Object Oriented Programming – BSC20924
Group A

Object Oriented Programming Essay

Due: 23 October 2024 12:00

Marcos José Fernandes Pinheiro
Student nº 74676

Main Differences between Abstract classes and Interfaces in C#.

In C#, *abstract classes* and *interfaces* are used to define contracts for other classes to implement, but they also serve different purposes and have distinct characteristics. Understanding their differences can help in designing flexible, maintainable, and efficient code.

1. Definition and Purpose:

- *Abstract Class*: A base class that can include both abstract methods (without implementation) and concrete methods (with implementation). Used when classes share functionality but need some methods to be implemented by derived classes.

- *Interface*: A contract that defines methods and properties without any implementation. Classes that implement an interface must provide concrete implementations for all methods.

2. Key Differences:

- *Multiple Inheritance*: Interfaces support multiple inheritances, while abstract classes do not.

- *Implementation*: Abstract classes can provide default method implementations. Interfaces, starting from C# 8.0, can offer default implementations but cannot store state.

- *Fields*: Abstract classes can have fields (variables), while interfaces cannot. They can only define methods, properties, events, or indexers but no internal data storage.

- *Use Case*: Abstract classes are for closely related classes, share common functionality, and need to inherit some behaviour whereas interfaces offer flexibility for unrelated classes that need to follow the same contract.

3. Syntax Comparison:

- ***Abstract Class*** example:

```
public abstract class Animal

{

    public abstract void MakeSound();

    public void Eat()
```

```
        {
            Console.WriteLine("Eating...");
        }
    }
```

- ***Interface*** example:

```
    public interface IAnimal
    {
        void MakeSound();
        void Eat();
    }
```

4. When to Use:

- Use **Abstract Classes** for hierarchical relationships and shared implementation. Good for use when you need to provide base functionality along with the enforcement of certain behaviors.

- Use **Interfaces** for flexibility and when multiple unrelated classes need to follow the same contract. Better for decoupling code and making it more modular.

Conclusion

Summarizing, abstract classes and interfaces in C# provide powerful tools for designing flexible and reusable code. Abstract classes are more suitable for situations where shared functionality and inheritance are needed, while interfaces allow for defining strict contracts without implementation, supporting more flexible and loosely coupled designs. The choice between them depends largely on the structure and requirements of the code you are developing.

Reference

Microsoft Docs (2024) *Abstract Classes and Interfaces in C#*. Available at: https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/abstract-and-sealed-classes-and-class-members (Accessed: 21 October 2024).

Richter, J. (2012) *CLR via C#*. 4th edn. Redmond, Washington: Microsoft Press.