クラス

クラスとインスタンス

オブジェクト指向(OOP)言語は、「継承」「多様性」「カプセル化」の3つの特徴を持ち合わせます
OOP はこれらの基本的な要素を「クラス」という機能で実現します

クラスは一連の論理的に関連性のある処理をまとめたデータ定義です
クラスは実世界の物「オブジェクト」を表すのに非常に優れた手段であるといえます
例えば、「猫」は種や性別などのデータを持ち、食べたり寝たりという基本動作を持ちます
クラスはデータを「フィールド」、動作を「メソッド」としてサポートします

私たちは、これまでのプログラムでも一つのクラスを作ってきました
C# プログラムは、必ず一つ以上のクラスと Main() メソッドを持つ必要があるからです
そう **class Test** という一番最初の文がクラスの宣言だったのです
ここの **class** キーワードを使用することで、クラスを宣言できます

[attributes] [modifiers] class identifier [:base-list] { class-body }[;]

attributes には属性、modifiers にはクラス修飾子をそれぞれ指定します
identifier は、宣言するクラスの識別子を、base-list はこのクラスの基底クラスなどを指定します
class-body には、このクラスの本文を記述します

attributes、modifiers、base-list に関連する技術については
現時点では説明できないので、クラスの基本を網羅してから説明します

クラスは宣言するだけでは使用することはできません
クラスを実際に**メモリに生成**する必要があります
クラスの実体を**インスタンス**と呼び、生成することを**インスタンス化**と呼びます

インスタンスの生成には **new 演算子**を使用します
new 演算子は指定したクラスの**インスタンス型の参照**を返します
インスタンス型の参照とは、すなわちインスタンスが格納されているメモリのアドレスを表します

new constructor();

constructor() には、そのクラスの「コンストラクタ」を指定します
コンストラクタはインスタンスを初期化するためのメソッドで
必ず**クラスと同じ名前**を持ち、デフォルトで存在しているメソッドです
（独自のコンストラクタの定義も可能ですが、それについては後記します）

コンストラクタが返したインスタンスの参照、すなわちオブジェクトは
一体どのような変数に格納すれば良いのでしょうか？
それは、当然その「クラス型」に型の合う変数です
Kitty クラスのインスタンスは Kitty 型変数に格納するという形になります

```
class Kitty {}

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

このプログラムは Kitty コンストラクタを呼び出しヒープにインスタンスを生成し
その参照を Kitty 型変数 obj に格納しています

WriteLine() メソッドは、このようなクラス型の変数を受け取ることができ
そのクラスの文字列情報をコンソールに出力します
どのような文字になるかはクラスによりますが、通常はそのクラスの名前が表示されるでしょう

メンバ変数

クラスの本文は「メンバ」と呼ばれるデータの定義になります
メンバは、そのクラスが持つデータや動作そのものです

クラスの持つデータは**メンバ変数**として保存します
メンバ変数は、クラスのメンバとして宣言された変数でありクラスと直接結び付けられます
メンバ変数へのアクセスは通常の変数と異なり、**インスタンスを指定**します

メンバ変数を宣言するには、クラスの本文でこれまでのように変数を宣言します
ただし、メンバは**セキュリティ**を持っています
他のクラスからアクセス可能か、あるいは外部からのアクセスを禁止するかという具合です

メンバへのアクセスの制御は、「アクセス修飾子」を最初に指定します
この場では、どこからでも自由にアクセスを許可する **public** を指定してください
例えば、次のメンバ変数はどこからでもアクセス可能な文字型の変数です

public string str;

アクセス修飾子は省略できますが、省略すると他のクラスからはアクセスできません

アクセス制御の詳細は、C#のクラスの基本を網羅してから詳しく説明します

メンバ変数は、クラスのインスタンスごとに独自のメモリ領域を持ちます
何らかのメンバへアクセスするには、常に次のような形を持ちます

object . member

object には、そのメンバを持つオブジェクトを指定します
member は、オブジェクトのアクセスしたいメンバを指定します

```
class Kitty {
 public string name;
}

class Test {
 static void Main() {
  Kitty obj = new Kitty();
  obj.name = "Rena";
  System.Console.WriteLine(obj.name);
 }
}
```

このプログラムは Kitty クラスのインスタンスを生成し
Kitty クラスのメンバ変数 name に文字列を代入し、それを出力しています

これは、obj オブジェクトが持つ name というプロパティを設定したという概念になります
このプロパティをどのように使うかは、そのクラスやプログラマに委ねられます
メンバ変数は、通常は参照する前に初期化する必要があります

しかし、上のプログラムだけでは一体何がクラスのメリットなのか理解できないかもしれません
たしかに Kitty クラスのオブジェクトを一つしか扱わないなら、クラスの意味はありません
オブジェクト指向のメリットは、これを再利用するということにあります

例えば、あなたが猫の管理をコンピュータを使って行う場合
上のように一度 Kitty クラスを定義してしまえば、あなたが何匹猫を飼っても
ゼロからその猫のためにプログラムを記述する必要はなく
Kitty のインスタンスを一つ生成すれば、それだけで新しい猫を管理できるのです

```
class Kitty {
 public string name;
}

class Test {
 static void Main() {
  Kitty rena = new Kitty();
  Kitty yuki = new Kitty();
  Kitty mimi = new Kitty();

  rena.name = "RENA";
  yuki.name = "YUKI";
  mimi.name = "MIMI";

  System.Console.WriteLine(rena.name);
  System.Console.WriteLine(yuki.name);
  System.Console.WriteLine(mimi.name);
 }
}
```

rena、yuki、mimi の3つの Kitty クラスのインスタンスを生成しています
メンバはインスタンスごとにメモリの独自の領域に割り当てられているので
例えば、rena.name と yuki.name はまったく別のメモリの位置を表します

これで、三匹の猫の名前を管理することができます
Kitty クラスを書きなおして、さらに新しいメンバ変数を追加すれば
多くの猫に関するプロパティを管理することができるのです

このように、オブジェクトのデータを管理するメンバ変数群を**フィールド**とも呼びます
この辺の用語はずいぶんと入り組んでいますが、整理してください
基本的に、オブジェクトのデータとはフィールドのことであり
フィールドの実体はメンバ変数であるという感覚です

因みに、フィールドには**クラス型の変数を含める**ことも可能です
その場合 A クラスのメンバ変数に B クラス型変数があるとして
A クラスのメンバ変数の B のメンバ変数 C にアクセスする時は
A . B . C というように記述してアクセスすることができます

```
class B {
 public string str;
}

class A {
 public B objB;
}

class Test {
 static void Main() {
  A objA = new A();
  objA.objB = new B();
```

```
    objA.objB.str = "Kitty on your lap";
    System.Console.WriteLine(objA.objB.str);
  }
}
```

このプログラムは A クラスのフィールドに B クラス型メンバ変数を持ちます
B クラスは string 型のメンバ変数を持っています

A クラスのオブジェクト objA が持つ B クラスのオブジェクト objB にアクセスするには
objA . objB というようにし、そこからさらに objA . objB . str とすることで
objB のメンバにアクセスすることができます
この入れ子関係は、例え何段階になっても可能になっています

---

class

[attributes] [modifiers] class identifier [:base-list] { class-body }[;]

クラスを宣言します

attributes - 属性を指定します
modifiers - アクセス修飾子を指定します
identifier - クラスの名前を指定します
base-list - 基底クラスを指定します
class-body - クラスの本体を定義します

new constructor()

指定コンストラクタを呼び出しインスタンスを生成します

constructor() - インスタンスを生成するクラスのコンストラクタを指定します

前のページへ　戻る　次のページへ