CHAPTER 18

Abstract

An abstract class provides a partial implementation that other classes can build on. When a class is declared as abstract, it means that the class can contain incomplete members that must be implemented in derived classes, in addition to normal class members.

Abstract Members

Any member that requires a body can be declared abstract – such as methods, properties, and indexers. These members are then left unimplemented and only specify their signatures, while their bodies are replaced with semicolons.

```
abstract class Shape
{
  // Abstract method
  public abstract int GetArea();

  // Abstract property
  public abstract int area { get; set; }

  // Abstract indexer
  public abstract int this[int index] { get; set; }

  // Abstract event
  public delegate void MyDelegate();
```


M. Olsson, *C# 8 Quick Syntax Reference*, https://doi.org/10.1007/978-1-4842-5577-3_18

```
  public abstract event MyDelegate MyEvent;

  // Abstract class
  public abstract class InnerShape {};
}
```

Abstract Example

In the following example, the class has an abstract method named `GetArea`.

```
abstract class Shape
{
  private int x = 100, y = 100;
  public abstract int GetArea();
}
```

If a class derives from this abstract class, it is then forced to override the abstract member. This is different from the `virtual` modifier, which specifies that the member may optionally be overridden.

```
class Rectangle : Shape
{
  public int GetArea() { return x * y; }
}
```

The deriving class can be declared abstract as well, in which case it does not have to implement any of the abstract members.

```
abstract class Rectangle : Shape {}
```

An abstract class can also inherit from a non-abstract class.

```
class NonAbstract {}
abstract class Abstract : NonAbstract {}
```

If the base class has virtual members, these can be overridden as abstract to force further deriving classes to provide new implementations for them.

```
class MyClass
{
  void virtual Dummy() {}
}

abstract class Abstract : MyClass
{
  void abstract override Dummy() {}
}
```

An abstract class can be used as an interface to hold objects made from derived classes.

```
Shape s = new Rectangle();
```

It is not possible to instantiate an abstract class. Even so, an abstract class may have constructors that can be called from derived classes by using the `base` keyword.

```
Shape s = new Shape(); // compile-time error
```

Abstract Classes and Interfaces

Abstract classes are similar to interfaces in many ways. Both can define member signatures that deriving classes must implement, yet neither one of them can be instantiated. The key differences are first that the abstract class can contain non-abstract members, while the interface cannot. And second, a class can implement any number of interfaces but only inherit from one class, abstract or not.

```
// Defines default functionality and definitions
abstract class Shape
{
  public int x = 100, y = 100;
  public abstract int GetArea();
}
class Rectangle : Shape {} // class is a Shape

// Defines an interface or a specific functionality
interface IComparable
{
  int Compare(object o);
}
class MyClass : IComparable {} // class can be compared
```

An abstract class, just like a non-abstract class, can extend one base class and implement any number of interfaces. An interface, however, cannot inherit from a class. It can inherit from another interface, which effectively combines the two interfaces into one.