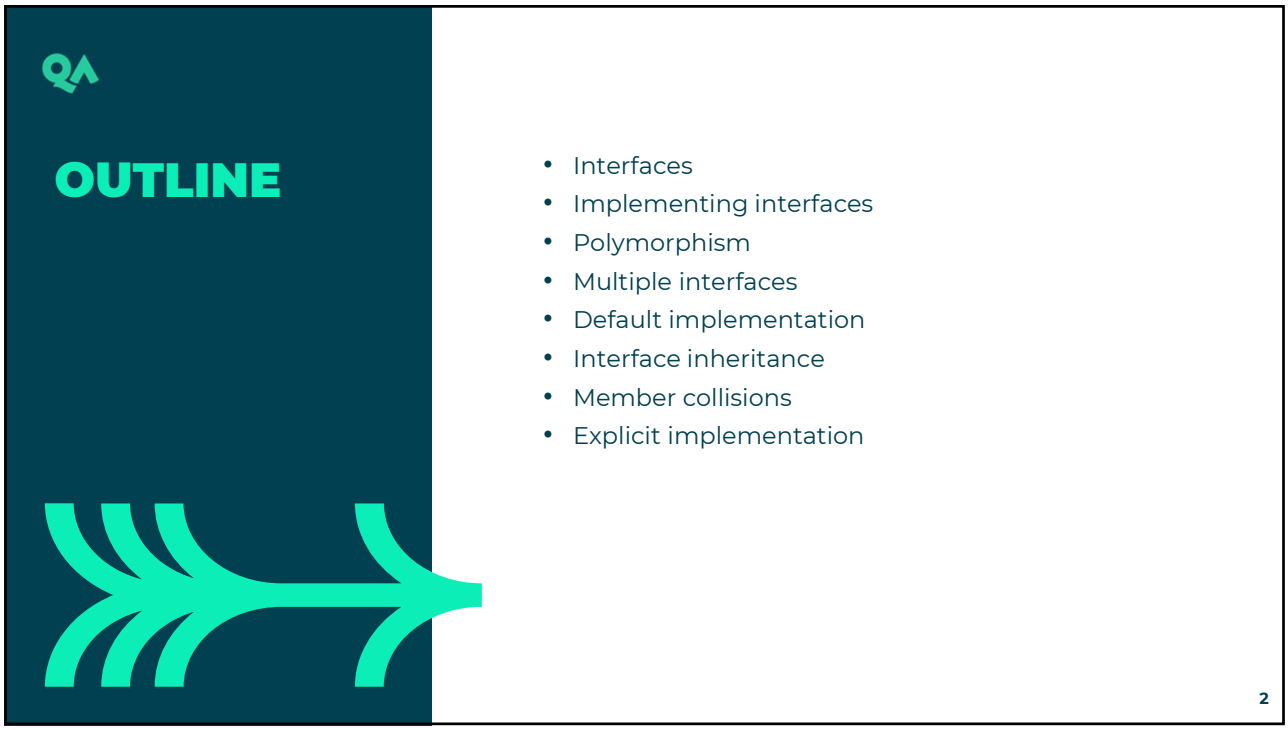

Interfaces

1


OUTLINE
Interfaces
Implementing interfaces
Polymorphism
Multiple interfaces
Default implementation
Interface inheritance
Member collisions
Explicit implementation
2

2

INTERFACES

- An interface contains definitions for a group of related functionalities
- A class or struct that implements the interface must provide the implementation code for all members
- From C# 8.0, an interface member can provide a default implementation for a member
- A class or struct can implement many interfaces, whereas a class can only inherit from one base class and structs cannot inherit from any base classes
- Interfaces conventionally are identified with a capital 'I', followed by a verb that describes the group of functionalities e.g., IComparable, IControllable, IDisposable,, and IPlayable

3

3

DEFINING AN INTERFACE

- Define an interface using the **interface** keyword
- Interfaces can contain *methods*, *properties*, *indexers*, and *events*
- Interface members are implicitly **public** and **abstract**

```
public interface IDrawable
{
    1 reference
    void Draw(Graphics g);// no implementation code
}
```

4

4

IMPLEMENTING AN INTERFACE

- List the interfaces after a colon and the base class (if also using inheritance)
- All non-default members must be implemented

```
public abstract class Shape
{
    1 reference
    public abstract double Area { get; }
}
```

```
public interface IDrawable
{
    1 reference
    void Draw(Graphics g);// no implementation code
}
```

```
public class Rectangle : Shape, IDrawable
{
    1 reference
    public int Width { get; set; }
    1 reference
    public int Height { get; set; }
    1 reference
    public override double Area => Width * Height;

    1 reference
    public void Draw(Graphics g)
    {
        // implementation code goes here;
    }
}
```

5

5

POLYMORPHISM

- An interface defines a collection of related functionalities
- A class or struct that implements an interface 'can do' those functions, i.e., they play that role
- An interface type can be used as a method parameter, return type, or as the type in a generic collection
- Any implementing class or struct can be used where the interface type is expected

6

6

POLYMORPHISM EXAMPLE

```
Graphics canvas = new();

void ProcessDrawable(IDrawable id)
{
    if (id is not null)
    {
        id.Draw(canvas);
    }
}

List<Shape> shapes = new() { new Rectangle(), new Rectangle() };
foreach (Shape shape in shapes)
{
    if (shape is IDrawable s)
    {
        s.Draw(canvas);
        //or:
        ProcessDrawable(shape as IDrawable);
    }
}
```

7

7

MULTIPLE INTERFACES

- A class or struct can implement multiple interfaces

```
public interface IComparable<T> {
  int CompareTo(T obj);
}
```

```
public interface IDrawable
{
    1 reference
    void Draw(Graphics g);// no implementation code
}
```

```
public class Rectangle : Shape, IDrawable, IComparable<Rectangle>
{
    3 references
    public int Width { get; set; }
    1 reference
    public int Height { get; set; }
    1 reference
    public override double Area => Width * Height;

    0 references
    public int CompareTo(Rectangle? other)
    {
        return Width - other.Width;
    }
    3 references
    public void Draw(Graphics g)
    {
        // implementation code goes here;
    }
}
```

8

8

DEFAULT IMPLEMENTATION

- From C# 8.0, interface members can provide a default implementation
- **IEmployee** provides *default implementation* for the **GetTaxAmount** method

```
public interface IEmployee
{
    1 reference
    public int ID
    {
        get;
        set;
    }
    1 reference
    public string Name
    {
        get;
        set;
    }
    3 references
    public double Salary
    {
        get;
        set;
    }
    0 references
    public double GetTaxAmount()
    {
        return Salary * 0.05;
    }
}
```

```
public class Employee : IEmployee
{
    0 references
    public Employee(double salary)
    {
        Salary = salary;
    }
    1 reference
    public int ID { get; set; }
    1 reference
    public string Name { get; set; }
    3 references
    public double Salary { get; set; }
}
```

```
IEmployee employeeA = new Employee(2500);
Console.WriteLine($"The Tax amount is {employeeA.GetTaxAmount()}");
Console.ReadKey();

Employee employeeB = new Employee(2500);
Console.WriteLine($"The Tax amount is {employeeB.GetTaxAmount()}");
Console.ReadKey();
```

CS1061 'Employee' does not contain a definition for 'GetTaxAmount' and no accessible extension method 'GetTaxAmount' accepting a first argument of type 'Employee' could be found (are you missing a using directive or an assembly reference?)

9

9

INTERFACES CAN INHERIT FROM INTERFACES

- **IFullySteerable** inherits **ISteerable**
- Any implementing class or struct must implement all the members of both **ISteerable** and **IFullySteerable**

```
public interface ISteerable
{
    1 reference
    void TurnLeft();
    1 reference
    void TurnRight();
}
```

```
public interface IFullySteerable : ISteerable
{
    1 reference
    void GoUp();
    1 reference
    void GoDown();
}
```

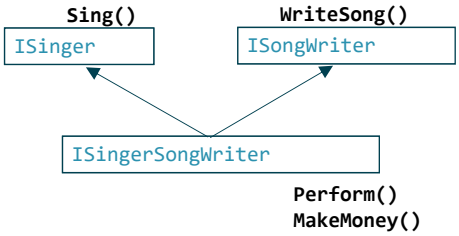
```
public class Drone : IFullySteerable
{
    1 reference
    public void GoDown()
    {
        // go down implementation;
    }
    1 reference
    public void GoUp()
    {
        // go up implementation;
    }
    1 reference
    public void TurnLeft()
    {
        // turn left implementation;
    }
    1 reference
    public void TurnRight()
    {
        // turn right implementation;
    }
}
```

10

10

MULTIPLE INTERFACE INHERITANCE

- **ISingerSongWriter** inherits both **ISinger** and **ISongWriter**
- Implementing classes and structs must implement:
  - Sing
  - WriteSong
  - Perform
  - MakeMoney



11

11

MULTIPLE INTERFACE MEMBER COLLISIONS

- Member collisions can arise when multiple interfaces use the same member name for semantically different functionality
- You can only implement one version of the member using *implicit* interface implementation

```
public interface ICowboy
{
    0 references
    void Draw(Graphics g);
}
```

```
public interface IDrawable
{
    4 references
    void Draw(Graphics g);
}
```

```
public class CowboyShape : ICowboy, IDrawable
{
    4 references
    public void Draw(Graphics g)
    {
        // only one implementation;
    }
}
```

12

12

EXPLICIT INTERFACE IMPLEMENTATION

- You can implement interface members *explicitly*, which includes the interface name as part of the member name

```
public class CowboyShape : ICowboy, IDrawable
{
    3 references
    public void Draw(Graphics g)
    {
        // implicit drawable implementation;
        Console.WriteLine("Drawing a cowboy shape");
    }
    1 reference
    void ICowboy.Draw(Graphics g)
    {
        // explicit cowboy implementation;
        Console.WriteLine("Reach for the sky, mister!");
    }
}
```

```
CowboyShape cs = new CowboyShape();
cs.Draw(canvas);// Drawing a cowboy shape

IDrawable id = cs;
id.Draw(canvas);// Drawing a cowboy shape

ICowboy ic = cs;
ic.Draw(canvas);// Reach for the sky, mister!
```

13

13

SUMMARY

- Interfaces
- Implementing interfaces
- Polymorphism
- Multiple interfaces
- Default implementation
- Interface inheritance
- Member collisions
- Explicit implementation

14

14

15