	codewitharrays.in freelance project available to buy contact on 8007592194	
SR.NO	Project NAME	Technology
1	Online E-Learning Platform Hub	React+Springboot+MySql
2	PG Mates / RoomSharing / Flat Mates	React+Springboot+MySql
3	Tour and Travel management System	React+Springboot+MySql
4	Election commition of India (online Voting System)	React+Springboot+MySql
5	HomeRental Booking System	React+Springboot+MySql
6	Event Management System	React+Springboot+MySql
7	Hotel Management System	React+Springboot+MySql
8	Agriculture web Project	React+Springboot+MySql
9	AirLine Reservation System / Flight booking System	React+Springboot+MySql
10	E-commerce web Project	React+Springboot+MySql
11	Hospital Management System	React+Springboot+MySql
12	E-RTO Driving licence portal	React+Springboot+MySql
13	Transpotation Services portal	React+Springboot+MySql
14	Courier Services Portal / Courier Management System	React+Springboot+MySql
15	Online Food Delivery Portal	React+Springboot+MySql
16	Muncipal Corporation Management	React+Springboot+MySql
17	Gym Management System	React+Springboot+MySql
18	Bike/Car ental System Portal	React+Springboot+MySql
19	CharityDonation web project	React+Springboot+MySql
20	Movie Booking System	React+Springboot+MySql

	freelance_Project available to buy contact on 8007592194	
21	Job Portal web project	React+Springboot+MySql
22	LIC Insurance Portal	React+Springboot+MySql
23	Employee Management System	React+Springboot+MySql
24	Payroll Management System	React+Springboot+MySql
25	RealEstate Property Project	React+Springboot+MySql
26	Marriage Hall Booking Project	React+Springboot+MySql
27	Online Student Management portal	React+Springboot+MySql
28	Resturant management System	React+Springboot+MySql
29	Solar Management Project	React+Springboot+MySql
30	OneStepService LinkLabourContractor	React+Springboot+MySql
31	Vehical Service Center Portal	React+Springboot+MySql
32	E-wallet Banking Project	React+Springboot+MySql
33	Blogg Application Project	React+Springboot+MySql
34	Car Parking booking Project	React+Springboot+MySql
35	OLA Cab Booking Portal	React+NextJs+Springboot+MySql
36	Society management Portal	React+Springboot+MySql
37	E-College Portal	React+Springboot+MySql
38	FoodWaste Management Donate System	React+Springboot+MySql
39	Sports Ground Booking	React+Springboot+MySql
40	BloodBank mangement System	React+Springboot+MySql

41	Bus Tickit Booking Project	React+Springboot+MySql
42	Fruite Delivery Project	React+Springboot+MySql
43	Woodworks Bed Shop	React+Springboot+MySql
44	Online Dairy Product sell Project	React+Springboot+MySql
45	Online E-Pharma medicine sell Project	React+Springboot+MySql
46	FarmerMarketplace Web Project	React+Springboot+MySql
47	Online Cloth Store Project	React+Springboot+MySql
48	Train Ticket Booking Project	React+Springboot+MySql
49	Quizz Application Project	JSP+Springboot+MySql
50	Hotel Room Booking Project	React+Springboot+MySql

51	Online Crime Reporting Portal Project	React+Springboot+MySql
52	Online Child Adoption Portal Project	React+Springboot+MySql
53	online Pizza Delivery System Project	React+Springboot+MySql
54	Online Social Complaint Portal Project	React+Springboot+MySql
55	Electric Vehical management system Project	React+Springboot+MySql
56	Online mess / Tiffin management System Project	React+Springboot+MySql
57		React+Springboot+MySql
58		React+Springboot+MySql
59		React+Springboot+MySql
60		React+Springboot+MySql

Spring Boot + React JS + MySQL Project List

Sr.No	Project Name	YouTube Link
1	Online E-Learning Hub Platform Project	https://youtu.be/KMjyBaWmgzg?si=YckHuNzs7eC84-IW
2	PG Mate / Room sharing/Flat sharing	https://youtu.be/4P9cIHg3wvk?si=4uEsi0962CG6Xodp
3	Tour and Travel System Project Version 1.0	https://youtu.be/-UHOBywHaP8?si=KHHfE_A0uv725f12
4	Marriage Hall Booking	https://youtu.be/VXz0kZQi5to?si=llOS-QG3TpAFP5k7
5	Ecommerce Shopping project	https://youtu.be/vJ_C6LkhrZ0?si=YhcBylSErvdn7paq
6	Bike Rental System Project	https://youtu.be/FIzsAmIBCbk?si=7ujQTJqEgkQ8ju2H
7	Multi-Restaurant management system	https://youtu.be/pvV-pM2Jf3s?si=PgvnT-yFc8ktrDxB
8	Hospital management system Project	https://youtu.be/IynIouBZvY4?si=CXzQs3BsRkjKhZCw
9	Municipal Corporation system Project	https://youtu.be/cVMx9NVyI4I?si=qX0oQt-GT-LR_5jF
10	Tour and Travel System Project version 2.0	https://youtu.be/_4u0mB9mHXE?si=gDiAhKBowi2gNUKZ

Sr.No	Project Name	YouTube Link
11	Tour and Travel System Project version 3.0	https://youtu.be/Dm7nOdpasWg?si=P_Lh2gcOFhlyudug
12	Gym Management system Project	https://youtu.be/J8_7Zrkg7ag?si=LcxV51ynfUB7OptX
13	Online Driving License system Project	https://youtu.be/3yRzsMs8TLE?si=JRI_z4FDx4Gmt7fn
14	Online Flight Booking system Project	https://youtu.be/m755rOwdk8U?si=HURvAY2VnizIyJlh
15	Employee management system project	https://youtu.be/lD1iE3W_GRw?si=Y_jv1xV_BljhrD0H
16	Online student school or college portal	https://youtu.be/4A25aEKfei0?si=RoVgZtxMk9TPdQvD
17	Online movie booking system project	https://youtu.be/Lfjv_U74SC4?si=fiDvrhhrjb4KSlSm
18	Online Pizza Delivery system project	https://youtu.be/Tp3izreZ458?si=8eWAOzA8SVdNwlyM
19	Online Crime Reporting system Project	https://youtu.be/0UlzReSk9tQ?si=6vN0e70TVY1GOwPO
20	Online Children Adoption Project	https://youtu.be/3T5HC2HKyT4?si=bntP78niYH802I7N

C# BASIC INTERVIEW QUESTIONS & ANSWERS

MUHAMMAD AFZAL

LINKEDIN :HTTPS://WWW.LINKEDIN.COM/IN/MUHAMMAD-AFZAL-DEARBUG/

1. What is C#?

Ans: C# (pronounced "C-sharp") is a programming language developed by Microsoft. It is used to create a wide range of applications, including web, mobile, and desktop applications. C# is part of the .NET framework, which provides tools and libraries for building these applications. It is known for being easy to learn and use, especially for those familiar with other programming languages like Java or C++.

2. What are the key features of C#?

Ans: The key features of C# include:

1. Simple Syntax: C# has a clear and easy-to-read syntax, making it beginner-friendly.

2. Object-Oriented: C# supports object-oriented programming (OOP) principles like inheritance, encapsulation, polymorphism, and abstraction.

3. Type-Safe: C# ensures that the type of variables is checked at compile time, reducing errors.

4. Automatic Memory Management: C# has a garbage collector that automatically manages memory allocation and deallocation.

5. Rich Standard Library: C# has a large library of pre-built classes and methods, making development faster and easier.

6. Interoperability: C# can work with other languages and technologies, especially within the .NET framework.

7. Component-Oriented: C# supports building applications using reusable components.

8. Versioning and Scalability: C# is designed to support versioning and can handle the growth of applications efficiently.

9. Asynchronous Programming: C# provides features for easy implementation of asynchronous operations, which helps in writing responsive applications.

10. Cross-Platform: With .NET Core, C# applications can run on Windows, Linux, and macOS.

3. What is the difference between a class and an object in C#?

Ans: In C#, the difference between a class and an object is:

1. Class:

- A class is a blueprint or template for creating objects.
- It defines properties, methods, and events that the objects created from the class will have.
- Example: `public class Car { public string color; public void Drive() { } }`

2. Object:

- An object is an instance of a class.
- It represents a specific example of the class with actual values for the properties.

- Example: `Car myCar = new Car(); myCar.color = "red"; myCar.Drive();`

In simple terms, think of a class as a recipe, and an object as the actual dish made from that recipe.

4. What are the different types of classes in C#?

Ans: In C#, there are several types of classes, each serving different purposes:

1. Standard Classes:

- The most common type of class used to create objects.
- Example: `public class Car { }`

2. Static Classes:

- Cannot be instantiated. They contain only static members.
- Useful for utility or helper methods.
- Example: `public static class MathUtilities { public static int Add(int a, int b) { return a + b; } }`

3. Abstract Classes:

- Cannot be instantiated directly. Must be inherited by other classes.
- Can have abstract methods (without implementation) and non-abstract methods (with implementation).
- Example: `public abstract class Animal { public abstract void MakeSound(); }`

4. Sealed Classes:

- Cannot be inherited by other classes. Used to prevent further inheritance.

- Example: `public sealed class FinalClass { }`

5. Partial Classes:

- Allows a class to be split into multiple files. All parts are combined into a single class at compile time.

- Useful for managing large classes.

- Example: Part 1: `public partial class MyClass { }` Part 2: `public partial class MyClass { }`

6. Nested Classes:

- A class defined within another class. Can be used to logically group classes.

- Example: `public class OuterClass { public class InnerClass { } }`

Each type of class serves a unique purpose and is used based on the requirements of the application being developed.

5. Explain the concept of inheritance in C#.

Ans: Inheritance in C# is a feature that allows a class to inherit properties and methods from another class. It helps in creating a hierarchical relationship between classes, promoting code reuse and reducing redundancy.

6. What are access Modifiers in C#?

Ans: Access modifiers in C# define the visibility and accessibility of classes, methods, and other members. They control how the members of a class can be accessed from other parts of the code. Here are the main access modifiers:

1. **Public**:
   - The member is accessible from any other code in the same assembly or another assembly that references it.
   - Example: public int myPublicVar;
2. **Private**:
   - The member is accessible only within the class or struct in which it is declared.
   - Example: private int myPrivateVar;
3. **Protected**:
   - The member is accessible within its class and by derived class instances.
   - Example: protected int myProtectedVar;
4. **Internal**:
   - The member is accessible only within files in the same assembly.
   - Example: internal int myInternalVar;
5. **Protected Internal**:
   - The member is accessible within its class and derived classes, but only within the same assembly.
   - Example: protected internal int myProtectedInternalVar;
6. **Private Protected**:

    - The member is accessible within its class and by derived class instances, but only within the same assembly.
    - Example: private protected int myPrivateProtectedVar;

7. What is polymorphism in C#?

Ans: Polymorphism in C# is a concept that allows objects to be treated as instances of their base class rather than their actual class. It enables a single interface to be used for different underlying forms (data types). There are two types of polymorphism in C#:

1. **Compile-time Polymorphism (Method Overloading)**:
    - Achieved by having multiple methods with the same name but different parameters.
2. **Run-time Polymorphism (Method Overriding)**:
    - Achieved by using inheritance and virtual/override keywords.
    - Allows a derived class to provide a specific implementation of a method that is already defined in its base class.

8. What is encapsulation in C#? Why is it important?

Ans: Encapsulation in C# is a principle of object-oriented programming that involves bundling the data (variables) and the methods (functions) that operate on the data into a single unit, called a class. It restricts direct access to some of an object's components, which can prevent the accidental modification of data. Encapsulation is achieved by using access modifiers to control the visibility of the class members.

Why Encapsulation is Important:

1. **Data Hiding**:
   - Encapsulation allows you to hide the internal state of an object and only expose a controlled interface to the outside world.
   - This helps in protecting the integrity of the data by preventing unauthorized or unintended access.
2. **Improved Maintainability**:
   - By controlling access to the internal state of the object, encapsulation makes it easier to modify and maintain the code.
   - Changes to the internal implementation do not affect other parts of the code that use the object.
3. **Enhanced Flexibility**:
   - Encapsulation allows you to change the internal workings of a class without affecting the external code that uses the class.
   - This makes the code more flexible and adaptable to change.
4. **Reusability**:
   - Encapsulated code can be more easily reused in different parts of a program or in different projects.
   - Well-defined interfaces ensure that the encapsulated code can be reused without knowing the details of its implementation.

9. What is an abstraction in C#?

Ans: Abstraction in C# is a principle of object-oriented programming that involves hiding complex implementation details and showing only the essential features of an object. It allows developers to focus on what an object does instead of how it does it. Abstraction can be achieved using abstract classes and interfaces.

Key Concepts of Abstraction:

- **Abstract Class**: A class that cannot be instantiated on its own and can contain abstract methods (without implementation) and non-abstract methods (with implementation).
- **Interface**: A contract that defines a set of methods and properties that a class must implement, without providing any implementation itself.

10. What is an interface in C#? How is it different from an abstract class?

Ans: An interface in C# is a contract that defines a set of methods, properties, events, or indexers that a class or struct must implement. Interfaces do not contain any implementation; they only specify the signatures of the members. Classes that implement an interface must provide the implementation for all the members defined in that interface.

Key Features of Interfaces:

- **No Implementation**: Interfaces only define methods and properties without any code.
- **Multiple Inheritance**: A class can implement multiple interfaces, allowing for more flexible designs.
- **Cannot Contain Fields**: Interfaces cannot contain fields or any data members, only method signatures and properties.

Difference Between Interface and Abstract Class:

Feature	Interface	Abstract Class
Implementation	No implementation; only method signatures	Can have both abstract and non-abstract methods with implementation
Multiple Inheritance	A class can implement multiple interfaces	A class can inherit from only one abstract class
Members	Can contain methods, properties, events, and indexers	Can contain methods, properties, fields, and constructors
Access Modifiers	All members are public by default; cannot have access modifiers	Can have various access modifiers (public, protected, etc.)
Field Declaration	Cannot contain fields	Can contain fields

When to Use:

- Use **interfaces** when you want to define a contract that can be implemented by multiple classes with potentially different implementations.
- Use **abstract classes** when you want to provide a common base with some shared code (implementation) that can also enforce certain methods to be overridden in derived classes.

11. What is the difference between struct and class in C#?

Ans: Struct: A struct is a value type in C# used to define lightweight objects that contain related data. Structs are stored on the stack and cannot inherit from other types, but they can implement interfaces.

Class: A class is a reference type in C# used to create complex objects that can contain data and methods. Classes are stored on the heap, support inheritance, and can have a default constructor.

12. What is a constructor in C#? How is it different from a method?

Ans: A **constructor** in C# is a special method that is automatically called when an instance of a class is created. Its main purpose is to initialize the object's data members. Constructors have the same name as the class and do not have a return type.

Key Features of Constructors:

- **Initialization**: Used to set initial values for object properties.
- **Same Name as Class**: The name of the constructor is always the same as the class name.
- **No Return Type**: Constructors do not have a return type, not even void.
- **Overloading**: You can have multiple constructors in a class with different parameters (constructor overloading).

Difference Between a Constructor and a Method:

1. **Purpose**:
   - A constructor is used to initialize an object when it is created.
   - A method performs a specific operation or action on an object after it has been created.
2. **Naming**:
   - A constructor has the same name as the class.
   - A method can have any name.
3. **Return Type**:

  - A constructor does not have a return type.
  - A method must have a return type, even if it is void.

13. What is a destructor in C#? When is it called?

Ans: A **destructor** in C# is a special method that is used to perform cleanup operations on an object before it is removed from memory. Destructors have the same name as the class but are prefixed with a tilde (~).

Key Features of Destructors:

- **Automatic Invocation**: Destructors are called automatically by the garbage collector when an object is no longer in use and is about to be collected.
- **No Parameters or Return Type**: Destructors cannot take parameters and do not have a return type.
- **Single Destructor**: A class can only have one destructor.

When is a Destructor Called?

- A destructor is called when the garbage collector determines that there are no more references to an object, which means it is no longer accessible.
- The exact timing of when the destructor is called is not guaranteed; it depends on the garbage collection process.

14. Explain the concept of method overloading in C#.

Ans: Method overloading in C# is a feature that allows a class to have multiple methods with the same name but different parameters. This enables you to define different behaviors for the same method name based on the input it receives. Method overloading improves code readability and allows methods to be called with varying arguments.

15. Explain the concept of method overriding in C#.

Ans: Method overriding in C# is a feature that allows a derived class to provide a specific implementation of a method that is already defined in its base class. This enables polymorphism, where a method can behave differently based on the object that is calling it.

The method in the base class must be marked with the virtual or abstract keyword to allow overriding. **virtual**: Indicates that the method can be overridden in derived classes but does not have to be. **abstract**: Indicates that the method must be overridden in derived classes and has no implementation in the base class.

16. What is the difference between "const" and "readonly" in C#?

Ans: const: A const is a compile-time constant in C# that must be initialized at the time of declaration and cannot be changed afterwards. It is implicitly static.

readonly: A readonly field can be assigned either at the time of declaration or in the constructor of the class. Once assigned, its value cannot be changed. It can be either instance or static.

17. What is the difference between "static" and "non-static" methods in C#?

Ans: Static Methods: Static methods belong to the class itself rather than to any specific instance of the class. They can be called without creating an object of the class. Static methods can only access static members (variables or methods) of the class.

Non-Static Methods: Non-static methods belong to an instance of the class. They require an object of the class to be called and can access both instance members (variables or methods) and static members of the class.

18. What are access modifiers in C#?

Ans: public: Accessible from any other code.

private: Accessible only within the same class.
protected: Accessible within the same class and by derived classes.
internal: Accessible within the same assembly but not from another assembly.
protected internal: Accessible within the same assembly and by derived classes.
private protected: Accessible within the same class and derived classes within the same assembly.

19. What is a namespace in C#?

Ans: A **namespace** in C# is a container that organizes classes, interfaces, enums, structs, and delegates. It helps to group related types together and avoid naming conflicts by providing a way to distinguish between different types that have the same name. Namespaces also make it easier to manage and navigate large codebases.

20. Explain the concept of properties in C#.

Ans: Properties in C# are members of a class that provide a flexible mechanism to read, write, or compute the values of private fields. They are a way to expose data while encapsulating the internal implementation and adding logic for getting and setting the values.

21. What is the difference between "ref" and "out" keywords in C#?

Ans: In C#, both ref and out keywords are used to pass arguments by reference to methods, allowing the method to modify the argument's value. However, they have key differences in how they are used and what they enforce.

ref Keyword:

1. **Initialization**: The variable passed as a ref parameter must be initialized before it is passed to the method.
2. **Read and Write**: The called method can read and modify the value of the argument.
3. **Bidirectional**: Useful when the method needs to both read from and write to the variable.

out Keyword:

1. **Initialization**: The variable passed as an out parameter does not need to be initialized before it is passed to the method. However, it must be assigned a value within the method before the method returns.

2. **Write Only**: The called method must assign a value to the out parameter before it returns, effectively making it write-only.
3. **Unidirectional**: Useful when the method needs to return multiple values or assign a value to an uninitialized variable.

22. What is an exception in C#? How is exception handling done?

Ans: An **exception** in C# is an event that occurs during the execution of a program that disrupts the normal flow of instructions. It usually indicates an error or an unexpected condition, such as trying to divide by zero, accessing an array out of bounds, or attempting to open a file that does not exist.

Exception Handling in C#:

Exception handling in C# is done using the try, catch, finally, and throw keywords. This mechanism allows you to catch and handle exceptions gracefully, ensuring that your program can recover from errors or fail gracefully.

23. What are delegates in C#?

Ans: Delegates in C# are types that represent references to methods with a specific signature and return type. They are used to encapsulate a method call as an object, allowing methods to be passed as parameters, assigned to variables, and called dynamically.

24. Explain the concept of events in C#.

Ans: Events in C# are a way for a class to provide notifications to other classes or objects when something of interest occurs. They are built upon delegates and provide a mechanism for publishing and subscribing to actions, enabling a form of communication between objects.

25. What is a lambda expression in C#?

Ans: A **lambda expression** in C# is a concise way to represent anonymous methods using a simplified syntax. It allows you to create inline functions that can be used as delegates or expression tree types. Lambda expressions are particularly useful for writing short and clear code, especially when working with LINQ (Language Integrated Query) and other scenarios where small functions are needed.

26. What is LINQ in C#?

Ans: LINQ (Language Integrated Query) in C# is a set of features that allows developers to write queries directly in C# using a syntax that is similar to SQL. LINQ enables querying and manipulating data from various sources, such as collections, databases, XML, and more, in a consistent and readable manner.

27. What are the different types of collections in C#?

Ans: In C#, collections are used to store and manage groups of related objects. The .NET Framework provides several types of collections, each with its own characteristics and use cases. Here are the main types of collections in C#:

Arrays: Fixed-size collection for storing similar items.
List<T>: Dynamic collection for storing items with variable size.
Dictionary<TKey, TValue>: Collection of unique key-value pairs.
HashSet<T>: Collection of unique items with no duplicates.
Queue<T>: FIFO collection for processing items in order.
Stack<T>: LIFO collection for managing elements in reverse order.
LinkedList<T>: Efficient collection for frequent insertions and deletions.
ObservableCollection<T>: Collection that supports notifications for data binding.

28. What is the difference between an Array and a List in C#?

Ans: The main differences between an **Array** and a **List** in C# revolve around their size, flexibility, and functionality.

Array:

- Fixed size, basic functionality, allows multi-dimensional arrays, generally better performance for indexed access.

List:

- Dynamic size, rich functionality with methods for manipulation, generic type-safe, better for scenarios where the size of the collection can change.

29. What is a Hashtable in C#?

Ans: A **Hashtable** in C# is a collection that stores key-value pairs. It is part of the System.Collections namespace and provides a way to quickly access values based on their associated keys. Hashtables are implemented using a hash table data structure, which allows for fast retrieval of values.

30. What is the difference between a Stack and a Queue in C#?

Ans: The main difference between a **Stack** and a **Queue** in C# lies in how they store and retrieve elements. They are both collections, but they operate on different principles:

Stack: LIFO structure where the last added item is the first to be removed. Used for reversing and backtracking.

Queue: FIFO structure where the first added item is the first to be removed. Used for processing items in order.

31. What are generics in C#?

Ans: Generics in C# are a feature that allows you to define classes, interfaces, and methods with a placeholder for the data type. This means you can create a single implementation that works with any data type while maintaining type safety. Generics help to avoid code duplication and improve performance by eliminating the need for boxing and unboxing with value types.

32. Explain the concept of threading in C#.

Ans: Threading in C# is a programming concept that allows the execution of multiple threads concurrently within a single application. A thread is the smallest unit of processing that can be scheduled by an operating system. Threading enables applications to perform multiple tasks at the same time, improving performance and responsiveness, especially in scenarios where tasks can be executed independently.

33. What is the "lock" statement in C#?

Ans: The **lock** statement in C# is a synchronization construct used to ensure that a block of code runs safely when accessed by multiple threads. It helps to prevent race conditions by allowing only one thread to enter a specific section of code at a time. When a thread locks an object, other threads that attempt to enter the same block of code must wait until the lock is released.

34. What is asynchronous programming in C#?

Ans: Asynchronous programming in C# is a programming model that allows a program to perform tasks without blocking the main thread. This means that while one task is waiting (like for a web request or file I/O), other tasks can continue to run. This helps improve the responsiveness of applications, especially user interfaces and server applications.

35. What are async and await in C#?

Ans: async and **await** are keywords in C# that are used to implement asynchronous programming. They enable developers to write code that performs long-running operations without blocking the main thread, allowing for better responsiveness and performance in applications.

36. What is a Task in C#?

Ans: In C#, a **Task** represents an asynchronous operation. It is part of the Task Parallel Library (TPL) and is used to manage and execute asynchronous code. A Task provides a way to run work on a background thread and to handle the completion of that work.

37. Explain the concept of garbage collection in C#.

Ans: Garbage collection (GC) in C# is an automatic memory management feature that helps reclaim memory occupied by objects that are no longer in use, preventing memory leaks and optimizing resource usage. The .NET runtime includes a built-in garbage collector that handles this process.

38. What are partial classes in C#?

Ans: Partial classes in C# allow a single class definition to be split across multiple files. This feature is useful for organizing large classes into smaller, more manageable pieces and enabling multiple developers to work on the same class simultaneously without causing conflicts.

39. What are sealed classes in C#? Why are they used?

Ans: A **sealed class** in C# is a class that cannot be inherited by any other class. When you declare a class as sealed, you prevent any other class from deriving from it. This is useful in various scenarios where you want to restrict the extensibility of a class for security, performance, or design reasons.

Why Use Sealed Classes?

1. **Preventing Inheritance**: Sealed classes are useful when you want to prevent other developers from deriving from your class, ensuring that the intended functionality remains intact.
2. **Security**: In scenarios where a class contains sensitive logic or data, sealing the class helps protect it from being extended in ways that might compromise its security.
3. **Simplifying Code**: Sealing a class can simplify the design by eliminating the complexity that comes with inheritance and polymorphism, making the codebase easier to understand and maintain.
4. **Performance Benefits**: Sealed classes can lead to optimizations in method calls, as the runtime can assume that the method implementations are fixed and won't change through inheritance.

40. What are static classes in C#?

Ans: A **static class** in C# is a class that cannot be instantiated, and all its members (fields, methods, properties, etc.) must be static. Static classes are used to group related utility functions or data without the need to create an instance of the class.

41. What is the difference between “==” and “Equals” in C#?

Ans: == Operator: The == operator checks if two things are the same. It looks at the actual values for numbers and characters, and for objects, it checks if they are the exact same object.

Equals Method: The Equals method is used to see if two objects are equal. It can be changed in custom classes to compare specific properties instead of just checking if they are the same object.

Key Difference: Use == for basic comparisons and Equals for checking if two objects have the same important information.

42. What is type casting in C#?

Ans: Type casting in C# is the process of converting a variable from one type to another. This is necessary when you want to use a value of one type as if it were another type. Type casting can be done in two main ways: **implicit casting** and **explicit casting**.

1. Implicit Casting:

- **Definition**: Implicit casting occurs automatically when converting a smaller type to a larger type. For example, converting an int to a float does not require any special syntax.

2. Explicit Casting:

- **Definition**: Explicit casting requires a cast operator because it involves converting a larger type to a smaller type or changing between incompatible types. This may lead to data loss or exceptions if the conversion is not valid.

43. Explain the concept of nullable types in C#.

Ans: A nullable type is defined by appending a question mark (?) to a value type, such as int?, float?, bool?, etc. This indicates that the variable can hold either a value of that type or null. Nullable types can be used when you want to represent the absence of a value, such as when working with databases where a column can be NULL.

44. What is the difference between "String" and "StringBuilder" in C#?

Ans:

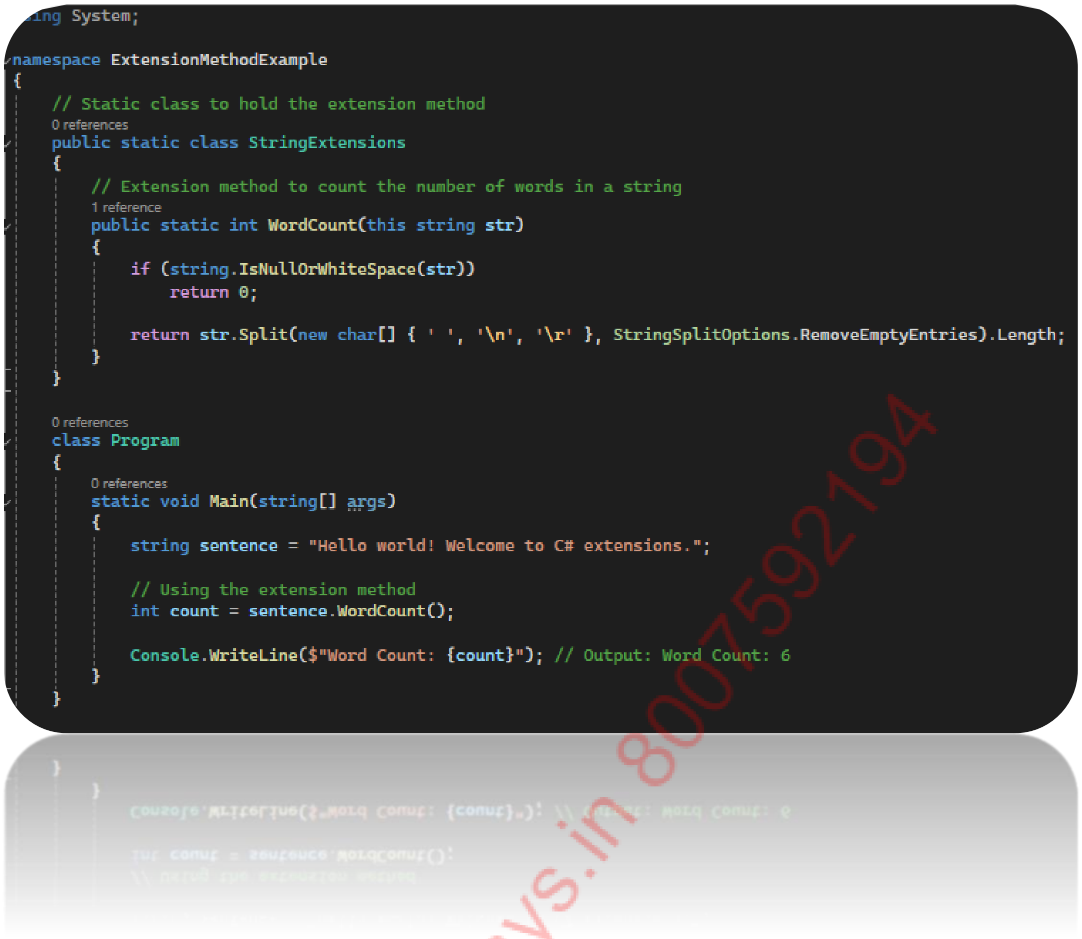
String: A String is immutable, which means that once a string is created, it cannot be changed. Any modification creates a new string object. Use String when you have a fixed or small amount of text that won't change often.

StringBuilder: StringBuilder is mutable, meaning that you can modify the contents without creating a new object. This makes it more efficient for repeated modifications. Use StringBuilder when you need to build or modify strings frequently, such as in loops or when concatenating large amounts of text.

45. What are extension methods in C#? Provide an example.

Ans: Extension methods in C# are a feature that allows you to add new methods to existing types without modifying their source code. This is useful for enhancing the functionality of classes or interfaces from libraries or frameworks you do not own or cannot change.

Example of an Extension Method:



```
using System;

namespace ExtensionMethodExample
{
    // Static class to hold the extension method
    0 references
    public static class StringExtensions
    {
        // Extension method to count the number of words in a string
        1 reference
        public static int WordCount(this string str)
        {
            if (string.IsNullOrWhiteSpace(str))
                return 0;

            return str.Split(new char[] { ' ', '\n', '\r' }, StringSplitOptions.RemoveEmptyEntries).Length;
        }
    }

    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            string sentence = "Hello world! Welcome to C# extensions.";

            // Using the extension method
            int count = sentence.WordCount();

            Console.WriteLine($"Word Count: {count}"); // Output: Word Count: 6
        }
    }
```

46. What is the difference between “IEnumerable” and “IQueryable” in C#?

Ans:

IEnumerable: IEnumerable is an interface that allows you to iterate over a collection of objects in memory. It executes queries after fetching all data, making it suitable for in-memory collections.

IQueryable: IQueryable is an interface that allows you to query data from an external data source, such as a database. It builds a query that is executed on the server side, improving performance by retrieving only the necessary data.

47. What is the purpose of the "using" statement in C#?

Ans: The using statement in C# is used to ensure that resources are properly released when they are no longer needed. It is specifically designed for managing resources that implement the IDisposable interface, such as file handles, database connections, and network streams.

48. What is the difference between value types and reference types in C#?

Ans: Value types store the actual data directly in their memory location. They are usually allocated on the stack, which makes access to them faster. Common value types include: (int, float, char, bool, struct). When a value type is assigned to another value type, a copy of the data is made. Changes to one variable do not affect the other.

Reference types store a reference (or address) to the actual data, which is located in the heap. They are allocated on the heap, which can make access slightly slower due to indirection. Common reference types include: (string, class, interface, array). When a reference type is assigned to another reference type, both variables point to the same data in memory. Changes made through one variable will affect the other.

49. What is boxing and unboxing in C#?

Ans: Boxing: Boxing is the process of converting a value type (like int or float) into a reference type (an object), allowing it to be treated as an object.

Unboxing: Unboxing is the process of converting a reference type back into a value type, retrieving the original value from the object.

50. What are attributes in C#?

Ans: Attributes in C# are special types of metadata that can be added to code elements such as classes, methods, properties, and fields. They provide additional information about the behavior of these elements and can be used by the compiler or runtime for various purposes.

51. Explain the concept of reflection in C#.

Ans: Reflection in C# is a feature that allows you to inspect and interact with your code's metadata at runtime. This means you can find out information about classes, methods, and properties, and even create objects or call methods dynamically while the program is running.

52. What is the difference between early binding and late binding in C#?

Ans: Early Binding: Early binding in C# is when the compiler resolves method calls and property access during compile time. This means that the compiler checks and verifies that the methods and properties exist and are accessible, resulting in faster execution because the calls are directly mapped in the compiled code. Early binding provides better type safety and helps catch errors during the compilation process.

Late Binding: Late binding in C# is when the method calls and property access are resolved at runtime. This allows more flexibility, as the exact type or method to be invoked does not need to be known at compile time. However, it comes with a performance cost since the program must determine which method or property to call while it is running. Late binding can lead to runtime errors if the methods or properties do not exist or are not accessible.

https://www.youtube.com/@codewitharrays

https://www.instagram.com/codewitharrays/

https://t.me/codewitharrays Group Link: https://t.me/ccee2025notes

+91 8007592194 +91 9284926333

codewitharrays@gmail.com

https://codewitharrays.in/project