

Reusing Code through Interfaces



Gill Cleeren
CTO Xpirit Belgium
[@gillcleeren](https://twitter.com/gillcleeren)



Overview



Understanding interfaces

Implementing and using interfaces

Interfaces and polymorphism



Understanding Interfaces





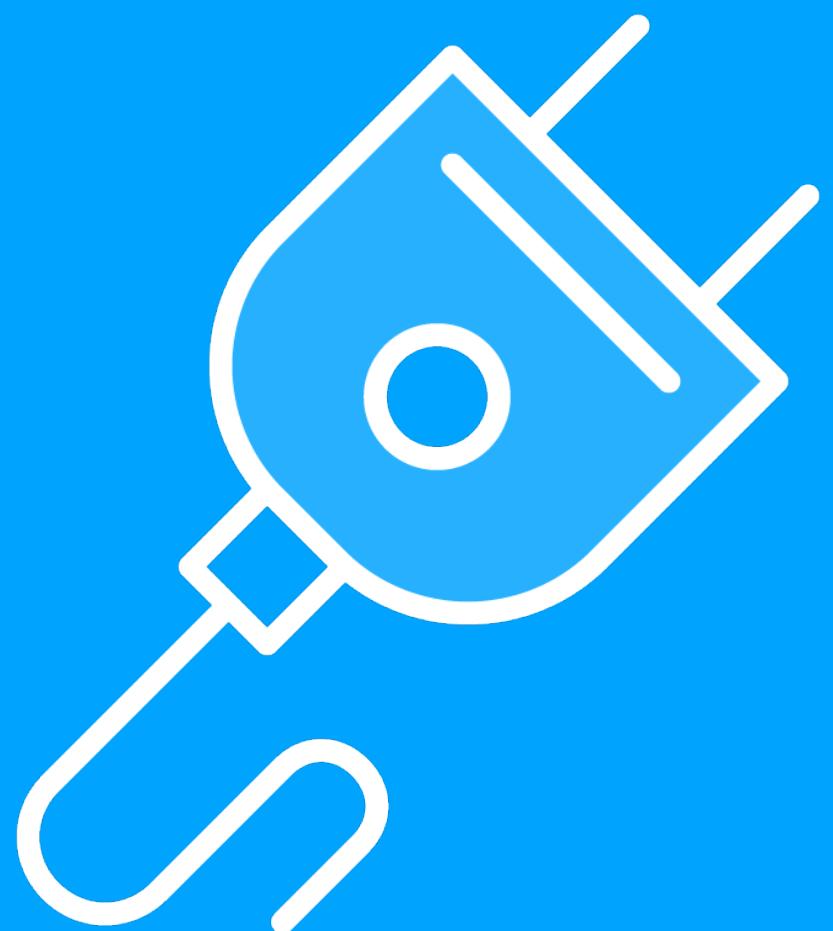
Remember abstract classes?

Can't be instantiated

Can contain abstract methods

**Guarantee to be implemented by
inheriting classes**





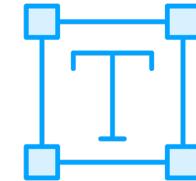
Introducing interfaces

Contract that must be implemented by classes use the interface

Contains definitions for set of related functionalities that classes must implement



Introducing Interfaces



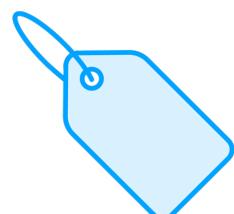
Use interface keyword



Typically contain no implementation code, though possible since C# 8



Can't be instantiated



Name typically starts with “I”



Members are public by default



```
internal interface ISaveable
{
    string ConvertToStringForSaving();
}
```

A First Interface

No implementation included for methods or properties



```
public class Order: ISaveable
{
    public string ConvertToStringForSaving()
    {
        return "Order";
    }
}
```

Implementing an Interface

By implementing the interface, we know (the contract) that the type has an implementation for the method(s)

Method must have same name and signature



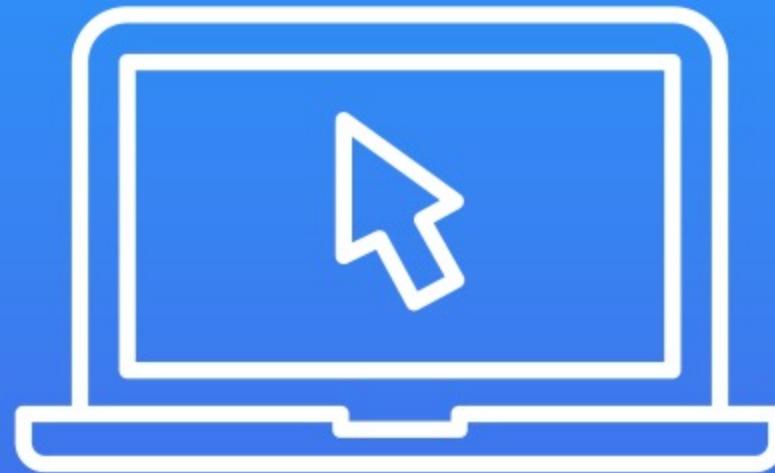
```
ISaveable s1 = new ISaveable(); //error
```

Instantiating an Interface

No implementation so no instantiation



Demo



Creating an interface
Implementing the interface



Implementing and Using Interfaces



Creating Interfaces

Methods

Properties

Events

Indexers



```
public interface ISaveable
{
    string Save();
    bool CanSave { get; set; }
    event EventHandler SaveableChanged;
}
```

A Real-world Interface





Implementing interfaces

- All members must be implemented
- One or more interface can be implemented
- Interfaces can inherit from other interfaces
 - All members from entire hierarchy must receive an implementation



Implementing Multiple Interfaces

```
public class Order: ISaveable, ILoggable
{
    public void Save()
    {
        ...
    }

    public void Log()
    {
        ...
    }
}
```



Implementing an Interface Explicitly

```
public class Order: ISaveable, ILoggable
{
    public void ISaveable.Save()
    {
        ...
    }

    public void ILoggable.Log()
    {
        ...
    }
}
```





Adding Implementations to an Interface

Possible but rarely used!



Demo



Implementing multiple interfaces



.NET Comes with Many Interfaces

IEnumerable

IDisposable

ICloneable

IComparable

IList/ICollection

ISerializable



The ICloneable Interface

```
namespace System
{
    //
    // Summary:
    //     Supports cloning, which creates a new instance of a class with the
    //     same value
    //     as an existing instance.
    public interface ICloneable
    {
        //
        // Summary:
        //     Creates a new object that is a copy of the current instance.
        //
        // Returns:
        //     A new object that is a copy of this instance.
        object Clone();
    }
}
```



```
public class Order: ISaveable, ICloneable
{
    public object Clone()
    {
        throw new NotImplementedException();
    }
}
```

Implementing ICloneable



Demo



Implementing ICloneable





Interfaces and Polymorphism



```
ISaveable s1 = new Product();
s1.Save();

s1.UseProduct(); //error since not defined on ISaveable interface
```

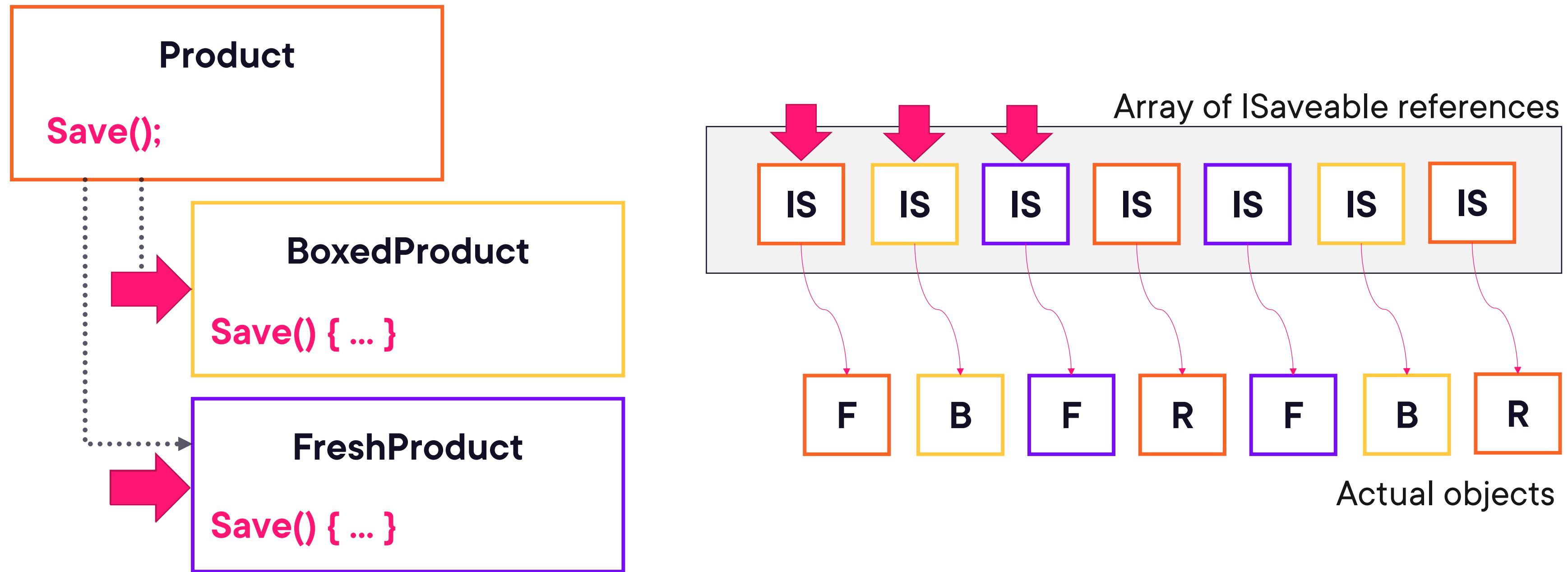
Using an Interface Type Reference



Using Polymorphism with Interfaces



Working with an Array of Interface Variables



Demo



Using polymorphism with interfaces



Summary



Interfaces help with abstraction

Define a contract

Can be used in polymorphic way

.NET comes with many interfaces built-in



**“Hey Bethany! The new application is ready.
Your staff can start using it!”**





“Oh, that is great!

**It works as expected, we won't run out of
stock anymore! No more days without pie!!”**





**Congratulations
on finishing this course!**

