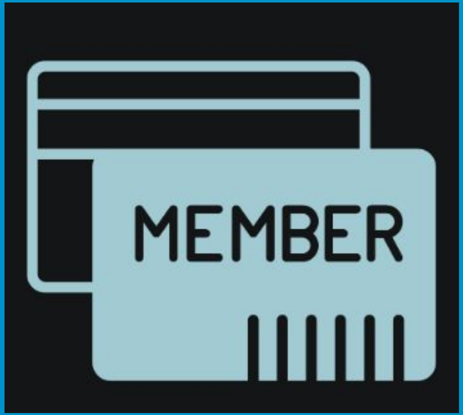
Interfaces

Completely Abstract

Abstract members only!



THE
END

JUST JOSHING, SQUID.

OOPs A PIE

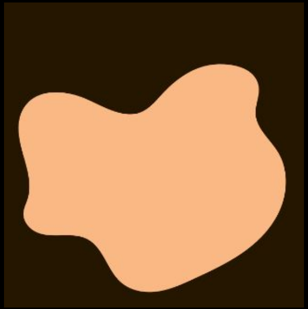
Stands for:

A - abstraction

P - polymorphism

I - inheritance

E - encapsulation



Another way to achieve Abstraction is with Interfaces.

C# Abstraction (w3schools.com)

Data abstraction is the process of hiding certain details and showing only essential information to the user.

Shopping without a list

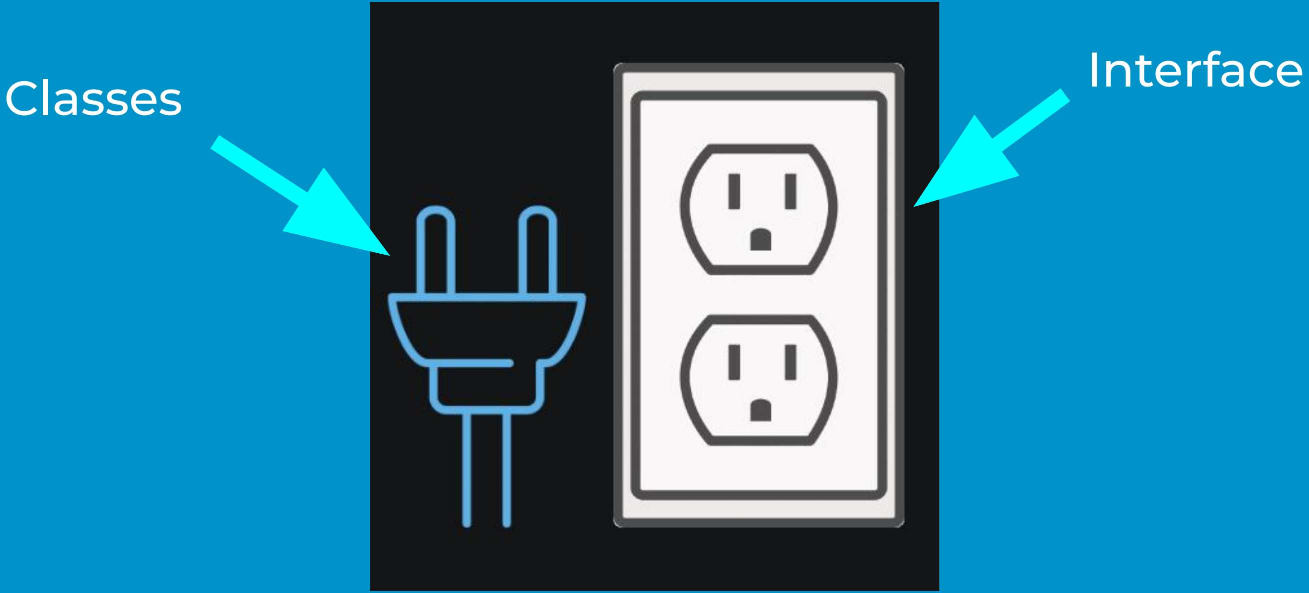
List ensures you don't forget anything

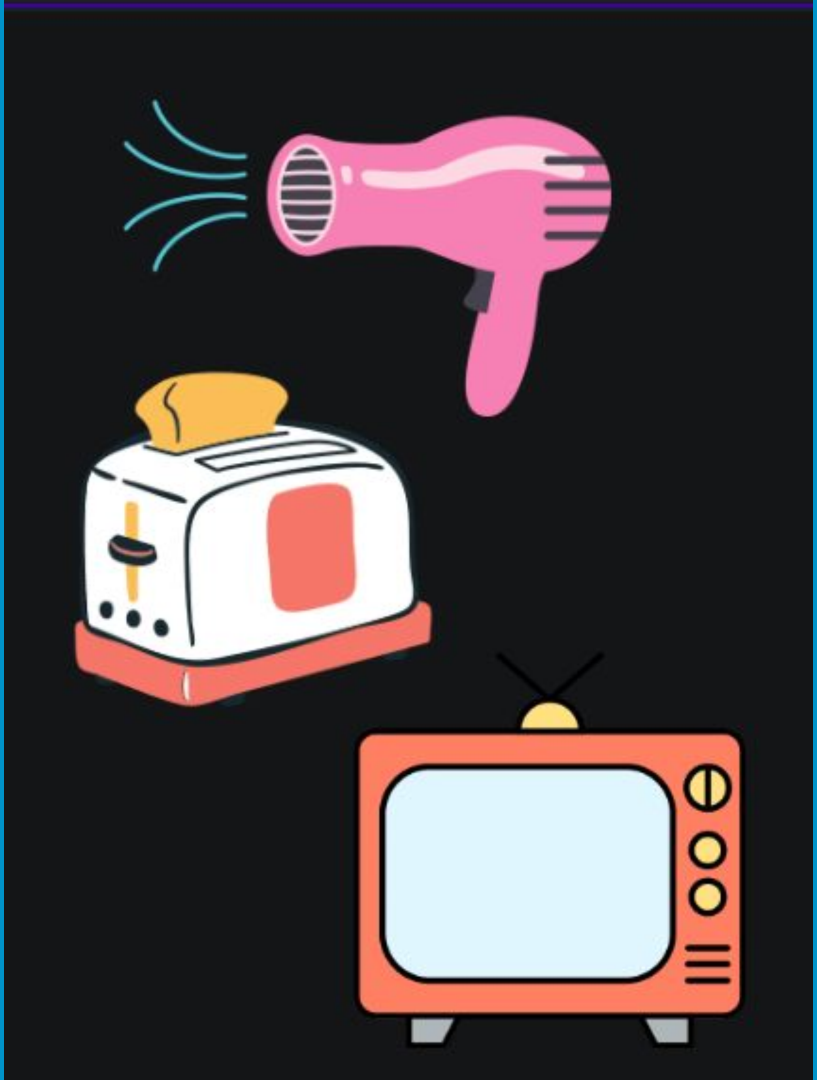
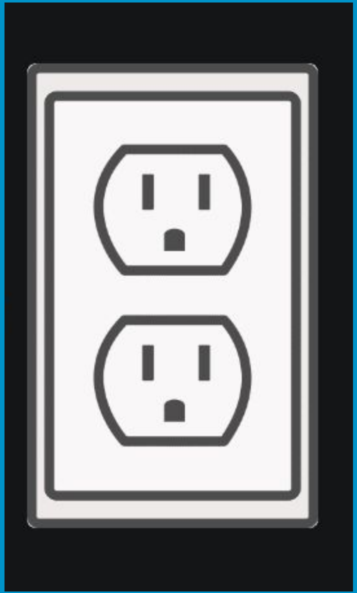
Interfaces act like a list for your code

Interfaces - Contract

Electrical Outlet

- The outlet (interface) promises that you'll receive electrical power if you plug into it.
- Various devices (implementing classes) can plug into the outlet. Each device might use the power in different ways, but all receive power.

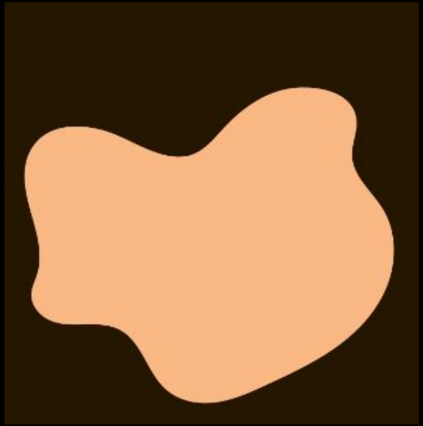


enforce a contract

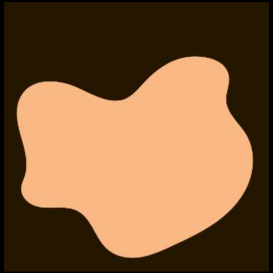
Bottom line:

Contract ➔

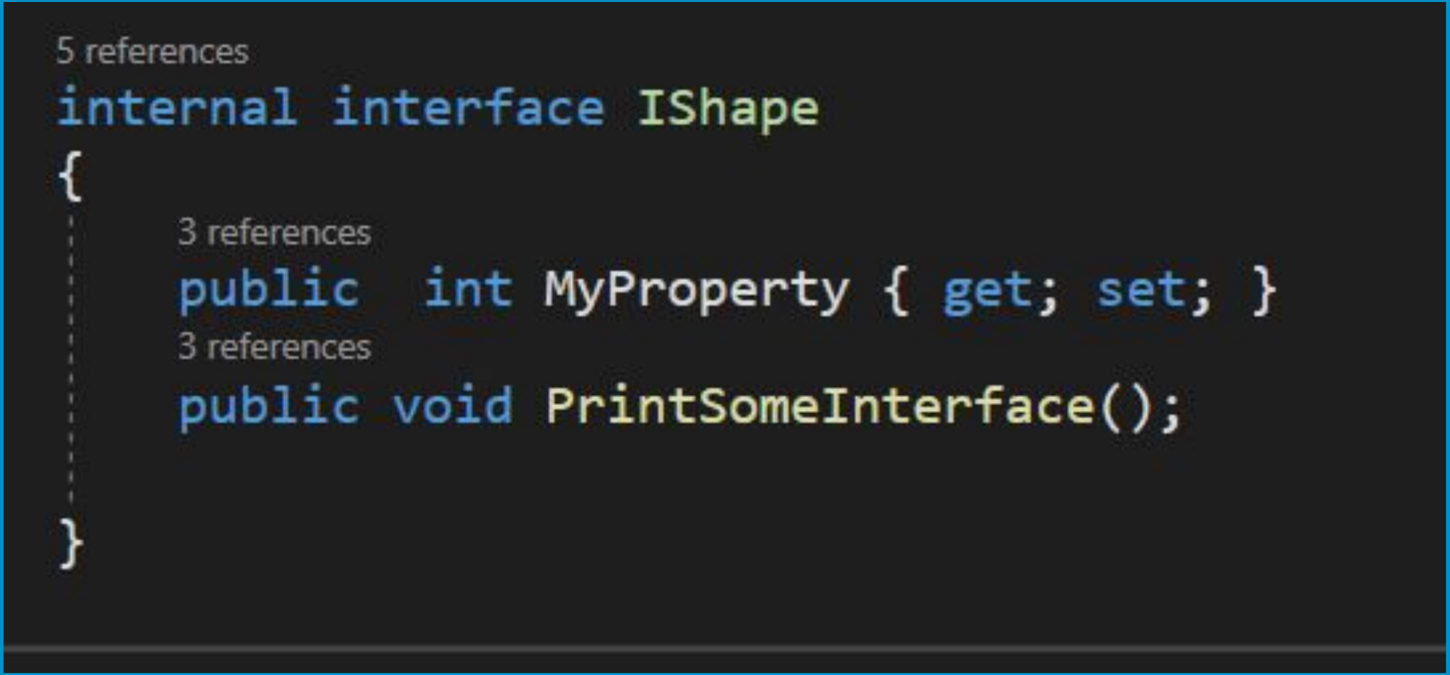


Completely Abstract

An interface is a completely abstract class which **contains only abstract members.** (think stubbed out methods)



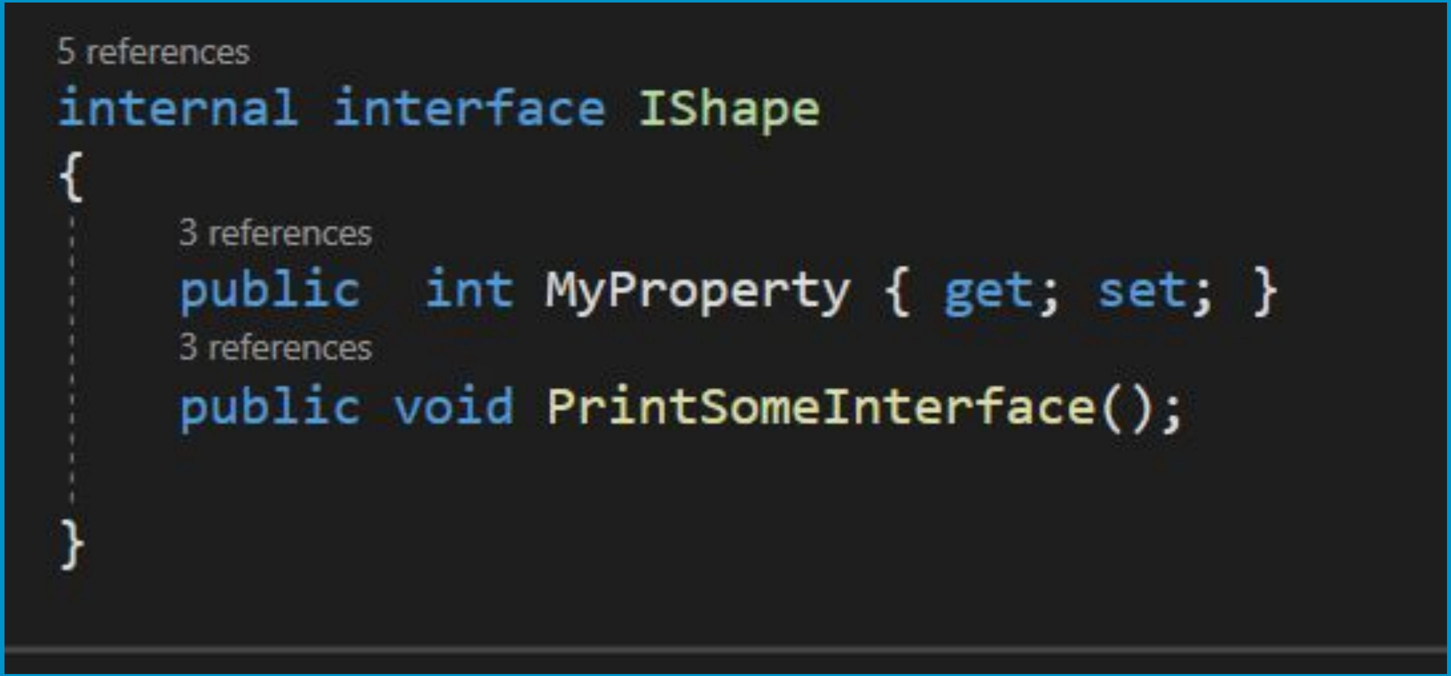
The interface itself provides no functionality.



```
5 references
internal interface IShape
{
        3 references
        public  int MyProperty { get; set; }
        3 references
        public void PrintSomeInterface();

}
```

** everything will be public in an interface

You do not need to use the override keyword

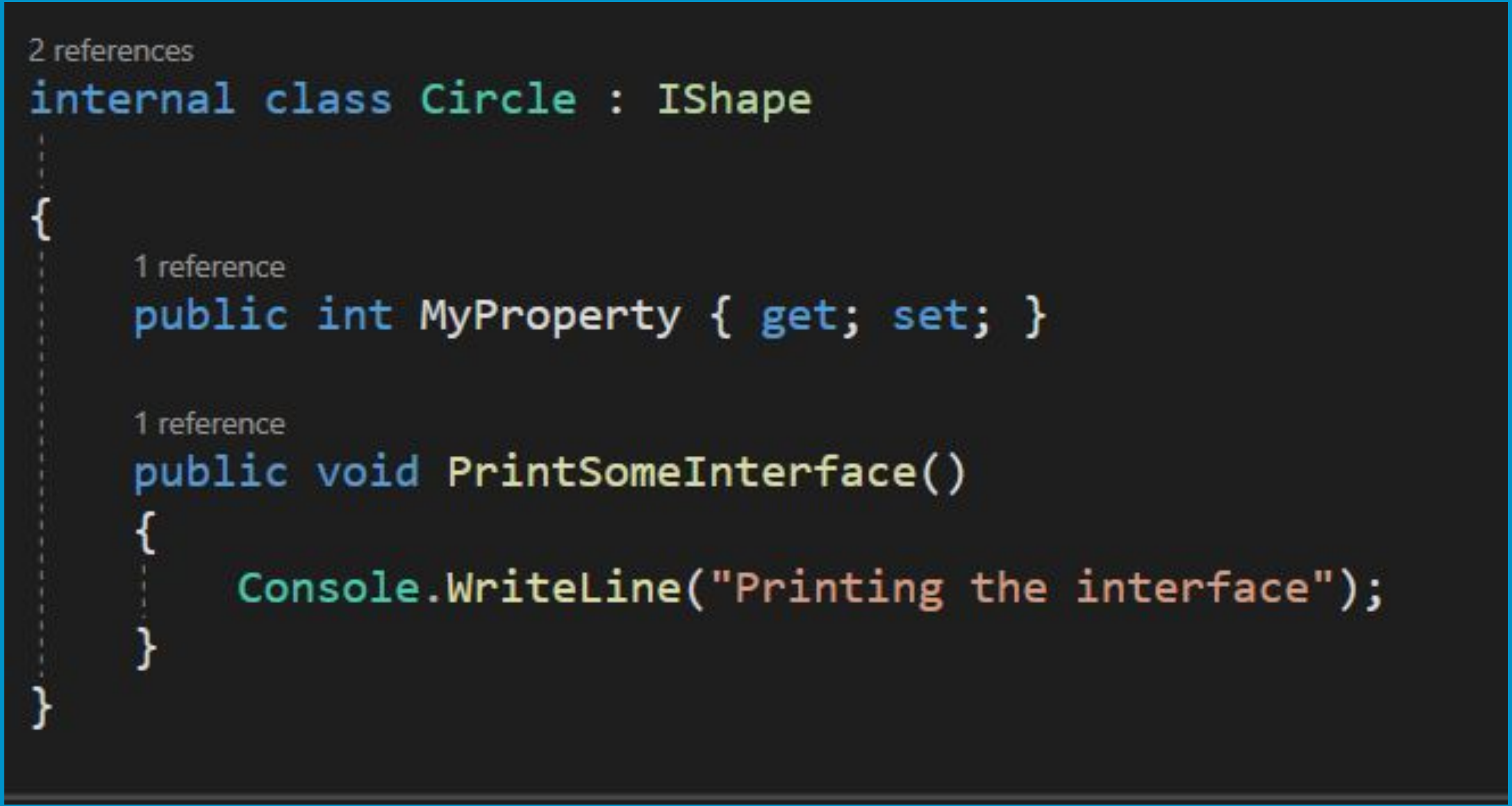


```
5 references
internal interface IShape
{
    3 references
    public  int MyProperty { get; set; }
    3 references
    public void PrintSomeInterface();

}
```

The interface itself provides no functionality.

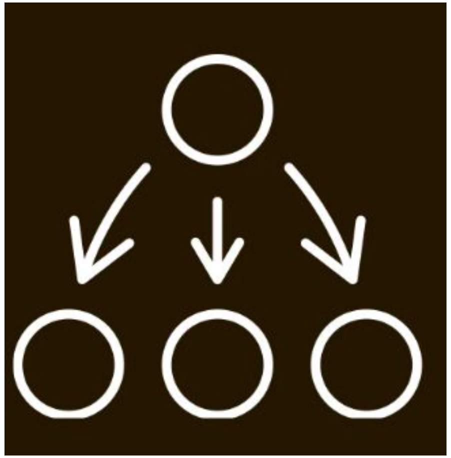
Classes <u>conform</u> to interfaces

Interface naming convention - will start with the letter I



```
2 references
internal class Circle : IShape

{
    1 reference
    public int MyProperty { get; set; }

    1 reference
    public void PrintSomeInterface()
    {
        Console.WriteLine("Printing the interface");
    }
}
```

You can implement multiple interfaces



Interfaces Specify Behavior

An interface is NOT a class. It is different from abstract class or base class. A class will **implement** an interface

Remember:

An interface does not care about the implementation. It merely requires that it is implemented.

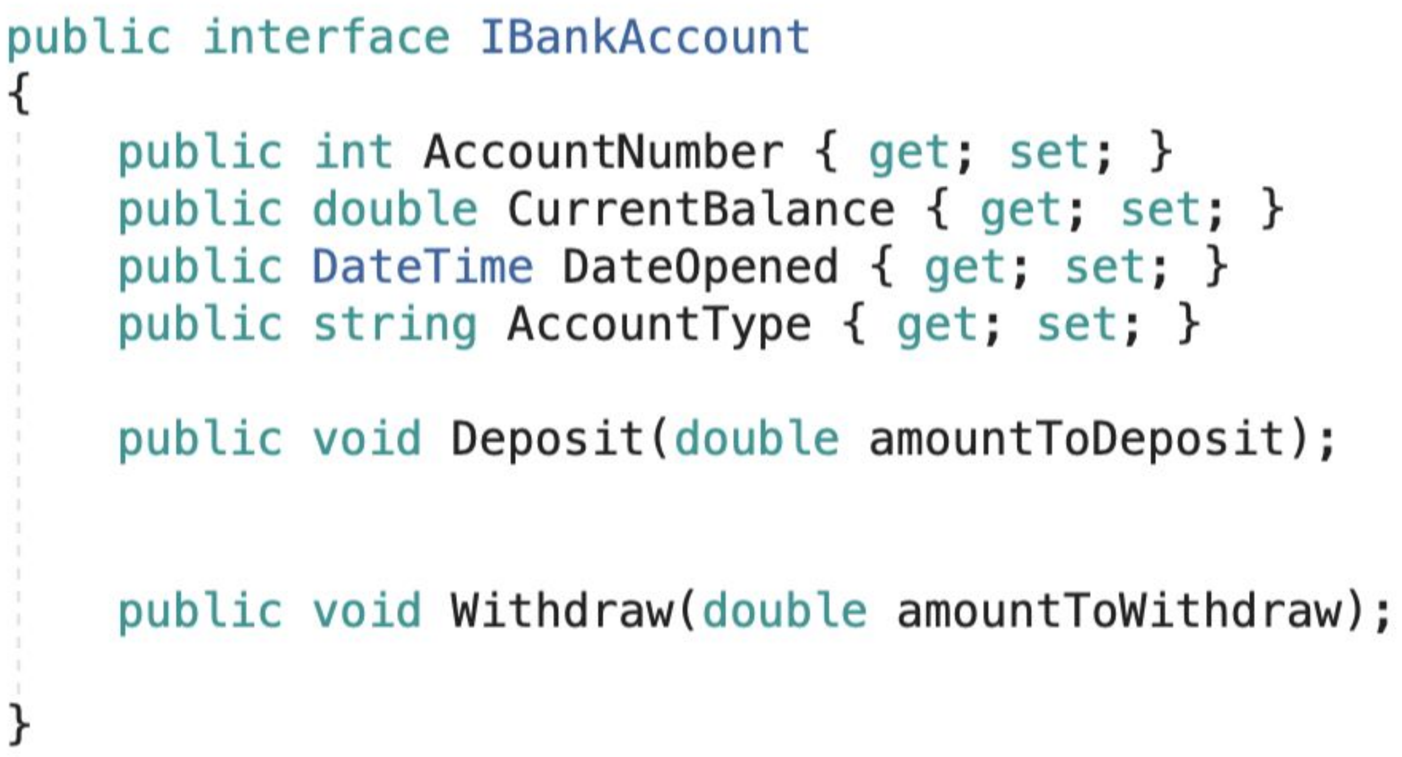
Polymorphism

- polymorphism

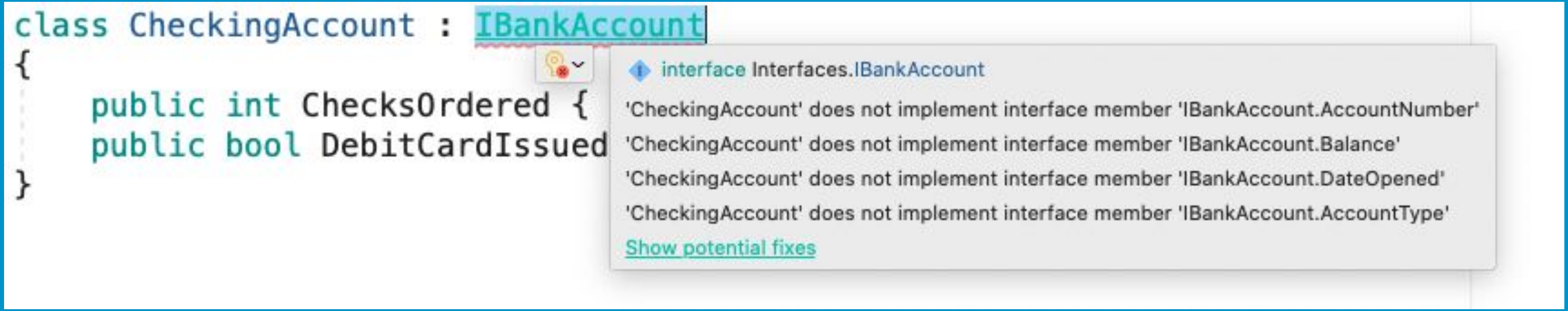
Benefits of Interfaces

– Adds ability to classes
– Acts as a checklist (contract)
– Focuses on functionality of your code because it doesn’t worry about implementation
– Supports polymorphism, allowing objects of different classes to be treated as objects of a common interface and processed uniformly.
– Makes code less coupled - the implementation of one class can be changed without affecting any other classes that use it, as long as the interface remains consistent

capital I



```
public interface IBankAccount
{
    public int AccountNumber { get; set; }
    public double CurrentBalance { get; set; }
    public DateTime DateOpened { get; set; }
    public string AccountType { get; set; }

    public void Deposit(double amountToDeposit);


    public void Withdraw(double amountToWithdraw);

}
```

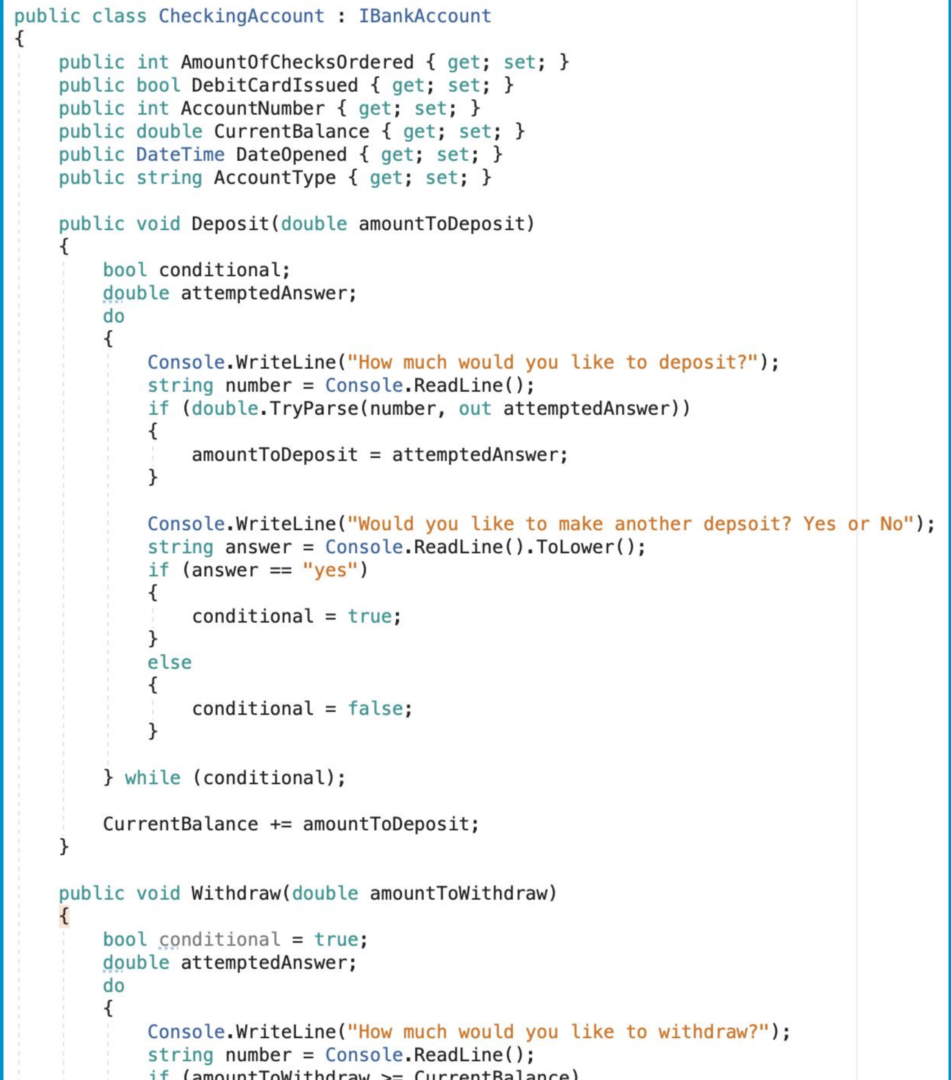


```
class CheckingAccount : IBankAccount
{
    public int ChecksOrdered {
    public bool DebitCardIssued
}
```

interface Interfaces.IBankAccount

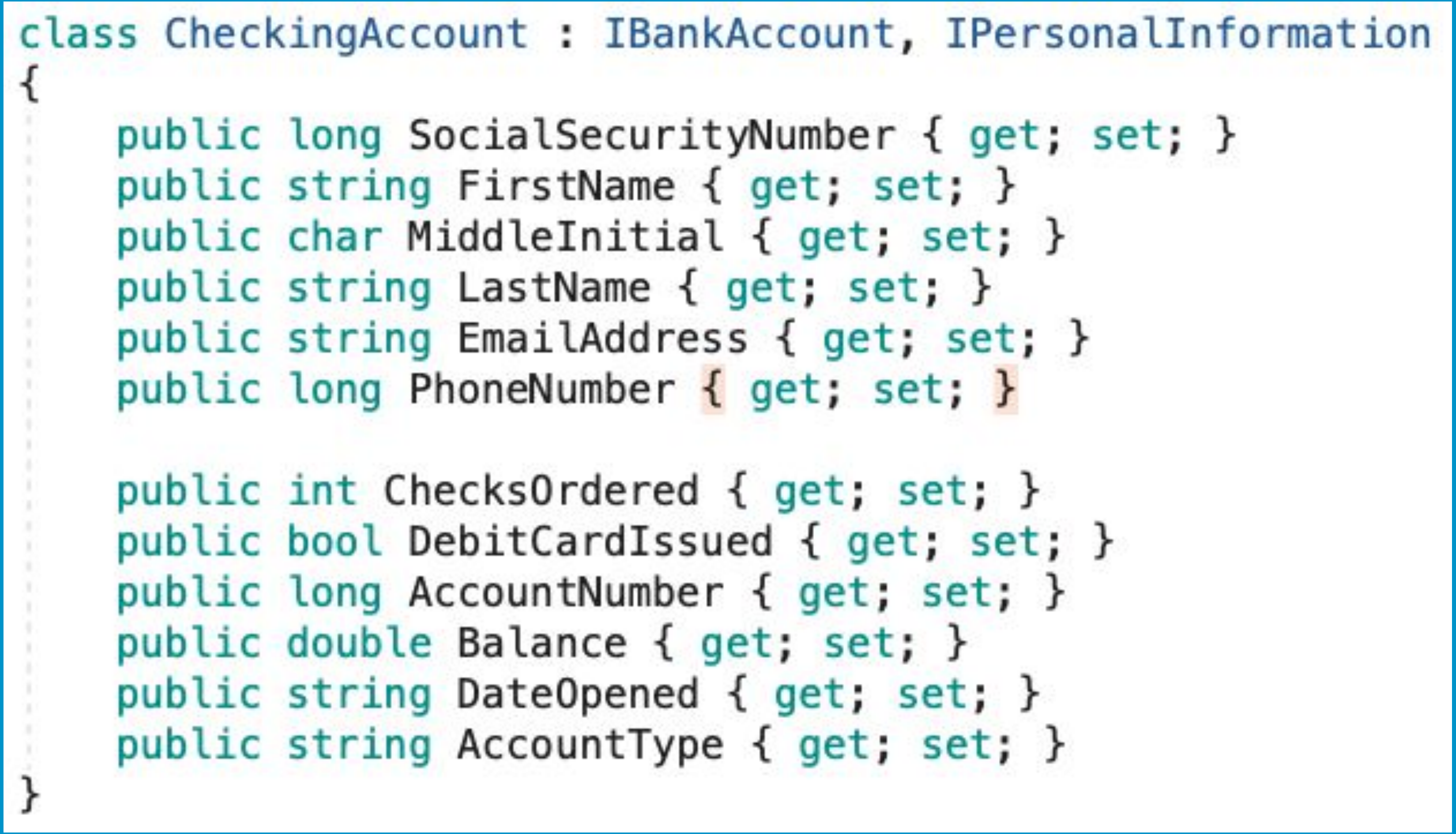
'CheckingAccount' does not implement interface member 'IBankAccount.AccountNumber'
'CheckingAccount' does not implement interface member 'IBankAccount.Balance'
'CheckingAccount' does not implement interface member 'IBankAccount.DateOpened'
'CheckingAccount' does not implement interface member 'IBankAccount.AccountType'

Show potential fixes



```
public class CheckingAccount : IBankAccount
{
    public int AmountOfChecksOrdered { get; set; }
    public bool DebitCardIssued { get; set; }
    public int AccountNumber { get; set; }
    public double CurrentBalance { get; set; }
    public DateTime DateOpened { get; set; }
    public string AccountType { get; set; }

    public void Deposit(double amountToDeposit)
    {
        bool conditional;
        double attemptedAnswer;
        do
        {
            Console.WriteLine("How much would you like to deposit?");
            string number = Console.ReadLine();
            if (double.TryParse(number, out attemptedAnswer))
            {
                amountToDeposit = attemptedAnswer;
            }

            Console.WriteLine("Would you like to make another depsoit? Yes or No");
            string answer = Console.ReadLine().ToLower();
            if (answer == "yes")
            {
                conditional = true;
            }
            else
            {
                conditional = false;
            }

        } while (conditional);

        CurrentBalance += amountToDeposit;
    }

    public void Withdraw(double amountToWithdraw)
    {
        bool conditional = true;
        double attemptedAnswer;
        do
        {
            Console.WriteLine("How much would you like to withdraw?");
            string number = Console.ReadLine();
            if (amountToWithdraw >= CurrentBalance)
```

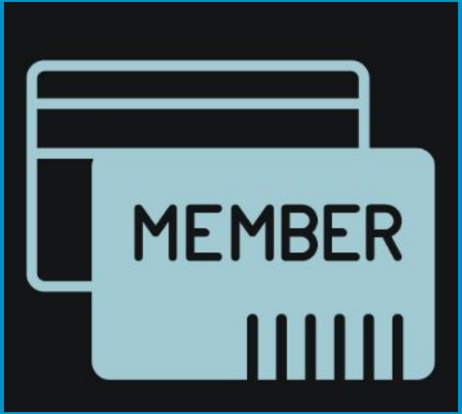
A class can conform to multiple interfaces



```
class CheckingAccount : IBankAccount, IPersonalInformation
{
    public long SocialSecurityNumber { get; set; }
    public string FirstName { get; set; }
    public char MiddleInitial { get; set; }
    public string LastName { get; set; }
    public string EmailAddress { get; set; }
    public long PhoneNumber { get; set; }

    public int ChecksOrdered { get; set; }
    public bool DebitCardIssued { get; set; }
    public long AccountNumber { get; set; }
    public double Balance { get; set; }
    public string DateOpened { get; set; }
    public string AccountType { get; set; }
}
```

Takeaway

Abstract members only!



Extending Interfaces:

- An interface can inherit from another interface.
- It allows for the creation of new interfaces based on existing ones.
- When one interface inherits from another, it takes on all the member declarations of the inherited interface.

Contract Definition: An interface defines a contract that classes or structs can implement. It specifies "what" operations can be done but not "how" they're done.
No Implementation: Interfaces do not provide implementation for their members.
Multiple Inheritance: C# does not support multiple inheritance for classes, but they do allow a class to implement multiple interfaces.
Polymorphism: Interfaces support polymorphism, enabling you to interact with different objects in a consistent way by referring to their interface type.