49 そのままは使えない「抽象クラス」

そのままは使えない、必ず子クラスを作って追加実装をしなければいけない、「抽象クラス（abstract class）」というものがあります。

リスト42がその例で、(2)のClass1クラスが抽象クラスです。(2)の先頭行には、abstractと記述しています。(3)のOutputメソッドは、やはりabstractと書かれた抽象メソッドで、中身の記述がありません。(4)のClass2クラスでは、(5)でOutputメソッドをオーバーライドして、処理を記述しています。Class1ではOutputメソッドの外側を記述し、Class2で内側を記述しているようなものです。Class2の段階まで持っていけば、(1)のようにインスタンス化して利用できます。

Class1は直接はインスタンス化して利用できません。(1)のClass2をClass1に書き換えると、**図54**のように下に波線が引かれ、そこにマウスポインターを持っていくとエラーメッセージが現れます。抽象クラスはだめよ、というわけです。

抽象クラスは、「何をするか」の大枠だけを決めておいて、「どのようにするか」を子クラスにまかせる仕組みと言えるでしょう。

似たような仕組みに、次に説明する「インタフェース」があります。C#で.NET Frameworkのクラスライブラリを使っていると、名前が「I」で始まるインタフェースによく遭遇します。例えば、IListインタフェースの実装として、ArrayListクラスが存在するなどです。

リスト42●抽象クラスの例（Program.csの一部）

```
class Program {
  static void Main(string[] args) {
    Class2 c = new Class2(); —(1)
    c.Output();
  }
}

abstract class Class1 {
  protected string S = "こんにちは！";
  public abstract void Output(); —(3)
}                                —(2)

class Class2 : Class1 {
  public override void Output() {
    Console.WriteLine(S);        (5)(4)
  }
}
```

図54●抽象クラスであるClass1は、インスタンス化できない

```
static void Main(string[] args) {
  Class1 c = new Class1();
  c.Output();
}
```

class csConsole01.Class1

エラー:
抽象クラスまたはインターフェイス 'csConsole01.Class1' のインスタンスを作成できません。

50 抽象クラスより厳しい「インタフェース」

インタフェースは、抽象クラスと同様に、機能の大枠を定めるものです。**リスト43**がその例で、(1)がInterface1インタフェース、(2)がそれを実装したClass1クラスです。クラスを継承する時と同様にコロンを使い、Class1クラスがInterface1の実装であることを示しています。

インタフェースは、抽象クラスよりも厳しい制約を課すものと言えます。リスト43はインタフェースを使ったことでリスト42から変更せざるを得なかった部分があります。それは文字列変数Sの宣言です。リスト42では抽象クラスClass1の中に記述していましたが、これをInterface1に書くことはできません。インタフェースはメンバー変数を持てないからです。

C#ではクラスの多重継承（複数の親を持つクラスを作る）はできません。でも、複数のインタフェースを実装するクラスを作ることは可能です。

リスト43●インタフェースとそれを実装するクラス（Program.csの一部）

```
class Program {
  static void Main(string[] args) {
    Class1 c = new Class1();
    c.Output();
  }
}

interface Interface1 {
  void Output();          (1)
}

class Class1 : Interface1 {
  string S = "こんにちは！";
  public void Output() {
    Console.WriteLine(S);   (2)
  }
}
```