リソース

using ステートメント

using ディレクティブがローカルスコープで利用できない理由として
using キーワードのもう一つの姿である **using ステートメント**があります

using ステートメントは**リソース**を取得、実行、解放をパックして行う機能を持ち
ここで使う「リソース」の定義は System.IDisposable インターフェイスを実装する型を指します
IDisposable は System 名前空間で次のように定義されています

public interface IDisposable

このインターフェイスはいたって単純なもので
値を返さない、パラメータを受け取らない単純な一つのメソッドを宣言しています

void Dispose();

このメソッドは、リソースを解放する時に呼び出されるメソッドです
using ステートメントは、終了時にこのメソッドを呼び出すため
リソースは必ず IDisposable インターフェイスを実装する必要があるのです
using ステートメントの構文は次のようになっています

using (expression | type identifier = initializer) statement...

expression はリソースに変換できる型の式を指定します
式を指定しない場合は、もう一方のローカル変数宣言を宣言することになります
type は変数の型、identifier は識別子、initializer は初期化子を指定します

statement には、埋め込みステートメントを指定します
ここで、式やローカル変数宣言で行った参照を実行し、操作します
using のエンドポイントに到達すると、プログラムは自動的に Dispose() を呼び出します

```
using System;

class Kitty : IDisposable {
 private string name;
 void IDisposable.Dispose() {
  Console.WriteLine("Dispose : " + name);
 }
 public string Name {
  get { return name; }
  set { name = value; }
 }
}

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

このプログラムは、Kitty クラスを using ステートメントでリソースとして扱っています
プログラムを実行すると、次のような結果になりました

Dispose : Kitty on your lap

Dispose() メソッドが実行されていることがわかりますね
それも、Kitty は Dispose() メソッドを隠蔽しているので
ちゃんと IDisposable 型にキャストしていることもここから確認できます

この using の動作は、じつは try-finally に等しいのです
以下のプログラムは、じつは上の using を使ったプログラムに等しいです

```
using System;

class Kitty : IDisposable {
 private string name;
 void IDisposable.Dispose() {
  Console.WriteLine("Dispose : " + name);
 }
 public string Name {
  get { return name; }
  set { name = value; }
 }
}

class Test {
 static void Main() {
  Kitty obj = new Kitty();
  try {
   obj.Name = "Kitty on your lap";
  }
  finally {
   if (obj != null) ((IDisposable)obj).Dispose();
  }
 }
}
```

例外の扱いに慣れている場合は、この方が書きやすいかもしれませんが
やはり using ステートメントを用いた方が明らかにスマートです
GUI のルートウィンドウの破棄やデータ伝送をパッケージ化したクラスなどでは
一般的に何らかの後処理が必要になるため、リソース化によってより効率的にプログラムできます

using ステートメントのリソース処理で複数のリソースを扱いたい場合
通常は、using ステートメントをネスティングすることで実現できます

```
using System;

class Kitty : IDisposable {
 private string name;
 void IDisposable.Dispose() {
  Console.WriteLine("Dispose : " + name);
 }
 public string Name {
  get { return name; }
  set { name = value; }
 }
}

class Test {
 static void Main() {
  using (Kitty rena = new Kitty()) {
   using (Kitty yuki = new Kitty()) {
    rena.Name = "RENA";
    yuki.Name = "YUKI";
   }
  }
 }
}
```

Main() メソッドで using ステートメントがネストされていることに注目してください
この場合、Dispose() が呼び出されるのは内側で指定されたリソースからです

しかし、多くのリソースを同時に扱う場合、これでは可読性が低下します
そこで、次のようにカンマ , で区切ることで複数の変数を指定できます

```
using System;

class Kitty : IDisposable {
 private string name;
 void IDisposable.Dispose() {
  Console.WriteLine("Dispose : " + name);
 }
 public string Name {
  get { return name; }
  set { name = value; }
 }
}

class Test {
 static void Main() {
  using (Kitty rena = new Kitty() , yuki = new Kitty()) {
   rena.Name = "RENA";
   yuki.Name = "YUKI";
  }
 }
}
```

このプログラムでは、1つの using ステートメントで
同時に複数のローカル変数を指定しています
ただし、異なる型の変数を宣言することはできません

---

using ステートメント

using (expression | type identifier = initializer) statement

リソースを生成、実行します
制御がエンドポイントに到達するとリソースを解放します

expression - リソース型の式を指定します
type - リソース型を指定します
identifier - ローカル変数の識別子を指定します
initializer - 初期化子を指定します

前のページへ　　戻る　　次のページへ