### **Important Microsoft notes: (Immutable By Design)**

If you don’t need it public, don’t have it public. Think *private* first.

**Idea for immutable:** Is to choose when you modify the state or value.

* **Race condition:** When multiple threads or processes access and try to manipulate shared data concurrently, the final outcome depends on the particular order in which their operations are executed.
* **Keywords for collection and classes:**
  + *Readonly -> For public immutable fields*
  + *Sealed or immutable -> All your classes/objects should have this as best practice.*
  + ***Avoid public*** *Methods & Setters*
    - Only make these public if necessary.
  + *Init -> When in properties this is preferred over set (setters).*
  + *Records*
  + *Auto Properties*
* **Difference between fields and properties:** 
  + Fields are like raw data storage within an object. You can directly access and modify them (if allowed).
  + Properties are like controlled access points to data. They use methods (getters/setters) to read or write data, often adding logic or validation in the process.
* **Properties Tips:**
  + If a setter is removed, the set is the same as a field readonly.
    - Public int Y { get; }
  + If you want to be able modify the properties internally only you can use the keyword *Private.*
    - Public int Y { get; private set; }
  + You can modify the properties when you inherit it.
    - Public int Y { get; protected set; }
* **Advantages of *init* instead of a parameter in a constructor:**
  + Constructor parameters are often used to initialize the state (fields or properties) of a new object upon its creation. With the *init keyword* you can decide whether to assign the parameter's value to an internal member during initialization.
* **Tips for making *List* immutable:**
  + Removing the setter is not the best way to make this list immutable for the public.
    - *List.Names.Add() = “Dylan”* or *List.names[2] = “Dylan”* you can directly modify it bypassing any prior configuration.
  + Having *Ireadonly* instead of *List* you can avoid the add method or directly modifying the index.
* **Tips for Records:**
  + They typically expose their data through **properties** and strongly encourage (or even enforce) **immutability**. This means that once a record instance is created, its state (the values of its properties) ideally shouldn't change.
  + To "change" a record, you create a new one with the updated values rather than altering the original. This preserves its immutability, often using concise syntax like "with-expressions."
    - In this new instance you can change the values using the *with* keyword.
  + Record comparison is by value.
  + You can use methods within the records.
  + You can use access modifiers the same as classes.
* **CAUTION:** Be aware that if a class (ChildClass) inherits from ParentClass, an instance of ChildClass can be supplied to a method expecting ParentClass without type compatibility problems. This is standard behavior.