Introduction
Debug points
Different Debug windows
Editing
Conditional break points
Data inspector
Conditional compilation
Types class (i.e. Abstract, sealed etc.)
Generics
File system in Depth
Data serialization (JSON,XML)
Base library features
Lambda expression
Extension methods
LINQ (with DataTable, List, etc.)
ORM tool
Security & Cryptography
Dynamic type
Database with C# (CRUD)

Introduction:

- Debugging is a crucial aspect of software development that involves identifying and fixing errors or bugs in a program. It is a systematic process of locating, isolating, and fixing problems within a computer program. Developers use various tools and techniques to debug their code and ensure that it behaves as expected.

Debug Points:

- Debug points, also known as breakpoints, are markers that developers set in their code to pause its execution at a specific point. When the program reaches a debug point, it stops, allowing the developer to inspect the current state of variables, analyze the program's flow, and identify potential issues.

Different Debug Windows:

- ***Code/Source Window:***
  - This window displays the source code of the program being debugged. Developers can set breakpoints, navigate through the code, and inspect variables in this window.
- ***Call Stack Window:***
  - It shows the sequence of function calls that led to the current point in the code. This helps developers understand the program's execution flow and identify the origin of an issue.
- ***Watch Window:***
  - Developers use this window to monitor the values of specific variables or expressions during debugging. It provides a real-time view of the variables, aiding in identifying unexpected changes or incorrect values.
- ***Output Window:***
  - This window displays messages generated during the debugging process, such as console outputs, error messages, and information from the debugger itself.
- ***Locals Window:***
  - It shows the local variables within the current scope of the code. Developers can inspect and modify these variables during debugging.

Editing:

- During debugging, developers might need to make temporary changes to the code to test hypotheses or apply quick fixes. Some debuggers allow for code edits during the debugging session, but these changes are usually not permanent and won't affect the original source code.

Conditional Breakpoints:

- Conditional breakpoints are breakpoints that only pause the execution of the program if a specified condition is met. This allows developers to focus on specific scenarios or occurrences, making the debugging process more efficient.

Data Inspector:

- The data inspector is a tool that enables developers to examine and analyze the values of variables, data structures, and memory during debugging. It provides a detailed view of the data, helping developers identify issues related to incorrect values or unexpected changes.

Conditional Compilation:

- Conditional compilation involves including or excluding portions of code during the compilation process based on specified conditions. This is often used to create different versions of a program for various environments or configurations. During debugging, conditional compilation can be used to selectively include or exclude code segments, aiding in isolating and fixing specific issues.

Types Of Classes ➖

Abstract Class:

- Definition: An abstract class is a class that cannot be instantiated on its own and may contain abstract methods (methods without a body).
- Purpose: Abstract classes serve as a base class for other classes. They provide a common interface and partial implementation that derived classes must complete.

Sealed Class:

- Definition: A sealed class is a class that cannot be inherited by other classes. It is marked with the `sealed` keyword.
- Purpose: Sealed classes are used when you want to restrict the inheritance of a class. Instances of sealed classes cannot be used as a base for other classes.

Static Class:

- Definition: A static class is a class that cannot be instantiated, and all its members (fields, methods, etc.) must be static.
- Purpose: Static classes are used to define utility functions, extension methods, or constants. They provide a way to organize related functionality without the need for instantiation.

Partial Class:

- Definition: A partial class is a class that can be defined in multiple files within the same assembly. Each part is marked with the `partial` keyword.
- Purpose: Partial classes are often used to split a large class into smaller, more manageable files. This is useful in scenarios where code generation tools are used, and developers need to extend the functionality without modifying the generated code.

Concrete Class:

- Definition: A concrete class is a class that can be instantiated, and it may provide a full implementation of its methods.
- Purpose: Concrete classes are the most common type of classes and are used to create objects. They can be instantiated, and their methods can be called directly.

Interface:

- Definition: An interface is a type that defines a contract specifying a set of methods that a class must implement. Interfaces cannot contain implementation.
- Purpose: Interfaces provide a way to achieve multiple inheritance in C#. Classes can implement one or more interfaces to define common behavior.

Generics ➖

- Generics introduces the concept of type parameters to .NET, which make it possible to design classes and methods that defer the specification of one or more types until the class or method is declared and instantiated by client code. For example, by using a generic type parameter `T`, you can write a single class that other client code can use without incurring the cost or risk of runtime casts or boxing operations, as shown here:

Generics overview

- Use generic types to maximize code reuse, type safety, and performance.
- The most common use of generics is to create collection classes.
- The .NET class library contains several generic collection classes in the System.Collections.Generic namespace. The generic collections should be used whenever possible instead of classes such as ArrayList in the System.Collections namespace.
- You can create your own generic interfaces, classes, methods, events, and delegates.
- Generic classes may be constrained to enable access to methods on particular data types.
- Information on the types that are used in a generic data type may be obtained at run-time by using reflection.

Advantages and disadvantages of generics

There are many advantages to using generic collections and delegates:

- Type safety. Generics shift the burden of type safety from you to the compiler. There is no need to write code to test for the correct data type because it is enforced at compile time. The need for type casting and the possibility of run-time errors are reduced.
- Less code and code is more easily reused. There is no need to inherit from a base type and override members. For example, the LinkedList<T> is ready for immediate use. For example, you can create a linked list of strings with the following variable declaration.
- 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. For example, the Predicate<T> generic delegate allows you to create a method that implements your own search criteria for a particular type and to use your method with methods of the Array type such as Find, FindLast, and FindAll.

1. Generic Controllers

Controllers in Web API handle HTTP requests and generate HTTP responses. By using generics, you can create generic controllers that operate on different types. This is useful when you have common CRUD (Create, Read, Update, Delete) operations for multiple entities.

2. Generic Services

Services in Web API often encapsulate business logic or data access logic. Generics can be applied to services to create reusable components that work with different types of entities.

3. Generic DTOs (Data Transfer Objects)

DTOs are used to transfer data between the client and server. Generics can be employed to create generic DTOs that accommodate different types of data.

4. Generic Repositories

When working with databases or external data sources, generic repositories can be beneficial. They allow you to define common data access methods that work with different entity types.

File system in Depth ➖

1. System.IO Namespace:

- The System.IO namespace contains classes for working with files, directories, and paths in .NET applications. Key classes include File, Directory, and Path.

2. Working with Paths:

- The Path class provides methods for working with file and directory paths. It helps in creating, combining, and manipulating file paths in a platform-independent manner.

3. File Operations:

- The File class provides static methods for reading from and writing to files, checking file existence, and other file-related operations.

4. Directory Operations:

- The Directory class provides static methods for working with directories, such as creating, moving, and deleting them.

5. Uploading and Downloading Files in Web API:

- When dealing with file uploads in an ASP.NET Web API, you handle HTTP requests with multipart/form-data content type. You can access files from the HttpContext.Current.Request.Files collection.

Data serialization (JSON,XML) ➖

- Serialization is the process of converting an object's state (including its fields, properties, and other attributes) into a format that can be easily stored or transmitted, and later reconstructed. The primary purpose of serialization is to persistently store or transmit the object's state in a way that it can be reconstructed at a later time or in a different environment.

JSON Serialization:

- JSON is a lightweight data interchange format that is easy for humans to read and write and easy for machines to parse and generate.
- It represents data as key-value pairs and is widely used for data exchange between web servers and clients.

Binary Serialization:

- Binary serialization involves converting object data into a binary format.
- It is more space-efficient compared to human-readable formats like JSON or XML but is not easily human-readable.
- Binary serialization is often used in scenarios where performance and efficiency are critical.

XML Serialization Attributes:

- Programming languages, such as C# in the .NET framework, provide attributes that can be applied to classes and members to control the serialization process.
- For example, in C#, the [XmlElement], [XmlAttribute], and [XmlIgnore] attributes are commonly used for XML serialization.

Data serialization is the process of converting complex data structures or objects into a format that can be easily stored, transmitted, or reconstructed.

JSON (JavaScript Object Notation), XML (eXtensible Markup Language), Binary Serialization (e.g., in .NET)

JSON Serialization : - Lightweight and human-readable. - Widely used in web development. - Supports complex data structures.

XML Serialization : - Document-based format. - Extensively used in configuration files. - Verbosity can lead to larger file sizes.

Binary Serialization : Compact and efficient for binary data. - Commonly used in programming languages like C# (.NET).

Lambda expression ➖

A lambda expression in programming is a concise way to represent an anonymous function—a function without a name. Lambda expressions are a feature available in many modern programming languages and are particularly common in functional programming paradigms. They are often used to define inline functions or to pass functions as arguments to higher-order functions.

The general syntax of a lambda expression is as follows:

```
(parameters) => expression
```

Here's a breakdown of the components:

- **parameters:** The input parameters of the function.
- **=>:** The lambda operator, indicating the separation between parameters and the function body.
- **expression:** The body of the function, which produces the result.

Lambda expressions are powerful tools for writing concise and expressive code, especially when working with functional programming concepts or when functions are used as first-class citizens. They allow developers to write more compact code without sacrificing readability.

Extension methods ➖

An extension method in C# is a special kind of static method that allows you to add new methods to existing types without modifying them. This concept was introduced in C# 3.0 and provides a way to extend the functionality of classes, even those that you don't have access to or cannot modify.

Extension Method allows you to inject additional methods without modifying, deriving or recompiling the original class , struct or Interface.

key characteristics :

Static Method: Extension methods are static methods within static classes.
Defined in a Static Class:
- Extension methods must be defined in a static class. This class is typically used as a container for related extension methods.

First Parameter is Modified Type with this Keyword:
- The first parameter of an extension method specifies the type being extended, and it is prefixed with the this keyword.

LINQ ➖

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.

A *query* is an expression that retrieves data from a data source. Different data sources have different native query languages, for example SQL for relational databases and XQuery for XML. Developers must learn a new query language for each type of data source or data format that they must support. LINQ simplifies this situation by offering a consistent C# language model for kinds of data sources and formats. In a LINQ query, you always work with C# objects.

You use the same basic coding patterns to query and transform data in XML documents, SQL databases, .NET collections, and any other format when a LINQ provider is available.

Three Parts of a Query Operation

All LINQ query operations consist of three distinct actions:

1. **Obtain the data source.**
2. **Create the query.**
3. **Execute the query.**

The Data Source

- The data source in the preceding example is an array, which supports the generic IEnumerable<T> interface. This fact means it can be queried with LINQ. A query is executed in a foreach statement, and foreach requires IEnumerable or IEnumerable<T>. Types that support IEnumerable<T> or a derived interface such as the generic IQueryable<T> are called *queryable types*.

The Query

- The query specifies what information to retrieve from the data source or sources. Optionally, a query also specifies how that information should be sorted, grouped, and shaped before being returned. A query is stored in a query variable and initialized with a query expression. You use C# query syntax to write queries.

ORM Tool ➖

- Object-Relational Mapping (ORM) is a programming technique used to interact with databases in a more object-oriented manner. In the context of ASP.NET Web API, an ORM tool facilitates the seamless integration of the web API with a

relational database.
- One popular ORM tool used in ASP.NET is Entity Framework (EF). Entity Framework is an open-source ORM framework provided by Microsoft, and it simplifies the data access code in your application.

What Is ORM (Object Relational Mapping)?

- Object Relational Mapping, also known as Mapping Tool, in computer science is a programming technique used to convert data between systems of incompatible types through object-oriented programming languages.
- In simple terms, it is as if the ORM creates a “virtual database” and reflects this “virtual database” in a traditional database, such as SQL Server for example.

Dynamic Type ➖

- The dynamic type is a static type, but an object of type dynamic bypasses static type checking. In most cases, it functions like it has type object. The compiler assumes a dynamic element supports any operation. Therefore, you don't have to determine whether the object gets its value from a COM API, from a dynamic language such as IronPython, from the HTML Document Object Model (DOM), from reflection, or from somewhere else in the program. However, if the code isn't valid, errors surface at run time.

- Dynamic objects expose members such as properties and methods at run time, instead of at compile time. Dynamic objects enable you to create objects to work with structures that don't match a static type or format. For example, you can use a dynamic object to reference the HTML Document Object Model (DOM), which can contain any combination of valid HTML markup elements and attributes. Because each HTML document is unique, the members for a particular HTML document are determined at run time.

Security and Cryptography ➖

Security and cryptography play crucial roles in ensuring the confidentiality, integrity, and authenticity of information in various domains, including computer systems, communication networks, and data storage. Here's an overview of security and cryptography:

Security:

Information Security:

- **Confidentiality:** Protecting information from unauthorized access.
- **Integrity:** Ensuring that information remains unaltered and trustworthy.
- **Availability:** Ensuring that information is accessible when needed.

Network Security:

- **Firewalls:** Protecting networks by controlling incoming and outgoing traffic.
- **Intrusion Detection/Prevention Systems (IDS/IPS):** Identifying and responding to potential security threats.
- **Virtual Private Networks (VPNs):** Securing communication over public networks.

Application Security:

- **Secure Coding Practices:** Writing code with security in mind to prevent vulnerabilities.
- **Authentication and Authorization:** Verifying user identity and controlling access to resources.

Physical Security:

- **Access Controls:** Restricting physical access to facilities and equipment.
- **Surveillance Systems:** Monitoring and recording activities in physical spaces.

Security Policies and Procedures:

- **Risk Management:** Identifying, assessing, and mitigating potential risks.
- **Incident Response:** Developing plans to respond to security incidents.

Cryptography:

Basic Concepts:

- **Encryption:** Converting plaintext into ciphertext to protect data.
- **Decryption:** Reverting ciphertext back to plaintext using a key.
- **Key Management:** Handling the generation, distribution, storage, and destruction of cryptographic keys.

Types of Cryptography:

- **Symmetric Cryptography:** Uses a single key for both encryption and decryption.
- **Asymmetric Cryptography (Public-Key Cryptography):** Uses a pair of public and private keys for encryption and decryption.
- **Hash Functions:** Produces a fixed-size output (hash) based on input data, commonly used for data integrity verification.

Protocols and Standards:

- **SSL/TLS (Secure Socket Layer/Transport Layer Security)**: Ensures secure communication over a computer network.
- **IPsec (Internet Protocol Security):** Secures Internet Protocol (IP) communication.
- **PGP (Pretty Good Privacy) and GPG (GNU Privacy Guard)**: Used for secure email communication.

Blockchain and Cryptocurrencies:

- **Blockchain:** A distributed and decentralized ledger secured through cryptographic principles.
- Cryptocurrencies: Use cryptographic techniques to secure transactions and control the creation of new units.

Quantum Cryptography:

- **Post-Quantum Cryptography:** Developing cryptographic algorithms resistant to attacks by quantum computers.

Security and cryptography are dynamic fields, evolving to address new challenges and technologies. Continuous learning and adaptation are essential to stay ahead of emerging threats and vulnerabilities.