アクセス修飾子

メンバのセキュリティ

クラスのメンバには、全て「宣言済みのアクセス可能性」を持ちます
これは、クラスのメンバが不用意に外部のクラスからアクセスされるのを防ぐためです

便利な機能を持つライブラリは外部から操作するためのメソッドを提供すれば良いだけで
内部事情に関するメソッドなどは公開する必要はありません
このようなメソッドやメンバ変数は、不用意に操作されるとプログラムの動作に影響します
そこで、外部や派生クラスからそのメソッドにアクセスできなくすることができます

デフォルトで、メンバは **private** アクセス修飾子を持ちます
private は自分が定義されているクラス以外はいかなるアクセスも許さないことを表します
これ以外に **public** を私たちは今まで指定してきましたが
public は private に対照で全ての位置からのアクセスを許可することを意味します

```
class Kitty {
 private string name;
 public void setName(string name) {
  this.name = name;
 }
 public string getName() {
  return name;
 }
}

class Test {
 static void Main() {
  Kitty rena = new Kitty();
  //rena.name = "RENA"; //コンパイルエラー
  rena.setName("RENA");
  System.Console.WriteLine(rena.getName());
 }
}
```

Kitty クラスの name メンバ変数は private です(private はデフォルトなので省略しても良い)
name メンバ変数には、Kitty クラスからはアクセス可能ですが、外部からはアクセスできません
そのため、このクラスは name を設定する setName() と取得する getName() を提供します

直接外部から name にアクセスした場合はコンパイラがエラーを出します
外部のクラスは setName() や getName() を使って間接的に name にアクセスします
実際にこのようなアプローチは、実用的なレベルで良く見かける設計です
なぜならば、通常メンバ変数の変更は他の何らかの動作に関連付けられることが多く
メンバ変数が変更された場合、他の処理をしなければならないことがあるからです

例えば、GUI ウィンドウを表示するか否かを表す visible という bool 変数があるとする
これが変更されれば、当然ウィンドウの描画処理を行う必要が出てくる
そのため開発者は、visible を private にし、setVisible() メソッドを代わりに提供するだろう

ところが、関係のない外部からはアクセスされたくないが
保守や拡張を行うのに**派生クラスからのアクセスは必要**とする場合はどうだろう
便利な機能は、できる限り基底クラスから受け継いでおきたいものです
このような場合は **protected** 修飾子を指定します

```
class Kitty {
 protected void Write() {
  System.Console.WriteLine("Kitty on your lap");
 }
}

class TokyoMM : Kitty {
 new public void Write() {
  base.Write();
  System.Console.WriteLine("Tokyo mew mew");
 }
}

class Test {
 static void Main() {
  TokyoMM obj = new TokyoMM();
  obj.Write();
 }
}
```

Kitty クラスの Write() メソッドは protected です
TokyoMM クラスは Kitty の派生クラスであり、このクラスは Write() を隠蔽します
派生クラスである TokyoMM から Kitty.Write() にアクセスすることは可能です
ただし、Test クラスから Kitty.Write() にアクセスしようとするとコンパイラはエラーを出します

ところで、隠蔽の時には異なるアクセス修飾子を指定することができます
ただし、オーバーライドを行う場合は**アクセス修飾子は変更できません**
では、隠蔽の時に private を指定すると、一体どうなってしまうのでしょうか？
private は外部のクラスから見れば、そのメンバを隠蔽することに等しいです

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

```
 }
}

class TokyoMM : Kitty {
 new private void Write() {
  System.Console.WriteLine("Tokyo mew mew");
 }
}

class Test {
 static void Main() {
  TokyoMM obj = new TokyoMM();
  obj.Write();
 }
}
```

このプログラムは非常に重要な意味を持っています
Test クラスの Main() メソッドから、アクセス不可の TokyoMM.Write() にアクセスしています
TokyoMM.Write() は Kitty.Write() を隠蔽しています

驚くことに、このプログラムは問題なくコンパイルできます
private を TokyoMM.Write() に指定することで、TokyoMM.Write() は隠蔽されたのです
Test クラスからはアクセスができないため、obj.Write() は Kitty.Write() にアクセスします

さて、これまで見てきたアクセス修飾子は C++ や Java 言語でも見られる代表的なものです
そのため、C++ や Java を学習したことがある方はよく見てきたものでしょう
しかし、C# ではこの他に **internal** というアクセス修飾子を持ちます

internal は現在のコンパイル単位でのみアクセスを許可するというものです
他のアクセス修飾子は他のアクセス修飾子と同時に使用することはできませんが
internal は protected internal というように、同時に使用することができます

「現在のコンパイル単位」という意味は、中間言語を理解するうえで重要です
この internal 修飾子は**アセンブリ**と呼ばれる、高度な C# 技術に関わります
これについては、C# の基礎を網羅した上で解説したいと思います

---

public
あらゆる位置からのアクセスを許可することを表します

protected
クラス外部からのアクセスを拒否するが
このクラスの派生クラスからのアクセスを許可します

private
クラスの外部からのアクセスを全て拒否します
このメンバには同一クラス内からのみアクセスできます

internal
現在のコンパイル単位でのみアクセスを許可します

前のページへ　戻る　次のページへ