bytes to store the object to memory, a database, or a file. Its main purpose is to save the state of an object in order to be able to recreate it when needed. The reverse process is called deserialization.

Serialization is the process of converting the state of an object into a form that can be persisted or transported. The complement of serialization is deserialization, which converts a stream into an object. Together, these processes allow data to be stored and transferred.

- Serialization stores the state of objects, i.e., member variable values, to persistent storage such as a disk.
- Deserialization is the reverse of serialization. It is a process of reading objects from a file where they have been stored.
- Serialization is used to transfer data from one application to another.

How serialization works:





1. JSON serialization and deserialization

The System.Text.Json namespace provides functionality for serializing to and deserializing from JavaScript Object Notation (JSON). *Serialization* is the process of converting the state of an object, that is, the values of its properties, into a form that can be stored or transmitted. The serialized form doesn't include any information about an object's associated methods. *Deserialization* reconstructs an object from the serialized form.

2. XML serialization

XML serialization converts (serializes) the public fields and properties of an object, and the parameters and return values of methods, into an XML stream that conforms to a specific XML Schema definition language (XSD) document. XML serialization results in strongly typed classes with public properties and fields that are converted to a serial format (in this case, XML) for storage or transport.

Because XML is an open standard, the XML stream can be processed by any application, as needed, regardless of platform.

5. Base library features

The base class library contains standard programming features such as Collections, XML, DataType definitions, IO (for reading and writing to files), Reflection and Globalization to name a few. All of which are contained in the System namespace. As well, it contain some non-standard features such as LINQ, ADO.NET (for database interactions), drawing capabilities, forms and web support.

BCL(base class library) is a subset of the Framework class library (FCL). The class library is a collection of reusable types that are closely integrated with CLR. The base Class library provides classes and types that are helpful in performing day-to-day operations .

Features:

- Represent base data types and exceptions.
- Encapsulate data structures.
- Perform I/O.
- Access information about loaded types.
- Invoke .NET security checks.
- Provide data access, rich client-side GUI, and server-controlled, client-side GUI.

Base Class Library Namespace	**Brief Description**
System	Contains the fundamentals for programming such as the data types, console, match and arrays, etc.
System.CodeDom	Supports the creation of code at runtime and the ability to run it.

System.Collections	Contains Lists, stacks, hashtables and dictionaries
System.ComponentModel	Provides licensing, controls and type conversion capabilities
System.Configuration	Used for reading and writing program configuration data
System.Data	Is the namespace for ADO.NET
System.Deployment	Upgrading capabilities via ClickOnce
System.Diagnostics	Provides tracing, logging, performance counters, etc. functionality
System.DirectoryServices	Is the namespace used to access the Active Directory
System.Drawing	Contains the GDI+ functionality for graphics support
System.EnterpriseServices	Used when working with COM+ from .NET
System.Globalization	Supports the localization of custom programs

System.IO	Provides connection to file system and the reading and writing to data streams such as files
System.Linq	Interface to LINQ providers and the execution of LINQ queries
System.Linq.Expressions	Namespace which contains delegates and lambda expressions
System.Management	Provides access to system information such as CPU utilization, storage space, etc.
System.Media	Contains methods to play sounds
System.Messaging	Used when message queues are required within an application, superseded by WCF
System.Net	Provides access to network protocols such as SSL, HTTP, SMTP and FTP
System.Reflection	Ability to read, create and invoke class information.
System.Resources	Used when localizing a program in relation to language support on web or form controls
System.Runtime	Contains functionality which allows the management of runtime behavior.

System.Security	Provides hashing and the ability to create custom security systems using policies and permissions.
System.ServiceProcess	Used when a windows service is required
System.Text	Provides the StringBuilder class, plus regular expression capabilities
System.Threading	Contains methods to manage the creation, synchronization and pooling of program threads
System.Timers	Provides the ability to raise events or take an action within a given timer period.
System.Transactions	Contains methods for the management of transactions
System.Web	Namespace for ASP.NET capabilities such as Web Services and browser communication.
System.Windows.Forms	Namespace containing the interface into the Windows API for the creation of Windows Forms programs.
System.Xml	Provides the methods for reading, writing, searching and changing XML documents and entities.

6. Lambda expression

Lambda expressions are anonymous functions that contain expressions or sequences of operators.

- All lambda expressions use the lambda operator =>, which can be read as “goes to” or “becomes”.
- The left side of the lambda operator specifies the input parameters, and the right side holds an expression or a code block that works with the entry parameters.
- Usually, lambda expressions are used as predicates or instead of delegates (a type that references a method).

Expression Lambdas

- Parameter => expression
- Parameter-list => expression
- Count => count + 2;
- Sum => sum + 2;
- n => n % 2 == 0

- The lambda operator => divides a lambda expression into two parts. The left side is the input parameter and the right side is the lambda body.
- Use Lambda Expression whenever you feel you can reduce your lines of code. Keep in mind the maintainability while reducing the number of lines of code. People think that Lambda Expressions are awkward looking code, but I'm telling you its worth it to use them in code wisely.

7. Extension methods

Extension methods enable you to "add" methods to existing types without creating a new derived type, recompiling, or otherwise modifying the original type. Extension methods are static methods, but they're called as if they were instance methods on the extended type.

C# extension method is a static method of a static class, where the "this" modifier is applied to the first parameter. The type of the first parameter will be the type that is extended.

Extension methods are only in scope when you explicitly import the namespace into your source code with a using directive.

Benefits of extension methods

- Extension methods allow existing classes to be extended without relying on inheritance or changing the class's source code.
- If the class is sealed, there is no concept of extending its functionality. For this, a new concept is introduced, in other words, extension methods.
- This feature is important for all developers, especially if you would like to use the dynamism of the C# enhancements in your class's design.

Important points for the use of extension methods

- An extension method must be defined in a top-level static class.
- An extension method with the same name and signature as an instance method will not be called.
- Extension methods cannot be used to override existing methods.
- The concept of extension methods cannot be applied to fields, properties, or events.
- Overuse of extension methods is not a good style of programming.

8. LINQ (with DataTable, List, etc.)

- Language-Integrated Query (LINQ) is the name for a set of technologies based on the integration of query capabilities directly into the C# language.
- Traditionally, queries against data are expressed as simple strings without type checking at compile time or IntelliSense support.
- When you write queries, the most visible "language-integrated" part of LINQ is the query expression. Query expressions are written in a declarative *query syntax*.
- By using query syntax, you perform filtering, ordering, and grouping operations on data sources with a minimum of code.

How to enable LINQ querying of your data source

- **In-memory data**

  There are two ways you enable LINQ querying of in-memory data. If the data is of a type that implements IEnumerable<T>, you query the data by using LINQ to Objects. If it doesn't make sense to enable enumeration by implementing the IEnumerable<T> interface, you define LINQ standard query operator methods, either in that type or as extension methods for that type.

- **Remote data**

  The best option for enabling LINQ querying of a remote data source is to implement the IQueryable<T> interface.

9. ORM tool

Object-relational mapping (ORM) is a way to align programming code with database structures. ORM uses metadata descriptors to create a layer between the programming language and a relational database. It thus connects object-oriented program (OOP) code with the database and simplifies the interaction between relational databases and OOP languages.

The idea of ORM is based on abstraction. The ORM mechanism makes it possible to address, access and manipulate objects without having to consider how those objects relate to their data sources. ORM lets programmers maintain a consistent view of objects over time, even as the sources that deliver them, the sinks that receive them and the applications that access them change.

How object-relational mapping works

ORM translates information about states and codes object-oriented programs create that are often difficult to understand. It creates a structured map that explains how objects are related to different tables (data) without knowing how the data is structured. In other words, it creates a logical model of the program with a high level of abstraction, i.e., without specifying the underlying code details.

ORM manages the mapping details between a set of objects and underlying relational databases, XML repositories or other data sources and sinks. It simultaneously hides the often changing details of related interfaces from developers and the code they create. In many cases, ORM changes can incorporate new technology and capabilities without requiring changes to the code for related applications.

Advantages and benefits of object-relational mapping

- ORM hides and encapsulates changes in the data source, so that when data sources or their APIs change, only ORM needs to change to preserve the appropriate associations -- not the applications that use ORM. This capacity makes it easier to maintain applications, lets developers take advantage of new classes as they become available and makes it easy to extend ORM-based applications.
- The program model can be used to connect the application with the SQL code without rewriting the code, saving a lot of time for developers.
- ORM also makes it easier and more cost-effective to maintain the application over time because it automates object-to-table and table-to-object conversion and requires less code compared to embedded SQL and handwritten stored procedures.

Disadvantages of object-relational mapping

- Critics have said that ORM can lead to an erosion in application speed and performance due to the extra code that is generated for abstraction. They believe that using stored procedures is a better way to avoid this problem. In some cases, depending on ORM might result in poorly designed databases and make it harder to maintain applications.

- Incorrect mapping between data tables and objects can also create application problems that may be difficult to recognize and can affect overall performance.

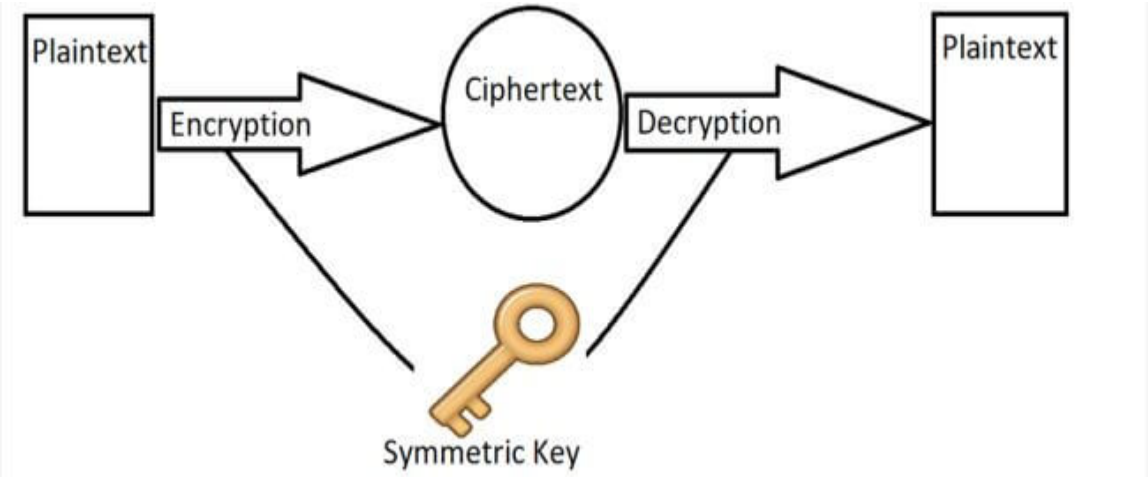
10. Security & Cryptography

Security and cryptography are essential components in modern software development to protect sensitive data, ensure privacy, and prevent unauthorized access. This documentation provides an overview of various security and cryptography techniques, including encryption, hashing, authentication, and authorization.

Encryption

Encryption is the process of converting plaintext into ciphertext to protect data confidentiality. It ensures that only authorized parties can access and understand the encrypted information. There are two main types of encryption:

- **Symmetric Encryption:** In symmetric encryption, the same key is used for both encryption and decryption. Common symmetric encryption algorithms include **AES** (Advanced Encryption Standard) and **DES** (Data Encryption Standard).
- **Asymmetric Encryption:** Asymmetric encryption uses a pair of keys: a public key for encryption and a private key for decryption. Public keys are widely distributed, while private keys are kept secret. Popular asymmetric encryption algorithms include **RSA** and **ECC** (Elliptic Curve Cryptography).



Hashing

Hashing is a one-way cryptographic function that converts input data (plaintext) into a fixed-size string of characters (hash value). Unlike encryption, hashing cannot be reversed to obtain the original input. Common hash functions include:

- MD5 (Message Digest Algorithm 5)
- SHA-1 (Secure Hash Algorithm 1)
- SHA-256, SHA-384, SHA-512: Variants of the SHA algorithm with different output sizes.

Hashing is commonly used for data integrity verification, password hashing, and digital signatures.

Choose an algorithm

- Data privacy:
    - Aes
- Data integrity:
    - HMACSHA256
    - HMACSHA512
- Digital signature:
    - ECDsa
    - RSA
- Key exchange:
    - ECDiffieHellman
    - RSA
- Random number generation:
    - RandomNumberGenerator.GetBytes
    - RandomNumberGenerator.Fill
- Generating a key from a password:
    - Rfc2898DeriveBytes.Pbkdf2

classes and interfaces.

Class or Interface	Description
SymmetricAlgorithm	The top-level abstraction for all symmetric cryptography algorithms.
AsymmetricAlgorithm	The top-level abstraction for all asymmetric cryptography algorithms.
RC2	The RC2 abstraction; a subclass of SymmetricAlgorithm.
DES	The DES algorithm abstraction; a subclass of SymmetricAlgorithm (DES.cs).
TripleDES	The Triple DES algorithm abstraction, derived from SymmetricAlgorithm.
Rijndael	The Rijndeal abstraction, derived from SymmetricAlgorithm.
RSA	The RSA public key cipher abstraction, derived from AsymmetricAlgorithm.
DSA	The DSA public key cipher abstraction, derived from AsymmetricAlgorithm.
ICryptoTransform	The interface definition for encryption and decryption transforms; objects of this type are generated by methods on SymmetricAlgorithm.

- **AES (Advanced Encryption Standard):**
  - AES is a symmetric encryption algorithm widely used for securing sensitive data.
  - It operates on fixed-size blocks of data and supports key sizes of 128, 192, or 256 bits.
  - AES is considered highly secure and efficient, making it suitable for various applications such as data encryption, network security, and secure communication.
- **Digital Signature:**
  - A digital signature is a cryptographic technique used to verify the authenticity and integrity of digital messages or documents.
  - It involves the use of asymmetric cryptography, where the sender signs the message with their private key, and the recipient verifies the signature using the sender's public key.
  - Digital signatures provide non-repudiation, meaning the sender cannot deny sending the message, and ensure that the message has not been tampered with during transmission.

- **RSA (Rivest-Shamir-Adleman):**
  - RSA is a widely used asymmetric encryption algorithm named after its inventors.
  - It is based on the difficulty of factoring large prime numbers and relies on the mathematical properties of modular exponentiation.
  - RSA is commonly used for key exchange, digital signatures, and encryption in various secure communication protocols and applications.
- **Hashing:**
  - Hashing is a process of converting input data (of any size) into a fixed-size string of bytes using a hash function.
  - Hash functions are one-way mathematical algorithms that generate a unique hash value for each unique input.
  - Hashing is used for data integrity verification, password hashing, digital signatures, and cryptographic checksums.
- **Rijndael:**
  - Rijndael is a symmetric encryption algorithm that was selected as the Advanced Encryption Standard (AES) by the U.S. National Institute of Standards and Technology (NIST) in 2001.
  - It supports key sizes of 128, 192, or 256 bits and operates on blocks of data.
  - Rijndael is known for its flexibility, efficiency, and security, making it a popular choice for encrypting sensitive data in various applications.
- **DES (Data Encryption Standard):**
  - DES is a symmetric encryption algorithm developed by IBM in the 1970s and adopted as a federal standard in the United States.
  - It operates on 64-bit blocks of plaintext and uses a 56-bit key for encryption and decryption.
  - Despite being widely used in the past, DES is now considered obsolete due to its relatively short key length, which makes it vulnerable to brute-force attacks.
  - However, DES served as the foundation for more secure encryption algorithms, including Triple DES (3DES), which applies DES encryption three times sequentially with different keys for enhanced security.
  - While DES itself is no longer recommended for new applications, its legacy continues to influence modern encryption standards and techniques.

11. Dynamic type

In C#, the dynamic type allows you to store data of any type, similar to object. However, unlike object, which is resolved at compile time, the dynamic type is resolved at runtime. This means that the type checking and member resolution for dynamic objects occur at runtime rather than compile time.

Using the dynamic type can be helpful in scenarios where the type of an object is not known until runtime, such as when working with dynamic languages, COM interop, or when dealing with data from external sources.

- **Late Binding:** The dynamic type enables late binding, allowing you to defer type checking and member resolution until runtime. This can be useful when interacting with dynamically-typed languages or when working with data from external sources where the types are determined at runtime.
- **Runtime Type Resolution:** With dynamic, type checking and member resolution are deferred until runtime. This means that the compiler does not enforce type safety at compile time, leading to potential runtime errors if the wrong type or member is used.
- **Interoperability:** dynamic is often used in scenarios involving COM interop, dynamic languages like Python or JavaScript, or when working with frameworks that use reflection extensively. It facilitates easier integration with such systems by allowing C# code to adapt to the types and members of external components dynamically.
- **Dynamic Object Creation:** You can create dynamic objects using the ExpandoObject class or by using anonymous types. These dynamic objects can have properties and methods added or modified at runtime, providing flexibility in data manipulation.
- **Dynamic Method Invocation:** Methods can be invoked dynamically on dynamic objects, allowing for method calls to be determined at runtime. This enables scenarios where the method to be called is determined dynamically based on runtime conditions.
- **Iterating Over Dynamic Collections:** The dynamic type is commonly used when working with collections of heterogeneous types, such as JSON or XML data. It allows for iteration over dynamic collections without requiring explicit type conversions.

While the dynamic type offers flexibility, it comes with some trade-offs. Since type checking is deferred until runtime, errors related to type mismatches may not be caught until runtime, potentially leading to runtime exceptions. Additionally, the use of dynamic can make code harder to understand and maintain, as the types of variables and members are not immediately apparent from the code itself.

12. Database with C# (CRUD)

CRUD operations refer to the basic actions performed on data in a database: Create, Read, Update, and Delete. These operations are fundamental to most applications that interact with data storage systems.

Components

- **Model (Data Structure):**
  - Represents the structure of the data being stored.
  - Contains properties that map to database columns.
- **Business Logic Layer (BL):**
  - Contains the business logic that operates on the data.
  - Responsible for performing CRUD operations.
- **Data Access Layer (DAL):**
  - Handles communication with the database.
  - Executes queries and commands to perform CRUD operations.
- **Controller (Optional):**
  - Facilitates communication between the user interface and the business logic layer.
  - Orchestrates the flow of data between the user interface and the BL.

Operations

1. Create (Add)

Description: Adds new data to the database.

Steps:

- Collect input data.
- Validate input data.
- Call the appropriate method in the BL to add the data.
- BL interacts with DAL to insert the data into the database.

2. Read (Retrieve)

Description: Retrieves existing data from the database.

Steps:

- Call the appropriate method in the BL to retrieve data.
- BL interacts with DAL to fetch data from the database.
- Return the data to the caller.

3. Update (Modify)

Description: Modifies existing data in the database.

Steps:

- Collect input data (including the unique identifier of the record to be updated).
- Validate input data.
- Call the appropriate method in the BL to update the data.
- BL interacts with DAL to update the corresponding record in the database.

4. Delete (Remove)

Description: Deletes existing data from the database.

Steps:

- Collect input data (such as the unique identifier of the record to be deleted).
- Call the appropriate method in the BL to delete the data.
- BL interacts with DAL to remove the corresponding record from the database.