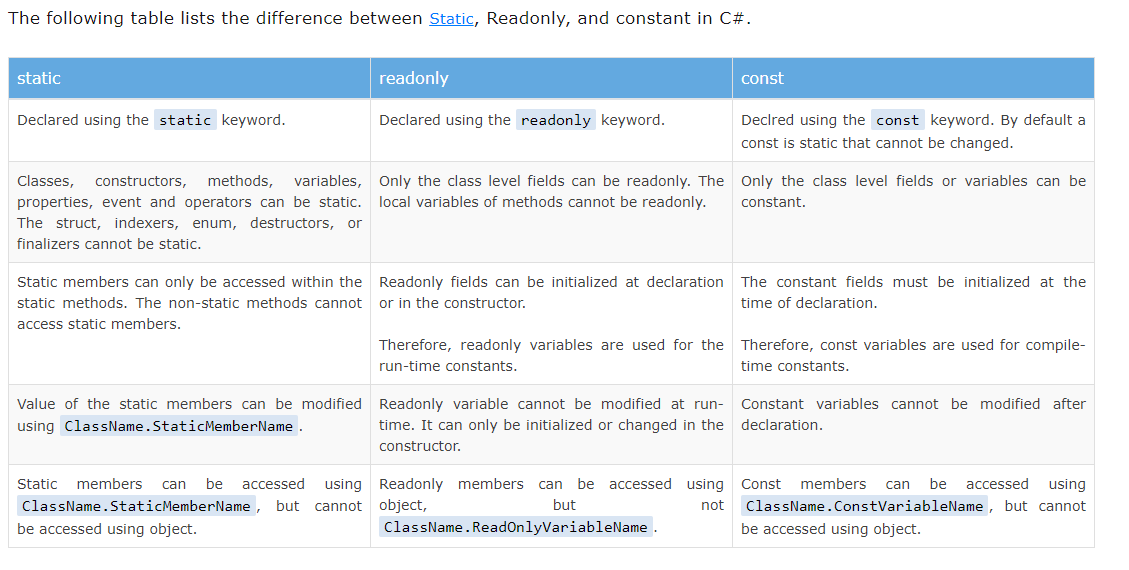
03 Object-Oriented Programming  
Test your knowledge  
1. What are the six combinations of access modifier keywords and what do they do?

* Public
* Private
* Protected
* Internal
* Protected Internal
* Private Protected

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

1. If you know the value will never, ever, *ever* change for any reason, use const.
2. If you're unsure of whether or not the value will change, but you don't want other classes or code to be able to change it, use readonly.
3. If you need a field to be a property of a type, and not a property of an instance of that type, use static.
4. A const value is also implicitly static.



3. What does a constructor do?

* A constructor is special member function shares the name of the class
* Constructor is used to create instance of the class
* Constructor can be overloaded with multiple parameters
* If there is no constructor then compiler provides a default constructor
* A default constructor is always replaced by the custom customer
* Constructor is used to initialize the class fields
* Constructor can not have any return type not even void

4. Why is the partial keyword useful?

The partial keyword indicates that other parts of the class, struct, or interface can be defined in the namespace. All the parts must use the partial keyword. All the parts must be available at compile time to form the final type. All the parts must have the same accessibility, such as public, private, and so on.

It provides a special ability to implement the functionality of a single class into multiple files and all these files are combined into a single class file when the application is compiled. A partial class is created by using a ***partial***keyword. This keyword is also useful to split the functionality of methods, interfaces, or structure into multiple files.

* When you want to chop the functionality of the class, method, interface, or structure into multiple files, then you should use *partial*keyword and all the files are mandatory to be available at compile time for creating the final file.
* The *partial*modifier can only present instantly before the keywords like struct, class, and interface.
* Every part of the partial class definition should be in the same assembly and [namespace](https://www.geeksforgeeks.org/c-namespaces/), but you can use a different source file name.
* Every part of the partial class definition should have the same accessibility as private, protected, etc.
* If any part of the partial class is declared as an abstract, sealed, or base, then the whole class is declared of the same type.
* The user is also allowed to use nested partial types.
* Dissimilar parts may have dissimilar base types, but the final type must inherit all the base types.

**Advantages :**

* With the help of partial classes, multiple developers can work simultaneously in the same class in different files.
* With the help of a partial class concept, you can split the UI of the design code and the business logic code to read and understand the code.
* When you were working with automatically generated code, the code can be added to the class without having to recreate the source file like in Visual studio.
* You can also maintain your application in an efficient manner by compressing large classes into small ones.

5. What is a tuple?

The word Tuple means “a data structure which consists of the multiple parts”. So tuple is a data structure which gives you the easiest way to represent a data set which has multiple values that may/may not be related to each other. It ***introduced in .NET Framework 4.0***. In tuple, *you can add elements from 1 to 8*. If you try to add elements greater than eight, then the compiler will throw an error. Tuples are generally used when you want to create a data structure which contains objects with their properties and you don’t want to create a separate type for that.

**Features of Tuples:**

* It allows us to represent multiple data into a single data set.
* It allows us to create, manipulate, and access data set.
* It return multiple values from a method without using *out*parameter.
* It can also store duplicate elements.
* It allows us to pass multiple values to a method with the help of single parameters.

6. What does the C# record keyword do?

Beginning with C# 9, you use the record keyword to define a [reference type](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/reference-types) that provides built-in functionality for encapsulating data. C# 10 allows the record class syntax as a synonym to clarify a reference type, and record struct to define a [value type](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/value-types) with similar functionality. You can create record types with immutable properties by using positional parameters or standard property syntax.

While records can be mutable, they're primarily intended for supporting immutable data models. The record type offers the following features:

* [Concise syntax for creating a reference type with immutable properties](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/record#positional-syntax-for-property-definition)
* Built-in behavior useful for a data-centric reference type:
  + [Value equality](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/record#value-equality)
  + [Concise syntax for nondestructive mutation](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/record#nondestructive-mutation)
  + [Built-in formatting for display](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/record#built-in-formatting-for-display)
* [Support for inheritance hierarchies](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/record#inheritance)

The preceding examples show some distinctions between records that are reference types and records that are value types:

* A record or a record class declares a reference type. The class keyword is optional, but can add clarity for readers. A record struct declares a value type.
* Positional properties are *immutable* in a record class and a readonly record struct. They're *mutable* in a record struct.

The remainder of this article discusses both record class and record struct types. The differences are detailed in each section. You should decide between a record class and a record struct similar to deciding between a class and a struct. The term *record* is used to describe behavior that applies to all record types. Either record struct or record class is used to describe behavior that applies to only struct or class types, respectively. The record type was introduced in C# 9; record struct types were introduced in C# 10.

7. 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.​

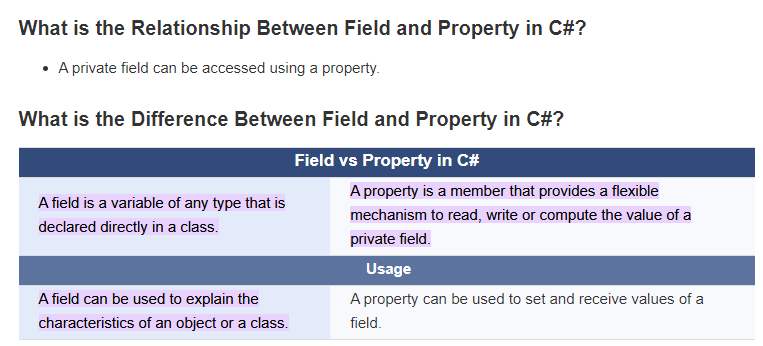
**Overloading**

* Must have at least two methods by the same name in the class.
* Must have a different number of parameters.
* If the number of parameters is the same, then it must have different types of parameters.
* Overloading is known as **compile-time polymorphism**.

**Overriding**

* Must have at least one method by the same name in both parent and child classes.
* Must have the same number of parameters.
* Must have the same parameter types.
* Overriding is known as **runtime polymorphism**​.

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



The difference between field and property in C# is that a field is a variable of any type that is declared directly in the class while property is a member that provides a flexible mechanism to read, write or compute the value of a private field.

9. How do you make a method parameter optional?

1. Use Parameter arrays
2. Default parameter
3. Use OptionalAttribute
4. Method Overloading

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

**Interface:**

Every single Method declared in an Interface will have to be implemented in the subclass. Only Events, Delegates, Properties (C#) and Methods can exist in a Interface. A class can implement multiple Interfaces.

**Abstract Class:**

Only Abstract methods have to be implemented by the subclass. An Abstract class can have normal methods with implementations. Abstract class can also have class variables beside Events, Delegates, Properties and Methods. A class can only implement one abstract class only due non-existence of Multi-inheritance in C#.

11. What accessibility level are members of an interface?

**interface:** The default and only access modifier supported is public and abstract.

12. True/False. Polymorphism allows derived classes to provide different implementations of the same method.

13. True/False. The override keyword is used to indicate that a method in a derived class is providing its own implementation of a method.

14. True/False. The new keyword is used to indicate that a method in a derived class is providing its own implementation of a method.

15. True/False. Abstract methods can be used in a normal (non-abstract) class.

16.True/False. Normal (non-abstract) methods can be used in an abstract class.

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

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

19. True/False. In a derived class, you can override a method that was neither virtual non abstract in the base class.

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

21. True/False. A class that implements an interface is allowed to have other members that aren’t defined in the interface.

22. True/False. A class can have more than one base class.

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

Working with methods  
1. Let’s make a program that uses methods to accomplish a task. Let’s take an array and reverse the contents of it. For example, if you have 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, it would become 10, 9, 8, 7, 6, 5, 4, 3, 2, 1.  
To accomplish this, you’ll create three methods: one to create the array, one to reverse the array, and one to print the array at the end.  
Your Mainmethod will look something like this:  
static void Main(string[] args) {  
int[] numbers = GenerateNumbers();  
Reverse(numbers);  
PrintNumbers(numbers);  
}  
The GenerateNumbersmethod should return an array of 10 numbers. (For bonus points, change the method to allow the desired length to be passed in, instead of just always being 10.)  
The PrintNumbersmethod should use a foror foreachloop to print out each item in the array. The Reversemethod will be the hardest. Give it a try and see what you can make happen. If you get stuck, here’s a couple of hints:  
Hint #1:To swap two values, you will need to place the value of one variable in a temporary  
location to make the swap:  
// Swapping a and b.  
int a = 3;  
int b = 5;  
int temp = a;  
a = b;  
b = temp;  
Hint #2:Getting the right indices to swap can be a challenge. Use a forloop, starting at 0 and going up to the length of the array / 2. The number you use in the forloop will be the index of the first number to swap, and the other one will be the length of the array minus the index minus 1. This is to account for the fact that the array is 0-based. So basically, you’ll be swapping array[index]with array[arrayLength – index – 1].

2. The Fibonacci sequence is a sequence of numbers where the first two numbers are 1 and 1, and every other number in the sequence after it is the sum of the two numbers before it. So the third number is 1 + 1, which is 2. The fourth number is the 2nd number plus the 3rd, which is 1 + 2. So the fourth number is 3. The 5th number is the 3rd number plus the 4th number: 2 + 3 = 5. This keeps going forever. The first few numbers of the Fibonacci sequence are: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...  
Because one number is defined by the numbers before it, this sets up a perfect opportunity for using recursion.  
Your mission, should you choose to accept it, is to create a method called Fibonacci, which takes in a number and returns that number of the Fibonacci sequence. So if someone calls Fibonacci(3), it would return the 3rd number in the Fibonacci sequence, which is 2. If someone calls Fibonacci(8), it would return 21. In your Mainmethod, write code to loop through the first 10 numbers of the Fibonacci sequence and print them out.

Hint #1:Start with your base case. We know that if it is the 1st or 2nd number, the value will be 1.

Hint #2:For every other item, how is it defined in terms of the numbers before it? Can you  
come up with an equation or formula that calls the Fibonaccimethod again?

Designing and Building Classes using object-oriented principles

1. Write a program that that demonstrates use of four basic principles of  
object-oriented programming /Abstraction/, /Encapsulation/, /Inheritance/ and  
/Polymorphism/.

2. Use /Abstraction/ to define different classes for each person type such as Student and Instructor. These classes should have behavior for that type of person.

3. Use /Encapsulation/ to keep many details private in each class.

4. Use /Inheritance/ by leveraging the implementation already created in the Person class to save code in Student and Instructor classes.

5. Use /Polymorphism/ to create virtual methods that derived classes could override to create specific behavior such as salary calculations.

6. Make sure to create appropriate /interfaces/ such as ICourseService, IStudentService, IInstructorService, IDepartmentService, IPersonService, IPersonService (should have person specific methods). IStudentService, IInstructorService should inherit from IPersonService. Person Calculate Age of the Person Calculate the Salary of the person, Use decimal for salary  
Salary cannot be negative number Can have multiple Addresses, should have method to get addresses Instructor Belongs to one Department and he can be Head of the Department Instructor will have added bonus salary based on his experience, calculate his years of experience based on Join Date Student  
Can take multiple courses Calculate student GPA based on grades for courses Each course will have grade from A to F Course Will have list of enrolled students Department Will have one Instructor as head Will have Budget for school year (start and end Date Time) Will offer list of courses

7. Try creating the two classes below, and make a simple program to work with them, as described below

Create a Color class: On a computer, colors are typically represented with a red, green, blue, and alpha (transparency) value, usually in the range of 0 to 255. Add these as instance variables. A constructor that takes a red, green, blue, and alpha value. A constructor that takes just red, green, and blue, while alpha defaults to 255 (opaque). Methods to get and set the red, green, blue, and alpha values from a Colorinstance. A method to get the grayscale value for the color, which is the average of the red, green and blue values.

Create a Ball class: The Ball class should have instance variables for size and color (the Color class you just created). Let’s also add an instance variable that keeps track of the number of times it has been thrown. Create any constructors you feel would be useful.  
Create a Pop method, which changes the ball’s size to 0.  
Create a Throw method that adds 1 to the throw count, but only if the ball hasn’t been popped (has a size of 0).

A method that returns the number of times the ball has been thrown.  
Write some code in your Main method to create a few balls, throw them around a few times, pop a few, and try to throw them again, and print out the number of times that the balls have been thrown. (Popped balls shouldn’t have changed.)