Introduction

In this article along with the demo project I will discuss Interfaces versus Abstract classes. The concept of Abstract classes and Interfaces is a bit confusing for beginners of Object Oriented programming. Therefore, I am trying to discuss the theoretical aspects of both the concepts and compare their usage. And finally I will demonstrate how to use them with C#.

Background

An Abstract class without any implementation just looks like an Interface; however there are lot of differences than similarities between an Abstract class and an Interface. Let's explain both concepts and compare their similarities and differences.

What is an Abstract class?

An abstract class is a special kind of class that cannot be instantiated. So the question is why we need a class that cannot be instantiated? An abstract class is only to be sub-classed (inherited from). In other words, it only allows other classes to inherit from it but cannot be instantiated. The advantage is that it enforces certain hierarchies for all the subclasses. In simple words, it is a kind of contract that forces all the subclasses to carry on the same hierarchies or standards.

What is an Interface?

An interface is not a class. It is an entity that is defined by the word Interface. An interface has no implementation; it only has the signature or in other words, just the definition of the methods without the body. As one of the similarities to Abstract class, it is a contract that is used to define hierarchies for all subclasses or it defines specific set of methods and their arguments. The main difference between them is that a class can implement more than one interface but can only inherit from one abstract class. Since C# doesn’t support multiple inheritance, interfaces are used to implement multiple inheritance.

Both together

When we create an interface, we are basically creating a set of methods without any implementation that must be overridden by the implemented classes. The advantage is that it provides a way for a class to be a part of two classes: one inheritance hierarchy and one from the interface.

When we create an abstract class, we are creating a base class that might have one or more completed methods but at least one or more methods are left uncompleted and declared `abstract`. If all the methods of an abstract class are uncompleted then it is the same as an interface but with the restriction that it cannot make a class inherit from it. The purpose of an abstract class is to provide a base class definition for how a set of derived classes will work and then allow the programmers to fill the implementation in the derived classes.

There are some similarities and differences between an interface and an abstract class that I have arranged in a table for easier comparison:

Feature	**Interface**	**Abstract class**
Multiple inheritance	A class may inherit several interfaces.	A class may inherit only one abstract class.
Default implementation	An interface cannot provide any code, just the	An abstract class can provide complete, default

	signature.	code and/or just the details that have to be overridden.
Constants	Only Static final constants.	Both instance and static constants are possible.
Core VS Peripheral	Interfaces are used to define the peripheral abilities of a class. In other words both Human and Vehicle can inherit from a IMovable interface.	An abstract class defines the core identity of a class and there it is used for objects of the same type.
Homogeneity	If the various implementations only share method signatures then it is better to use Interface.	If the various implementations are of the same kind and use common behaviour or status then abstract class is better to use.
Speed	Requires more time to find the actual method in the corresponding classes.	Fast
Adding functionality	If we add a new method to an Interface then we have to track down all the implementations of the interface and define implementation for the new method.	If we add a new method to an abstract class then we have the option of providing default implementation and therefore all the existing code might work properly.

Using the code

Let me explain the code to make it a bit easier. There is an `Employee` abstract class and an `IEmployee` interface. Within the Abstract class and the Interface entity I am commenting on the differences between the artifacts.

I am testing both the Abstract class and the Interface by implementing objects from them. From the `Employee` abstract class, we have inherited one object: `Emp_Fulltime`. Similarly from `IEmployee` we have inherited one object: `Emp_Fulltime2`.

In the test code under the GUI, I am creating instances of both `Emp_Fulltime` and `Emp_Fulltime2` and then setting their attributes and finally calling the `calculateWage` method of the objects.

Abstract Class Employee

```
using System;

namespace AbstractsANDInterfaces
{
   ///
   /// Summary description for Employee.
   ///

   public abstract class Employee
      {
```

```
    //we can have fields and properties
    //in the Abstract class
    protected String id;
    protected String lname;
    protected String fname;

    //properties
    public abstract String ID
        {
      get;
      set;
        }

    public abstract String FirstName
        {
      get;
      set;
        }

    public abstract String LastName
        {
      get;
      set;
        }
    //completed methods
    public String Update()
        {
      return "Employee " + id + " " +
                      lname + " " + fname +
           " updated";
        }
    //completed methods
    public String Add()
        {
      return "Employee " + id + " " +
                      lname + " " + fname +
           " added";
        }
    //completed methods
    public String Delete()
        {
      return "Employee " + id + " " +
                      lname + " " + fname +
           " deleted";
        }
    //completed methods
    public String Search()
        {
```

```
        return "Employee " + id + " " +
                        lname + " " + fname +
            " found";
        }

    //abstract method that is different
    //from Fulltime and Contractor
    //therefore i keep it uncompleted and
    //let each implementation
    //complete it the way they calculate the wage.
    public abstract String CalculateWage();

    }
}
```

Interface Employee

```
using System;


namespace AbstractsANDInterfaces
{
  /// <summary>
  /// Summary description for IEmployee.
  /// </summary>
  public interface IEmployee
    {
    //cannot have fields. uncommenting
    //will raise error!
//        protected String id;
//        protected String lname;
//        protected String fname;

    //just signature of the properties
    //and methods.
    //setting a rule or contract to be
    //followed by implementations.
    String ID
        {
      get;
      set;
        }

    String FirstName
        {
      get;
      set;
        }
    String LastName
        {
      get;
      set;
        }

    // cannot have implementation
    // cannot have modifiers public
    // etc all are assumed public
    // cannot have virtual

    String Update();

    String Add();

    String Delete();

    String Search();

    String CalculateWage();
    }
}
```

Inherited Objects

Emp_Fulltime:

```
using System;

namespace AbstractsANDInterfaces
{
  ///
  /// Summary description for Emp_Fulltime.
  ///

  //Inheriting from the Abstract class
  public class Emp_Fulltime : Employee
    {
    //uses all the properties of the
    //Abstract class therefore no
    //properties or fields here!

    public Emp_Fulltime()
        {
        }


    public override String ID
        {
```

```
        get
              {
          return id;
              }
        set
              {
                  id = value;
              }
          }

    public override String FirstName
          {
        get
              {
          return fname;
              }
        set
              {
                  fname = value;
              }
          }

    public override String LastName
          {
        get
              {
          return lname;
              }
        set
              {
                  lname = value;
              }
          }

    //common methods that are
    //implemented in the abstract class
    public new String Add()
          {
        return base.Add();
          }
    //common methods that are implemented
    //in the abstract class
    public new String Delete()
          {
        return base.Delete();
          }
    //common methods that are implemented
    //in the abstract class
    public new String Search()
```

```
        {
      return base.Search();
        }
    //common methods that are implemented
    //in the abstract class
    public new String Update()
        {
      return base.Update();
        }

    //abstract method that is different
    //from Fulltime and Contractor
    //therefore I override it here.
    public override String CalculateWage()
        {
      return "Full time employee " +
         base.fname + " is calculated " +
         "using the Abstract class...";
        }
    }
}
```

Emp_Fulltime2:

```
using System;

namespace AbstractsANDInterfaces
{
  ///
  /// Summary description for Emp_fulltime2.
  ///

  //Implementing the interface
  public class Emp_fulltime2 : IEmployee
    {
    //All the properties and
    //fields are defined here!
    protected String id;
    protected String lname;
    protected String fname;

    public Emp_fulltime2()
        {
      //
      // TODO: Add constructor logic here
      //
        }

    public String ID
        {
      get
            {
        return id;
            }
      set
            {
                id = value;
            }
        }

    public String FirstName
        {
      get
            {
        return fname;
            }
      set
            {
                fname = value;
            }
        }

    public String LastName
        {
      get
            {
        return lname;
            }
      set
            {
                lname = value;
            }
        }

    //all the manipulations including Add,Delete,
    //Search, Update, Calculate are done
    //within the object as there are not
    //implementation in the Interface entity.
    public String Add()
        {
      return "Fulltime Employee " +
                             fname + " added.";
        }

    public String Delete()
        {
      return "Fulltime Employee " +
                              fname + " deleted.";
          }

     public String Search()
          {
       return "Fulltime Employee " +
                             fname + " searched.";
          }

     public String Update()
          {
       return "Fulltime Employee " +
                              fname + " updated.";
          }

     //if you change to Calculatewage().
     //Just small 'w' it will raise
     //error as in interface
     //it is CalculateWage() with capital 'W'.
     public String CalculateWage()
          {
       return "Full time employee " +
                       fname + " caluculated using " +
           "Interface.";
          }
     }
}
```

Code for testing

```
//This is the sub that tests both
//implementations using Interface and Abstract
private void InterfaceExample_Click(object sender,
                                   System.EventArgs e)
{
  try
     {

          IEmployee emp;

          Emp_fulltime2 emp1 = new Emp_fulltime2();
     //has to be casted because of the interface!
          emp = (IEmployee) emp1;
          emp.ID = "2234";
          emp.FirstName= "Rahman" ;
          emp.LastName = "Mahmoodi" ;
     //call add method od the object
          MessageBox.Show(emp.Add().ToString());

     //call the CalculateWage method
```

```
        MessageBox.Show(emp.CalculateWage().ToString());


    }
  catch(Exception ex)
    {
        MessageBox.Show(ex.Message);
    }

}

private void cmdAbstractExample_Click(object sender,
                                   System.EventArgs e)
{

    Employee emp;
  //no casting is requird!
    emp = new Emp_Fulltime();


    emp.ID = "2244";
    emp.FirstName= "Maria" ;
    emp.LastName = "Robinlius" ;
    MessageBox.Show(emp.Add().ToString());

  //call the CalculateWage method
    MessageBox.Show(emp.CalculateWage().ToString());

}
```

Conclusion

In the above examples, I have explained the differences between an abstract class and an interface. I have also implemented a demo project which uses both abstract class and interface and shows the differences in their implementation.