コンストラクタ

インスタンスの初期化

これまで、クラスのメンバ変数はインスタンスを生成した後
それぞれのインスタンスに対して個別の初期化してきました

これは、1つや2つのオブジェクトに対しては対した作業量にはなりませんが
多くのメンバ変数をもつクラスのインスタンスを複数初期化するとなると話しは別です
そこで、インスタンスの初期化には**コンストラクタ**を使います

コンストラクタは、基本的にはメソッドと変わりませんが二つの特徴を持ちます
一つは**クラスと同じ名前**であり、クラスと同じメソッドはコンストラクタとなります
もう一つは**戻り値は無い**ため、コンストラクタには戻り値は指定しません

なぜ戻り値が無いのかというと、コンストラクタの戻り値はインスタンスへの参照であり
コンストラクタはインスタンス生成時以外に呼び出されることは無いからです
これ以外の点では、基本的にメソッドと同じで内部で何をしようがプログラマの勝手です
ただし、論理的に初期化と関連のある処理を記述することが美しいプログラムへの道です

```
class Kitty {
 public string strName , strSex;
 public Kitty(string str , bool bl) {
  strName = str;
  strSex = bl ? "雌猫" : "雄猫";
 }
 public void Write() {
  System.Console.WriteLine(
   "名前 = " + strName + " : 性別 = " + strSex
  );
 }
}

class Test {
 static void Main() {
  Kitty rena = new Kitty("RENA" , true);
  Kitty kaimu = new Kitty("カイム" , false);

  rena.Write();
  kaimu.Write();
 }
}
```

このプログラムの Kitty クラスは、インスタンスを生成する時に
Kitty(string , bl) 型のコンストラクタを呼び出し、メンバを初期化します

コンストラクタの戻り値は無いので、Kitty() には戻り値の指定が無いことに注意してください
void は何も値を返さないという意味ですが、コンストラクタはそれ以前に戻り値の指定をしません

コンストラクタが宣言されていないクラスでも、デフォルトコンストラクタが存在します
デフォルトコンストラクタは、何も引数を受け取らない public なコンストラクタです

フィールドの初期化

メンバ変数はコンストラクタ以外の方法でも初期化することが可能です
それは、あらかじめ**静的に**デフォルトの値を与えることです

これは、通常の変数の初期化のように代入演算子を用いて
メンバ変数に対してデフォルトの値を代入します
そうすると、インスタンスが生成されてコンストラクタが実行された直後に
そのメンバ変数に対して指定の値が代入されます

```
class Kitty {
 public string name = "RENA" , sex = "雌猫";
 public void Write() {
  System.Console.WriteLine("名前 = " + name + " : 性別 = " + sex);
 }
}

class Test {
 static void Main() {
  Kitty rena = new Kitty();
  Kitty yuki = new Kitty();
  yuki.name = "YUKI";

  rena.Write();
  yuki.Write();
 }
}
```

このプログラムを実行すると次のような結果が得られます

名前 = RENA : 性別 = 雌猫
名前 = YUKI : 性別 = 雌猫

Kitty クラスのメンバ変数 name と sex はそれぞれがデフォルトの値を持ちます
コンストラクタや明示的な代入が無ければデフォルトの値のままなので

結果として上のようなものが出力されたのです

---

null アクセス

C# では、初期化されていないメンバ変数の初期値というものが割り当てられています
そのため、初期化されていないメンバ変数にアクセスしても
最低限の動作はある程度守られるようにはなっています

ところが、クラス型のような「参照型」の変数の初期値はどうでしょう
「参照型」とは、実体が存在するメモリのアドレスを表す変数です
これは実体が存在して初めて保障されるものですから
インスタンス化される前は初期化のしようがありません

実は、このような参照型の初期値には **null リテラル** が割り当てられます
null はリテラルであり、どこへの参照も持っていない状態であることを意味します
プログラムのクラス型変数の保証はコンパイル時にはわかりません
そこで、もしプログラムが null へアクセスした場合 CLR はプログラムを停止させます

```
class Tmp {
 public string tmp;
}

class Kitty {
 public Tmp obj;
}

class Test {
 static void Main() {
  Kitty rena = new Kitty();
  rena.obj.tmp = "Kitty on your lap";
 }
}
```

このプログラムを実行すると**実行時例外**が発生します
例外とは、コンパイルエラーと異なり実行中の動的なエラーです

Kitty クラスにある Tmp クラス型のメンバ変数はデフォルトで null であり
これを初期化しないまま、Main() メソッドで Kitty.obj.tmp とアクセスしています
null、すなわちインスタンスが存在せず、これは実行できないので例外となるのです

例外の生成と処理方法については、後ほど詳しく解説します
この場では、null アクセスはプログラムを停止させてしまうということを知ってください

前のページへ　戻る　次のページへ