**Part02**

**What is a Copy Constructor?**

A **copy constructor** is a special constructor in a class (or struct) that initializes a new object as a copy of an existing object of the same type. It is used to create a new object by copying the values from another object.

In C#, if you don't explicitly define a copy constructor, the compiler provides a default one that performs a **shallow copy** (i.e., copying the references for reference types). However, you can define a custom copy constructor to control how the copying process should occur, especially if your class contains complex types or resources like arrays or objects.

**Key Points:**

* **Purpose**: The copy constructor allows you to create a new object by copying the values of an existing object.
* **Syntax**: It usually takes a single parameter of the same type as the class or struct.
* **When Used**: It’s typically used when you want to duplicate an object, especially for deep copying or when objects contain reference types.

**Why is it Important?**

* **Shallow Copy vs. Deep Copy**: By defining a copy constructor, you can control whether the object is copied by reference (shallow copy) or by creating a new instance for any referenced objects (deep copy).
* **Control**: It provides the ability to handle resources or complex members in a specific way when an object is copied.

In summary, a **copy constructor** is a powerful tool for object duplication, providing flexibility in handling complex objects or resource management.

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

An **Indexer** in C# allows an object to be accessed like an array using an index. It is useful when a class needs to represent a collection of data and provides a simple way to access or modify its elements through array-like indexing.

**When to Use an Indexer:**

1. **Collections or Arrays**: When you want to access elements in a collection (e.g., a list or array) using an index.
   * Example: A shopping cart where products are accessed by index.
2. **Encapsulating Data Access**: To control how data is accessed or modified.
   * Example: A grade book system where grades are updated using an index.
3. **Custom Data Structures**: For multi-dimensional arrays or matrices where elements are accessed using multiple indices.
   * Example: A matrix in a data analysis tool.
4. **Dictionary-like Behavior**: For collections where keys are used to access values, similar to a dictionary.
   * Example: Employee directory where employees are accessed by their unique ID.

**-Summarize keywords we have learnt last lecture**

1. **Access Modifiers**: These control the accessibility of classes, methods, properties, and fields within a program. The keywords include:
   * **private**: Limits access to the containing class or struct.
   * **protected**: Allows access within the containing class or struct, and by derived classes.
   * **internal**: Makes members accessible within the same assembly (project).
   * **public**: Makes members accessible from any other class or assembly.
   * **private protected**: Allows access to derived classes within the same assembly.
   * **internal protected**: Allows access to derived classes within the same assembly or from other assemblies.
2. **struct**: Defines a value type that can contain fields, properties, and methods. Structs cannot inherit from other types but can implement interfaces.
3. **class**: Defines a reference type that can have fields, properties, methods, and can support inheritance.
4. **Constructor**: A special method that initializes objects. A struct has a parameterized constructor, while a class has default and parameterized constructors.
5. **this**: Refers to the current instance of a class or struct, typically used in constructors or methods to distinguish between the instance variables and parameters.
6. **override**: Used to modify the behavior of a base class method. For example, the ToString() method can be overridden to return a custom string representation of the object.
7. **new**: Used to create an instance of a class or struct. Also used to hide base class members in derived classes.