**Task 2**

1. **Difference between Abstract Class & Sealed Class**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Abstract Class** | **Sealed Class** |
| Inheritance | Can be inherited by other classes | Cannot be inherited |
| Instantiation | Cannot be instantiated directly | Can be instantiated |
| Purpose | To provide a base class with common functionality that derived classes must implement | To prevent a class from being extended |
| Abstract Methods | Can contain abstract methods (methods without implementation) that must be implemented by derived classes | Cannot contain abstract methods |
| Use Case | Use when you have a base class with methods that must be implemented by subclasses | Use when you want to restrict inheritance for security, optimization, or design reasons |

1. **Difference between Struct & Class**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Struct** | **Class** |
| Type | Value type | Reference type |
| Memory Allocation | Allocated on the stack | Allocated on the heap |
| Inheritance | Cannot be inherited | Can be inherited |
| Default Constructor | Cannot have a default constructor | Can have a default constructor |
| Performance | Generally faster due to stack allocation | Slower due to heap allocation |
| Mutability | Typically immutable | Can be mutable or immutable |
| Use Case | Use for small data structures, typically containing only data | Use for complex objects requiring inheritance, polymorphism, etc. |

1. **Difference between Record & Class**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Record** | **Class** |
| Purpose | Primarily for storing immutable data | General-purpose type |
| Immutability | Typically immutable | Can be mutable or immutable |
| Syntax | Concise syntax for declaring data-centric types | Standard syntax |
| Equality | Value-based equality by default | Reference-based equality by default |
| Use Case | Use for data transfer objects, read-only data | Use for any type of object including those needing methods, behavior, and inheritance |

### 4. Mutable vs Immutable Classes in C#

#### **Mutable Classes**

In C#, a class is mutable by default. This means that the fields or properties of an instance of the class can be changed after the object is created.

#### **Immutable Classes**

An immutable class, on the other hand, does not allow changes to the fields or properties after the object is created. To make a class immutable, you can do the following:

1. Use readonly fields.
2. Make properties get only.
3. Ensure that the fields are set only once, typically through the constructor.

### 5. Tuples in C#

A tuple is a data structure that can hold a fixed number of items of different types. Tuples are often used to group multiple values together without creating a specific class to hold them.

You can create and use tuples in C# in the following ways:

// Creating a tuple

var tuple = (1, "Hello", 3.14);

// Accessing tuple elements

Console.WriteLine(tuple.Item1); // Output: 1

Console.WriteLine(tuple.Item2); // Output: Hello

Console.WriteLine(tuple.Item3); // Output: 3.14

// Deconstructing a tuple

(int number, string text, double value) = tuple;

Console.WriteLine(number); // Output: 1

Console.WriteLine(text); // Output: Hello

Console.WriteLine(value); // Output: 3.14