プロパティ

暗黙のアクセッサメソッド

オブジェクト指向のフィールドにはアクセスレベルが割り当てられていて
多くのプログラマはメンバ変数に他のクラスが直接アクセスすることはないようにプログラムします

保護された属性を取得する場合、プログラマはアクセッサメソッドを呼び出します
画像ファイルのサイズを取得するときに getWidth() というメソッドを呼び出したり
特定の座標をオブジェクトに指定するのに setPoint() メソッドを呼び出すようにです

なぜ、直接アクセスせずにアクセッサメソッドを通す方法が推奨されるのかというと
例えば属性値を変更する時、与えられたデータが正常なものかをメソッドで判断し
正しいものであればセットし、そうでなければ設定を拒否することができます

ところが、プログラマによってはアクセッサメソッドに奇妙な名前をつけることがありますし
型などにおいても、アクセッサメソッドは一貫性を持ちません
C# 言語は**プロパティ**と呼ばれる技術でこれを見事にカバーしています

プロパティとは、暗黙のアクセッサメソッドです
通常のメンバにアクセスするように見せかけ、内部ではメソッドを通します
クラスを作成するプログラマはこれまでのようにアクセッサメソッドを作成し
そのクラスを利用するプログラマはアクセッサメソッドを意識せずにメンバにアクセスできます

プロパティの宣言は通常のメンバを宣言する場合とほとんど同じです

[attributes] [modifiers] type identifier {accessor-declaration}

attributes は属性、modifiers は修飾子
type はプロパティの型、indentifier はこのプロパティの名前です

accessor-declaration には、プロパティのアクセッサを宣言します
このアクセッサがプロパティの本体であり、暗黙のアクセッサメソッドとなります

set {accessor-body}
get {accessor-body}

set はプロパティへのアクセスがあった時に呼び出されるアクセッサメソッド
get はプロパティの要求があった時に呼び出されるアクセッサメソッドです
それぞれ、**set アクセッサ宣言**、**getアクセッサ宣言**と呼びます

アクセッサに対する代入行為があった場合、暗黙的に set アクセッサが呼び出されます
set アクセッサは、void 型の戻り値でプロパティ型のパラメータを一つ持っているメソッドに等しいです
代入された暗黙的な値は、常に **value という名前**を持ちます

代入以外の形でプロパティが参照された場合は get アクセッサが呼び出されます
get アクセッサはプロパティ方の戻りとを持つノンパラメータのメソッドに等しいです

```
class Kitty {
 private string strName;
 public string StrName {
  get {
   System.Console.WriteLine("strName を要求");
   return strName;
  }
  set {
   System.Console.WriteLine(value + "を設定");
   strName = value;
  }
 }
}

class Test {
 static void Main() {
  Kitty obj = new Kitty();
  obj.StrName = "Kitty on your lap";
  string str = obj.StrName;
 }
}
```

このプログラムは、Kitty クラスの strName メソッドにアクセスするために
アクセッサメソッドとなる StrName プロパティを用意しています

それぞれ、get アクセッサが呼び出された場合も、set アクセッサが呼び出された場合も
コンソールに文字列を出力するように設計されているので
このプログラムを実行すれば、プロパティが暗黙的に働いていることを確認できます

このプログラムを C++ や Java 形式のアクセッサメソッドで書けば次のようになるでしょう
C# のプロパティを使うか、従来のアクセッサメソッドを定義する形にするかは好みの問題でしょう
しかし、C# を使う以上は直感的に記述できるプロパティを使うべきといえるでしょう

```
class Kitty {
 private string strName;
 public string GetStrName() {
  System.Console.WriteLine("strName を要求");
```

```
  return strName;
 }
 public void SetStrName(string value) {
  System.Console.WriteLine(value + "を設定");
  strName = value;
 }
}

class Test {
 static void Main() {
  Kitty obj = new Kitty();
  obj.SetStrName("Kitty on your lap");
  string str = obj.GetStrName();
 }
}
```

実際に上のプロパティを使ったプログラムはこれと同じことをやっています

MSIL レベルでは「プロパティ」という概念は存在しません
プロパティはコンパイル時にアクセッサメソッドに変換されているのです

```
.method private hidebysig static void  Main() cil managed
{
  .entrypoint
  // Code size       25 (0x19)
  .maxstack  2
  .locals (class Kitty V_0,
           string V_1)
  IL_0000:  newobj     instance void Kitty::.ctor()
  IL_0005:  stloc.0
  IL_0006:  ldloc.0
  IL_0007:  ldstr      "Kitty on your lap"
  IL_000c:  callvirt   instance void Kitty::set_StrName(string)
  IL_0011:  ldloc.0
  IL_0012:  callvirt   instance string Kitty::get_StrName()
  IL_0017:  stloc.1
  IL_0018:  ret
} // end of method Test::Main
```

これが、先ほどのプロパティを使ったプログラムの Main() メソッドの MSIL です
set や get アクセッサ宣言は set_プロパティ名()、get_プロパティ名() メソッドに変換されています
ただし、C# からアクセッサメソッドを明示的に呼び出すことはできません
（すなわち、get_StrName() を指定することはできない）

因みに、プロパティもメソッド同様に**継承**されます
プロパティをオーバーライドするということも可能になっています
ただし、この時はアクセッサレベルではなくプロパティレベルでオーバーライドされるので
set アクセッサをオーバーライドした場合は、get アクセッサもオーバーライドされます

場合によっては、メンバ変数を変更されたくないということもあるでしょう
このような場合は、set アクセッサを省略することで読み取り専用プロパティを作れます

```
class Kitty {
 private string strName;
 public string StrName {
  get {
   System.Console.WriteLine("strName を要求");
   return strName;
  }
 }
}

class Test {
 static void Main() {
  Kitty obj = new Kitty();
  obj.StrName = "Kitty on your lap";
  //string str = obj.StrName; //取得できない
 }
}
```

StrName プロパティは get アクセッサしか持たないため
このプロパティは読み取り専用となります

前のページへ　　戻る　　次のページへ