Interfaces

In this lesson, another important topic of C# abstraction is covered, Interfaces.

WE'LL COVER THE FOLLOWING

- What Is an Interface?
- Declaration
- Rules to be Followed
- Implementation
- Advantages of Interfaces

What Is an Interface?

> An interface is just like an `abstract class` but only specifies the behavior that a class **must** *implement*.

An interface can be used to achieve **100%** abstraction as it can **only contain abstract members**(*what is to be done*) and no implementation details (*how to be done*) for these members. In this way, interfaces satisfy the definition of abstraction. The implementation details of the members declared in an interface are totally up to the classes implementing that interface.

> An interface can be thought of as a **contract** that a class has to fulfill while implementing that interface. According to this contract, the class that implements an interface has to implement all the abstract members declared in that very interface.

Declaration

An interface is declared just like a class but using the `interface` keyword:

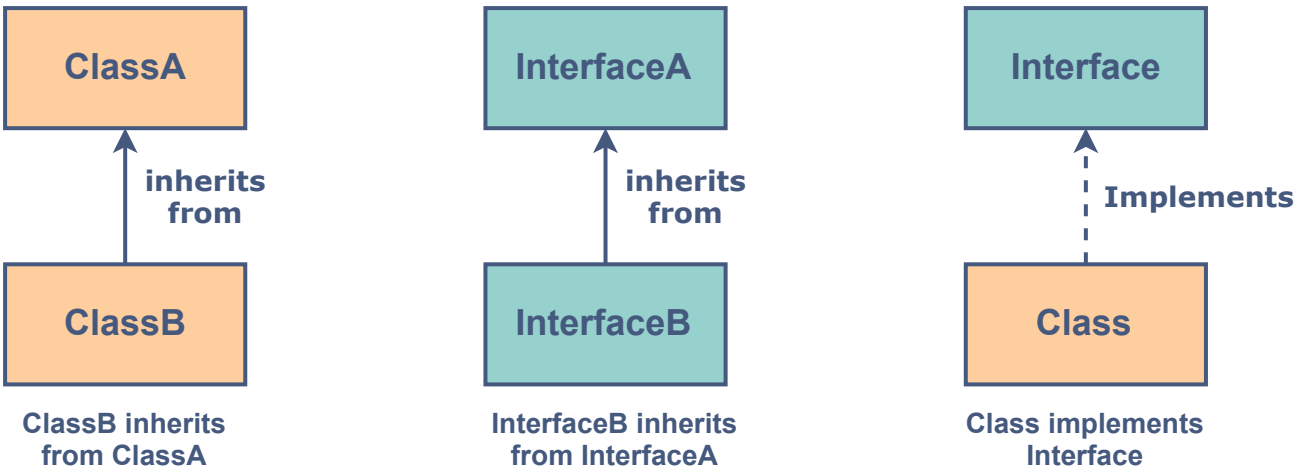
An interface is declared just like a class but using the `interface` keyword.

```
interface IInterfaceName {
  // abstract member(s) go here
}
```

Notice the additional `I` prepended to the interface name. It is used as an indicator to let the users know that this is an interface rather than a class.

Rules to be Followed

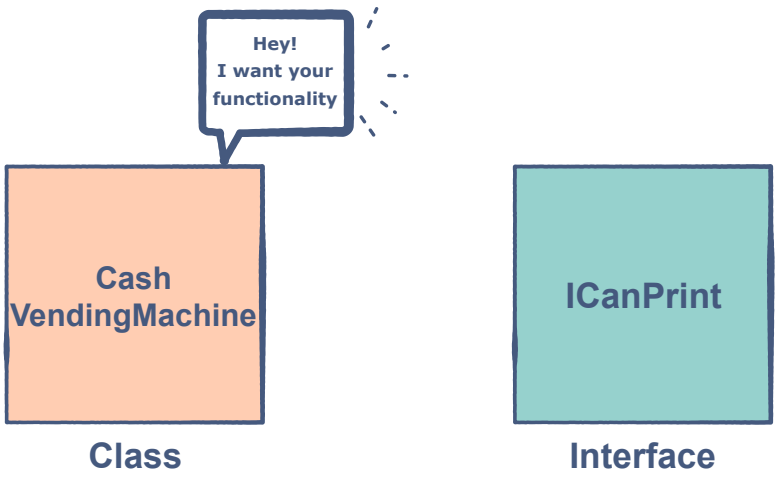
- An interface can only have:
    - Properties(s)
    - Method(s)
    - Event(s)
    - Indexers(s)
- All the members declared in an interface are **by default** `public abstract` so we do not need to write these modifiers while declaring the members.
- Just like an `abstract` class, an `interface` cannot be instantiated.
- To use an interface, a class **must** implement all of the `abstract` member(s) declared in it. We will learn, very soon, how to implement one.
- An interface **cannot** have constructor(s) in it.
- A class cannot extend from more than one class, but it can implement **any number** of interfaces.
- An interface can inherit from another interface.
- An interface cannot be declared `sealed` or `read-only`.

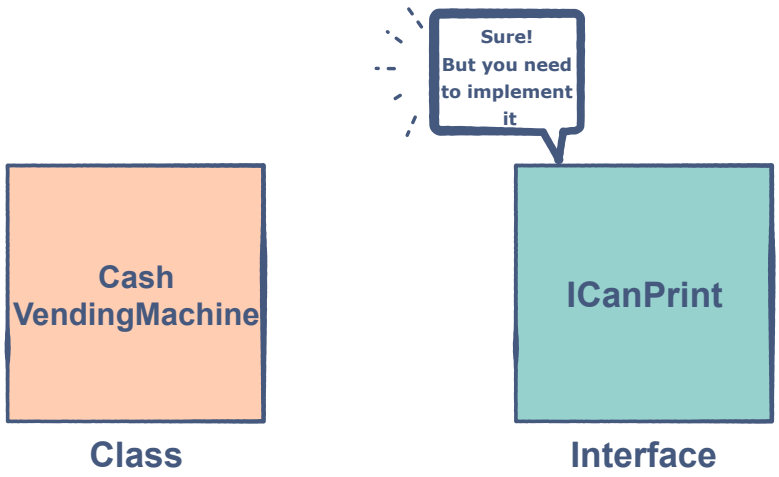


Implementation

Let's see interfaces in action by extending the example we used in the previous lesson. Consider that we want to have an additional receipt printing feature implementation in our `CashVendingMachine`. We want to print out the receipts whenever a transaction is successful, just like an ATM. For this, we have:

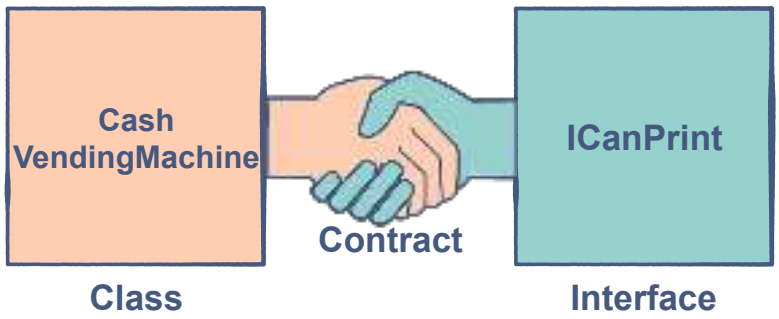
- A base `abstract` class named `EDM` (Electronic Dispensing Machine).
- A derived class named `CashVendingMachine`
- A derived class named `CardVendingMachine`
- An interface named `ICanPrint`





2 of 3

```
class CashVendingMachine : ICanPrint
```



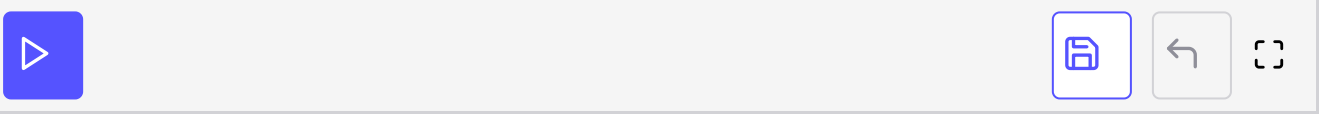
An interface can be thought of a contract

3 of 3

The code can be seen below:

```
// Base abstract class
abstract class EDM {

  public EDM() {
    //Parameter-less constructor
  }
  public abstract void Transact(bool successful);
```

```
  public void Dispense() {
    Console.WriteLine("{0} is dispensing the product!",this.GetType().Name);
    //this.GetType().Name is an inbuilt functionality of C#

    //to get the class name from which the method is being called
  }

}

// ICanPrint interface
interface ICanPrint {
  // Public abstract method
  void PrintReceipt();
}

class CashVendingMachine : EDM, ICanPrint { //Inherits from EDM, Implements ICanPrint

    public override void Transact(bool successful) {
      if(successful) {
        Console.WriteLine("The transaction was successfully completed!");
        PrintReceipt();
      }
      else Console.WriteLine("The transaction was unsuccessful!");
  }
  // Implementing the abstract method from ICanPrint
  public void PrintReceipt() {
    Console.WriteLine("Printing your receipt...");
  }

}



class Demo {

  public static void Main(string[] args) {
    // Creating the objects
    EDM cashVendy = new CashVendingMachine();

    cashVendy.Dispense();   // Calling methods from CashVendingMachine
    cashVendy.Transact(true);
    Console.WriteLine();
    cashVendy.Transact(false);

  }

}
```



> The highlighted line shows how to implement an interface syntactically.

Advantages of Interfaces

- Interfaces allow us to achieve *100% abstraction*.

- Interfaces can be used to achieve *loose coupling* in an application. This means that a change in one class doesn't affect the implementation of the other class. In other words, in an application, the code in a class becomes immutable and mutually exclusive since it doesn't get affected by changes in some other class.
- By using interfaces, one can break up complex designs and clear the dependencies between objects.
- Interfaces can be used to achieve *multiple inheritance* in C#. We will discuss this in the next lesson.