値型と参照型

値渡しと参照渡し

これまで、何度かオブジェクトは**参照型**であると説明しました
参照とはメモリの有効なアドレスを表す数値であり、それ自体に意味はありません
プログラムは参照を元にメモリの指定アドレスにアクセスしてインスタンスを操作します

これと対象の int などの変数は直接データそのものを保有する「値型」と呼びます
参照型と値型には、その性質から非常に興味深い動作をします

値型は、他の変数やメソッドにデータを転送する時は**内容をコピー**します
メソッドはパラメータを受け取りますが、それはコピーされた値です
つまり、メソッド内で受け取った値を変更しても**呼び出し元に影響はない**のです

```
class Test {
 static void Main() {
  int x = 0;
  getValue(x);
  System.Console.WriteLine(x);
 }
 static void getValue(int x) {
  x = 10;
 }
}
```

変数 x を Main() メソッドで宣言し、これを制的メソッド getValue() に渡します
getValue() では受け取った x に対して 10 を代入して制御を戻します
その後 WriteLine() で x を出力していますが、この結果はどうなると思いますか？
答えは 0 が出力されます

メソッドに値型が渡された時、メモリアドレスではなくデータのコピーが渡されます
そのため getValue() で x の値を変更しても、元の Main() 関数の x には関係がないのです

ところが、参照型の場合は話しが違います
参照型変数が持つ値は**メモリアドレス**そのものです
この値自体はコピーされてメソッドに渡されますが、値が参照するアドレスは同じです
つまり、どの位置から呼び出してもアクセスするメモリアドレスは同じということになります

```
class Test {
 string str;

 static void Main() {
  Test obj = new Test();
  Test tmp = obj;
  tmp.str = "Kitty on your lap";
  System.Console.WriteLine(obj.str);
 }
}
```

まず、Test クラスのインスタンスを生成し obj にその参照を格納ています
obj には何の操作もしないまま、次に別の Test 型の tmp 変数に参照を代入します
この時**インスタンスは複製されず**参照のコピーだけが渡されます

そして、tmp.str メンバ変数を初期化し WriteLine() で obj.str を出力します
変数 obj のメンバは初期化していないはずですが、このプログラムの結果は
Kitty on your lap という文字列をコンソールに出力します

なぜならば、変数 tmp も obj も同じメモリアドレス（すなわちインスタンス）を指しているわけで
そのうちどれを使ってアクセスしても意味は同じなのです

変数 obj 変数 tmp
　↓　　↓
インスタンス

無論、メソッドにコピーされるのも参照型の場合はメモリアドレスを表す値であり
この値を使ってアクセスする以上、参照するインスタンスは同じものです

```
class Test {
 string str;

 static void Main() {
  Test obj = new Test();
  getTest(obj);
  System.Console.WriteLine(obj.str);
 }
 static void getTest(Test obj) {
  obj.str = "Kitty on your lap";
 }
}
```

このプログラムは Kitty on your lap と文字列を出力します

参照を渡す

さて、値型変数を受けとってその内容を変更しても呼び出し元に影響はありませんでした
しかし、値型変数を受けとって呼び出し元の変数の内容を変更したい場合はどうしましょう

C言語ではポインタを使い、C++ や Perl 言語では参照を渡し
Java ではその方法を提供しない(それでも問題はない)方法をとっています

C# は **ref メソッドパラメータ** キーワードを用いてこれを実現します
ref キーワードはメソッドの仮引数指定時に型名の前に指定します
ref 修飾子を指定したパラメータを**参照パラメータ**と呼びます
参照パラメータは変数の値ではなく参照(メモリアドレス)を受け取ります

ただし、**呼び出し側も ref を指定**する必要があります
明示的に ref を指定してメソッドを呼び出すことで、プログラマはこの変数の内容が
メソッドによって変更される可能性があることを明確に知ることができます

```
class Test {
 static void Main() {
  int x = 0 , y = 0;
  getLocation(ref x , ref y);
  System.Console.WriteLine("x = " + x + " : y = " + y);
 }
 static void getLocation(ref int x , ref int y) {
  x = 100;
  y = 20;
 }
}
```

getLocation() メソッドは、座標を取得するメソッドをシュミレートしています
このような関連した二つの値を返すメソッドは
一つの値しか返せない戻り値では値を取得できません

そこで、ref 修飾子を指定してその変数の参照を受け取れば
メソッド内での変更は呼び出しもとの変数そのものに影響します
上のプログラムを実行すると x は 100、y は 20 という値を出力します

ref 修飾子で受ける変数は**必ず初期化されている**必要があります
初期化されていない変数を渡そうとするとコンパイルエラーが発生します
ところが、メソッドで変数を上書きする場合は意味のない初期化となってしまいます

受け取る参照が指す変数が初期化されている必要がない場合
ref ではなく **out メソッドパラメータ** キーワードを使用します
これは、参照先が初期化されている必要がないという以外で ref と同じです
out 修飾子を指定したパラメータを**出力パラメータ**と呼びます

```
class Test {
 static void Main() {
  int x , y;
  getLocation(out x , out y);
  System.Console.WriteLine("x = " + x + " : y = " + y);
 }
 static void getLocation(out int x , out int y) {
  x = 100;
  y = 20;
 }
}
```

このプログラムは先ほどのプログラムと同じですが
Main() メソッドの x と y 変数を初期化していないという点で違います
メソッドでの上書きのみを目的とする場合は out を指定しましょう
逆に、読み込みを行う可能性がある場合は ref を使って初期化を促します

---

ref

メソッドパラメータに指定し引数が
呼び出しもとの変数のポインタであることをコンパイラに通知します
変数は初期化されていなければなりません

out

メソッドパラメータに指定し引数が
呼び出しもとの変数のポインタであることをコンパイラに通知します

前のページへ　戻る　次のページへ