Introduction to Java

Special Care Set for Java Begineers/ Ex 3

・Notice!
The motivation of this page is to lead regular exercises.
We prepare many useful information in this page as hints of exercises, but this page has no answer.
If you have question, please don't hesitate to ask and discuss your teacher and teaching assistants.
このページは、通常の演習問題への導きとして作られています。
多くの有用な情報をヒントとして準備しておりますが、ここに回答が隠されているとかはありません。
もし、演習に対して質問等ございましたら、遠慮なく教員・TAに質問・議論をしてください。

Example Problem: クラス設計と機能 - Carクラスを例として -

クラスの設計は、Javaの根幹の問題です。
Cやでは、『どのような手続き』で問題を解くか？に焦点が当たっていましたが、
Javaでは『オブジェクト』を『どのように表現』するか？に焦点が当たります。
ここでは、例題として『車』をテーマにして、そのオブジェクトの持つ値と動作について考えていきましょう。

Class design is one of the important topic of Objective language such as Java.
Programming C is focused on procedure of the process, but Java focuses to express states and actions of objects.
In this example problem, let's think the object "Car" and its status and actions.

1. 車の持つ値 (State of the Car)

車を動かすのには、様々な値 (変数) があることがわかります。それらは、『外部から見える』ものと、『内部からしか見れない』ものがあります。
『内部からしか見れない』値を外部から見たり、変更したりすると、それは事故につながりますね。
ですから、車の持つ値を定義するときには、『外部から（勝手に）変更できるもの＝外部状態』と『内部からしか変更できないもの＝内部状態』とに分けて、
良く管理する必要があります。
The car object has a lot of state (variable) and it can separate as "see from outside - Public" and "hide to outside - Private".
If the value which hide to outside but it can changable from outside such as speed changes 60 to 100 by outside suddenly, it occurs car accidents.
Therefore, we should manage such values as "accessable from outside: Public state" and "accessable only internal: Private state".

外部状態 (public): アクセルを踏む量, ブレーキを踏む量, ハンドル位置, シフト位置, エアコンの温度設定, エアコン出力の量, ガソリン給油量... など
Public state: pressure of acceleration, pressure of break, position of handle, position of shift, setting of air conditioner, suppling amount of oil, etc.

内部状態 (private): 速度, 車の向き, エンジン回転量, 残存ガソリン量, ギア... など
Private state: speed, direction, engine rotation, amount of fuel, gear setting, etc.

実際には、車の『動作』にかかわる状態はほとんど内部状態であることがわかりますね。
また, 外部状態となっているものも、実際には『動作』に紐づけされるので, 次に説明する『動作』として定義するならば, ほぼすべての値が内部状態となることがわかります。
In fact, the states which related car motion are nearly able to regard as private state.
Also, public states is able to explain as other motion. Then, we can able to define as "Action" for the car object and almost all states regards as private state.

2. 車に対して行われる動作 (Action for the Car)

車を動かすということは、乗車する人間などが、車に対して何かしらの動作を行う事です。
車はひとりでに動くのではなく、誰かが動作をさせることで、車が動作することになります。

値と同じように、これも『内部動作』と『外部動作』の種類があります。

外部動作 (public): アクセル[踏む], ブレーキ[踏む], ハンドル[回す], シフト[移動する], エアコン温度[調節する], エアコン出力[調節する], ガソリン給油[入れる],... など

内部動作 (private): アクセルをのまれた量に対してエンジンの回転数を上げる, エンジン回転数とギアからタイヤを回転させる,... など

こうしてみると、オブジェクト指向のモノの考え方は、『どのようにして動かすか』という手続きで見るのではなく、『何がどうする

と、どうなるか』といった『状態』ベースで考えられることになります。

3. Carクラスを作る

上記1, 2のことを踏まえつつ、簡単なCarクラスを作成してみましょう。
今回の車は直進だけを考えて、速度と走行距離に関係する所だけを扱おうと思います。

・内部状態: ガソリン量 (double fuel), アクセル量 (double accel [0-1]), 馬力(double power), 燃費 (double consumption), 速度 (speed), 走行距離 (milage)

・外部動作: アクセルを踏んで速度を上げる (speedUp), ブレーキを踏んで速度を落とす (speedDown), 一定時間運転する (drive), 給油する (putFuel), 速度を見る (getSpeed), 走行距離を見る (getMilage)

```
/* Carクラスのスケルトン */
public class Car {
    private double fuel;
    private double speed;
    private double consumption;


    private double milage;



    public void speedUp(){}
    public void speedDown(){}
    public void driveCar(int minute){}
    public void putFuel(double fuel){}
    public double getSpeed(){}
    public double getMilage(){}
    public double getFuel(){}
    public void printState(){}

}
```

外部動作を扱うためのメソッド名で良く使われるのは、get... や set... などです。これは, 内部状態を表すメンバー変数に対して、『動作』として明示的に値を設定・取り出しを行うことを意味しています。
何故こんな遠回りをするかと言うと、内部状態を表すメンバー変数をもしpublicにしておいた場合に便利だからと言って変なところで適当に変数設定をしてしまう可能性があり、それがバグや状態エラーに繋がる可能性があるからです。
汎用的で便利に扱おうとすると不都合が多くなる上に、『物体』を定義する上でこちらの方が自然である（動作として明確に定義できる）ことから、Javaではこのような方法を取ることが多いです。

4. コンストラクタを利用する

コンストラクタとは、クラスをインスタンス化する時に呼び出される『初期化』を行うためのメソッドです。
クラスで定義しているオブジェクトは、インスタンス化することで実体を持ち、それを複数個生成することが出来るようになります。
例えば、こんな感じです。

```
/* CarDemoクラスによるCarオブジェクトの定義 */
public class CarDemo {
    public static void main(String[] argc){
        Car preo = new Car();
        Car toppo = new Car();
        Car Celsior = new Car();
        Car fit = new Car();

    }
}
```

上記の書き方では、ただCarオブジェクトを配置しただけですが、一応Carオブジェクトを使用できる状態になっています。
では、Carクラスにコンストラクタを定義して、車を定義した時に同時に燃費とガソリン量を設定するようにしましょう。

```
/* CarDemoクラスによるCarオブジェクトの定義2 */
public class CarDemo {
    public static void main(String[] argc){
        Car preo = new Car(9.0, 30);
        Car toppo = new Car(18.0, 30);
        Car Celsior = new Car(9.0, 84);
        Car fit = new Car(20.0, 42);

    }
```

```
}



public class Car {

    private double fuel;
    private double speed;
    private double consumption;
    private double milage;



    Car(double consumption, double fuel){
        this.consumption = consumption;
        this.fuel = fuel;
    }


    //...
}
```

授業中にも説明しましたが、this とは、その『インスタンス』で宣言されている内部変数に対して表すものです。
引数と内部変数の文字が同じ場合、そのメソッド中で優先的に扱われるのは『引数』の方です。
ですから、メソッド内部やクラス内部で扱う変数の場合は、明示的に this.fuel などといった形で表しておくと、間違いがありません。

5. 内部動作・外部動作を定義する

外側の定義が終われば、今度は内側をどんどんとコーディングしていきましょう。
クラスの内部はほぼC/C++と同じです。それぞれの手続きを記入していきましょう。

```
/* Carクラスの完成型 */
class Car{

    private int speed;

    private double fuel;

    private double consumption;

    private double milage;

    Car(double consumption, double fuel){

        this.consumption = consumption;

        this.fuel = fuel;

        this.milage = 0;

    }


    public void speedUp(){

        if(speed < 100) this.speed += 10;

    }


    public void speedDown(){

        if(speed > 0) this.speed -= 10;

    }


    public void driveCar(int minute){

        double dist = speed * (double)minute / 60.0;

        if(this.fuel < dist / this.consumption){

            this.milage += fuel * consumption;

            this.fuel = 0;

        }

        else{

            this.milage += dist;
```

```
        this.fuel -= dist / this.consumption;
    }
}

public void putFuel(double fuel){
    this.fuel += fuel;
}

public double getSpeed(){
    return this.speed;
}

public double getMilage(){
    return this.milage;
}

public double getFuel(){
    return this.fuel;
}

public void printState(){
    System.out.println("Speed: " + this.speed + "km/h, Traveled: " + this.milage + "km, Fuel left: " + this.fuel +"L.");
}
}
```

6. メインメソッドから使用するクラスのメソッドを扱う

最後に、クラスによって定義されたオブジェクトをいろいろな形で使っていくようにしましょう。
4. のソースコードで、メインメソッド中にオブジェクトを様々な形でインスタンス化しました。
ここからは、みなさんで、インスタンス化されたオブジェクトのメソッドをどんどん使って、遊んでみてください。
なお、これがどのような挙動をするかを確認したい場合は、先週（Special Care Set Ex2）のExample 2.『Jeliotによる挙動の確認』を参照してください。