(3) カプセル化

**(3．1)
カプセル化の意味**

カプセル化とは何のことだろう？

**(3．2)
大域変数**

カプセル化しないとどうなるの？

**(3．3)
変数とメソッド**

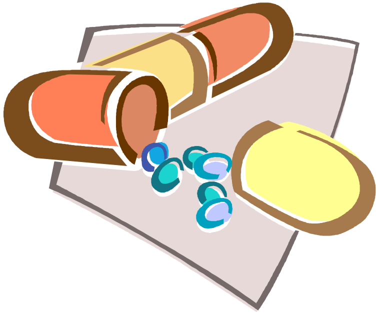
カプセル化されたオブジェクトの中身は？

**(3．4)
Javaでのカプセル化**

Javaでのカプセル化を少し見てみる。

(3．1) カプセル化の意義

- ■前節(2)では、プログラムを分ける単位として、オブジェクトを導入しました。
- ■しかしながら、それだけで“オブジェクト指向プログラミング”とは言えません。スライド(1．2)に記した各々の項目について、本節以下では説明していきます。最初は、カプセル化です。
- ■カプセル化(encapsulation)とは、風邪薬のカプセルと同じ単語です。風邪薬のカプセルは、粉末とか液体の中身の風邪薬を隠して、飲みやすいようにするものです。オブジェクト指向におけるカプセル化も、ある意味では似たような働きをします。



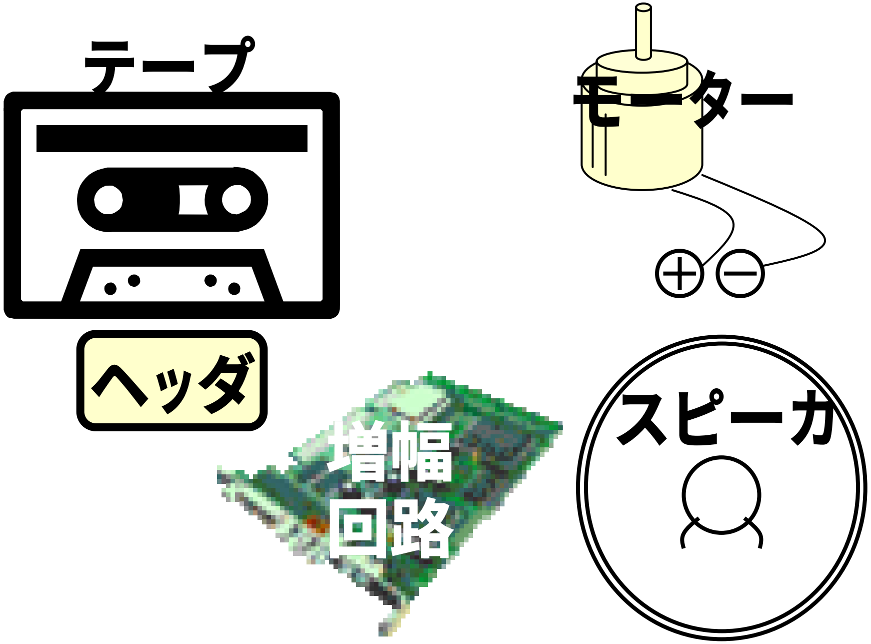
(3.1.1) テープデッキでのカプセル化

■カプセル化はプログラムだけに適用される方法ではありません。

■ここでは、テープデッキ(ウォークマンを思い浮かべてください)でのカプセル化を考えて見ます。

■ここでは、テープデッキを単純化して、次の部品から構成されるとします。

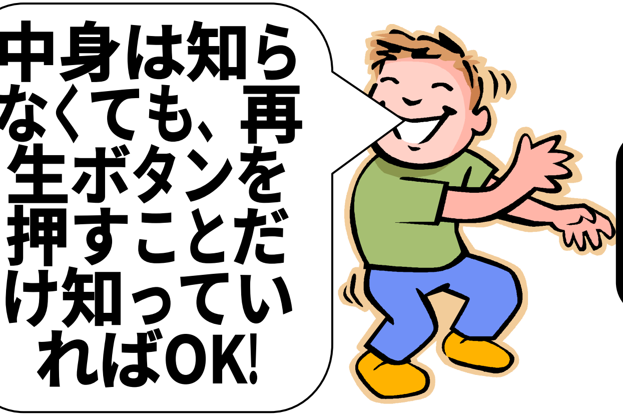
- ヘッダ(テープの読み取り器)
- モーター
- 増幅回路
- その他、操作しないもの(テープ、スピーカ)

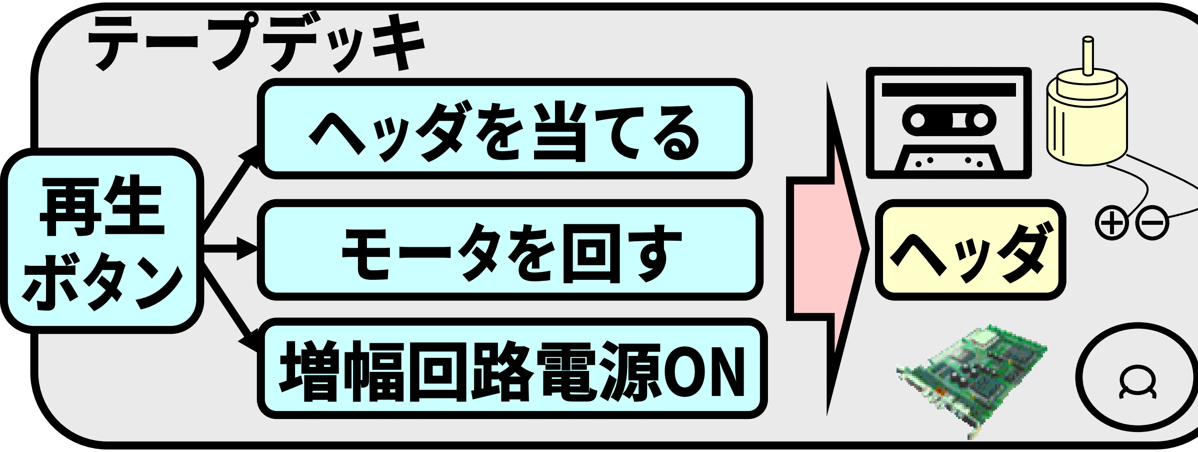

テープ
モーター
⊕⊖
ヘッダ
増幅回路
スピーカ

■カプセル化した場合、しない場合について、次のスライドで見て見ます。

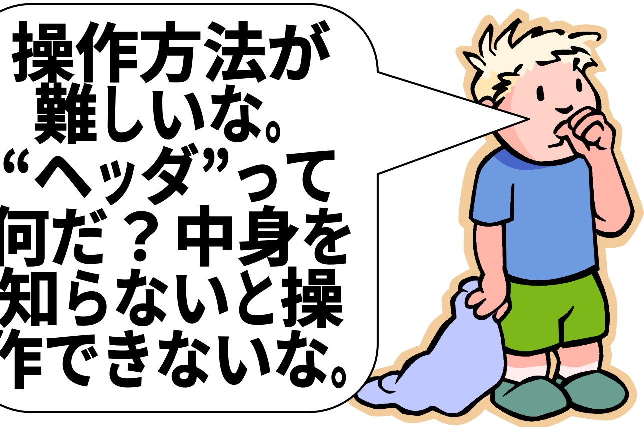
(3．1．2) 再生時の処理

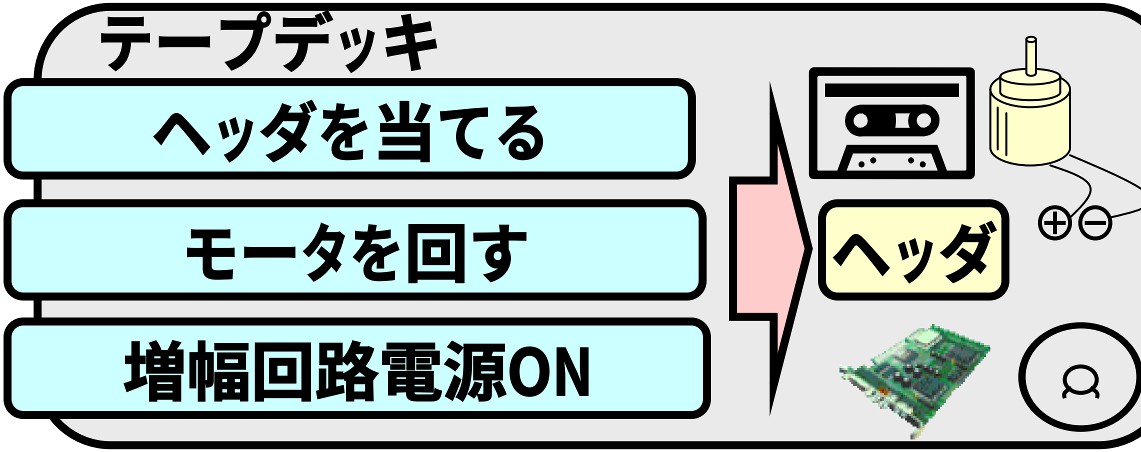
■カプセル化されたテープデッキでの再生


中身は知らなくても、再生ボタンを押すことだけ知っていればOK!


テープデッキ
再生ボタン
ヘッダを当てる
モータを回す
増幅回路電源ON
ヘッダ
⊕⊖

■カプセル化されていないテープデッキでの再生


操作方法が難しいな。“ヘッダ”って何だ？中身を知らないと操作できないな。

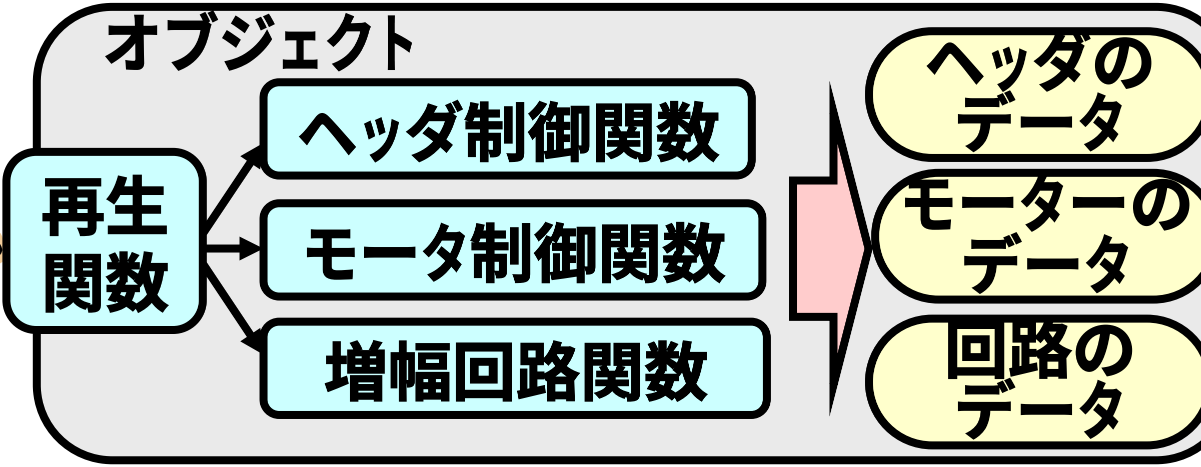

テープデッキ
ヘッダを当てる
モータを回す
増幅回路電源ON
ヘッダ
⊕⊖

■部品 (中身) を隠すほうが、使いやすい訳です。

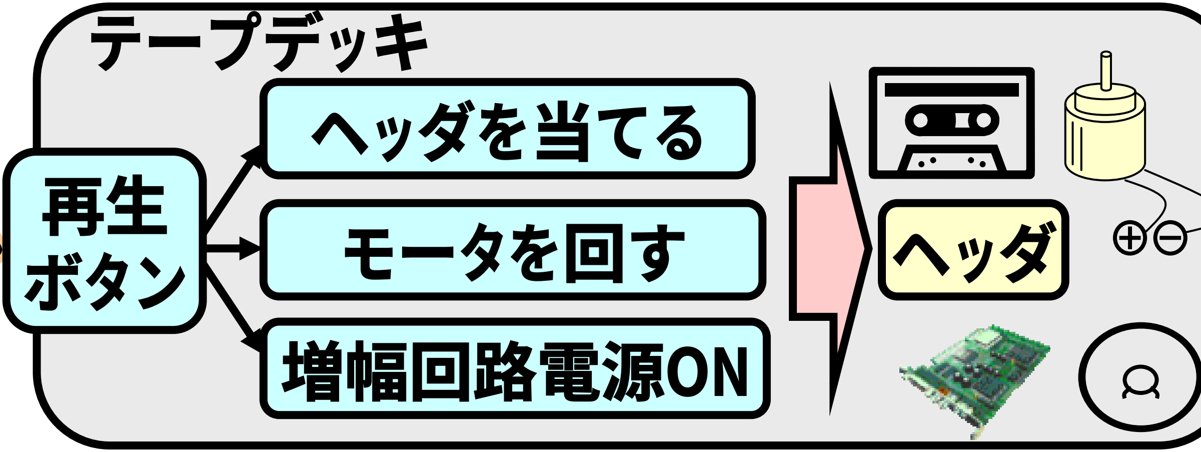
(3．1．3) プログラムの場合

- プログラムでは、大雑把に言うと、隠すべき中身 (部品) は、データということになります。

オブジェクト (プログラム) の中身を知らなくても、使い方は解るよ！

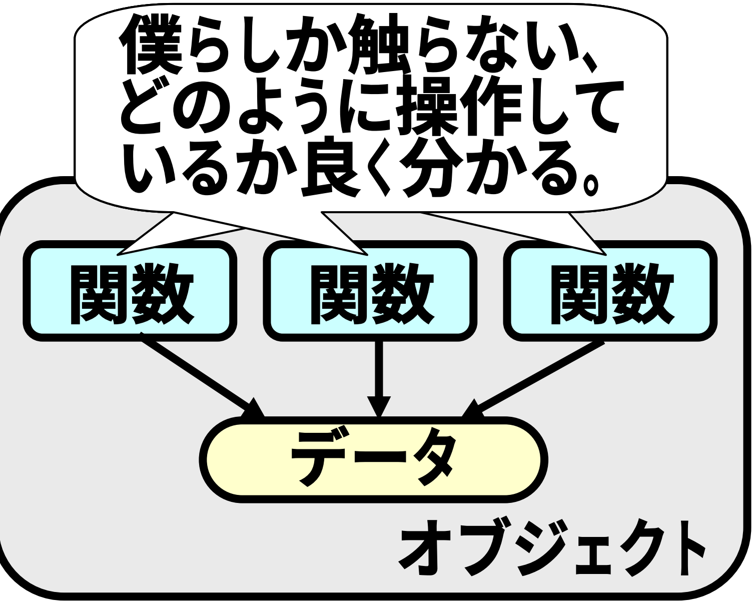

オブジェクト
再生関数
ヘッダ制御関数
モータ制御関数
増幅回路関数
ヘッダのデータ
モーターのデータ
回路のデータ

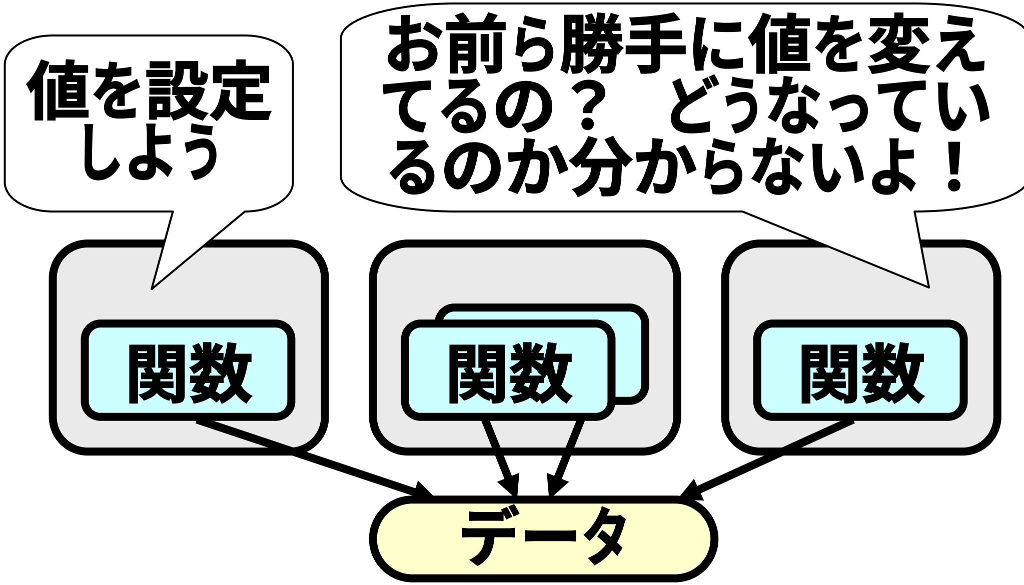
中身は知らなくても、再生ボタンを押すことだけ知っていればOK!


テープデッキ
再生ボタン
ヘッダを当てる
モータを回す
増幅回路電源ON
ヘッダ
⊕⊖

(3．2) 大域変数

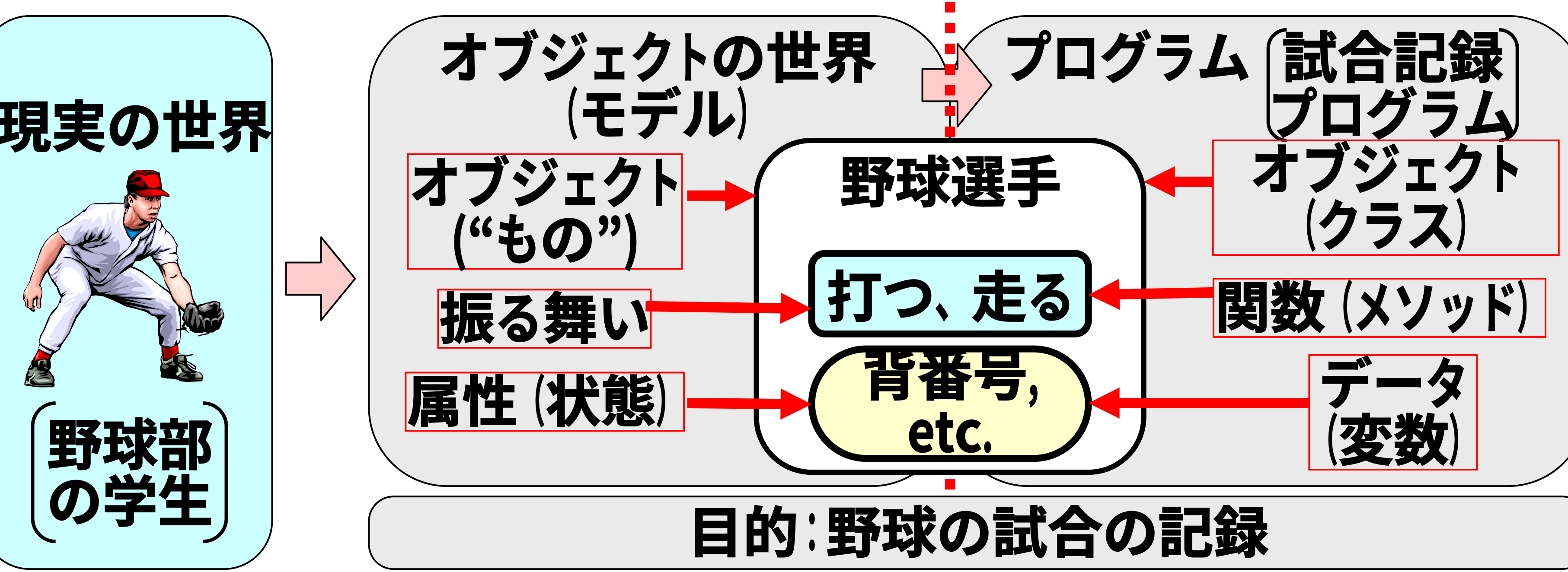
- カプセル化が有効な手法であることは理解して頂けたかと思います。
- 広い範囲のプログラムからアクセスされるデータ (これを大域変数と言います) があると、そのデータに関してどのような事態が起こるかは、その広い範囲のすべてを理解しないと分からなくなります。これは、プログラムを分かりにくくする原因となります。
- カプセル化は、このような事態を防ぎます。


僕らしか触らない、どのように操作しているか良く分かる。
関数
関数
関数
データ
オブジェクト


値を設定しよう
お前ら勝手に値を変えてるの？　どうなっているのか分からないよ！
関数
関数
関数
データ

（3．3．1）属性

- ■オブジェクトは、データをカプセル化（＝外部より隠蔽）します。それでは、オブジェクトはどのようなデータをカプセル化するのでしょうか？
- ■プログラムの観点からデータと記してきたものは、モデリングの観点からは、属性（スライド(2.1)での状態と思ってください）と呼びます。

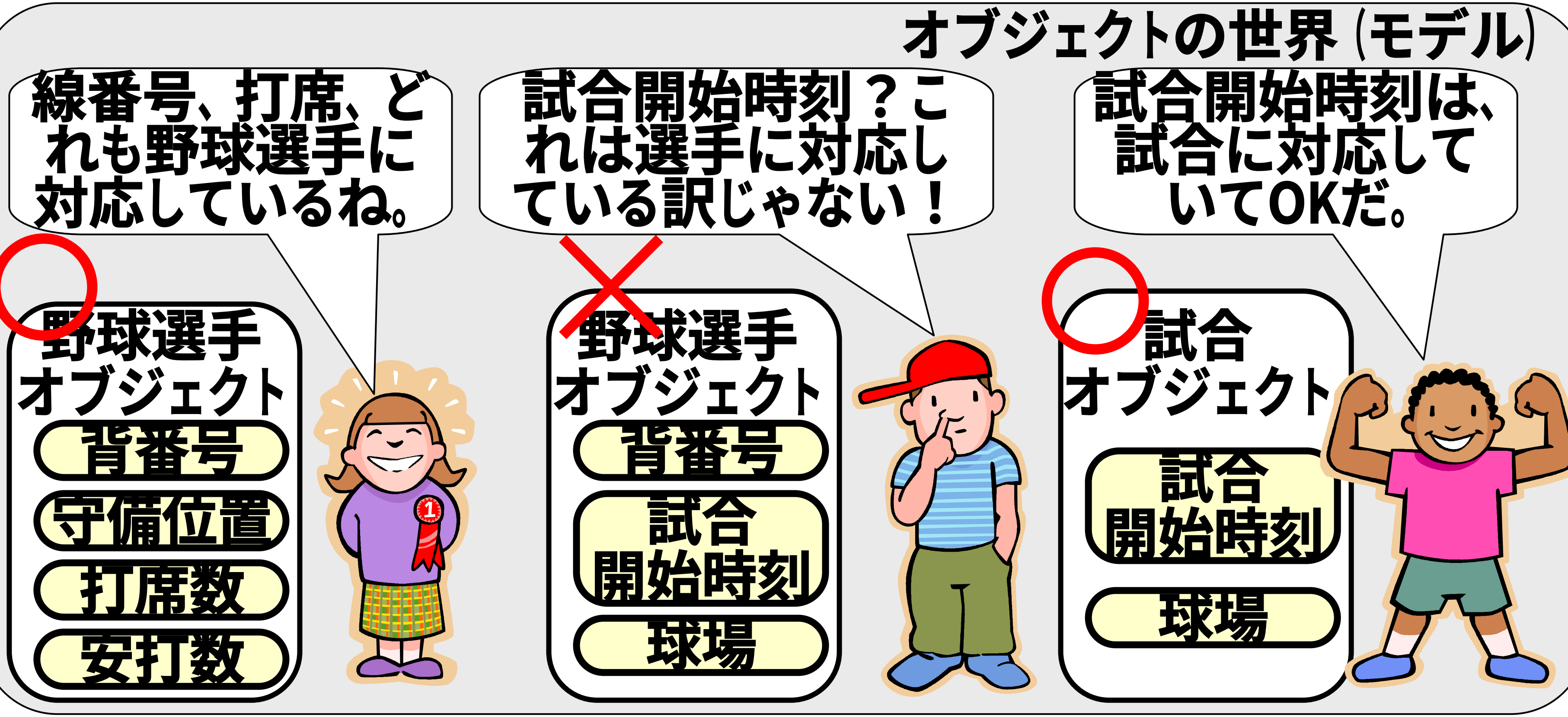


現実の世界
野球部の学生
オブジェクトの世界（モデル）
プログラム（試合記録プログラム）
オブジェクト（“もの”）
振る舞い
属性（状態）
野球選手
打つ、走る
背番号, etc.
オブジェクト（クラス）
関数（メソッド）
データ（変数）

目的：野球の試合の記録

(3．3．2) オブジェクトと属性

■オブジェクトが持つ属性は、そのオブジェクトに対応するものになる訳です。オブジェクトと考えた“もの”に対応しないものを属性に持つと、プログラムのレベルでの混乱を招くことになります。



オブジェクトの世界（モデル）

線番号、打席、どれも野球選手に対応しているね。

試合開始時刻？これは選手に対応している訳じゃない！

試合開始時刻は、試合に対応していてOKだ。

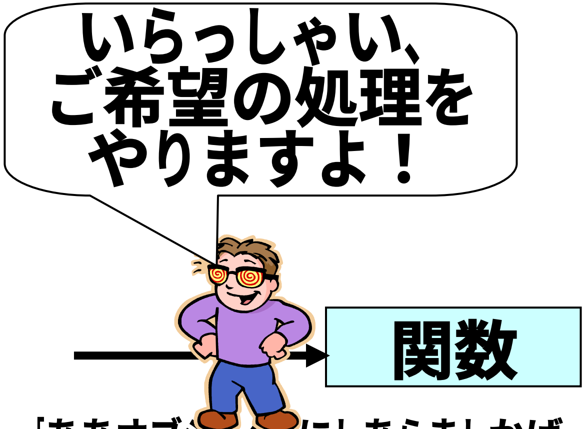
○ 野球選手オブジェクト
- 背番号
- 守備位置
- 打席数
- 安打数

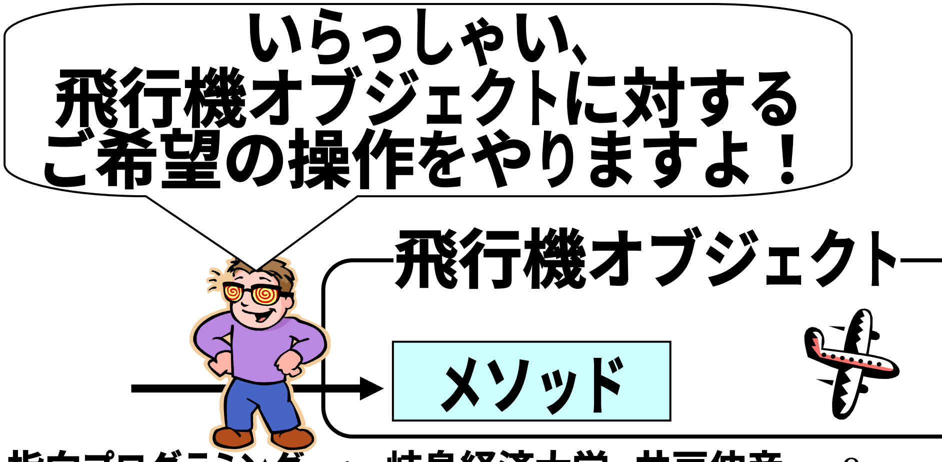
× 野球選手オブジェクト
- 背番号
- 試合開始時刻
- 球場

○ 試合オブジェクト
- 試合開始時刻
- 球場

（３．３．３）メソッド

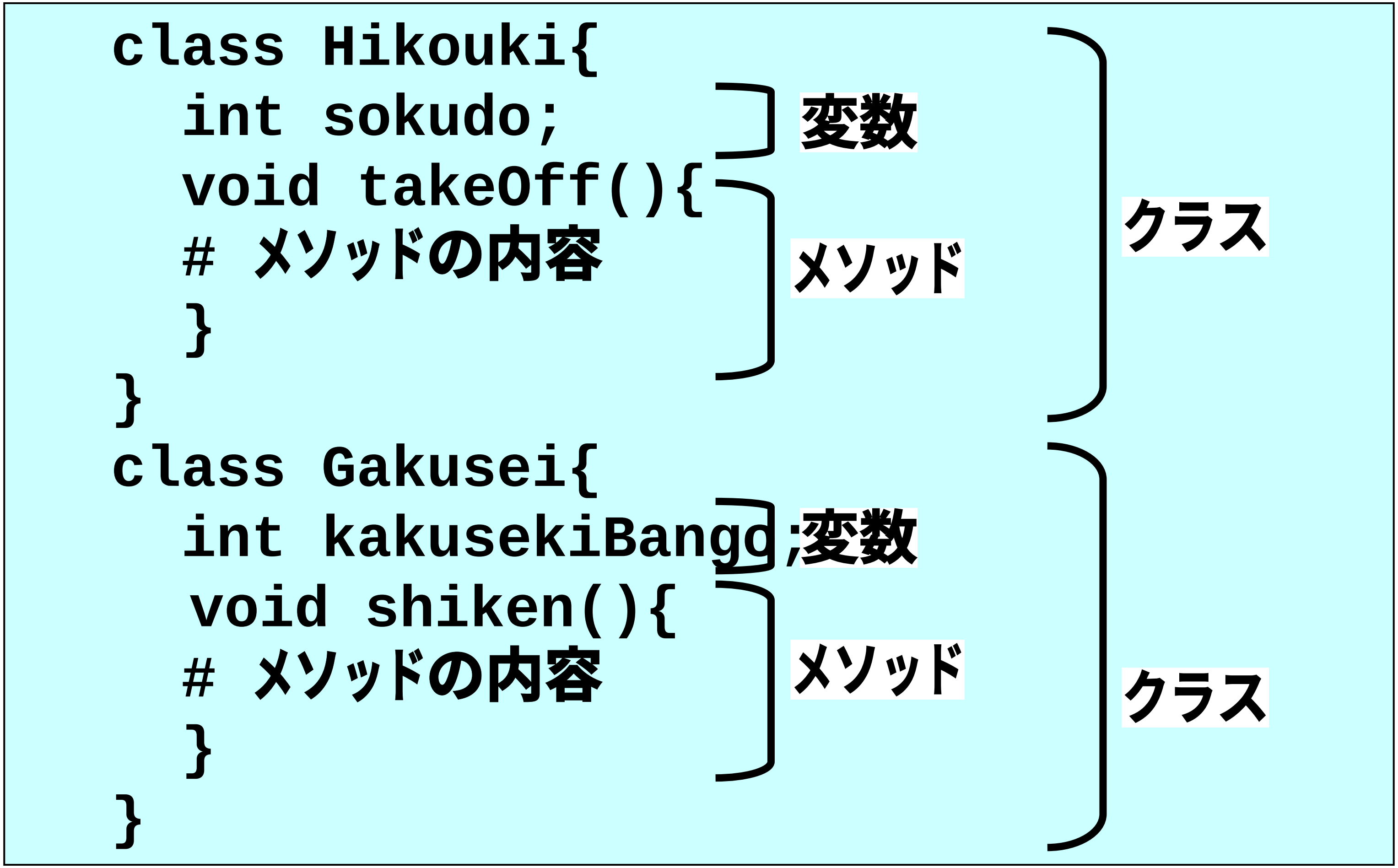
- ■スライド(3.3.1)にも記したように、オブジェクトの“振舞い”を記述するコードを、“メソッド”と言います。
- ■メソッドの実体は、コードの固まりであり、従来関数（サブルーチン、プロシージャ、ファンクション）と呼ばれていたものと変わりません。
- ■しかしながら、“関数”などの呼び名と、“メソッド”という呼び名には、考え方の違いが込められています。すなわち、メソッドは、単たる処理ではなく、オブジェクトに対する操作と考えます。


いらっしゃい、
ご希望の処理を
やりますよ！
関数


いらっしゃい、
飛行機オブジェクトに対する
ご希望の操作をやりますよ！
飛行機オブジェクト
メソッド

（3．4）Javaでの記述

■まず、単純にデータ（＝変数）とメソッドとを示します。



```
class Hikouki{
  int sokudo;
  void takeOff(){
  # メソッドの内容
  }
}
class Gakusei{
  int kakusekiBango;
   void shiken(){
  # メソッドの内容
  }
}
```

■Javaオブジェクト指向スライド（2）

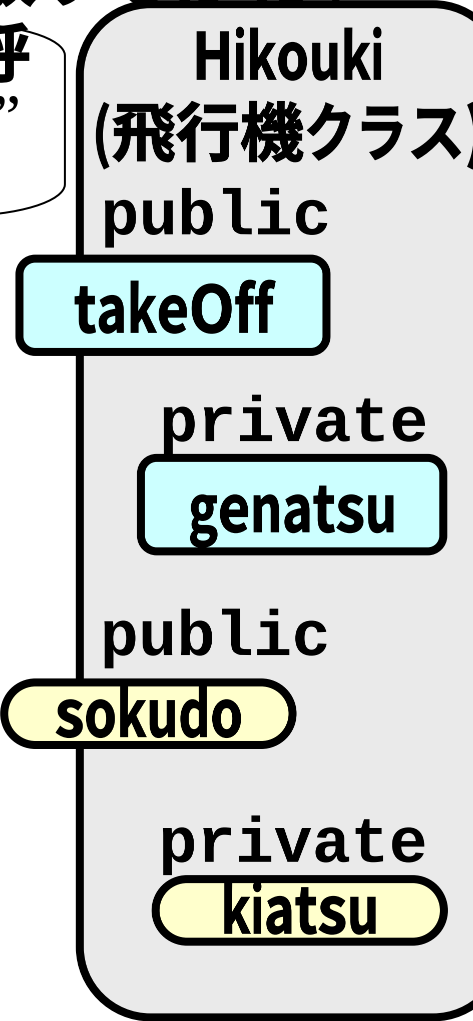
(3．4．1) カプセル化の記述

■オブジェクトの中の変数（データ）とメソッドとを、オブジェクトの外に見せるか、見せないで隠蔽するかは、次のように記述します。

```
   class Hikouki{
      public int sokudo;
      private int kiatsu;
      public void takeOff(){
      // メソッドの内容
      }
      private void genatsu(){
      // メソッドの内容
      }
   }
```

"takeOff"は外から呼べるけど、"genatsu"は呼べない。

"sokudo"は外かアクセス出来る、"genatsu"はアクセス出来ない。

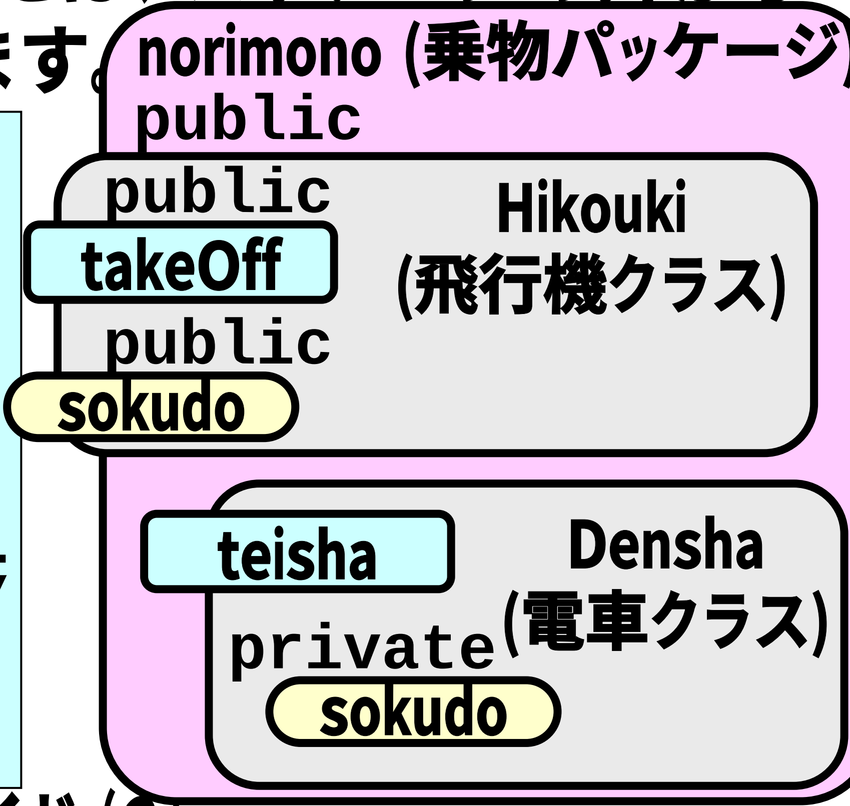

Hikouki
(飛行機クラス)
public
takeOff
private
genatsu
public
sokudo
private
kiatsu

■Javaオブジェクト指向スライド（3）

(3.4.2) パッケージ

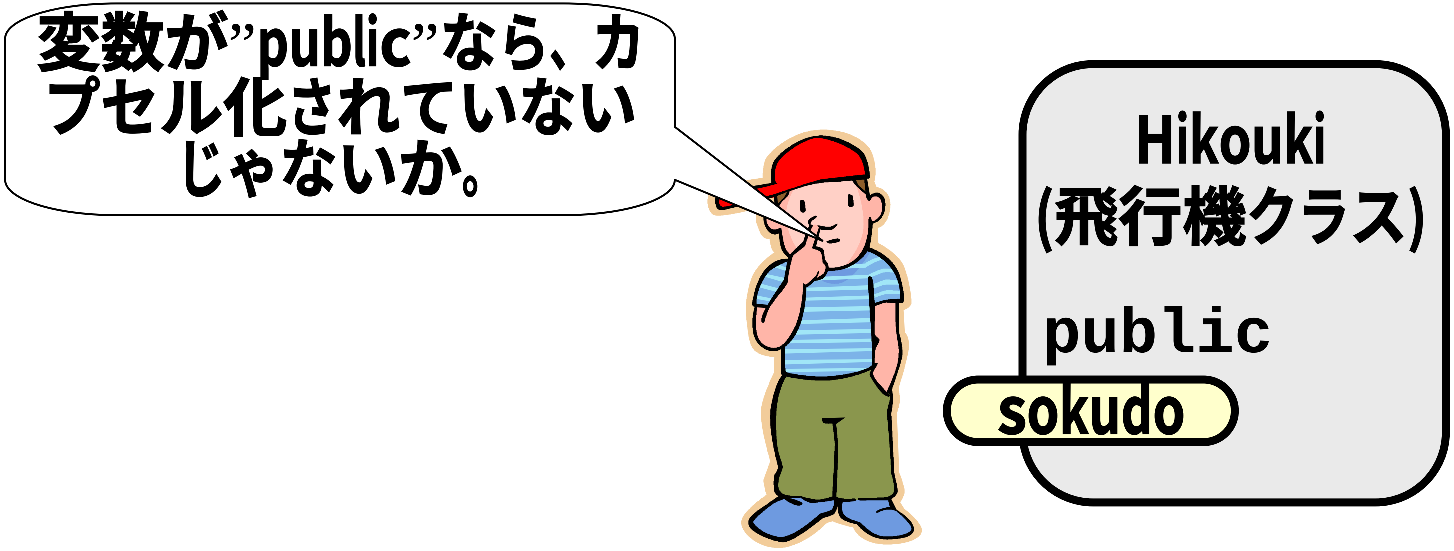
- ■Javaには、1つ以上のクラスをまとめて、パッケージというプログラムの単位を作ることが出来ます。
- ■パッケージにおいてもカプセル化が行われます。詳細は省略しますが、”public”とは、パッケージの外からもアクセスできることを示します。

```
packege norimono;
public class Hikouki{
  public int sokudo;
  public void takeOff(){
  }
}
class Densha{
  private int kakusekiBango;
  void teisha(){
  }
}
```


norimono (乗物パッケージ)
public
public
takeOff
Hikouki
(飛行機クラス)
public
sokudo
teisha
Densha
(電車クラス)
private
sokudo

(3．4．3) “public”の変数

- ■スライド(3.4.1),(3.4.2)に示したJavaでの記法は、カプセル化を可能にするものです。
- ■しかし、変数を”public”としたのでは、カプセル化したことになっていません。
- ■オブジェクト指向の趣旨に沿うならば、変数はすべて”private”とすることが基本となります。



変数が”public”なら、カプセル化されていないじゃないか。

Hikouki
(飛行機クラス)
public
sokudo

（3．5．1）基本演習：カードデッキ

- ■課題2－1で扱った、ブラックジャックについて考えます。
  - コピーしてきた“blackjack”のフォルダを開いてください。
  - ファイル“CardDeck.java”を、ダブルクリックにより開いてください。
- ■“CardDeck”クラスには、次のように変数が宣言されています。
  - `protected Card[] cards = new Card[SUM_OF_CARDS];`
  - これは、カードデッキが52枚のカードを持つことを示しています。

（３．５．２）基本演習（課題３－１）

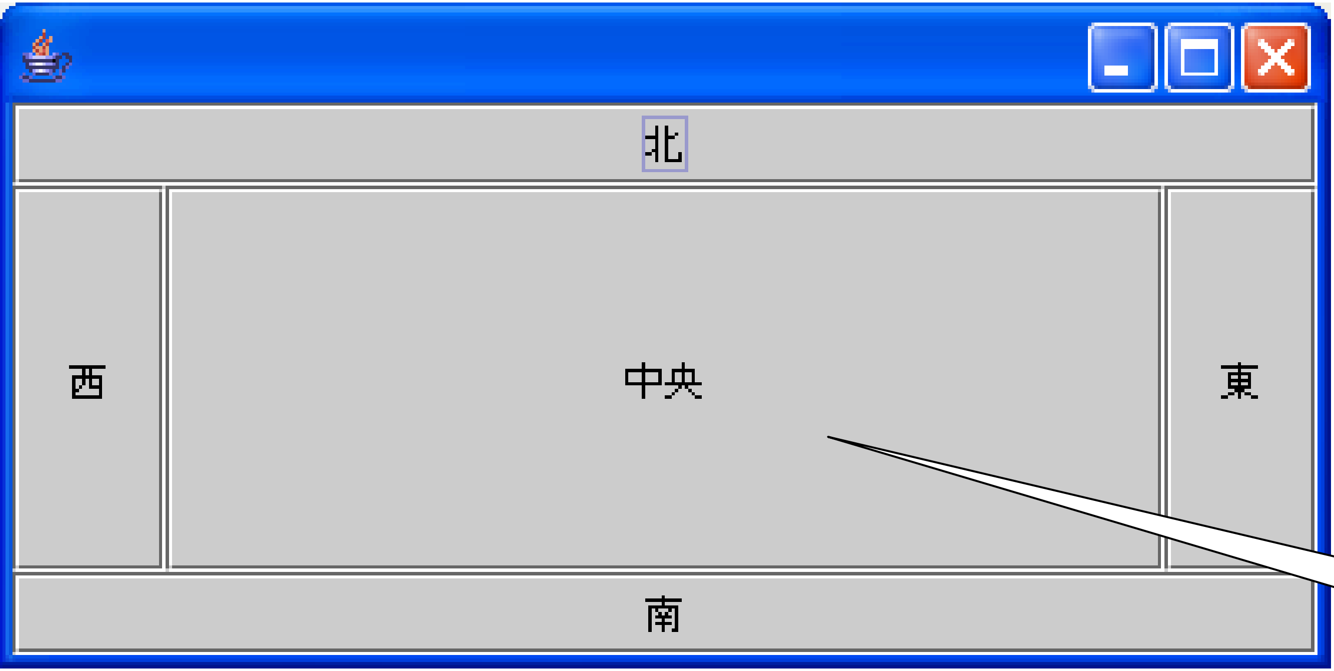
■“CardDeck”クラスには、次のようにメソッドが宣言されています。

- `public void shuffle() {  // 後略`
- `public Card getCard() {  // 攻略`
- これらは、カードデッキが持つ５２枚のカードへの操作を示しています。

■（課題３－１）“CardDeck”クラスは、どのようなデータを持ち、これを隠蔽して、どのような操作を外に見せているか、考えて見てください。

(3．6) 発展演習 (課題3－2)

■blackjackのプログラム、および、スライド (10) を参考にして、次のようなウインドウを表示するプログラムを作ってください。



すべてボタン

(4) UML

■ソフトウェアを開発するとき、いきなりプログラムを書いていくことは一般的に出来ません。まず、設計をする必要があります。

■ここまで、クラス(変数、メソッド)について見てきました。これらのことを文書に記しておくことは、設計の重要な部分のひとつです。

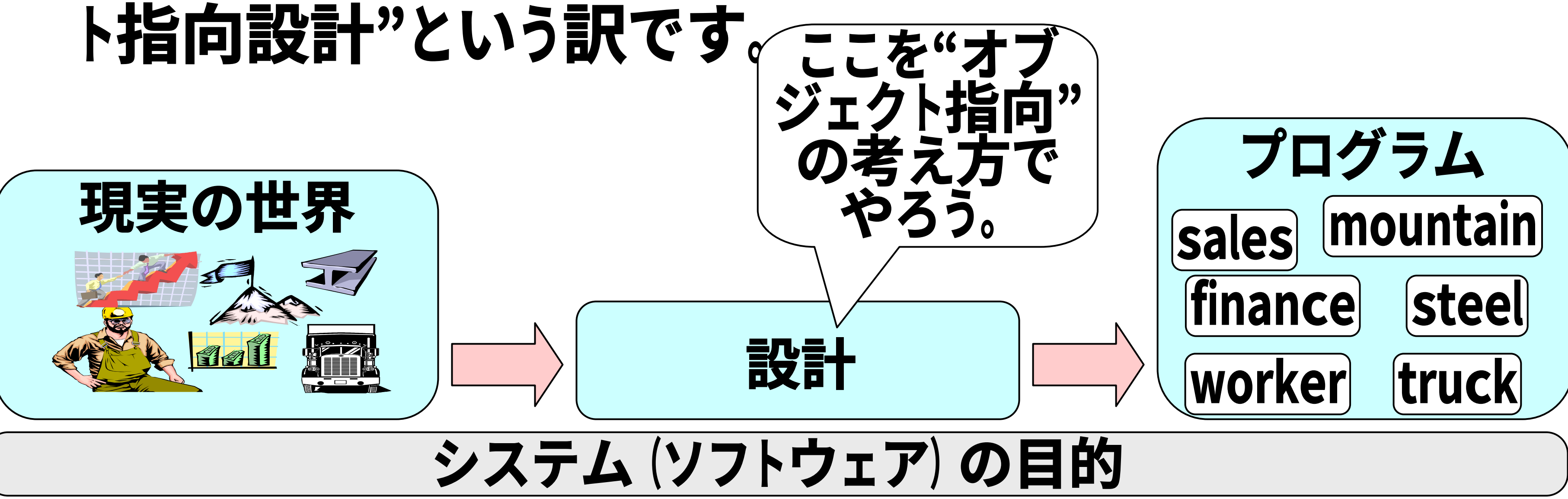
■それでは、どのように記しておけば良いのでしょうか？

■この問いに答えるため、本章(4)では、UMLについて記します。

■次章(5)から、UMLによる図が登場してきます。

(4．1) ソフトウェアの設計

- ■設計図とは、システムの要求をもとに、現実の世界とプログラムとを橋渡しするものです。これを“オブジェクト指向”の考えに基づいて作成することが、“オブジェクト指向設計”という訳です。


ここを“オブジェクト指向”の考え方でやろう。
現実の世界
設計
プログラム
sales
mountain
finance
steel
worker
truck
システム（ソフトウェア）の目的

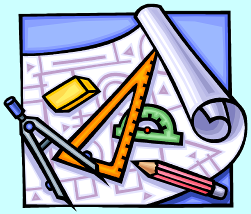
※：ソフトウェア開発の行程については、別のスライドにて説明します。

（4．2）UMLの位置づけ

■UML (Unified Modeling Language) は、オブジェクト指向の考え方に基づき、プログラムの元となるモデルの世界を視覚的に表現する記述方法です。すなわち、これが設計図です。

これを元にプログラムを作成したり、
人のプログラムを理解したりしたら
いいんだ。これが設計図だ。

UML



現実の世界

→

オブジェクトの世界（モデル）

売上
山
鉄鋼
資金
労働者
トラック

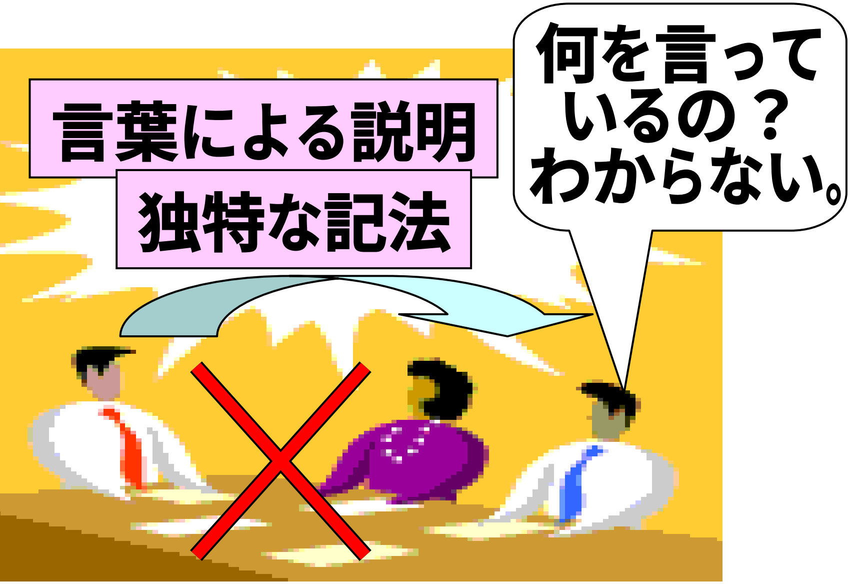
→

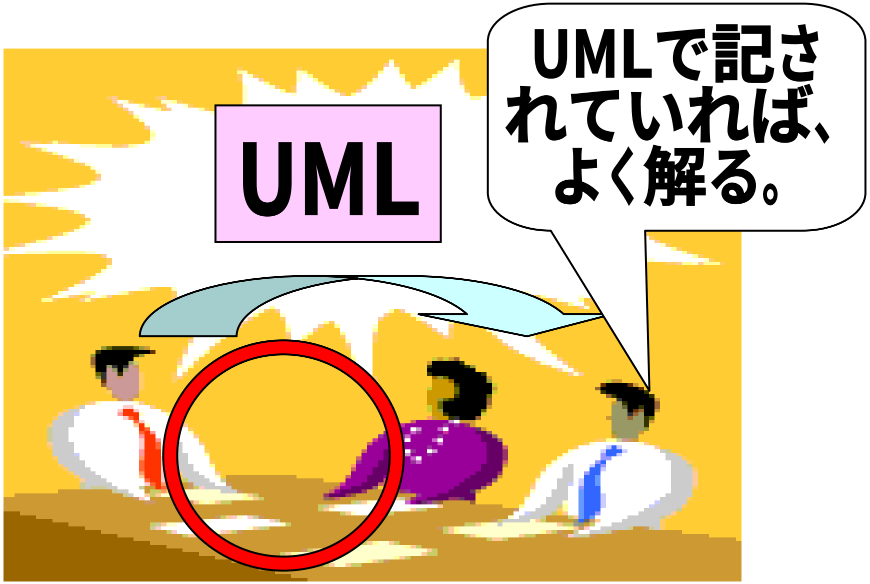
プログラムの世界

sales
mountain
finance
steel
worker
truck

(4．3) UMLの重要性

- ■UMLの重要な点は、設計図として標準化がなされていることです。実際、ソフトウェアの仕様をUMLで記述した本は、あちこちで見られるようになってきています。
- ■標準化された記法により、他人が作ったソフトウェアや昔に作られたソフトウェアに対する理解が容易に行えるようになります。

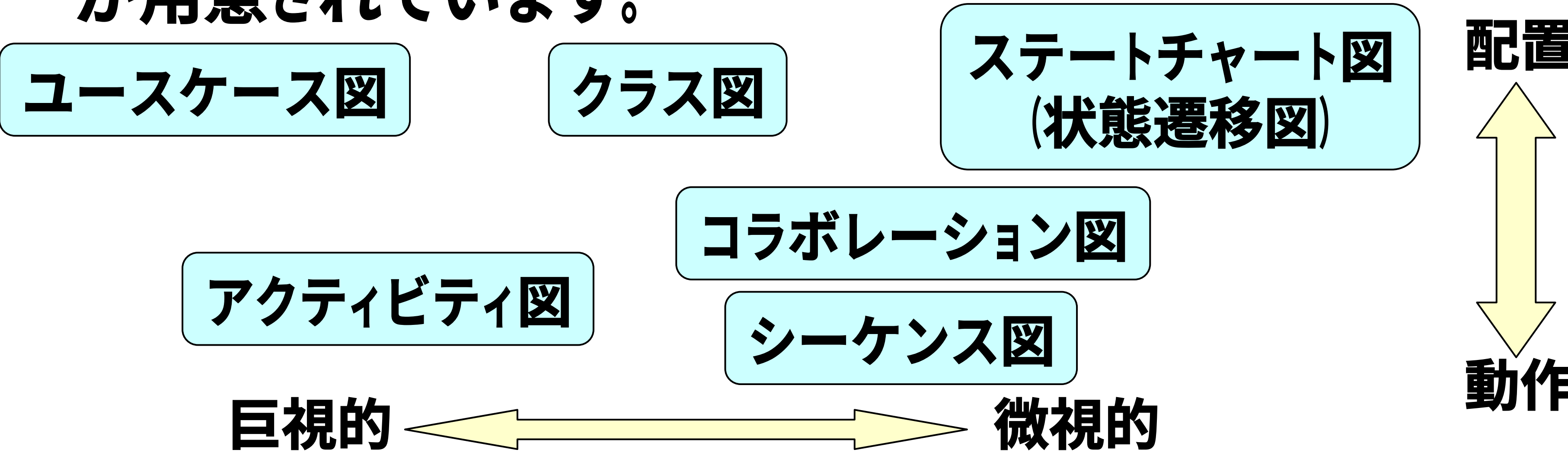

言葉による説明
独特な記法
何を言っているの？わからない。


UML
UMLで記されていれば、よく解る。

(4．4) UMLの図

■考えたオブジェクトがどのようなものであるのかを、プログラムを作る観点から整理するものがUMLです。

■従って、UMLにはプログラムの様々な面を記述する図が用意されています。


ユースケース図
クラス図
ステートチャート図（状態遷移図）
配置
コラボレーション図
アクティビティ図
シーケンス図
動作
巨視的
微視的

■本スライドでは、アクティビティ図、シーケンス図、クラス図を主に扱います。ユースケース図については、別スライドにて扱います。