Part01

Why can't a struct inherit from another struct or class in C#?

Structs are value types and have a specific memory layout. Allowing inheritance would introduce complexity in maintaining this layout.

Structs are intended to be lightweight and efficient. Allowing inheritance would introduce overhead.

How do access modifiers impact the scope and visibility of a class member?

• Private: Encapsulates members within the declaring class.

Scope: The member is accessible only within the class or struct in which it is declared.

Visibility: Other classes or structs, even those in the same namespace, cannot access private members.

• Protected: Allows access within the declaring class and its derived classes.

Scope: The member is accessible within its class and by derived class instances.

Visibility: Other classes or structs that do not inherit from the class cannot access protected members.

• Internal: Limits access to the same assembly.

Scope: The member is accessible within the same assembly but not from another assembly.

Visibility: Classes or structs outside the assembly cannot access internal members.

• Protected **Internal**: Combines protected and internal access.

Scope: The member is accessible within its class, derived classes, and any classes within the same assembly.

Visibility: Combines the features of protected and internal access modifiers.

• Public: Allows global access.

Scope: The member is accessible from any other class or struct.

Visibility: There are no restrictions; the member is accessible globally.

Why is encapsulation critical in software design?

for data protection, maintainability, flexibility, improved code quality, and overall system robustness. It helps in creating a modular, secure, and adaptable codebase that is easier to manage and evolve over time.

Data Hiding: The balance field is private, preventing direct access from outside the class.

Controlled Access: The Deposit and Withdraw methods provide controlled access to modify the balance.

Reusability: The [BankAccount] class can be reused in various financial applications without modification.

what is constructors in structs?

Constructors are special methods used to initialize objects. They are called when an instance of a class or struct is created.

Constructors have the same name as the class or struct and do not have a return type (not even void).

In Struct:

- Structs in C# cannot define parameterless (default) constructors.
- Structs can define constructors with parameters to provide custom initialization for its fields.

How does overriding methods like ToString() improve code readability?

• Meaningful Output: Provides a clear, descriptive representation of an object's state.

• Easier Debugging: Simplifies the process of understanding object contents during debugging.

• Consistent Format: Ensures a standardized way of displaying object information.

•Simplified Changes: Allows easy updates to object representation by modifying `ToString()` alone.

How does memory allocation differ for structs and classes in C#?

• **Structs (Value type)**: Stack-allocated, direct storage, copied by value, no garbage collection.

• **Classes(Reference type)**:: Heap-allocated, reference storage, copied by reference, managed by garbage collection.

Part02

What is copy constructor?

A copy constructor is a special type of constructor used to create a new object as a copy of an existing object. This constructor copies the values from the existing object to the new object, ensuring that each object has its own copy of the data.

Ex:

```
// Copy constructor

public MyClass(MyClass existingObject) {

 Value = existingObject.Value;

}
```

What is Indexer, when used, as business mention cases u have to utilize it?

An indexer in C# is a special type of property that allows an object to be indexed in the same way as an array. It enables instances of a class or struct to be accessed using array-like syntax, providing a more intuitive way to access and manipulate collections of data encapsulated within an object.

Used in :

• Custom Collections: When creating custom collection classes that need to provide easy, array-like access to their elements.

• Data Structures: For data structures where access to elements by index is a natural fit (e.g., matrices, lists).

• Encapsulation: To encapsulate internal data storage and provide a controlled interface for accessing and modifying elements.

1-Inventory Systems:

- Description: In an inventory management system, you might have a ProductCollection class that holds a collection of products.
- **Usage**: Using an indexer allows accessing products by their index in a natural way, enhancing code readability.

2- Lookup Tables:

- Description: In scenarios where you have lookup tables (e.g., for tax rates, currency conversion), an indexer can simplify accessing values.
- Usage: Implementing an indexer in a LookupTable class allows easy retrieval of values using a key.

Summarize keywords we have learnt last lecture

• Private: Encapsulates members within the declaring class.

• Protected: Allows access within the declaring class and its derived classes.

• Internal: Limits access to the same assembly.

Protected Internal**:** Combines protected and internal access.

• Public: Allows global access.

• this: Refers to the current instance of the class or struct.