Abstract

Abstraction in C# is the process to hide the internal details and showing only the functionality. The **abstract modifier** indicates the incomplete implementation. The keyword **abstract** is used before the class or method to declare the class or method as abstract. Also, the **abstract** modifier can be used with **indexers**, events, and **properties**.

Example:

```
public abstract void geek();
// this indicates the method 'geek()' is abstract

abstract class gfg
// this indicates the class 'gfg' is abstract
```

Abstract Method: A method which is declared abstract, has no “body” and declared inside the abstract class only. An abstract method must be implemented in all non-abstract classes using the override keyword. After overriding the abstract method is in the non-Abstract class. We can derive this class in another class, and again we can override the same abstract method with it.

Syntax:

```
public abstract void geek();
// the method 'geek()' is abstract
```

Abstract Class:
This is the way to achieve the abstraction in C#.

An Abstract class is never intended to be instantiated directly.

Abstract class can also be created without any abstract methods, We can mark a class abstract even if doesn’t have any abstract method.

The Abstract classes are typically used to define a **base** class in the *class hierarchy*.

Or in other words, an abstract class is an incomplete class or special class we can’t instantiate.

The purpose of an abstract class is to provide a blueprint for derived classes and set some rules what the derived classes must implement when they inherit an abstract

class.

We can use an abstract class as a base class and all derived classes must implement abstract definitions.

Important Points:

- Generally, we use abstract class at the time of *inheritance*.
- A user must use the *override* keyword before the method which is declared as abstract in child class, the abstract class is used to inherit in the child class.
- An abstract class cannot be inherited by structures.
- It can contains constructors or destructors.
- It can implement functions with non-Abstract methods.
- It cannot support multiple inheritance.
- It can’t be static.

Example

```
// Abstract class
public abstract class Animal
{
  // Abstract method (does not have a body)
  public abstract void animalSound();
  // Regular method
  public void sleep()
  {
    Console.WriteLine("Zzz");
  }
}

// Derived class (inherit from Animal)
public class Pig : Animal
{
  public override void animalSound()
  {
    // The body of animalSound() is provided here
    Console.WriteLine("The pig says: wee wee");
  }
}
```