У тех, кто только начинает осваивать C# часто возникает вопрос что такое интерфейс и зачем он нужен.  
  
Сначала о том, что можно найти по первой же ссылке в поисковике. В большинстве статей смысл интерфейса разъясняется как «договор» о том, что должен содержать класс, какие свойства и методы. Например у нас есть интерфейс:

public interface IForecast

{

int GetForecast(int value);

int Size { get; set; }

}

Например, предположим у нас есть два класса:

class First

и

class Second

Это два совсем разных класса, но пусть у них будет что-то общее, т.е. мы хотим, чтобы один метод работал с ними обоими без дополнительной логики. В таком случае мы можем реализовать общий интерфейс, т.е. объявить эти классы как

class First : IForecast

и

class Second : IForecast

Теперь мы можем сделать общий метод для них:

void AnyMethod(IForecast anyClass)

{

var value = anyClass.GetForecast(10);

}

Как видно, переменная value получит значение функции GetForecast от того класса, который будет передан в качестве параметра без дополнительных действий по приведению типов и т.п.

Можно пойти еще дальше и объявить массив (или list) таких объектов:

var array = new IForecast[2];

array[0] = new First();

array[1] = new Second();  
А потом можно например вычислить сумму всех свойств Size:

var summ = 0;

foreach (var forecast in array)

{

summ += forecast.Size;

}  
У объявления через интерфейс есть один недостаток: вам будут доступны только методы и свойства, объявленные в интерфейсе ~~и самом классе~~. Если ваш класс унаследован от базового, то для доступа к его методам придется делать приведение типа:

public class BaseClass

{

public int GetValue(int value)

{

return value \* 2;

}

}

public class Second: BaseClass, IForecast

IForecast frc;

frc = new Second();

var frcBase = (BaseClass) frc;

var result = frcBase.GetValue(45);  
Еще один фокус с интерфейсам можно провернуть используя возможность реализовать в одном классе два интерфейса с одинаковыми по сигнатуре, но разными по содержанию методами. Я не знаю какая от этого может быть практическая польза кроме уже упомянутой, но если вдруг вам не хватает фантазии придумывать имена функций, то можете вынести их в интерфейсы.  
  
Например добавим в наш проект еще один интерфейс:

public interface IForecast2

{

int GetForecast(int value);

}  
И создадим новый класс, наследующий оба интерфейса:

public class Third: IForecast, IForecast2

{

int IForecast.GetForecast(int value)

{

return value + Size;

}

public int Size { get; set; }

int IForecast2.GetForecast(int value)

{

return 2 \* value + Size;

}

}  
Это называется явной реализацией интерфейса. Теперь можно построить такую конструкцию:

var third = new Third {Size = 10};

var v1 = ((IForecast) third).GetForecast(100);

var v2 = ((IForecast2)third).GetForecast(100);

Console.WriteLine(v1+v2);  
Интересно, что при реализации таких интерфейсов методам не могут быть назначены модификаторы доступа (“public” или что-то еще), однако они вполне доступны как публичные через приведение типа.  
  
Как видно, это работает, но выглядит слишком громоздко. Чтобы как-то улучшить код, можно написать несколько иначе:

var third = new Third {Size = 10};

var bad = (IForecast) third;

var good = (IForecast2) third;

var v1 = bad.GetForecast(100);

var v2 = good.GetForecast(100);

Console.WriteLine(v1+v2)  
  
bad и good будут ссылаться на third, но иметь в виду разные интерфейсы. Привидение можно сделать один раз, а его результат потом использовать многократно. Возможно, что в некоторых случаях это сделает код более читаемым.  
  
Еще одно применение интерфейса — быстрое изготовление заглушек для функций.  
Например, вы в команде строите большой проект и ваш класс зависит от класса, который пишет коллега. Но еще не написал. Мудрый начальник подготовил интерфейсы всех основных классов еще на первом этапе разработки проекта и вы теперь можете не ждать коллегу, а реализовать класс-заглушку для своего класса. Т.е. временный класс не будет содержать логики, а будет только делать вид что работает: принимать вызовы и возвращать адекватные значения. Для этого надо только реализовать тот же интерфейс, что получил ваш коллега.  
  
Работающий проект со всеми примерами из этой статьи можно посмотреть здесь: [ссылка](https://github.com/viktand/testInterface)