Session 3 → Task 1:

Source Code: https://github.com/Y-Baker/.NET_InnoTech/tree/main/Session%233

1- **Abstract Class:**
indicate that a class is intended only to be a base class of other classes meaning you can't create an instance from this class.
Members marked as abstract must be implemented by non-abstract classes that derive from the abstract class.

Suppose you have a base class name vehicle and three classes inherit from vehicle (Car, Boat, Bicycle) and you don't want to allow create a vehicle object.

```
1. using System;
2.
3. namespace MyFirstProgram;
4. class Program
5. {
6.     static void Main(string[] args)
7.     {
8.         Car car = new Car();
9.         Bicycle bicycle = new Bicycle();
10.        Boat boat = new Boat();
11.
12.        // Vehicle vehicle = new Vehicle();  // Error: Cannot create an instance of the abstract type or interface 'Vehicle'
13.    }
14. }
15. abstract class Vehicle
16. {
17.     public int speed = 0;
18.
19.     public void go()
20.     {
21.         Console.WriteLine("This vehicle is moving!");
22.     }
23. }
24. class Car : Vehicle
25. {
26.     public int wheels = 4;
27.     int maxSpeed = 500;
28. }
29. class Bicycle : Vehicle
30. {
31.     public int wheels = 2;
32.     int maxSpeed = 50;
33. }
34. class Boat : Vehicle
35. {
36.     public int wheels = 0;
37.     int maxSpeed = 100;
38. }
39.
```

2- **Sealed Class**:

A sealed class is a class that cannot be inherited from. It can be instantiated (opposite of abstract) but cannot be a base class.

used to prevent further inheritance and to provide a final implementation of a class.

suppose you end class implementation and don't need to create another class inherit from it

```
1. using System;
2.
3. namespace MyFirstProgram;
4. class Program
5. {
6.     static void Main(string[] args)
7.     {
8.         Car car = new Car();
9.         // Vehicle vehicle = new Vehicle(); // Error: Cannot create an instance of the abstract type or interface 'Vehicle'
10.     }
11. }
12. abstract class Vehicle
13. {
14.     public int speed = 0;
15.
16.     public void go()
17.     {
18.         Console.WriteLine("This vehicle is moving!");
19.     }
20. }
21. sealed class Car : Vehicle
22. {
23.     public int wheels = 4;
24.     int maxSpeed = 500;
25. }
26.
27. // class CarChild : Car  // 'CarChild': cannot derive from sealed type 'Car'
28. // {
29. //     public int wheels = 4;
30. //     int maxSpeed = 300;
31. // }
32.
```

Session 3 → Task 2:

Class is mutable by default

Mutable Class: those whose state can be modified after they are instantiated
for example array and list as u can add and remove elements after creation

```
public class MutablePerson
{
    public string Name { get; set; }
    public int Age { get; set; }
}
```

Immutable Class: those who's designed to be unchangeable once they are created.
for example strings so we use string Builder (mutable) before to prevent creating new string after any modification

```
public class ImmutablePerson
{
    public string Name { get; }
    public int Age { get; }

    public ImmutablePerson (string name, int age)
    {
        Name = name;
        Age = age;
    }
}
```

How to change between them:

By remove setter from all attributes in the class and initial all of them in constructor

Session 3 → Task 3:

Tuple: like list but immutable meaning you can't change its item after creation, and you can name it's item to easy usability

Declaring new tuple:

```
var tuple = ("Youse", "Computer Engineering");
Console.WriteLine($"Name: {tuple.name}, Department: {tuple.department}");
```

Declaring named tuple:

```
var tuple = (name: "Youse", department: "Computer Engineering");
Console.WriteLine($"Name: {tuple.name}, Department: {tuple.department}");
```

Unpacking tuple:

```
(string name, string department) = tuple;
Console.WriteLine($"Name: {name}, Department: {department}");
```

References:

- https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/sealed
- https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/abstract
- https://medium.com/@jepozdemir/understanding-mutable-and-immutable-types-in-c-2c609fd75a12