静的メンバ

インスタンスの無いメンバ

各インスタンスは、それぞれが独自にメンバを保有するものでしたが
プログラムによっては、特定のクラスのデータを
全てのインスタンスで共有したい場合もあるでしょう

その他にも、固有の値しか持たないメンバ変数を全てのインスタンスに持たせることも
やはりメモリの消費につながるため、資源の無駄使いと批難されるでしょう
このような場合は**静的メンバ**を使ってこの問題を解決します

インスタンスを必要としないメンバには **static** 修飾子を指定します
そう、Main() メソッドに指定していた static キーワードがこれです
static は public などのアクセス修飾子と同様にクラス修飾子として指定します

静的なメンバはインスタンスの数に関係なく1つしかメモリに存在しません
static を指定したメンバはインスタンスを持たないため
インスタンスではなく**型を指定**して静的なメンバにアクセスします

```
class Kitty {
 public static string str;
 public void Write() {
  System.Console.WriteLine(str);
 }
}

class Test {
 static void Main() {
  Kitty rena = new Kitty();
  Kitty yuki = new Kitty();

  Kitty.str = "Kitty on your lap";

  rena.Write();
  yuki.Write();
 }
}
```

Kitty クラスの str メンバ変数は静的メンバであり、Write() メソッドはインスタンスメンバです
生成された二つのインスタンスは自分の Write() メソッドを実行します
これを実行し、コンソールに出力された結果は次のようになります

Kitty on your lap
Kitty on your lap

Write() メソッドは静的なメンバ変数 str を出力します
この変数は全てのインスタンスで共有されるので、変数 rena と yuki のどちらからアクセスしても
得られる内容は同じなので、結果として同じ文字列が2行出力されたのです

このように、静的メンバはインスタンスではなく直接クラスに結び付けらます
当然、静的メソッドもインスタンスメソッドと異なりインスタンスなしで動作します
ただし、静的メソッドは直接インスタンスメンバにアクセスすることはできなくなります
例えば、次のプログラムは間違っています

```
class Kitty {
 public string str;
 public static void Write() {
  System.Console.WriteLine(str);
 }
}
```

これは、静的メソッド Write() からインスタンスメンバ str にアクセスしています
同様にインスタンスメソッドを呼び出すことも当然ですができません

静的メソッドは、インスタンスが必要のない処理に使用します
例えば数学的な処理を返すメソッドはインスタンスの必要性はありません
二つの数を受けとって、大きい方を返すメソッドを作る時にインスタンスなど必要ないのです

```
class Math {
 public static int Max(int x , int y) {
  return x >= y ? x : y;
 }
}

class Test {
 static void Main() {
  System.Console.WriteLine(Math.Max(10 , 20));
 }
}
```

Math クラスの Max() メソッドは二つの整数から大きい方を返します
このメソッドにインスタンスは必要ないので、このメソッドは静的です
そのため、インスタンスではなく型を指定してアクセスすることができるのです

静的コンストラクタ

C# には「静的コンストラクタ」と呼ばれる新しい初期化ブロックが登場した
これは、C++ プログラマにとっては真新しいものだろう
この「静的コンストラクタは」Java で例えると「静的初期化ブロック」と同じ働きをする

静的コンストラクタは、コンストラクタに static 修飾子を付加します
ただし、静的コンストラクタには**アクセス修飾子はつけられません**
それは、静的コンストラクタはコンストラクタのように明示的に呼ばれるものではないからです
ついでに言うと、静的コンストラクタには引数を受け取ることもありません

静的コンストラクタは、CLR がクラスをロードした時に実行されます
これは、静的なメンバ変数を初期化する時などに用いることができます

```
class Kitty {
 static Kitty() {
  System.Console.WriteLine("Kitty on your lap");
 }
 public static void Temp() {}
}

class Test {
 static void Main() {
  Kitty.Temp();
 }
}
```

Kitty クラスは静的コンストラクタ Kitty() を持っています
Main() メソッドが Kitty クラスの静的メソッド Temp() を呼び出そうとすると
Kitty クラスはロードされ、それと同時に静的コンストラクタが実行されます

---

static

静的メンバを宣言する修飾子です
フィールド、メソッド、プロパティ、演算子、そしてコンストラクタに使えます

前のページへ　戻る　次のページへ