**1.What is C#??**

C# is an object-oriented, type-safe, and managed language that is compiled by .Net framework to generate Microsoft Intermediate Language.

**2. What is the difference between public, static, and void?**

**Public declared** variables or methods are accessible anywhere in the application.

|  |
| --- |
| public class MyClass {  public int myPublicVariable = 20;  public void myPublicMethod() {  // Do something  } } |

**Static declared** variables or methods are globally accessible without creating an instance of the class.

Static member is by default not globally accessible it depends upon the type of access modified used.

|  |
| --- |
| public class MyClass {  public static int myStaticVariable = 10;  public static void myStaticMethod() {  // Do something  } } |

**Void** is a type of modifier that states that the method or variable does not return any value.

|  |
| --- |
| public class MyClass {  public void myVoidMethod() {  // This method performs some action but does not return a value.  } } |

3. **What is an Object?**

* A class or struct definition is like a blueprint that specifies what the type can do.
* An object is basically a block of memory that has been allocated and configured according to the blueprint.
* A program may create many objects of the same class.
* Objects are also called instances, and they can be stored in either a named variable or in an array or collection.

**What is a class?**

A class is a template to create an object. It contains properties as well as methods. We can create many instances of objects from a single class.

**4. What are the type of class in C#?**

Abstract Class, Partial Class, Sealed Class, and Static Class.

|  |  |  |  |
| --- | --- | --- | --- |
| * An object cannot be created from the abstract class. * We use abstract keyword to declare an abstract class. * To use the abstract class, it must be inherited by a subclass. * An Abstract class contains both abstract and non-abstract methods. | * 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. | * 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. | * 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. * Static classes are sealed and therefore cannot be inherited. They are also not available for use in the same way as a regular class. |
| abstract class AbstractClass {  } | partial class PartialClass {  } | sealed class SealedClass {  } | static class StaticClass {  } |

**5. What is Encapsulations?**

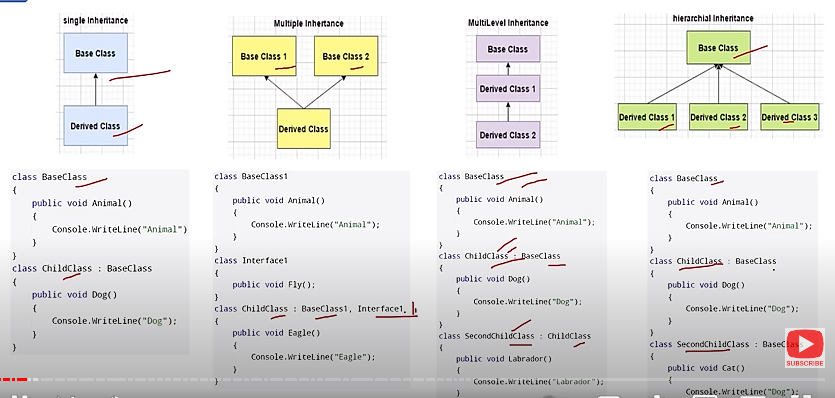
Encapsulation is a process of binding the data members and member functions into a single unit.

**6. What is Abstraction?**

Abstraction is a process of hiding the implementation details and displaying the essential features.

**7. What is Inheritance?**

Inheritance is a process of deriving the new class from an already existing class.



**8. What is Polymorphism?**

* When a message can be processed in different ways it is called polymorphism.
* polymorphism ability of a single interface to be used to refer to multiple implementations of a particular behaviour.
* Polymorphism means many forms.

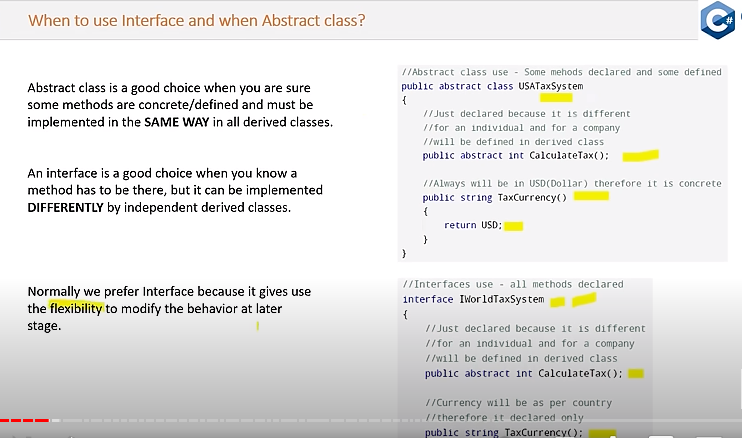
There are 2 types of polymorphism:

1. Compile time polymorphism also known as Overloading.
2. Run time polymorphism also known as Overriding.

**9.  What’s the difference between an abstract class and interface class?**

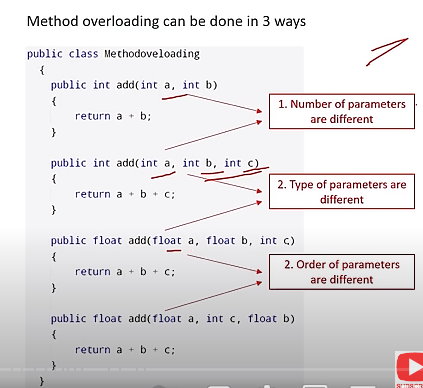
|  |  |
| --- | --- |
| **Abstract Class** | **Interface Class** |
| Abstract classes have method declaration and method definition | interfaces have method declaration only |
| Abstract class can have fields | Interface cannot have fields |
| Abstract classes support access specifiers | whereas interfaces don't support access specifiers. |
| Abstract class can have constructors | Interface class cannot have constructors |
| Abstract classes have normal variables and constant variables | interfaces do not have variables. |
| Abstract class cannot have multiple inheritance | Interfaces can have multiple inheritance |
|  |  |

Abstract class and Interface can only use for inheritance not for object creation.



**10. What is method Overloading?**

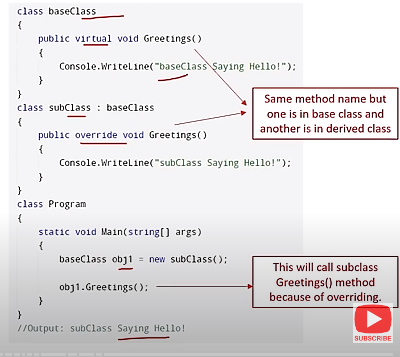
Creating multiple methods in a class with the same name but with different parameters and different types is called method overloading.



**11. What is method Overriding?**

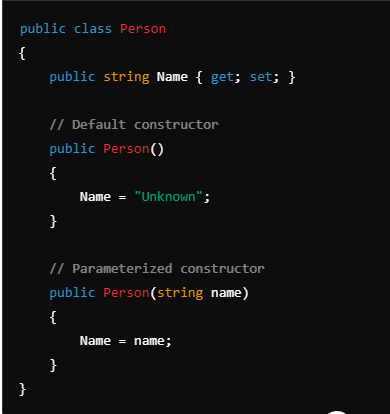
**Overriding** means to change the functionality of a method without changing the signature.

* You can override a method in base class by creating a similar method in derived class.
* This can be done by using virtual/override keywords.



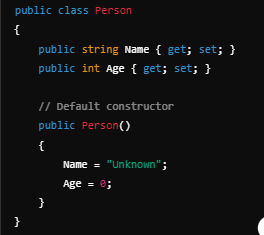
**12. Define Constructors**

* A constructor is a member function in a class that has the same name as its class.
* The constructor is automatically invoked whenever an object class is created.
* It constructs the values of data members while initializing the class.



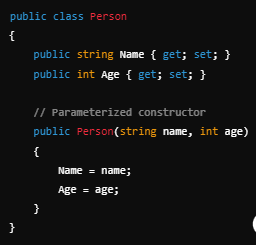
* 1. **Default Constructor** A constructor that takes no parameters. If no constructor is defined, C# automatically provides a default constructor.

**Usage**: Initializes object members to default values.



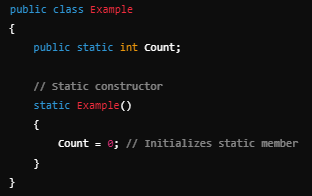
* 1. **Parameterized Constructor** A constructor that takes one or more parameters. This allows you to initialize object members with specific values at the time of creation.

**Usage**: Enables flexible object creation with custom values.



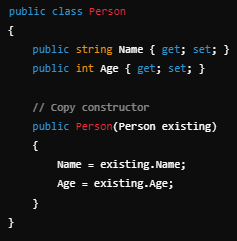
* 1. **Static Constructor** A constructor that initializes static members of a class. It is called automatically before any static members are accessed or any instance of the class is created.

**Usage**: To set up static data or perform actions that need to happen only once.



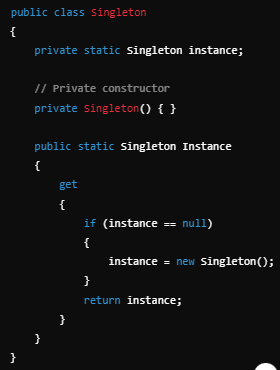
* 1. **Copy Constructor** A constructor that creates a new object as a copy of an existing object. C# does not provide a built-in copy constructor, but you can define one manually.

**Usage**: Useful for creating a new instance that is a copy of an existing instance.



* 1. **Private Constructor** A constructor that is declared as private, preventing the class from being instantiated from outside. Often used in singleton patterns or static classes.

**Usage**: To restrict instantiation and control access to class instances.



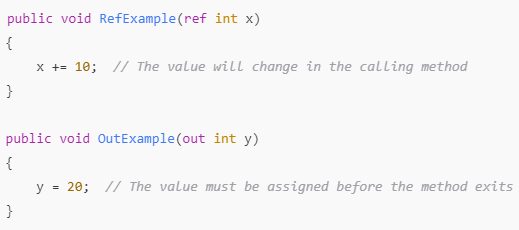
**13. What is Jagged Arrays?**

The Array which has **elements of type array** is called jagged Array. The elements can be of different dimensions and sizes. We can also call jagged Array as an Array of arrays.

EX: int[][] jaggedarr = new int[4][]

**14. What is the difference between ref & out parameters?**

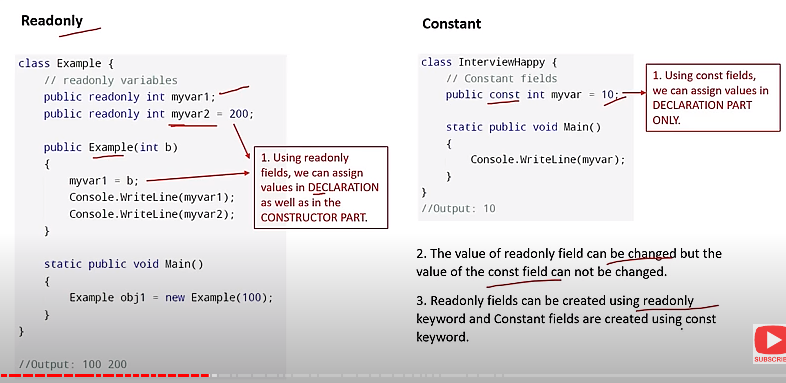
An argument passed as **ref** must be initialized before passing to the method whereas **out** parameter needs not to be initialized before passing to a method.



**15. What is the difference between constants and read-only?**

**Constant** variables are declared and initialized at compile time. The value can’t be changed afterward.

**Read-only** is used only when we want to assign the value at run time.

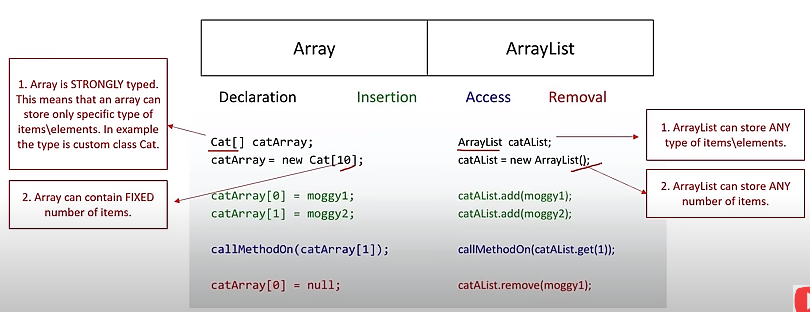


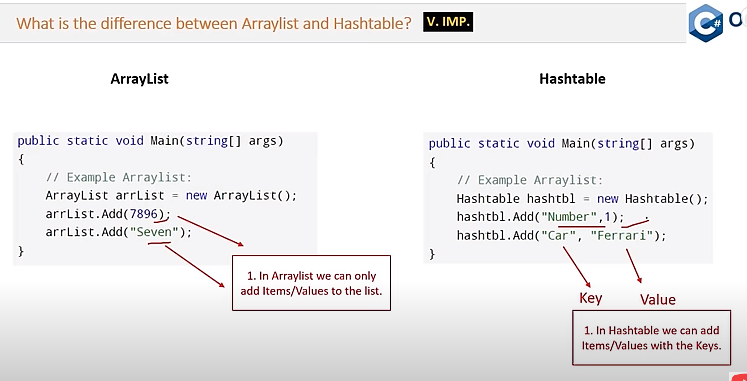
**15. What are sealed classes in C#?**

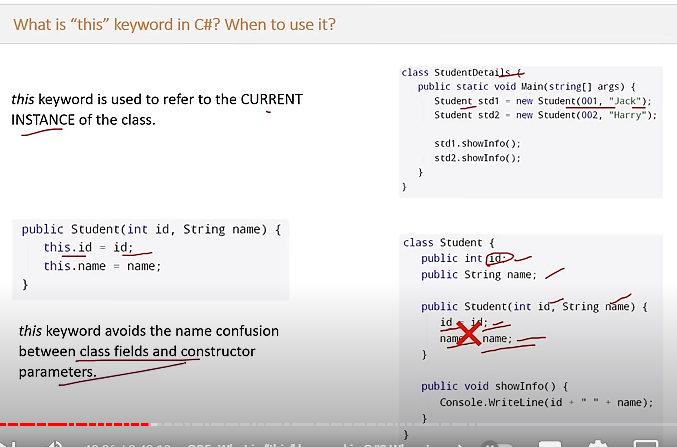
We create sealed classes when we want to restrict the class to be inherited. Sealed modifier used to prevent derivation from a class. If we forcefully specify a sealed class as base class, then a compile-time error occurs.

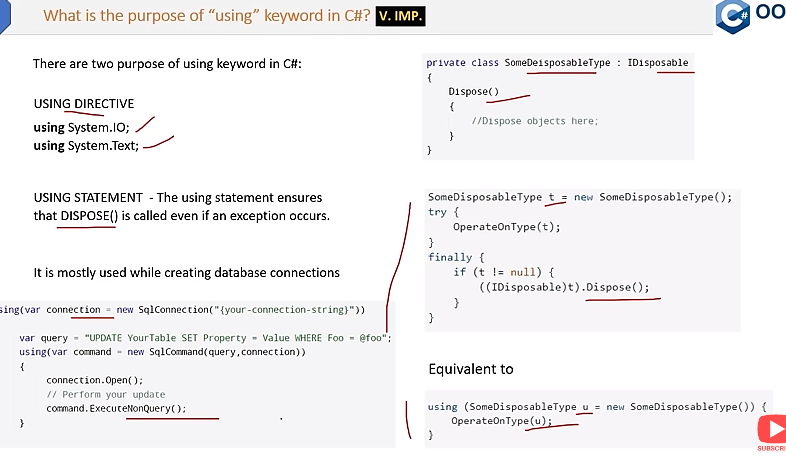
**16. What is the difference between Array and Arraylist?**

**In an array**, we can have items of the same type only. The size of the array is fixed when compared. To **an arraylist** is like an array, but it doesn’t have a fixed size.







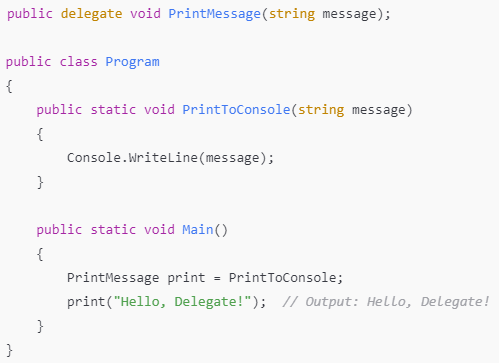


**17.  What are delegates?**

Delegates are same are function pointers in C++, but the only difference is that they are type safe, unlike function pointers. Delegates are required because they can be used to write much more generic type-safe functions.

* Delegates allow methods to be passed as parameters.
* Delegates can be used to define callback methods.

EX : public delegate int PerformCalculation (int x, int y);



* Method Indirection
* Event Handling
* Callback Mechanisms
* Asynchronous Programming
* Decoupling Components

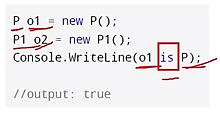
**18. What is the difference between a Class and a Struct?**

**Structs** are value-type variables, and **classes** are reference types. Structs stored on the Stack causes additional overhead but faster retrieval. Structs cannot be inherited.

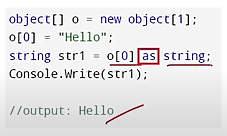
|  |  |
| --- | --- |
| **Class** | **Struct** |
| **Classes** are reference types | **Structs** are value-type variables |
| Class can be inherited | Structs cannot be inherited |
| Supports polymorphism and can implement interfaces | No support for polymorphism |
| Can have parameter less constructor and constructors with parameters | Can have parameter less constructor |
| Suitable for complex objects, business logic | Suitable for small data structures |

**19. What is difference between “is” and “as” operators in c#?**

**“is” operator** is used to check the compatibility of an object with a given type, and it returns the result as Boolean.



**“as” operator** is used for casting of an object to a type or a class.



**20. Is C# code is managed or unmanaged code?**

C# is managed code because Common language runtime can compile C# code to Intermediate language.

**21. What is meant by a Partial Class?**

A [partial class](https://www.simplilearn.com/tutorials/c-sharp-tutorial/partial-class-in-c-sharp) effectively breaks a class's definition into various classes in the same or other source code files.

**22. What are reference types and value types?**

A value type holds a data value inside its memory space. Reference type, on the other hand, keeps the object’s address where the value is stored. It is, essentially, a pointer to a different memory location.

**23. What are generics in C# .NET?**

**Generics** in C# allow you to define classes, methods, interfaces, and delegates with placeholders for the types they operate on.

To reduce code redundancy, raise type safety, and performance, generics can be used to make code classes that can be reused. Collection classes can be created using generics.

**24. What is an object pool in .NET?**

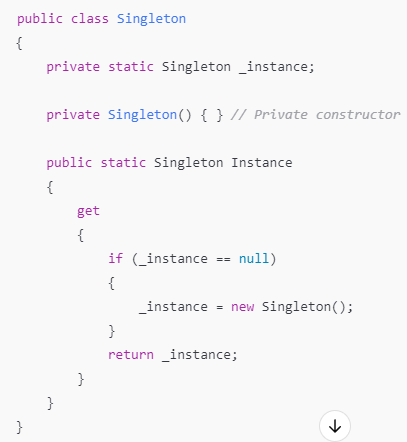
A container that has objects which are ready to be used is known as an object pool. It helps in tracking the object which is currently in use and the total number of objects present in the pool. This brings down the need for creating and re-creating objects.

**25. What is Singleton design pattern in C#?**

Singleton design pattern in C# has just one instance that gives global access to it.

This is particularly useful when only one instance of a class is needed to control access to shared resources such as configuration settings, logging, or database connections.

Factory Pattern, Observer Pattern, Strategy Pattern, Builder Pattern, Adapter Pattern

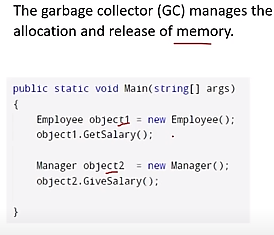


**26. What is Garbage Collection?**

Garbage collection (GC) is an automatic memory management feature that reclaims memory occupied by objects that are no longer in use, preventing memory leaks and improving application performance. The .NET runtime periodically identifies and collects unused objects.

Automatic Memory Management

* + Identifies Unused Objects
  + Prevents Memory Leaks
  + Efficient Memory Usage
  + No Manual Cleanup



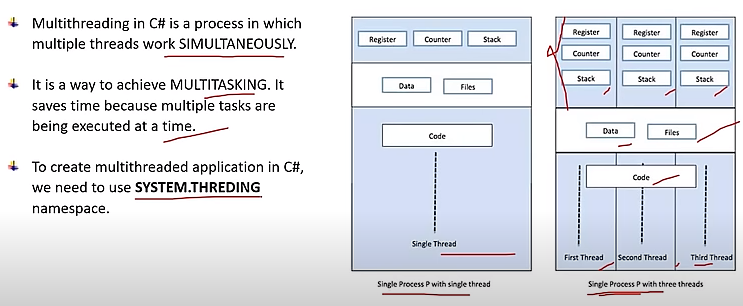
**27. What is boxing and unboxing in C#? How can you avoid performance issues related to boxing?**

Boxing is the process of converting a value type to an object type, and unboxing is the reverse process.

|  |  |
| --- | --- |
| int i = 123;  // The following line boxes i.  object o = i; boxing | o = 123;  i = (int)o; // unboxing |

### 28. What is Multithreading with .NET?

Multi-threading refers to the use of multiple threads within a single process. Each thread here performs a different function.



**29. What are the differences between IEnumerable and IQueryable?**

***IEnumerable:***

* Suitable for in-memory collections like lists and arrays.
* Executes queries locally in memory.
* Pulls all data into memory before filtering, sorting, or projecting.
* Limited support for deferred execution.
* Suitable for LINQ to Objects.

**IQueryable:**

* Designed for querying data stores like databases.
* Executes queries on the data source, minimizing data transfer.
* Supports deferred execution with expressions.
* Suitable for LINQ to Entities (Entity Framework) and other data access technologies.

**30. What are the SOLID Principles?**

SOLID is an five design principles that help developers create more understandable, flexible, and maintainable software:

* S: Single Responsibility Principle (SRP)
* O: Open/Closed Principle (OCP)
* L: Liskov Substitution Principle (LSP)
* I: Interface Segregation Principle (ISP)
* D: Dependency Inversion Principle (DIP)

Single Responsibility Principle (SRP)

A class should have one, and only one, reason to change. This means that a class should only have one responsibility or job.

Open/Closed Principle (OCP)

Software entities (classes, modules, functions, etc.) should be open for extension but closed for modification. This means you should be able to add new functionality without changing existing code.

Liskov Substitution Principle (LSP)

Objects of a superclass should be replaceable with objects of a subclass without affecting the correctness of the program. Essentially, subclasses should behave as expected when used in place of their parent classes.

Interface Segregation Principle (ISP)

No client should be forced to depend on methods it does not use. This means that interfaces should be specific to their clients rather than general-purpose.

Dependency Inversion Principle (DIP)

High-level modules should not depend on low-level modules. Both should depend on abstractions. Additionally, abstractions should not depend on details; details should depend on abstractions.

### 31. What is the difference between Finalize () and Dispose () methods?

**Dispose ()** is called when we want for an object to release any unmanaged resources with them.

**Finalize ()** is used for the same purpose, but it doesn’t assure the garbage collection of an object.

