1. What are the six combinations of access modifier keywords and what do they do?

Private Access Modifier

Objects that implement private access modifiers are accessible only inside a class or a structure. We can’t access them outside the class they are created.

Public Access Modifier

Objects that implement public access modifiers are accessible from everywhere in our project. There are no accessibility restrictions.

Protected Access Modifier

The protected keyword implies that the object is accessible inside the class and in all classes that derive from that class.

Internal Access Modifier

The internal keyword specifies that the object is accessible only inside its own assembly but not in other assemblies.

(These two are search.)

Protected Internal Access Modifier

The protected internal access modifier is a combination of protected and internal. As a result, we can access the protected internal member only in the same assembly or in a derived class in other assemblies (projects).

Private Protected Access Modifier

The private protected access modifier is a combination of private and protected keywords. We can access members inside the containing class or in a class that derives from a containing class, but only in the same assembly(project). Therefore, if we try to access it from another assembly, we will get an error.

1. What is the difference between the static, const, and readonly keywords when applied to a type member?

The static keyword is used to specify a static member, which means static members are common to all the objects and they do not tie to a specific object. This keyword can be used with classes, means this belonging to the class level.

Constant fields or local variables must be assigned a value at the time of declaration and after that, they cannot be modified. By default constant are static, hence you cannot define a constant type as static.

A readonly field can be initialized either at the time of declaration or within the constructor of the same class. Therefore, readonly fields can be used for run-time constants.

1. What does a constructor do?

When an instance of a class or a struct is created, its constructor is called. Constructors enable us to set default values, limit instantiation, and write code that is flexible and easy to read.

1. Why is the partial keyword useful?

The partial keyword is used to create partial types. It allows us to write partial class, interface, struct and method in two or more separate source files.

1. What is a tuple?

A tuple is a data structure that contains a sequence of elements of different data types. It can be used where you want to have a data structure to hold an object with properties, but you don't want to create a separate type for it.

1. What does the C# record keyword do?

use the record keyword to define a reference type that provides built-in functionality for encapsulating data.

1. What does overloading and overriding mean?

Overloading occurs when two or more methods in one class have the same method name but different parameters..

Overriding occurs when two methods have the same method name and parameters. One of the methods is in the parent class, and the other is in the child class. Overriding allows a child class to provide the specific implementation of a method that is already present in its parent class.​

1. What is the difference between a field and a property?

a field is a variable that is defined inside a class. It can be used to define the characteristics of an object or a class.

a property is a member of the class that provides an abstraction to set (write) and get (read) the value of a private field.

public class Person{

  private int age; // Field "age" declared

  private string name; // Field "name" declared

  public int Age{ // Property for age used inside the class

    get{        // getter to get the person's age

      return age;

    }

    set{        // setter to set the person's age

      age = value;

    }

  }

1. How do you make a method parameter optional?

We can implement optional parameters by using default value, method overloading.

1. What is an interface and how is it different from abstract class?

An interface defines a contract. Any class or struct that implements that contract must provide an implementation of the members defined in the interface. An interface may define a default implementation for members. It may also define static members in order to provide a single implementation for common functionality.

An abstract class allows you to create functionality that subclasses can implement or override. An interface only allows you to define functionality, not implement it. And whereas a class can extend only one abstract class, it can take advantage of multiple interfaces.

1. What accessibility level are members of an interface?

Public: Access is not restricted.

Protected: Access is limited to the containing class or types derived from the containing class.

Internal: Access is limited to the current assembly.

Private: Access is limited to the containing type.

protected internal: Access is limited to the current assembly or types derived from the containing class.

private protected : Access is limited to the containing class or types derived from the containing class within the current assembly.

True or False

12. True. Polymorphism allows derived classes to provide different implementations

of the same method.

13. True. The override keyword is used to indicate that a method in a derived class is

providing its own implementation of a method.

14. True. The new keyword is used to indicate that a method in a derived class is

providing its own implementation of a method.

1. False. Abstract methods can be used in a normal (non-abstract) class.
2. True. Normal (non-abstract) methods can be used in an abstract class.

17. True. Derived classes can override methods that were virtual in the base class.

18. True. Derived classes can override methods that were abstract in the base class.

An abstract method must be implemented in all non-abstract classes using the override keyword. After overriding, the abstract method is in the non-Abstract class. We can derive this class in another class, and again we can override the same abstract method with it.

19. False.

In a derived class, you can override a method that was neither virtual non abstract in the

base class.

1. True. A class that implements an interface does not have to provide an implementation for all of the members of the interface.

21. True. A class that implements an interface is allowed to have other members that

aren’t defined in the interface.

Not sure.

1. False. A class can have more than one base class.

Multiple inheritance is not supported in C# because the programmers believe it adds excessive complexity to the language while making not providing enough benefit.

23. True. A class can implement more than one interface.