Solid Principles Use Case

1. Single Responsibility Principle (SRP)

Definition: A class should have only one reason to change, meaning it should have only one job or responsibility.

Use Case in .NET:

- **Service Classes**: In a typical .NET application, service classes should focus on a single domain or functionality. For instance, a `` `UserService` `` should only handle user-related operations, while an `` `OrderService` `` should manage order-related tasks.

- **Example**:

```csharp
public class UserService
{
    private readonly IUserRepository _userRepository;
    public UserService(IUserRepository userRepository)
    {
        _userRepository = userRepository;
    }

    public void CreateUser(User user)
    {
        _userRepository.Add(user);
    }
}
```

2. Open/Closed Principle (OCP)

Definition: Software entities should be open for extension but closed for modification.

Use Case in .NET:

- **Extension Methods**: Use extension methods to add functionality to existing classes without modifying them.
- **Example**:

```csharp
public static class StringExtensions
{
    public static bool IsValidEmail(this string str)
    {
        return Regex.IsMatch(str, @"^[^@\s]+@[^@\s]+\.[^@\s]+$");
    }
}
```

Solid Principles Use Case

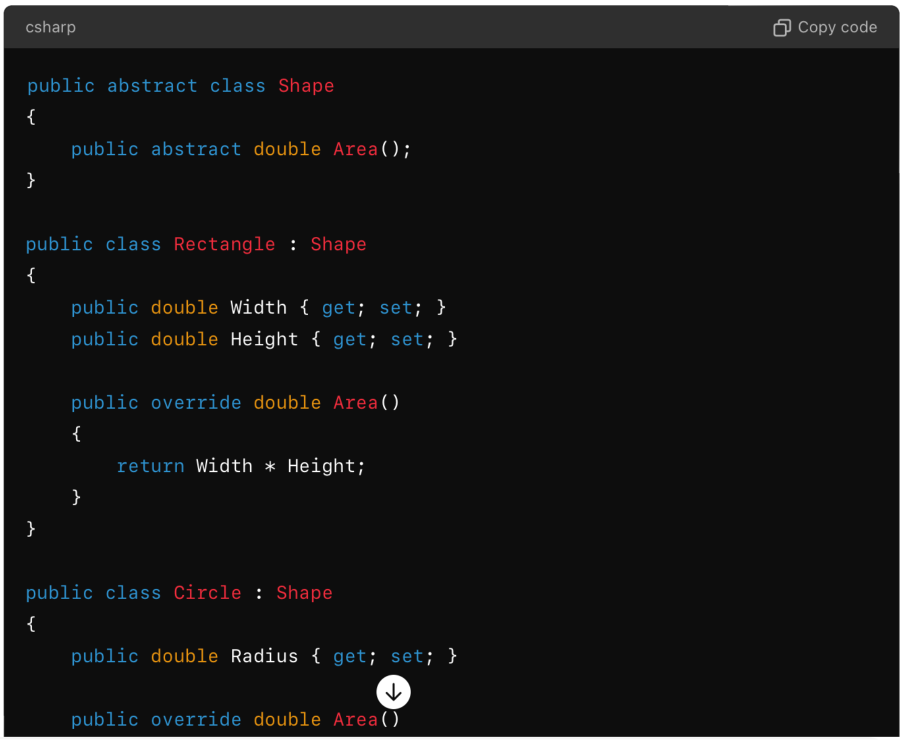
3. Liskov Substitution Principle (LSP)

Definition: Objects of a superclass should be replaceable with objects of a subclass without affecting the correctness of the program.

Use Case in .NET:

- **Inheritance Hierarchies**: Ensure that subclasses can be used interchangeably with their base classes without altering the expected behavior.

- **Example**:



```csharp
public abstract class Shape
{
    public abstract double Area();
}

public class Rectangle : Shape
{
    public double Width { get; set; }
    public double Height { get; set; }

    public override double Area()
    {
        return Width * Height;
    }
}

public class Circle : Shape
{
    public double Radius { get; set; }

    public override double Area()
```

4. Interface Segregation Principle (ISP)

Definition: Clients should not be forced to depend on interfaces they do not use.

Use Case in .NET:

- **Splitting Large Interfaces**: Split large interfaces into smaller, more specific ones so that implementing classes only need to worry about the methods that are relevant to them.

Solid Principles Use Case

- **Example**:

```csharp
public interface IPrinter
{
    void Print(Document doc);
}

public interface IScanner
{
    void Scan(Document doc);
}

public class MultiFunctionPrinter : IPrinter, IScanner
{
    public void Print(Document doc) { /*...*/ }
    public void Scan(Document doc) { /*...*/ }
}

public class SimplePrinter : IPrinter
{
    public void Print(Document doc) { /*...*/ }
}
```

5. Dependency Inversion Principle (DIP)

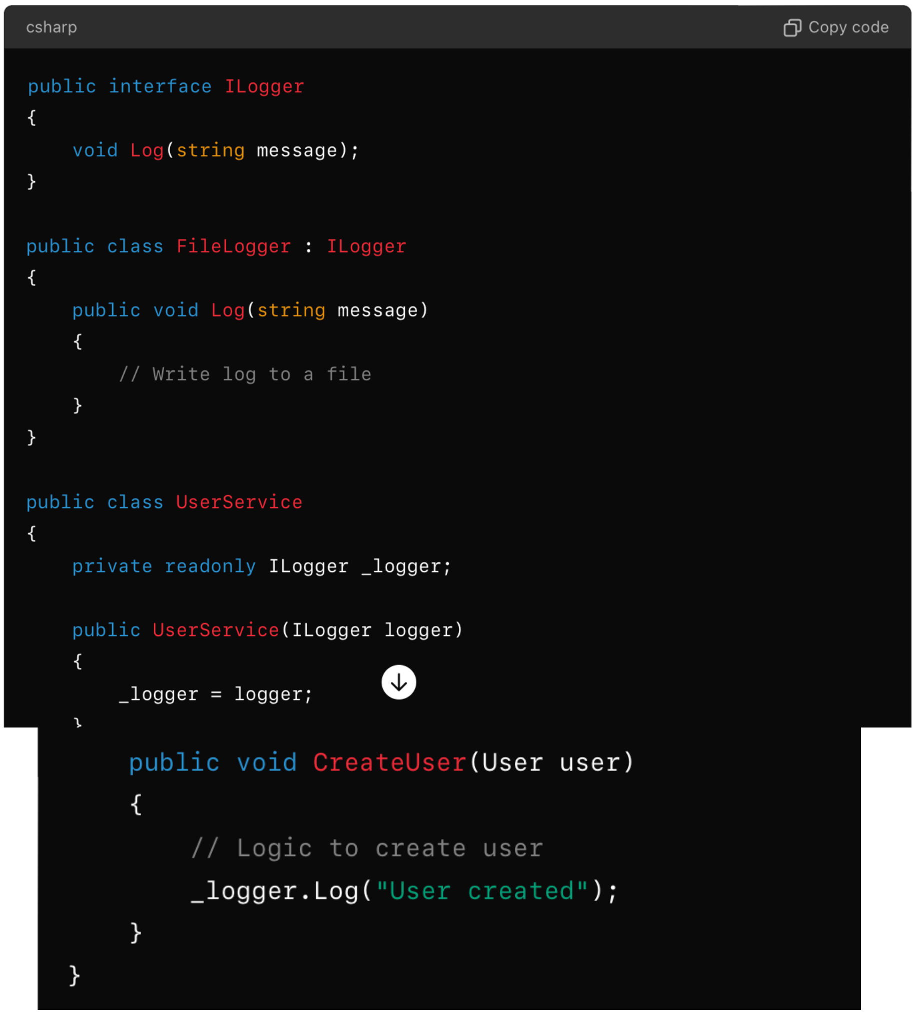
Definition: High-level modules should not depend on low-level modules. Both should depend on abstractions. Abstractions should not depend on details. Details should depend on abstractions.

Use Case in .NET:

- **Dependency Injection**: Use dependency injection to decouple classes from their dependencies.

Solid Principles Use Case

- **Example**:



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

public class FileLogger : ILogger
{
    public void Log(string message)
    {
        // Write log to a file
    }
}

public class UserService
{
    private readonly ILogger _logger;

    public UserService(ILogger logger)
    {
        _logger = logger;
    }
    public void CreateUser(User user)
    {
        // Logic to create user
        _logger.Log("User created");
    }
}
```