C++講座補講
クラスの使い方

以下、名無しに変わりましてVIPが補講します

今日の内容

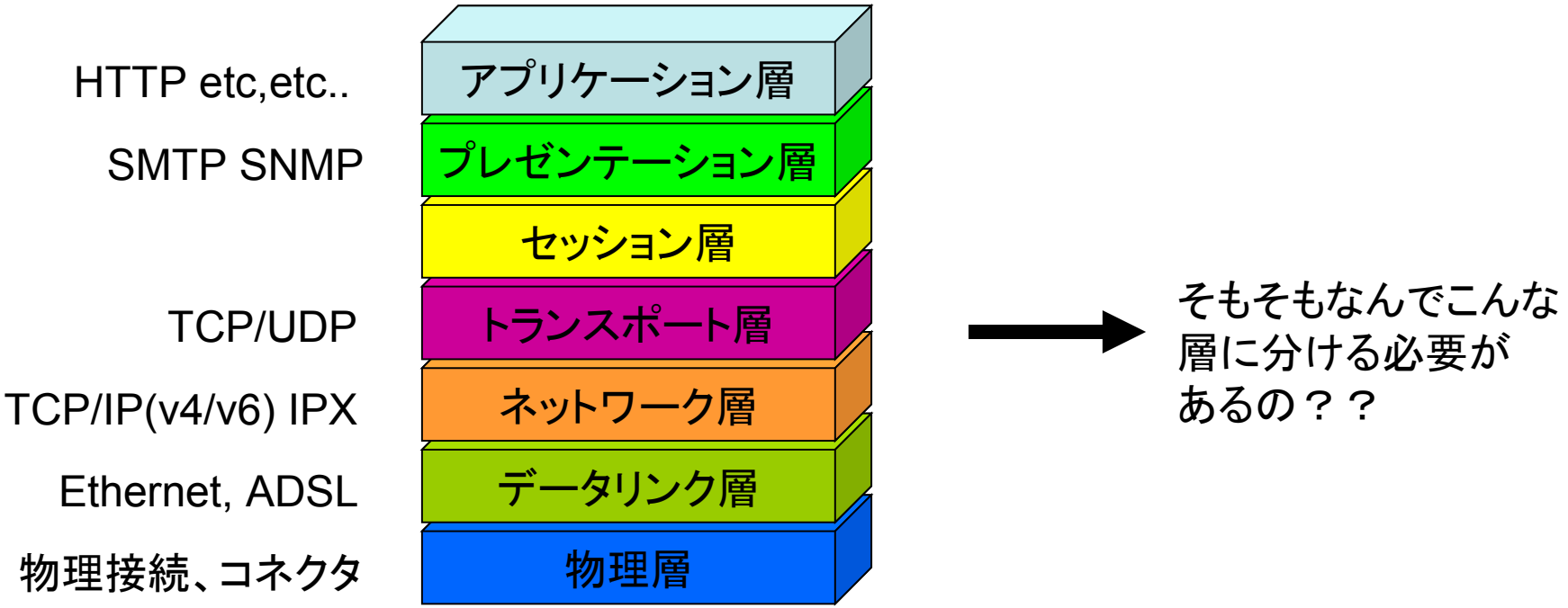
- インターフェースとレイヤリング
- クラスの作り方

抽象化とレイヤリング

- オブジェクト指向プログラミングにおける抽象化とは？
  - 各々の事象をより「一般化」させること
  - そもそもこの定義自体が抽象的すぎますが・・・
- レイヤリング
  - システムをそれぞれ「階層化」して分割する

レヤリングモデル

- 必ず出てくるOSI参照モデル
  - 正直「もう見飽きた」という人が多いでしょうが・・・



レイヤリング無しで
ブラウザのプログラムを書く

/* URL をサーバに送る */
＃場合１：TCP/IP を使ってる場合　・・・・・
＃場合１の１：無線を使ってる場合　・・・・・・
＃場合１の１の１：暗号化無し無線使ってる場合・・・・・
＃場合１の１の２：WEPを使ってる場合・・・
＃場合１の１の２の１：802.11a を使ってる場合
　電波をこういう波形でこういう風に出して、こういう電波がきたらどうこう

＃場合１の１の２の２：802.11bを使ってる場合
　電波をこういう波形でこういう風に出して、こういう電波がきたらどうこう

＃場合１の２：有線を使ってる場合
＃場合１の２の１：Ethernetを使ってる場合
＃場合１の２の１の２：1000Base-T を使ってる場合
　電線にこういう風に電気を流して、こういう電気が来たら・・・
　　：（中略）
＃場合２：IPXを使ってる場合・・・・

ブラウザのプログラムを書く

```
/* URL をサーバに送る */
#場合1：TCP/IP を使ってる場合 ・・・・・
#場合1の1：無線を使ってる場合 ・・・・・・
#場合1の1の1：暗号化無し無線使ってる場合・・・・・
#場合1の1の2：WEPを使ってる場合・・・
#場合1の1の2の1：802.11a を使ってる場合
  電波をこういう波形でこういう風に出して、こういう電波がきたらどうこう

#場合1の1の2の2：802.11bを使ってる場合
  電波をこういう波形でこういう風に出して、こういう電波がきたらどうこう

#場合1の2：有線を使ってる場合
#場合1の2の1：Ethernetを使ってる場合
#場合1の2の1の2：1000Base-T を使ってる場合
  電線にこういう風に電気を流して、こういう電気が来たら・・・
     :（中略）
#場合2：IPXを使ってる場合・・・・
     :
     :
```

```
 ∧_∧
(･ω･;)
U　 U)
 し ⌒J
```

・・・・さすがにこんなことまで
把握してプログラム書けないんよ

```
    |
  ＼ ＿ ／
＿ （m） ＿
    |ミ|
  ／ ｀ ＼
```

ピコーン

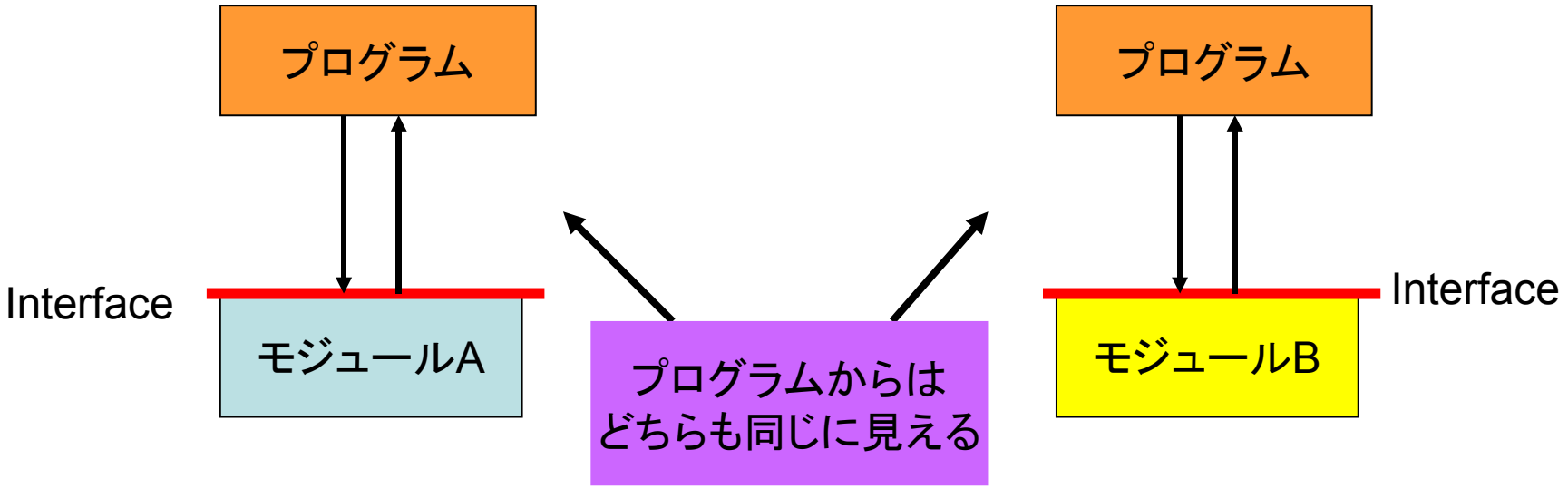
```
 ∧_∧
( ･ω･)
(U　U
 し ⌒J
```

各部分はそれぞれ専門の奴に
書いてもらえばいんじゃね？

→ 抽象化の概念

抽象化

- 階層ごとに「インターフェース」を決めて、それぞれを分離する
- 階層の上のプログラムはインターフェースの向こう側に誰がいるか関知しない



身近な(？)例

- デバイスドライバ
  - プログラムを書く人はどんなデバイスが繋がっているか意識せずに書ける
  - DirectX
    - グラボがRadeonだろうがGeForceだろうがお構いなし
    - (・・・・でも、最近は2強だし、特定ボード用にチューンすることも珍しくないけどね・・・・)
- プラグイン

ブラウザのプログラムを書い「た」が・・・

```
/* URL をサーバに送る */
＃場合1：TCP/IP を使ってる場合　・・・・・
＃場合1の1：無線を使ってる場合　・・・・・・
＃場合1の1の1：暗号化無し無線使ってる場合・・・・・
＃場合1の1の2：WEPを使ってる場合・・・
＃場合1の1の2の1：802.11a を使ってる場合
  電波をこういう波形でこういう風に出して、こういう電波がきたらどうこう

＃場合1の1の2の2：802.11bを使ってる場合
  電波をこういう波形でこういう風に出して、こういう電波がきたらどうこう

＃場合1の2：有線を使ってる場合
＃場合1の2の1：Ethernetを使ってる場合
＃場合1の2の1の2：1000Base-T を使ってる場合
  電線にこういう風に電気を流して、こういう電気が来たら・・・
      :（中略）
＃場合2：IPXを使ってる場合・・・・
      :
      :
```

```
   :Λ_Λ
:((ﾟωﾟ`)):  な、何日も徹夜してやっと
    U　U)　　書き上げたお・・・・・・
    し ⌒J
```

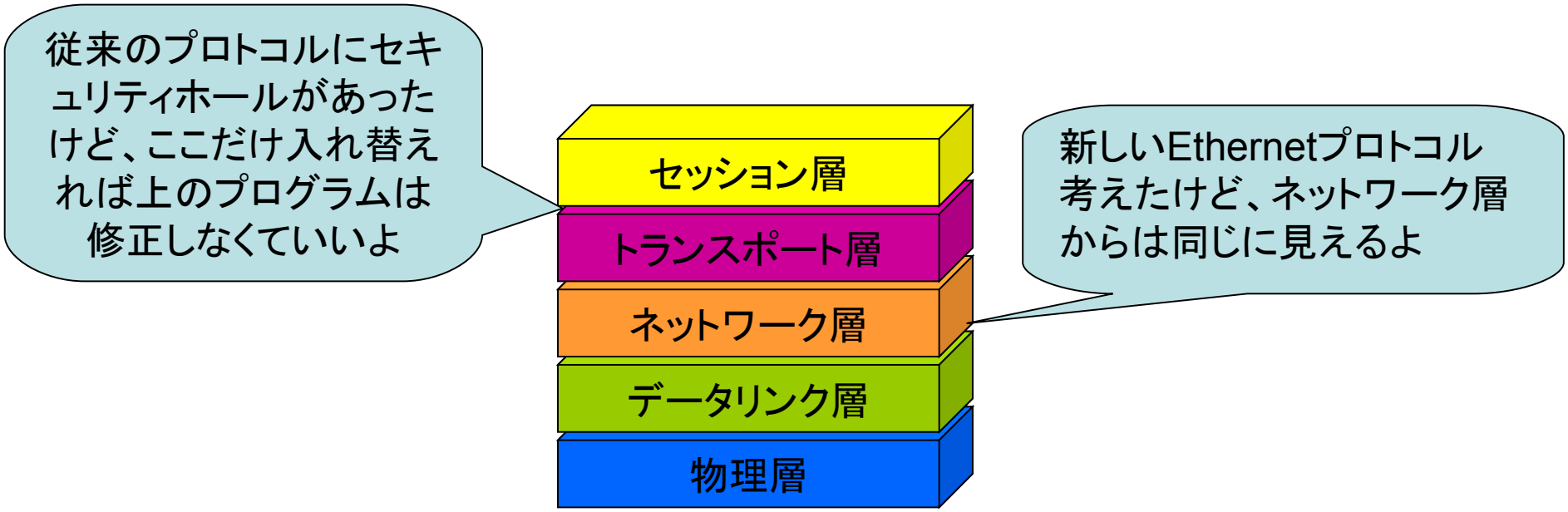
```
   :Λ_Λ　　！？
:((´ﾟ'ωﾟ)):
    U　U)
    し ⌒J
```

今日C社から新しい無線カードが出たそうなんで、対応してくださいね

```
     Λ_Λ
y=-(　ﾟωﾟ) ・゛・;; ﾀｰﾝ
   |　 U
   し ⌒J
```

抽象化の利点(2)

- 階層ごとに入れ替えができれば楽



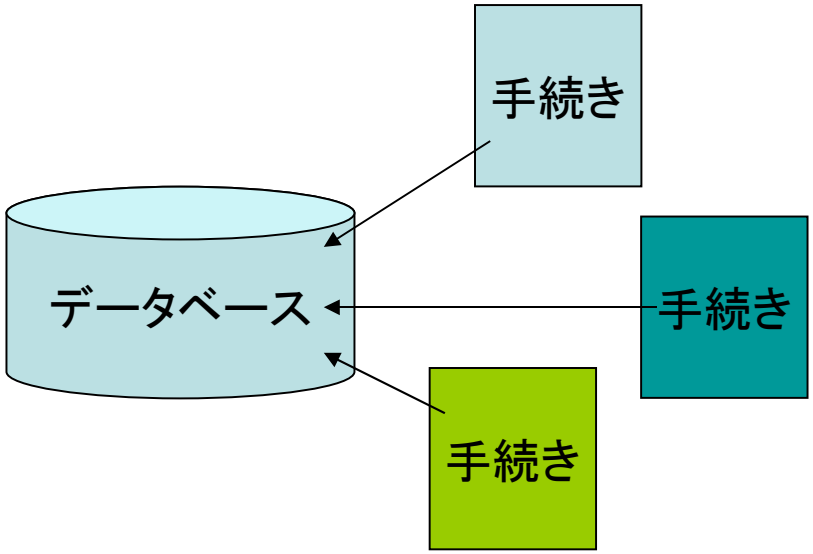
ここから言語の話

実際の業務システムの例

- レンタルビデオ屋を考えて見ましょう
- どんな処理が必要になるか？
- 単純に考えて・・・・
  - 入会・退会
  - レンタル・返却
  - 延長・延滞・督促処理
  - Etc.etc.…

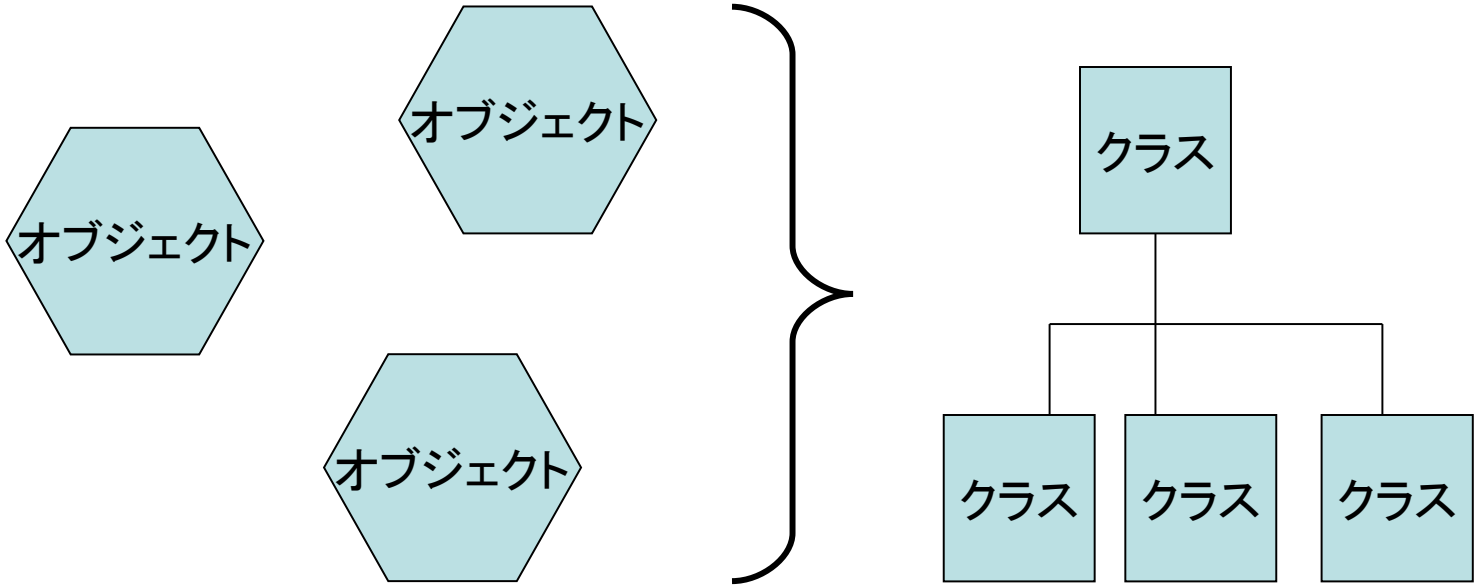
手続き型で考えると

- とりあえずデータを作る
  - 顧客データ、作品データ
- それぞれのデータに関する手続きを作る
  - 入会・退会・レンタル・返却・・・・・



オブジェクト指向で考えると

- 登場する物事をオブジェクトとして考えて
- それぞれのオブジェクトに対して一般化してみる（汎化）
- クラスを作る



レンタルビデオ屋にはどんなオブジェクトがある？

- とりあえず考えて見ましょう

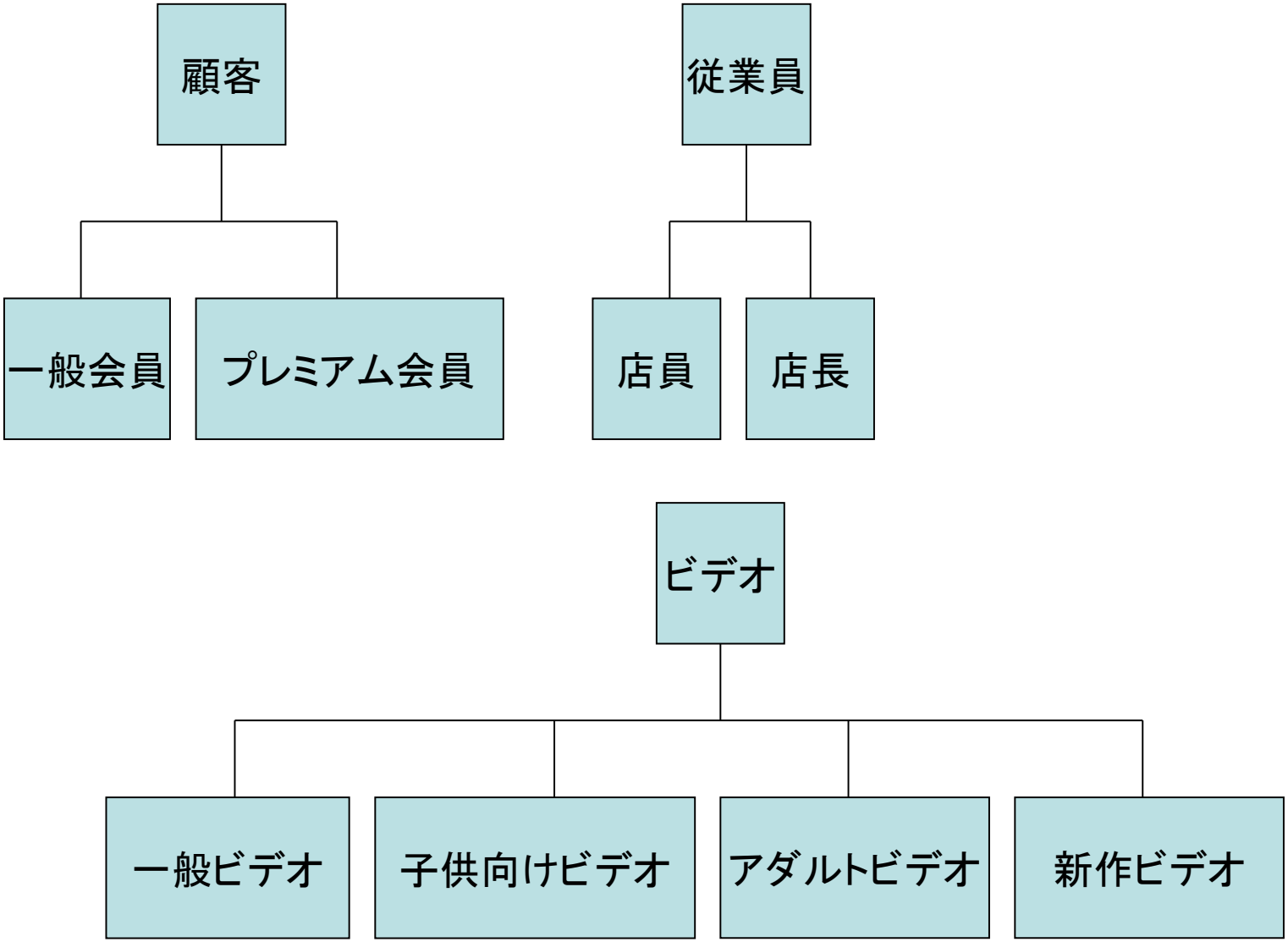
レンタルビデオ屋にはどんなオブジェクトがある？

- お客
- 店員
- 店長
- 新作ビデオ
- プレミアム会員
- ビデオ
- アダルトビデオ
- 子供向けビデオ

オブジェクトをまとめてみよう

- それぞれの登場オブジェクトは、上位の概念でまとめられる
  - 顧客
    - お客
    - プレミアム会員
  - 従業員
    - 店長
    - 店員
  - ビデオ
    - 一般ビデオ
    - 子供向けビデオ
    - アダルトビデオ
    - 新作ビデオ

これらを元にクラスを作るとすると？

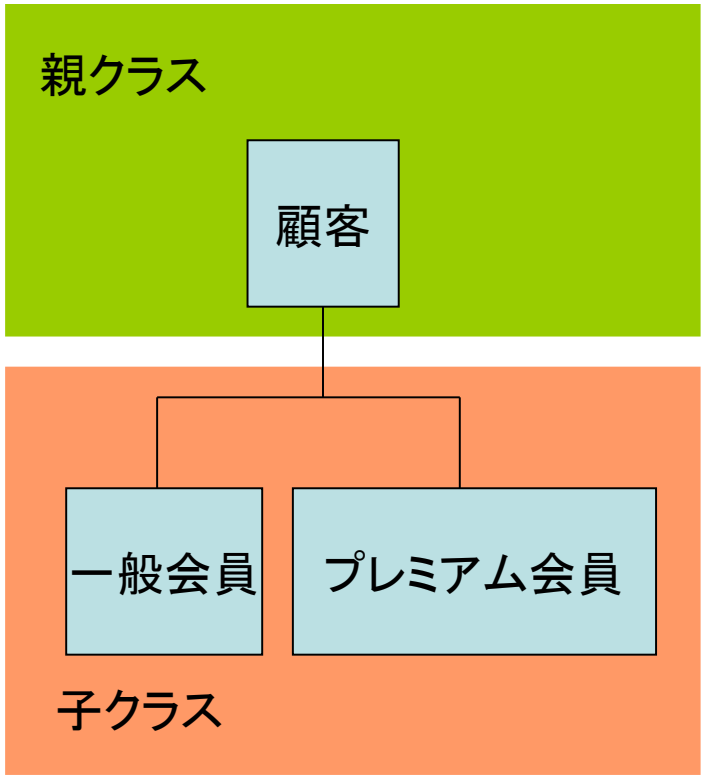


継承関係で書くと・・・

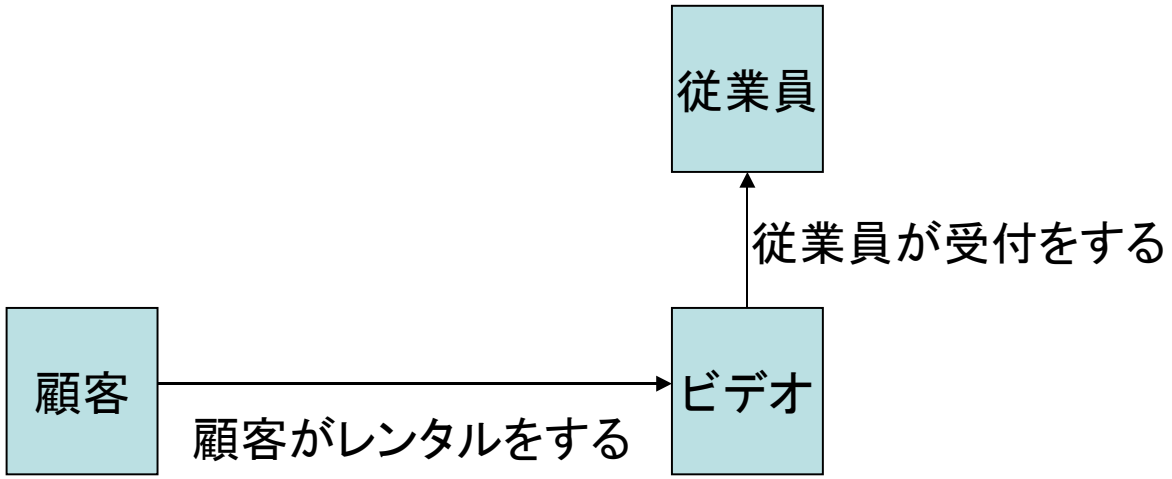
```
class Kokyaku {
        string name;
        …
        …
};

class Ippan : private Kokyaku {
        …
        …
};

class Premium : private Kokyaku {
        …
        …
};
```



それぞれのクラスの関係は？

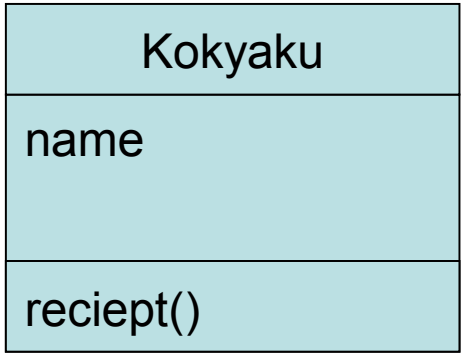


では、レシートを作ってみよう

- Receiptメソッド
- どこに作るのがいいか？

- 顧客に出すので顧客クラスかな？
  - これもいくらでも選択肢がある

顧客クラス



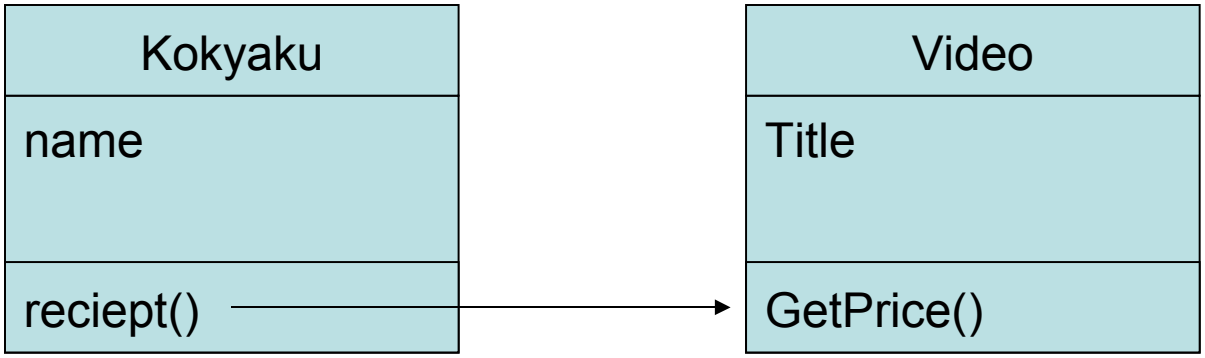
レシートを作るときにやること

- レンタルの値段計算
- レンタルしていいかチェック
- ポイントの計算
  - プレミアム会員は2倍！！

まずは、値段計算

- 値段はビデオによって違う
- ということで、ビデオオブジェクトに値段を聞くようにしよう

値段の問い合わせ

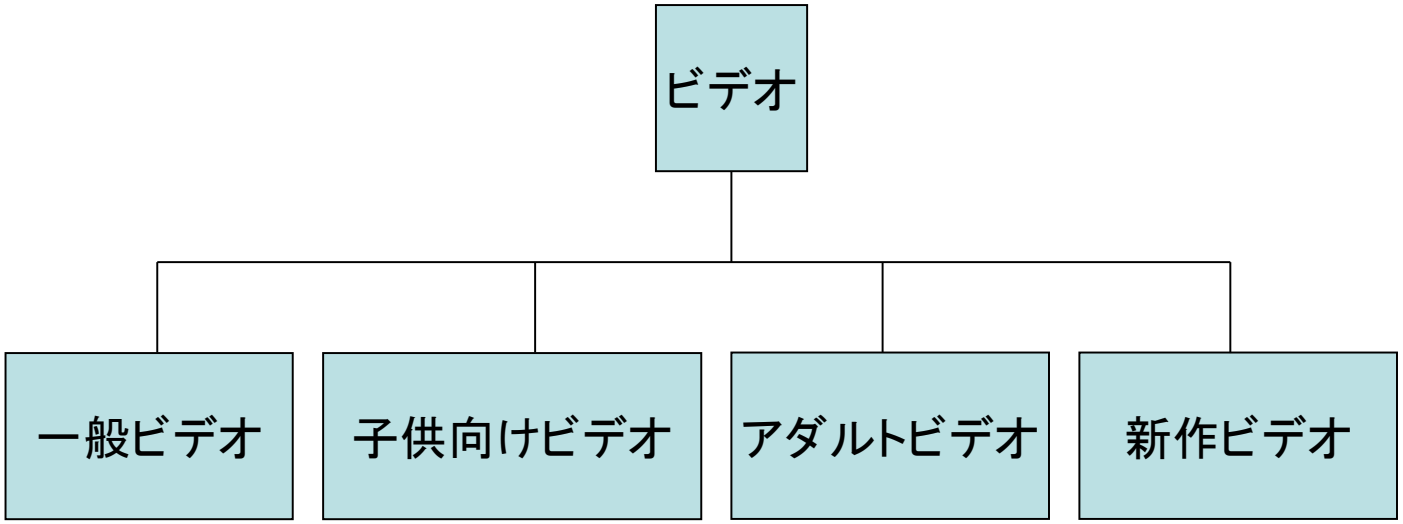


Receipt から GetPrice 呼び出し
video1->GetPrice()

値段計算

- それぞれのビデオの値段はどう出す？
  - 作品の種類によって違う
- ということは、それぞれの子クラスでポリモーフィックに呼び出せばいい！！

ビデオのクラス構造

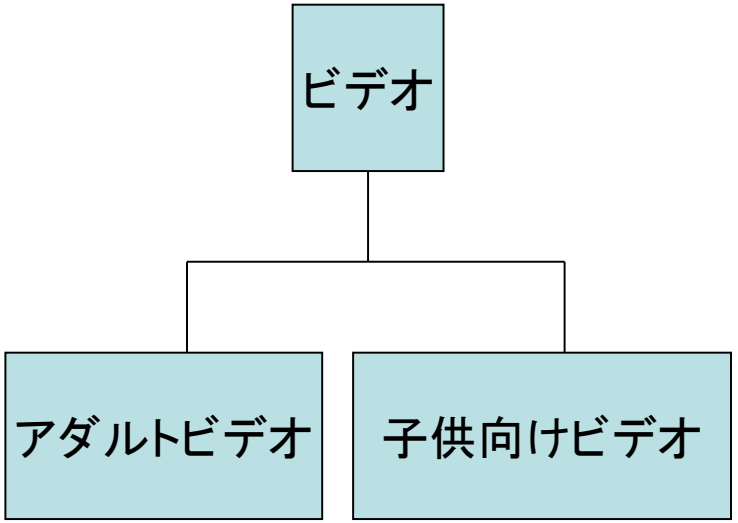


実際にはどう書くか？

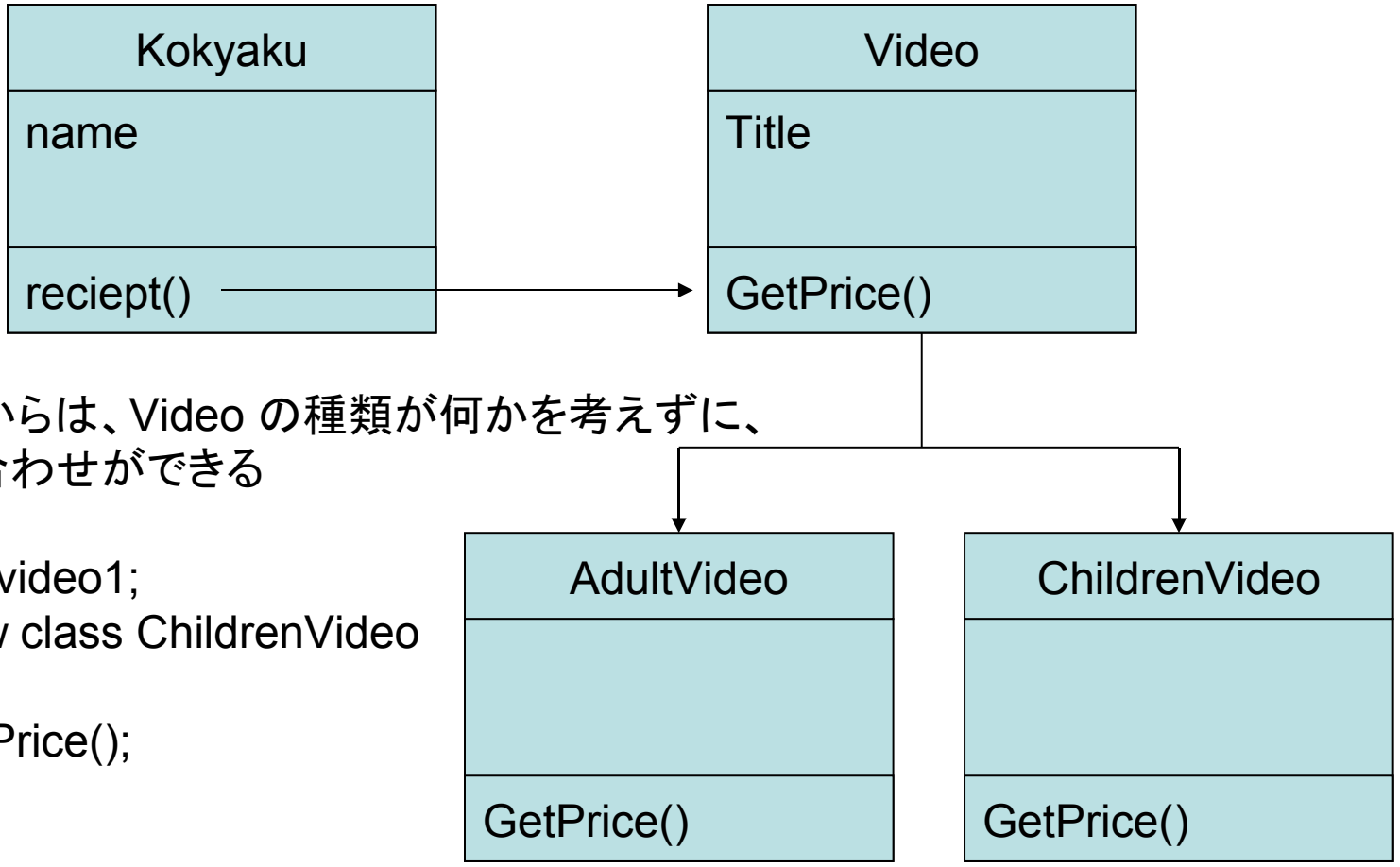
```
class Video {
        string title;
public:
        virtual getPrice();
        …
};

class AdultVideo : private Video {
public:
        getPrice();
        …
};

class ChlidrenVideo : private Video {
public:
        getPrice();
        …
};
```



値段の問い合わせ



Kokyaku 側からは、Video の種類が何かを考えずに、
値段の問い合わせができる

```
class video *video1;
video1 = new class ChildrenVideo

Video1->getPrice();
```

その他いろいろ・・・

- 「レンタルしていいかチェック」はどうすればいい？
  - 18歳未満はアダルト貸し出し不可
- 「ポイント計算」はどうする？
  - プレミアム会員は2倍

- 自分でも考えて見ましょう！

ポリモーフィズムと条件分岐

- 結局は条件分岐と一緒です
  - class に「type」フィールドをつけて、それぞれの手続きの中で if/switch で分岐
- しかし、クラスの種類が増えるたびに手続き側に書き直しが必要
  - そのオブジェクトに対する手続きがプログラム中に散らばってたとしたら？
- だからオブジェクト指向

Typeフィールドをつける場合

```
Reciept(Video video)
{
        if (Video->type == Adult) {
        …

        } else if (Video->type == Children) {
        } else if(….) {
        }else if (…)

}
```

オブジェクト指向の場合

```
Kokyaku::Reciept(Video *video)
{
        video->getPrice()
}
```

まとめ

- インターフェースとレイヤリング
- クラスの作り方