C#講習会　その2

基本編

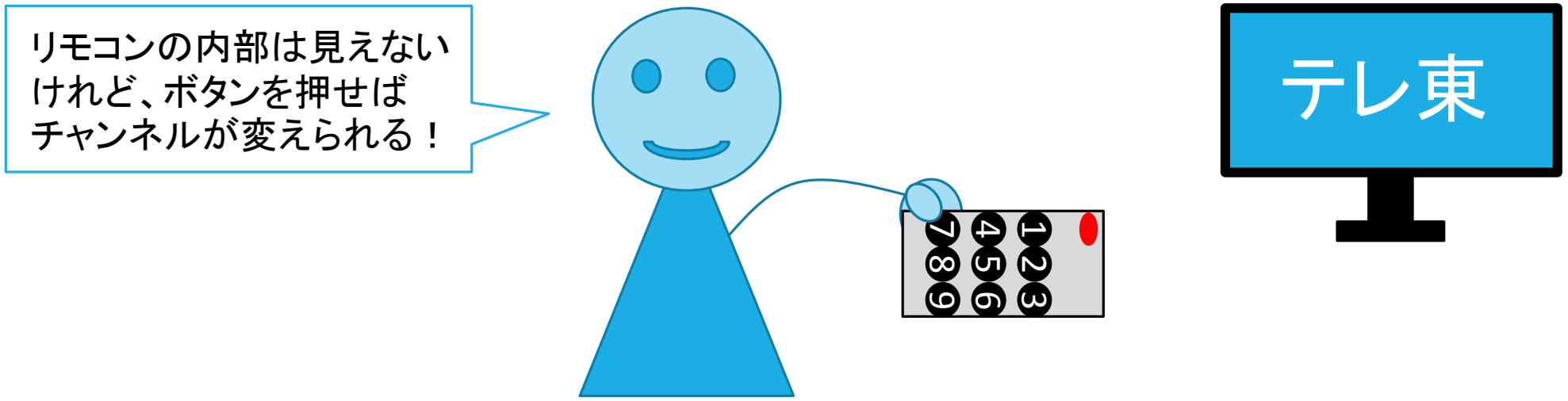
今回の内容

- カプセル化
- 継承
- 多態性

カプセル化

カプセル化とは？

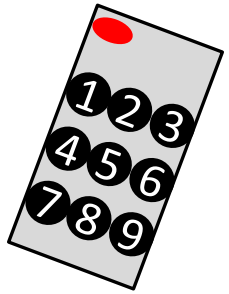
- クラスの内部実装を隠蔽すること
  - クラス利用者から中身が見えないようにする



カプセル化の目的

- 不正な書き換えを防止する
- 実装を変更したときに、利用者側の操作を変更する必要をなくす

利用者は内部構造をいじられない



内部構造が変わっても、利用者側の操作はボタンを押すだけ

アクセシビリティ

◆クラスのメンバ変数やメソッドに対して、どこからアクセスできるかという制限の度合い

アクセス装飾子	説明
public	どこからでもアクセス可能
private	クラス内部からのみアクセス可能
protected	クラス内部と、派生クラスの内部からのみアクセス可能

プロパティ

◆クラス外部から見るとメンバー変数のように振る舞い、クラス内部から見るとメソッドのように振舞うもの

考え方としては…

●メンバー変数はクラス外部から直接アクセス出来ないようにして、オブジェクトの状態の変更はすべてメソッドを通して行うべきだということ

プロパティの定義

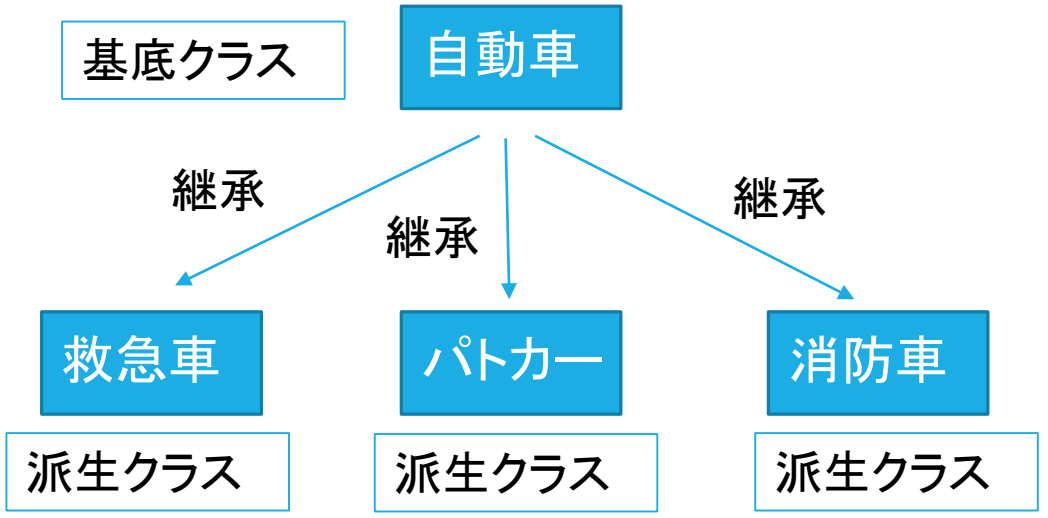
```
アクセス装飾子　型名　プロパティ名
{
        set {    値変更時の処理を書く        }

        get {    値取得時の処理を書く        }
}
```

set、getの中身は省略することもできる（自動プロパティ）

継承

継承とは？

- あるクラスから性質を受け継いだ新しいクラスを作ること
- クラスBがクラスAを継承したとき、クラスAを基底クラス、クラスBを派生クラスという



クラスの継承

```
class　派生クラス名　：　基底クラス名
{
　　派生クラスの定義
}
```

- 派生クラスは基底クラスのメンバ変数、メソッドを継承する

protected

- 基底クラスのprivateは派生クラスからアクセスできないが、protectedは派生クラスからアクセスできる

```
1   using System;
2
3   class Base
4   {
5       public int public_val;
6       private int private_val;
7       protected int protected_val;
8
9       public Base()
10      {
11          public_val = 0;
12          private_val = 0;
13          protected_val = 0;
14      }
15  }
```

```
16
17  class Derivation : Base
18  {
19      public Derivation()
20      {
21          public_val = 1; //アクセス可
22          private_val = 1; //アクセス不可
23          protected_val = 1; //アクセス可
24      }
25  }
```

コンストラクタ呼び出し

- 派生クラスのインスタンスが生成されたとき、派生クラスのコンストラクタが呼び出される前に、基底クラスのコンストラクタが呼び出される

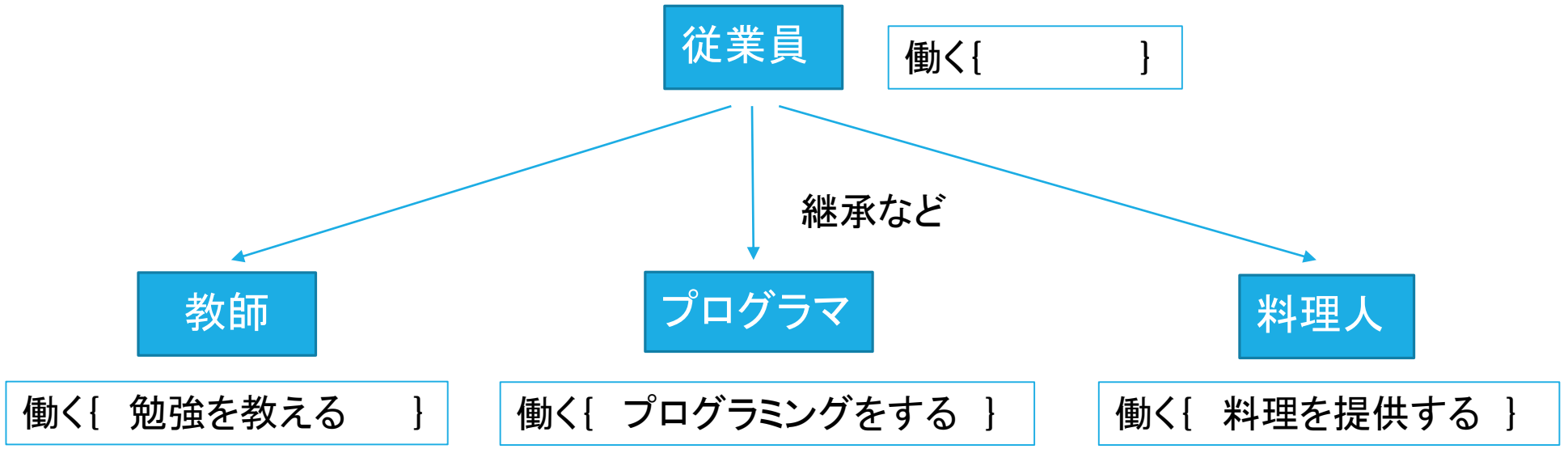
基底クラスのコンストラクタを明示的に呼び出す方法

```
派生クラスのコンストラクタ(引数) : 基底クラスのコンストラクタ(引数)
{
}
```

多態性

多態性とは？

◆同一のメソッド呼び出しに対して、オブジェクトによって異なる振る舞いをすること



オーバーロード

◆引数の型または引数の数が異なる同じ名前のメソッド、またはコンストラクタを同一クラス内で定義すること（多重定義）

```
using System;

class friends
{
    public string name { get; }
    public string animal { get; }

    public friends()
    {
        name = "NONAME";
        animal = "UNKOWN";
    }

    public friends(string name, string animal)
    {
        this.name = name;
        this.animal = animal;
    }
}
```

```
class Sample
{
    static void Main()
    {
        friends cellian = new friends();
        friends serval = new friends("サーバル", "サーバルキャット");

        Console.WriteLine("名前 : {0} 動物 : {1}", cellian.name, cellian.animal);
        Console.WriteLine("名前 : {0} 動物 : {1}", serval.name, serval.animal);
    }

}
```

仮想メソッド

- virtual装飾子をつけたメソッドを仮想メソッドという

- 仮想メソッドは派生クラスでオーバーライドすることができる

オーバーライド

- 仮想メソッドを派生クラスで再定義すること
- オーバーライドする際にはメソッドにoverride装飾子をつける

```
1   using System;
2
3   class Worker
4   {
5       public virtual void Work() { Console.WriteLine("働く"); }
6   }
7
8   class Programmer : Worker
9   {
10      public override void Work()
11      {
12          Console.WriteLine("プログラミングをする");
13      }
14  }
```

```
16  class Polymorphism
17  {
18      public static void Main()
19      {
20          Worker person1 = new Worker();
21          Programmer person2 = new Programmer();
22
23          person1.Work();
24          person2.Work();
25      }
26  }
```