I have seen many times an Interface instance generated from a class.

Why does use an Interface in this wise?

An Interface instance created only itself with the help of the derived class and we can access only this interface members through this instance.

How does this give an advantage?

Interfaces define that a class MUST be able to do something. This means that you know the object being worked on will do what you want to be able to do. It allows you greater freedom and advantages of OOP. This is a deep topic but a very basic example would be this:

```
public interface IAnimal
{
    string Speak();
}

public class Dog : IAnimal
{
    public string Speak()
    {
        return "Woof, woof";
    }
}

public class Cat : IAnimal
{
    public string Speak()
    {
        return "Meow";
    }
}

public class Parrot : IAnimal
{
    public string Speak()
    {
        return "Sqwark!";
    }
}
```

Then you could use any animal you like!

```
class Program
{
    static void Main(string[] args)
    {
        // Writes Woof, Woof
        IAnimal animal = new Dog();
        Console.WriteLine(animal.Speak());

        // Now writes Meow
        animal = new Cat();
        Console.WriteLine(animal.Speak());

        // Now writes Sqwark etc
        animal = new Parrot();
        Console.WriteLine(animal.Speak());
    }
}
```

This also allows you to then get into things like **Inversion Of Control** where you would take an item in like this and you could pass a dog, cat or parrot and the method would

always work, not knowing or caring which animal it was:

```
public void ShoutLoud(IAnimal animal)
{
    MessageBox.Show("Shout " + animal.Speak());
}
```

This then makes ShoutLoud **unit testable** because you could use a mock object rather than a real animal. It basically makes your code flexible and dynamic rather than rigid and tightly coupled.

Using an interface this way gives you the ability to create methods that use standard template of the interface. So here you might have many classes of printer that all inherit from `IPrinter`

```
class SamsungPrinter : IPrinter
{
    // Stuff and interface members.
}

class SonyPrinter : IPrinter
{
    // Stuff and interface members.
}

interface IPrinter
{
    void Print();
}
```

So for each type `SamsungPrinter`, `SonyPrinter`, etc. you can pre-process using something like

```
public static void PreProcessAndPrint(IPrinter printer)
{
    // Do pre-processing or something.
    printer.Print();
}
```

You know from inheriting from `IPrinter` and using that type in the method parameters that you can always safely use the `Print` method on what ever object is passed.
Of course there are many other uses for using interfaces. One example of their use is in design patterns, in particular the Factory and Strategy patterns.