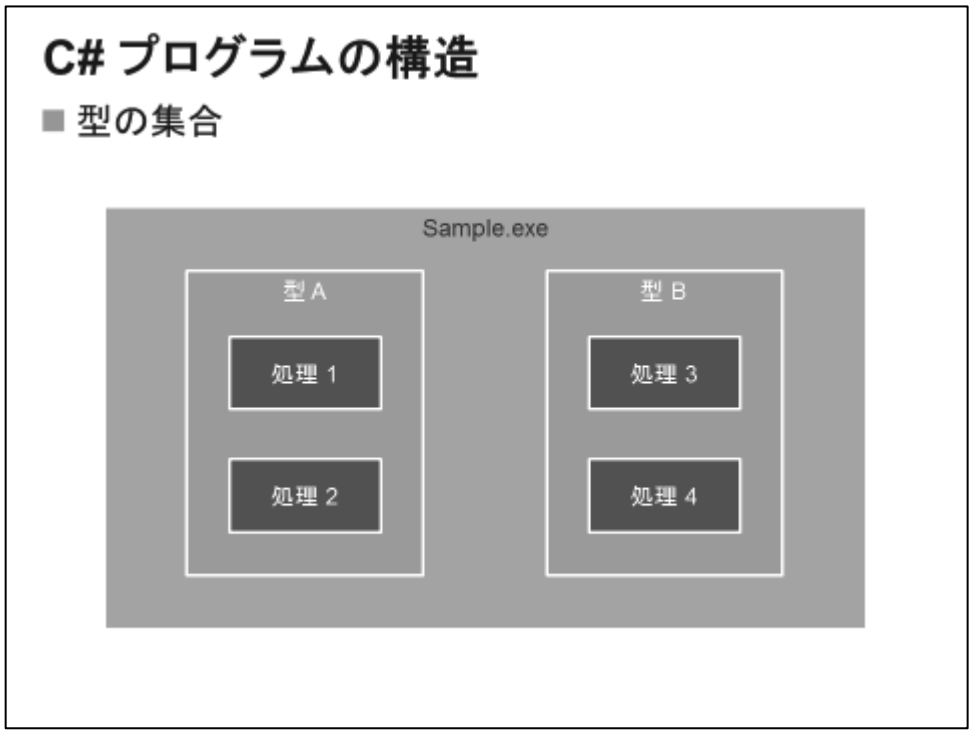


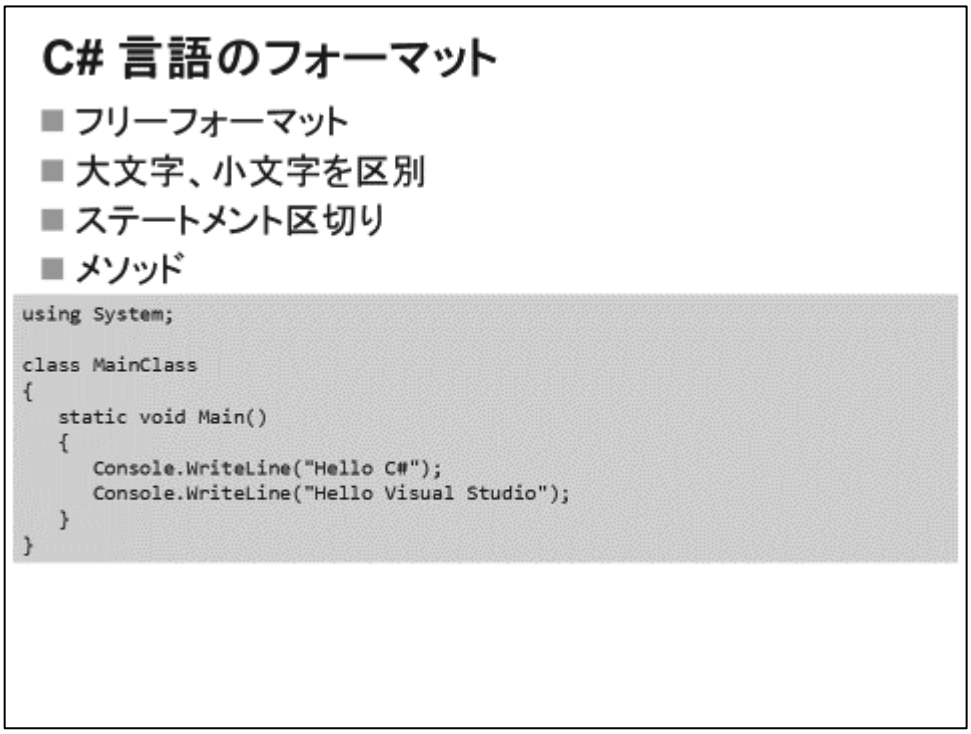
■　C# のプログラムは型の集合

C#で作成されるプログラムはすべて型の集合体(コレクション)として作成されます。処理のみを単独で記述せずに、何らかの型(ユーザー定義型)に含めます。

以下のコード例では、MainClass という名前で型を定義し、その中で Main という名前で処理を定義しています。

● **コード例：型**

```
1.  class MainClass
2.  {
3.     static void Main()
4.     {
5.        System.Console.WriteLine("Hello C#");
6.        System.Console.WriteLine("Hello Visual Studio");
7.     }
8.  }
```



■　フリーフォーマット

C# のソースコードの記述様式は、比較的自由に空白や改行を挿入することができます。

■　大文字、小文字を区別

C# では、大文字・小文字は厳密に区別されます(Case-Sensitive)。たとえば、「main」と「Main」、および「MAIN」は、まったく別の名前(識別子)として評価されます。

■　ステートメント区切り

実行させる命令は、ステートメントとして記述し、一つ一つのステートメントの区切りを「;」(セミコロン)であらわします。

■　メソッド

一般的なプログラミング言語と同様に、ある程度まとまった意味をもつ一連の処理は、ひとまとめであらわすことができます。そのまとまりを C# では「メソッド」と呼びます(他のプログラミング言語で「関数」や「プロシージャ」と呼ばれているものと同様です)。

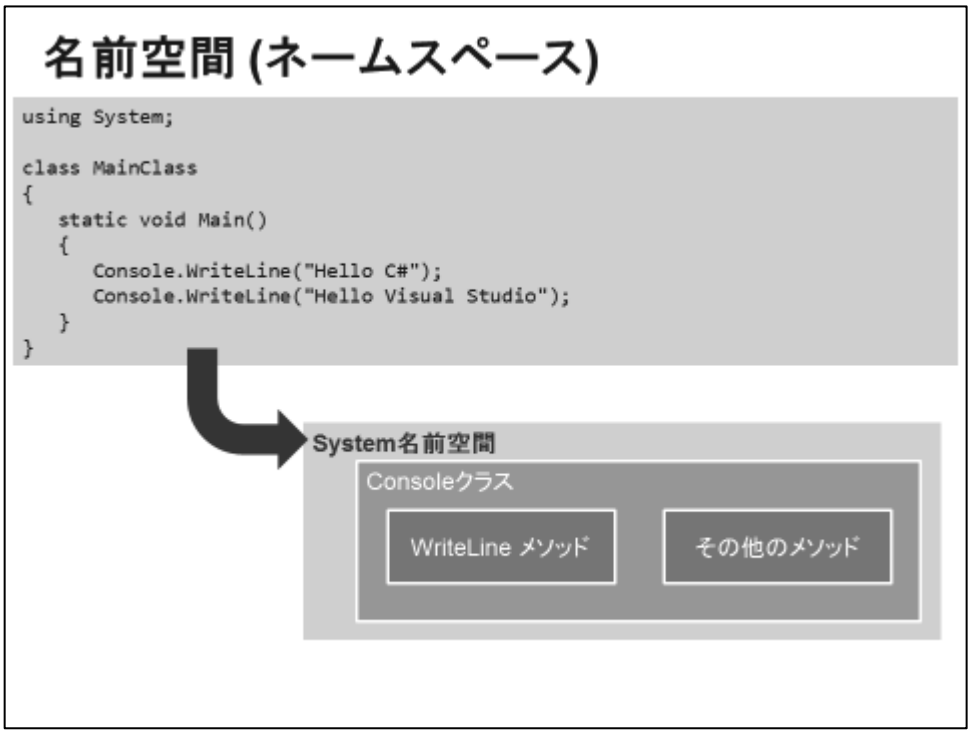
メソッドを用いることで、構造的に把握しやすいプログラムが作成できますが、C# ではメソッドのみを単独で記述することができません(前頁)。そこで、型のひとつである「クラス」の構造内でメソッドを定義します。

■　プログラムの開始点（エントリポイント）

メソッドを定義する場合、あらかじめ意味付けされた文字列(キーワード)を除けば、任意の名前をメソッドにつけることが可能です。しかし、「Main」という名前のメソッドは、特別な意味を持ちます。

C# のプログラムは、「Main」という名前のメソッドから処理が始まります。1 つのプログラム内に Main メソッドがひとつだけ定義されていれば、それがプログラムの開始点(エントリポイント)になります。また、Main メソッドを複数のクラスに定義することで、2 つ以上 Main メソッドを含んだプログラムを作成することも可能です。その場合は、エントリポイントにする Main メソッドを含むクラスを指定します。

上のコード例では、Main メソッドの中で、WriteLine メソッドにより「Hello C#」および「Hello Visual Studio」という文字列を表示しています。WriteLine メソッドは、.NET Framework クラスライブラリが提供するメソッドの一つです。.NET Framework が提供するメソッドも、何らかの型に含まれます。「WriteLine」の前の「Console」が、WriteLine メソッドが含まれるクラス名です。また、「Console」の前に記述される「System」は、Console クラスが含まれる名前空間です。



名前空間は、クラスなどの型を論理的にグループ化するものです。同じ名前のクラスであっても、それが含まれる名前空間が異なれば、異なるクラスとみなされるため、クラス名の重複を防ぐことができます。

名前空間は階層構造を形成することが可能です。.NET Framework クラスライブラリでは System 名前空間を頂点に複数の名前空間に枝分かれして、多くのクラスが管理されています。

特定の名前空間のクラスを利用するには、前ページの「System.Console」のように、「.」(ドット)付きの名前空間名でクラス名を修飾するか、または「using」キーワードで名前空間の使用を宣言して、暗黙的にクラス名を名前空間で修飾します。

■ コード例：名前空間

```
1.  using System;
2.
3.  class MainClass
4.  {
5.     static void Main()
6.     {
7.        Console.WriteLine("Hello C#");
8.        Console.WriteLine("Hello Visual Studio");
9.     }
10. }
```

ソースコード上に記される説明書きなど任意の文字列は、コメントとしてあらわします。

「/*　*/」は、コメントの開始位置と終了位置を明示した形式です。複数行にわたってコメントを書くことができます。

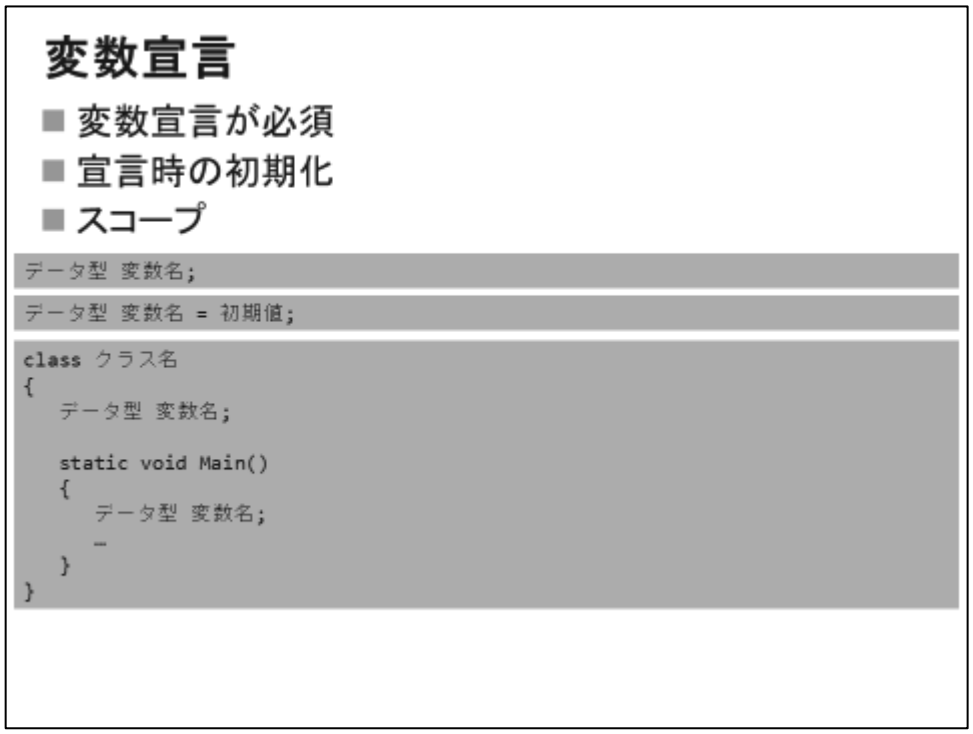
「//」は、単一行のみで利用するコメントです。//を記述した位置から、その行の終わりまでがコメントとして扱われます。

■ コード例：コメント

```
1.  using System; // 利用する名前空間の宣言
2.  class MainClass
3.  {
4.     /*
5.        アプリケーションの開始点
6.        文字列を 2 行出力して終了します
7.     */
8.     static void Main()
9.     {
10.       Console.WriteLine("Hello C#"); // 文字列を表示
11.       Console.WriteLine("Hello Visual Studio");
12.    }
13. }
```

■変数とデータ型■

- 変数宣言
- データ型
- 暗黙的に型指定された変数
- データ型の変換
- 値型と参照型
- ボックス化とボックス化解除



■　変数宣言が必須

C#で変数を利用するためには、必ず変数宣言が必要になります。変数宣言の記述法としては、以下のような形式を使用します。

```
データ型 変数名;
```

例えば、int というデータ型（整数を扱えるデータ型です）で、data という名前の変数を宣言するには、次のように記述します。

```
1.  int data;
```

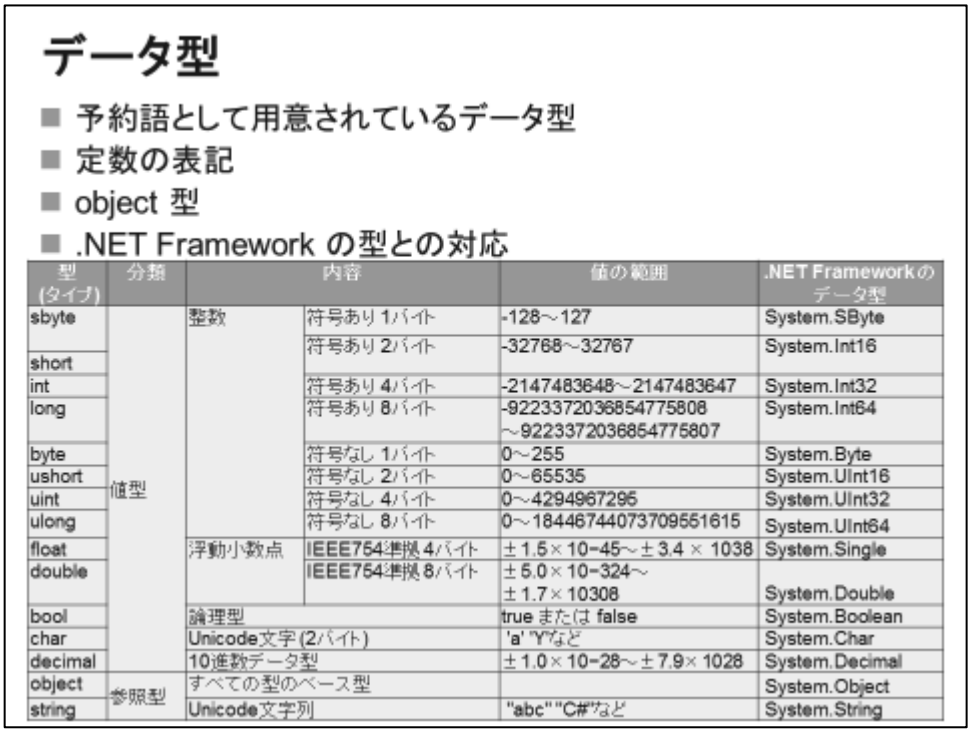
■　宣言時に初期化が可能

変数は宣言と同時に初期値を指定することも可能です。初期値を与える場合は、=(イコール)を使用して初期値を与えます。

```
1.  int data = 100;
```

■　スコープ

変数を宣言する場所は、クラス内であれば、どこで宣言しても文法上は正しいものになります。しかし、宣言する場所によって、その変数にアクセスできるコードの範囲が変わります。このアクセス可能な範囲をスコープと呼びます。メソッド内で宣言された変数は、そのメソッド内からのみ変数へのアクセスが行え、クラス内(メソッドの外側)で宣言された変数は、そのクラス内の各メソッドからアクセスが行えます。この章では、メソッド内で宣言される変数について扱います。なお、クラス内で宣言される変数については 3 章で扱います。



型 (タイプ)	分類	内容		値の範囲	.NET Framework のデータ型
sbyte	値型	整数	符号あり 1バイト	-128～127	System.SByte
short			符号あり 2バイト	-32768～32767	System.Int16
int			符号あり 4バイト	-2147483648～2147483647	System.Int32
long			符号あり 8バイト	-9223372036854775808 ～9223372036854775807	System.Int64
byte			符号なし 1バイト	0～255	System.Byte
ushort			符号なし 2バイト	0～65535	System.UInt16
uint			符号なし 4バイト	0～4294967295	System.UInt32
ulong			符号なし 8バイト	0～18446744073709551615	System.UInt64
float		浮動小数点	IEEE754準拠 4バイト	±1.5×10−45～±3.4 × 1038	System.Single
double			IEEE754準拠 8バイト	±5.0×10−324～ ±1.7×10308	System.Double
bool		論理型		true または false	System.Boolean
char		Unicode文字 (2バイト)		'a' 'Y'など	System.Char
decimal		10進数データ型		±1.0×10−28～±7.9×1028	System.Decimal
object	参照型	すべての型のベース型			System.Object
string		Unicode文字列		"abc" "C#"など	System.String

■　予約語（キーワード）として用意されているデータ型

C#では組み込み型は予約語(キーワード)を用いて表すことができます。C#の組み込み型は、整数、実数、文字、文字列など、一般的なデータ表現に利用されるデータ型が提供されています。

■　定数の表記

コード内に整数を記述する際、語尾に何もつけないときは、コンパイラは int、uint、long、ulong の順に照らし合わせ、値を格納可能な大きさのデータ型として扱います。また、コード内に小数点を含む数値を記述する場合は、語尾に何もつけないときは、コンパイラは double として扱います。それ以外のデータ型として扱いたいときは、以下のように数値の後にデータ型を表す値を指定します。

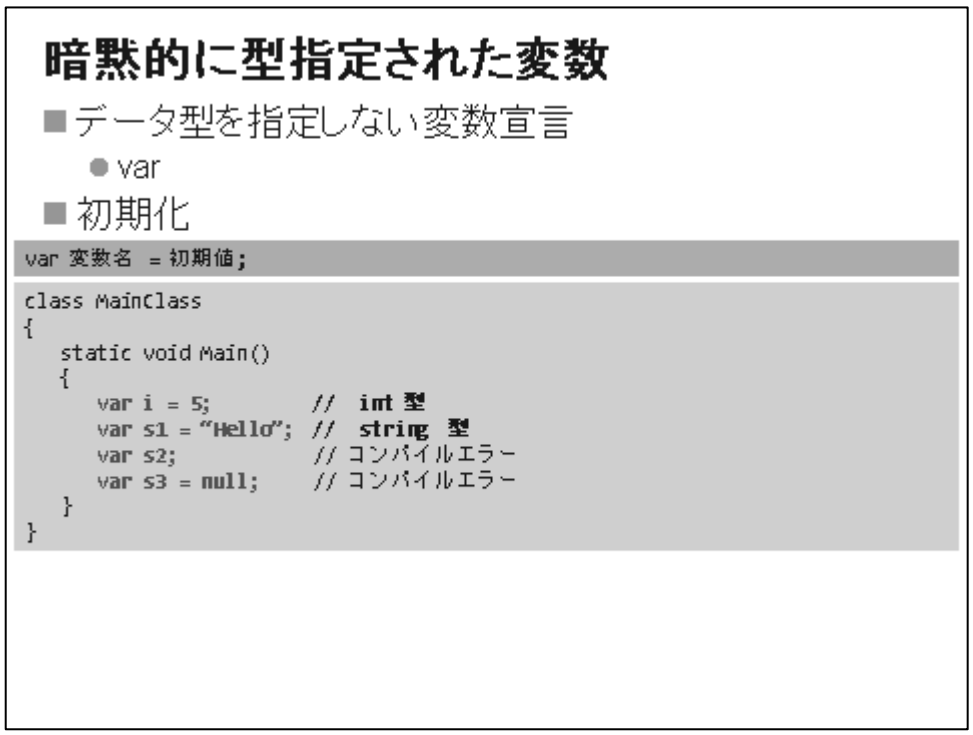
データ型	記述例
uint	100u、100U
ulong	100ul、100UL
float	3.14f、3.14F
double	3.14d、3.14D
decimal	1.05m、1.05M
char	'a'、'A'
string	"Hello"

■　object 型

object 型は、すべてのデータ型の基礎となる汎用的なデータ型です。すべてのデータ型の値が、object 型の変数に代入可能です。

■　C# の組み込み型は、.NET Framework の　CTS 型の別名

C# で提供されている組み込みのデータ型は、.NET Framework の　CTS (共通型システム)で提供される構造体やクラスの別名(エイリアス)にあたります。C# の型と、CTS の型は区別されません。コードでは、C# の型も CTS の型もどちらも利用可能です。



■　データ型を指定しない変数宣言

C#では var を利用すると、データ型を指定せずに変数を宣言することができます。データ型を指定せずに宣言した変数を、暗黙的に型指定された変数といいます。暗黙的に型指定された変数の記述法としては、以下のような形式を使用します。

```
var 変数名 = 初期値;
```

■　初期化

暗黙的に型指定された変数は、データ型を初期値から推測します。そのため、宣言時に必ず初期値を指定する必要があります。初期値を指定していない場合は、コンパイラがデータ型を推測できないため、コンパイルエラーになります。また、初期値として null を使用することもできません。

■　コード例：暗黙的に型指定された変数

```
1.  class MainClass
2.  {
3.     static void Main()
4.     {
5.        var i = 5;              //  int 型
6.        var s1 = "Hello";       //  string  型
7.        //var s2;               // コンパイルエラー
8.        //var s3 = null;        // コンパイルエラー
9.     }
10. }
```