Abstract Class versus Interface

<http://www.codeproject.com/Articles/11155/Abstract-Class-versus-Interface>

By **[Rahman Mahmoodi](http://www.codeproject.com/script/Membership/View.aspx?mid=444844)**, 7 Jan 2008

|  |  |  |  |
| --- | --- | --- | --- |
| |  |  | | --- | --- | |  |  | |  |

Top of Form

* [**Download source files - 8.75 KB**](http://www.codeproject.com/KB/cs/abstractsvsinterfaces/AbstractsVSInterfaces_src.zip)

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 from 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 same as an interface. 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 signature. | An abstract class can provide complete, default code and/or just the details that have to be overridden. |
| Access Modfiers | An interface cannot have access modifiers for the subs, functions, properties etc everything is assumed as public | An abstract class can contain access modifiers for the subs, functions, properties |
| 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 various implementations only share method signatures then it is better to use Interfaces. | If 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 (Versioning) | 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. |
| Fields and Constants | No fields can be defined in interfaces | An abstract class can have fields and constrants defined |

Bottom of Form