かるぼ式メモリー図ですべて解決 ～C#オブジェクト指向編～

By 刈歩 菜良 CTP featuring ちゃま いかを

【謝辞】
本プレゼンテーションは prototype.js の大家 shibuya.js の amachang 様の御好意で amachang 様のスクリプトを流用させて頂いております。
amachang 様のブログ ・・・ 「IT 戦記」 http://d.hatena.ne.jp/amachang/

前回までの復習

■値型と参照型

型定義時のコーディング上の違いは

値型 = **struct** キーワードで定義　　参照型 = **class** キーワードで定義

変数宣言時にデータを格納する場所が

ある → 値型　　**ない** → 参照型

変数宣言時の領域に格納されるのは

データそのもの → 値型　　**参照情報** → 参照型

■引数のデータの渡し方

引数でデータが渡されるのは

値型→変数の中身が**そのまま**渡される　　参照型→変数の中身が**そのまま**渡される

参照渡しの場合は

渡す変数に**別名が付く**と考えるとイメージしやすい。

復習問題 （できればセッション前にやってみてください。）

以下のコードをメモリー図で表現してください

①値型
```
int num;
num = 15;
```

②参照型
```
string str;
str = "愛田前";
```

③1 次元配列（値型） … 以降は、配列に入る値は省略されていますが、図には記述してください。
```
int[] arr;
arr = new int[5];
```

④2 次元配列（値型）
```
int[,] ar2;
ar2 = new int[2, 3];
```

⑤3 次元配列（値型）
```
int[,,] ar3;
ar3 = new int[2, 2, 2];
```

⑥ジャグ配列（値型）
```
int[][] j_ar2;
j_ar2 = new int[2][];
j_ar2[0] = new int[3];
j_ar2[1] = new int[5];
```

⑦1 次元配列（参照型）
```
string[] arr;
arr = new string[5];
```

⑧2 次元配列（参照型）
```
string[,] ar2;
ar2 = new string[2, 3];
```

⑨3 次元配列（参照型）
```
string[,,] ar3;
ar3 = new string[2, 2, 2];
```

⑩ジャグ配列（参照型）
```
string[][] j_ar2;
j_ar2 = new string[2][];
j_ar2[0] = new string[3];
j_ar2[1] = new string[5];
```

⑪引数のデータの渡し方
```
int x = 0;
int[] arr = {1, 2, 3};
method(x, arr);

public void method(int i, int[] a)
{
    i = 50;
    a[0] = 100;
}
```

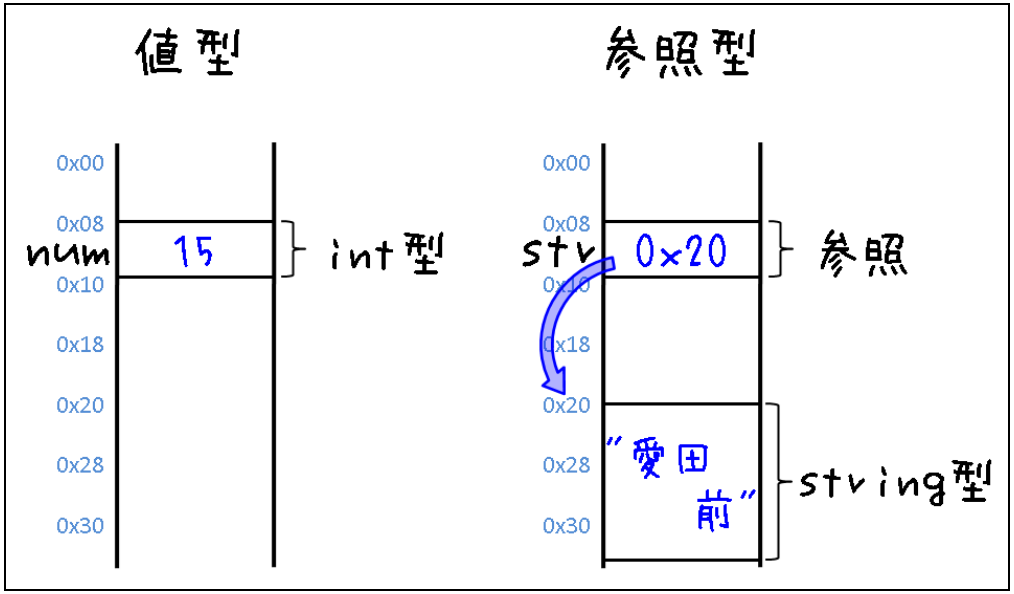
⑫引数のデータの渡し方(参照渡し)
```
int x = 0;
int[] arr = {1, 2, 3};
method(ref x, ref arr);

public void method(ref int i, ref int[] a)
{
    i = 50;
    a[0] = 100;
}
```

前回までのまとめ

・シンプルに考えよう
・覚えることは少なく

最低限これだけ覚えればＯＫ



本日のお題

■オブジェクト指向(前編)

```
oreClass oc;
oc = new oreClass();
oc.id = 33;

class oreClass
{
    public int id;
}
```

・コンストラクタ＆カプセル化

```
oreClass oc1, oc2;
oc1 = new oreClass(33);
oc2 = new oreClass(51);
x = oc1.getId();
y = oc2.getId();
x += 1;

class oreClass
{
    private int id;

    public oreClass(int i)
    {
        id = i;
    }

    public int getId()
    {
        return id;
    }
}
```

・参照型のフィールド（配列型）

```
oreClass oc1;
oc1 = new oreClass(33, 51);
arr = oc1.getIdArray();
arr[0] += 1;
arr = new int[3]{5, 10, 15};

class oreClass
{
    private int[] idArray;

    public oreClass(int a, int b)
    {
        idArray = new int[2]{a, b};
    }

    public int[] getIdArray()
    {
        return idArray;
    }
}
```

・参照型のフィールド（string 型）

```
oreClass oc1, oc2;
oc1 = new oreClass("かるぼ");
oc2 = new oreClass("ぽぴ");
str1 = oc1.getName();
str2 = oc2.getName();
str1 += "なら";

class oreClass
{
    private string name;

    public oreClass(string n)
    {
        name = n;
    }

    public string getName()
    {
        return name;
    }
}
```

・プロパティ

```
oreClass oc1;
oc1 = new oreClass(33, 51);
oc1.Ids[0] += 1;
oc1.Ids = new int[3]{5, 10, 15};

class oreClass
{
    private int[] idArray;

    public oreClass(int a, int b)
    {
        idArray = new int[2]{a, b};
    }

    public int[] Ids
    {
        get
        {
            return idArray;
        }
        set
        {
            idArray = value;
        }
    }
}
```

・インデクサ

```
oreClass oc1, oc2;
oc1 = new oreClass(33, 51);
oc1[0] += 1;
oc1 = new int[3]{5, 10, 15};

class oreClass
{
    private int[] idArray;

    public oreClass(int a, int b)
    {
        idArray = new int[2]{a, b};
    }

    public int this[int index]
    {
        get
        {
            return idArray[index];
        }
        set
        {
            idArray[index] = value;
        }
    }
}
```

・set アクセッサがなかったらどうなる？

次回予告

いよいよ最終回！！

静的フィールド/メソッド、継承，インターフェイス，ポリモフィズム など