変数のスコープ

変数の有効範囲

ここで、これまで幾分か曖昧だった変数の範囲と種類について説明したいと思います
変数には、それぞれアクセスできる範囲と破棄するタイミングがあります

変数は、メンバ変数と**ローカル変数**に区別されます
メンバ変数は、クラスに直結しているメンバであり、インスタンスと静的変数に分かれます
これに対し、ローカル変数はメソッドの内部で宣言された変数です

ローカル変数はメンバ変数と異なり**メソッド終了時に破棄**されます
これは参照型も同様で、破棄された参照型のインスタンスは解放の対象にセットされます

変数のアクセス範囲は { } で表され、これを **スコープ** と呼びます
メソッド内部でも、スコープを利用すればさらに変数を局所化して宣言できます
スコープから外部（親スコープ）の変数にアクセスすることはできますが
スコープの外部から内部の変数にアクセスすることはできません

```
class Test {
 static void Main() {
  string str1 = "Kitty on your lap";
  {
   string str2 = "Tokyo mew mew";
   System.Console.WriteLine(str1);
   System.Console.WriteLine(str2);
  }
  System.Console.WriteLine(str1);
  //System.Console.WriteLine(str2); //エラー
 }
}
```

Main() メソッドでは、さらにブロック { } を使用して子スコープを作成しています
そこで、新たに str2 という変数を宣言していますが、この変数はスコープ内でのみ有効です
そのため、ブロックを抜け出ると str2 はメモリから破棄されてしまいます

また、ローカル変数のスコープの場合は変数の隠蔽はできません
同一名の変数が親に存在する場合、コンパイラはエラーを発生させます
では、次のようなプログラムの場合はどうなるのでしょうか

```
class Test {
 public static int x;
 static void Main() {
  x = 10;
  int x;
 }
}
```

一見すると x = 10 で静的メンバ変数 x に 10 を代入し
その後にローカル変数として x を宣言し、メンバ変数を隠蔽しそうですが
じつは、ローカル変数宣言子の位置よりも前にローカル変数を参照するとエラーになります
Main() の1行目を Test.x と明示的にメンバ変数であることを示せばエラーにはなりません