名前空間

名前の階層

オブジェクト指向言語は、三大特性の一つとして「カプセル化」があります
名前空間は、このカプセル化を実現する一つの手段で
クラスなどに識別子に対して、その上にさらに別の階層を与えるものです

名前空間を用いることで、巨大なプロジェクトチームによる開発もより効率的に行えます
非オブジェクト指向言語ならば、それぞれのチームが互いに名前の競合に注意する必要がありますが
名前空間を用いれば、トップレベルの階層にそれぞれのチーム名を与えることで
以下の階層はどのような名前をつけても、他のチームと識別子が衝突することはありません

これまでのプログラムは、最上階層に class や interface を配置してきましたが
この上の最上位のスコープとして名前空間が入るのです
名前空間の宣言には **namespace** キーワードを用います

namespace name[.name1] ...] { type-declarations }

name には、この名前空間の名前となる**修飾識別子**を指定します
type-declarations に、名前空間の本文を指定します
名前空間の本文とは、これまで書いてきたようなクラスや構造体になります

名前空間をわかりやすく表現するならば、UNIX のディレクトリシステムに似ています
名前空間はディレクトリのように、階層化（ネスト）することも可能です

```
namespace KittyScope {
 class Kitty {}
}

class Test {
 static void Main() {
  //Kitty obj = new Kitty(); //Kitty はこの空間に存在しない
  KittyScope.Kitty obj = new KittyScope.Kitty();
 }
}
```

このプログラムは、KittyScope という名前空間に Kitty クラスを宣言しています
このクラスに別の名前空間からアクセスするには KittyScope を明示的に指定する必要があります
Test クラスは KittyScope ではないから、直接 Kitty クラスは見えません
もちろん、KittyScope 名前空間内からは見えるので、名前空間を指定する必要はありません

異なる名前空間は、名前が衝突することはありません
各名前空間で同名のクラスがあっても、それぞれのクラスは名前空間で識別されます

```
namespace KittyScope {
 class Kitty {
  public override string ToString() {
   return "Kitty on your lap";
  }
 }
}

namespace MainScope {
 class Kitty {
  public override string ToString() {
   return "Tokyo mew mew";
  }
 }

 class Test {
  static void Main() {
   MainScope.Kitty objA = new MainScope.Kitty();
   KittyScope.Kitty objB = new KittyScope.Kitty();
   System.Console.WriteLine(objA +"¥n" + objB);
  }
 }
}
```

このプログラムでは KittyScope と MainScope の名前空間で
Kitty クラスという同名のクラスが宣言されています
しかし、それぞれ宣言されている名前空間は異なるので衝突することはありません

因みに、Main() メソッドで MainScope.Kitty と指定してありますが
Test クラスも MainScope 空間内で宣言されているため、MainScope は省略できます

さらに、名前空間は別の名前空間を含むことができます
これによって、ルート空間からサブ空間へ階層化させることが可能です

```
namespace A {
 namespace B {
  namespace C {...
```

こうして階層化した名前空間の識別子にアクセスするには、ルートから順に指定してアクセスします
例えば、空間 C のクラス c の M() メソッドにアクセスするには A.B.C.c.M() と指定します
内部空間からのアクセスであれば、上位空間の指定は省略することが可能となります
同様のアクセスが B からならば、A と B の指定は省略してもかまいません

```
namespace A {
 namespace B {
  namespace C {
   class Kitty {
    public override string ToString() {
     return "Kitty on your lap";
    }
   }
  }
 }
}
class Test {
 static void Main() {
  A.B.C.Kitty obj = new A.B.C.Kitty();
  System.Console.WriteLine(obj);
 }
}
```

このような機能を用いれば、コンポーネントの目的に応じてカプセル化することも
社内専用ライブラリの開発チームの権威レベルを表すことも可能です

名前空間の階層は . トークンで識別子を区切って一度に宣言することも可能です
上の名前空間のネスト宣言は次のように宣言しても同じです

namespace A.B.C {...

このように、階層的な名前空間を一度に宣言することも可能です
少なくとも、上のプログラムよりは以下のプログラムのほうがスマートでしょう

```
namespace A.B.C {
 class Kitty {
  public override string ToString() {
  return "Kitty on your lap";
  }
 }
}
class Test {
 static void Main() {
  A.B.C.Kitty obj = new A.B.C.Kitty();
  System.Console.WriteLine(obj);
 }
}
```

ところで、名前空間の識別子が衝突してしまったらどうなるのでしょうか？
名前空間は名前の衝突を避ける手段として生まれたものですが
名前空間の修飾識別子が同名になることもありえる話です

じつは、この場合はエラーになりませんし、問題もありません
名前空間は実体を持つわけではなく、識別子の区分けにすぎません
名前空間の宣言で、修飾識別子が重複している場合は
それぞれが**同一の空間**と判断されて処理されることになります

```
namespace KittyScope {
 class Kitty {
  public override string ToString() {
   return "Kitty on your lap";
  }
 }
}

namespace KittyScope {
 class Test {
  static void Main() {
   Kitty obj = new Kitty();
   System.Console.WriteLine(obj);
  }
 }
}
```

プログラムを見るとKittyScopeという名前空間が重複して宣言されています
しかし、これはコンパイル時には同一の空間にまとめられるので問題はありません
Testクラスから見てもKittyクラスから見ても、互いに同じ空間にあると考えられます
これを利用すれば、異なる位置に同一の名前空間を書くことができます

---

namespece

namespace name[.name1] ...] { type-declarations }

名前空間を宣言します

name - 名前空間の修飾識別子を指定します
type-declarations - 名前空間の本文を指定します
