Interfaces in C#

Was ist ein Interface?

- Ein Interface ist eine Sammlung von Methoden.- und Eigenschaftendefinitionen, die von einer Klasse oder Struktur implementiert werden müssen.
- Interfaces definieren keine Implementierung der Methoden oder Eigenschaften, sondern nur ihre Signatur.
- **Warum Interfaces?**
  - Trennung von Schnittstellen und Implementierung
  - Erleichtert die Erstellung flexibler und erweiterbarer Software
  - Unterstützt die Verwendung von Abstraktionen in der Softwarearchitektur

Syntax eines Interfaces in C#

```
public interface IShape
{
    void Draw();
    double GetArea();
}
```

- Interface-Namen beginnen in der Regel mit einem "**I**".
- Alle Methoden im Interface sind implizit **public** und **abstract**.

Implementierung eines Interfaces

```
public class Circle : IShape
{
    public double Radius { get; set; }

    public void Draw()
    {
        Console.WriteLine("Drawing a Circle");
    }

    public double GetArea()
    {
        return Math.PI * Radius * Radius;
    }
}
```

- Die Klasse `Circle` implementiert das Interface `IShape` und definiert die Methoden `Draw` und `GetArea`.

Mehrfachvererbung durch Interfaces

- Klassen können in C# nur von einer Basisklasse erben, aber sie können mehrere Interfaces implementieren.
- Dies ermöglicht eine Art "Mehrfachvererbung", ohne die typischen Probleme, die bei Mehrfachvererbung in anderen Sprachen auftreten können.

Mehrfachvererbung durch Interfaces - Beispiel

```
public interface IShape
{
    void Draw();
}

public interface IColorable
{
    void SetColor(string color);
}

public class ColoredCircle : IShape, IColorable
{
    public void Draw()
    {
        Console.WriteLine("Drawing a Circle");
    }

    public void SetColor(string color)
    {
        Console.WriteLine($"Coloring Circle with {color}");
    }
}
```

Interfaces vs. Abstrakte Klassen

- **Abstrakte Klasse:**
  - Kann sowohl implementierte Methoden als auch abstrakte Methoden enthalten.
  - Unterstützt Felder und Konstruktoren.
  - Nur eine einzige abstrakte Klasse kann geerbt werden.
  - Weniger flexibel (es muss die **Is A** Beziehung erfüllt sein)
- **Interface:**
  - Enthält nur Methodensignaturen (bis auf Default-Implementierungen, ab C# 8.0).
  - Keine Felder, Konstruktoren oder Implementierungen (außer mit Default-Implementierungen).
  - Mehrere Interfaces können implementiert werden.

Interfaces vs. Abstrakte Klassen - Beispiel

```
public abstract class Animal
{
    public abstract void MakeSound();

    public void Sleep()
    {
        Console.WriteLine("Sleeping...");
    }
}

public interface IFlyable
{
    void Fly();
}

public class Bird : Animal, IFlyable
{
    public override void MakeSound()
    {
        Console.WriteLine("Bird chirping");
    }

    public void Fly()
    {
        Console.WriteLine("Flying high!");
    }
}
```

Interface-Vererbung

- Ein Interface kann auch von einem anderen Interface erben.
- Dies ermöglicht die Erstellung hierarchischer Interface-Strukturen.

Interface-Vererbung - Beispiel

```
public interface IShape
{
    void Draw();
}

public interface I3DShape : IShape
{
    double GetVolume();
}

public class Sphere : I3DShape
{
    public void Draw()
    {
        Console.WriteLine("Drawing a Sphere");
    }

    public double GetVolume()
    {
        return (4 / 3) * Math.PI * Math.Pow(5, 3);
    }
}
```

Default-Implementierung in Interfaces (C# 8.0)

- Seit C# 8.0 können Interfaces auch Standardimplementierungen für Methoden enthalten.
- Dies ermöglicht es, neue Methoden zu einem Interface hinzuzufügen, ohne dass bestehende Implementierungen brechen.

Default-Implementierung in Interfaces (C# 8.0) - Beispiel

```
public interface ILogger
{
    void Log(string message);

    // Default implementation
    void LogError(string error)
    {
        Log($"Error: {error}");
    }
}

public class ConsoleLogger : ILogger
{
    public void Log(string message)
    {
        Console.WriteLine(message);
    }
}
```

Interface als Rückgabewert und Parameter

- Interfaces können als Rückgabe- und Parameter-Typen verwendet werden, um Flexibilität zu schaffen.

Interface als Rückgabewert und Parameter - Beispiel

```
public interface IDatabase
{
    void Connect();
}

public class SqlDatabase : IDatabase
{
    public void Connect()
    {
        Console.WriteLine("Connecting to SQL Database");
    }
}

public class DatabaseManager
{
    public void InitializeDatabase(IDatabase database)
    {
        database.Connect();
    }
}
```

Beispiel für Dependency Injection mit Interfaces

- Interfaces spielen eine Schlüsselrolle in **Dependency Injection** und Inversion of Control (IoC).

```
public interface IService
{
    void Serve();
}

public class Service : IService
{
    public void Serve()
    {
        Console.WriteLine("Service Called");
    }
}

public class Client
{
    private readonly IService _service;

    public Client(IService service)
    {
        _service = service;
    }

    public void Start()
    {
        Console.WriteLine("Client started");
        _service.Serve();
    }
}
```

Fazit

- Interfaces sind ein mächtiges Werkzeug in C#, um Abstraktionen und lose Kopplung zu erreichen.
- Sie ermöglichen die Erstellung flexibler, erweiterbarer und wartbarer Software.
- Durch das Implementieren mehrerer Interfaces kann eine Klasse verschiedene Verantwortlichkeiten übernehmen.