Object-Orientated Programming with C#

Chapter 4

https://csharp.christiannagel.com

Topics

- Inheritance
- Access Modifiers
- Inheritance with Records
- Interfaces
- Generics
- Constraints

OO Foundations

Inheritance

Encapsulation

Polymorphism

Inheritance

- Declare methods virtual
- Use override to override methods

```
public class Shape
{
  public void Draw() => DisplayShape();

  protected virtual void DisplayShape()
  {
    Console.WriteLine($"Shape with {Position} and {Size}");
  }
}
```

```
public class Rectangle : Shape
{
  protected override void DisplayShape()
  {
    Console.WriteLine($"Rectangle at position {Position} with size {Size}");
  }
}
```

Inheritance Keywords

Keyword	Functionality
virtual	Make method overridable
override	Override a virtual method
new	Hide a method from a base class
abstract	Declare variables, cannot create instances Need to derive from Method: no implementation, needs to be overridden
sealed	Cannot derive from the class Cannot override method
base	Keyword to invoke members of the base class

Access Modifiers

Access Modifier	Description
private	Access only within the type
public	Access from everywhere
protected	Access from within the type and from derived types
internal	Access from all types in the same assembly
internal protected	Access from all types in the same assembly, and from derived types in a different assembly
private protected	Access from derived types in the same assembly

Inheritance with Records

```
public record Position(int X, int Y);

public record Size(int Width, int Height);

public abstract record Shape(Position Position, Size Size)
{
  public void Draw() => DisplayShape();

  protected virtual void DisplayShape()
  {
    Console.WriteLine($"Shape with {Position} and {Size}");
  }
}
```

```
public record Rectangle(Position Position, Size Size) : Shape(Position, Size)
{
  protected override void DisplayShape()
  {
    Console.WriteLine($"Rectangle at position {Position} with size {Size}");
  }
}
```

Interfaces

- Contract: implementing interfaces, using interfaces
- Interfaces can’t contain state
- Interfaces can contain implementation (C# 8, default interface members)

Predefined Interfaces

- IDisposable
- IFromattable
- IEquatable<T>
- …

Explicitly and Implicitly Implemented Interfaces

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

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

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

Default Interface Members

- Avoid breaking changes

```
public interface ILogger
{
  void Log(string message);
  public void Log(Exception ex) => Log(ex.Message);
}
```

- Can also be used for traits

Generics

- Generic type, define type on usage

```
public record LinkedListNode<T>(T Value)
{
  public LinkedListNode<T>? Next { get; internal set; }
  public LinkedListNode<T>? Prev { get; internal set; }
  public override string? ToString() => Value?.ToString();
}
```

Constraints

Constraint	Description
where T : struct	T must be a value type
where T : class	T must be a reference type
where T : class?	T must be a nullable or non-nullable reference type
where T : notnull	T must be a non-nullable type
where T : unmanaged	T must be a unmanaged type
where T : IFoo	T is required to implement IFoo
where T : Foo	T is required to derive from Foo
where T : new()	T must have a parameterless constructor
where T1 : T2	T1 must derive from the generic type T2

Summary

- Inheritance
- Encapsulation
- Polymorphism
- Generics