【Java言語】オブジェクト指向(概念と実体)(クラスとインスタンス)

参考　An introduction to c++ Programming and Object Oriented Programming　with Tutorials and Hands-On Examples by NATHAN METZLER

1.オブジェクト指向プログラミングの概要

オブジェクト指向プログラミング（略してOOP）では、データに重点が置かれ、手順には重点が置かれていません。

OOPは、ひな形(**クラス**)に定義された属性(**データ・フィールド**)とこのデータにアクセスする操作(**アクセッサ・メソッド**)、及び機能を実現する操作（**振る舞い・メソッド**）を含む実体(**インスタンスorオブジェクト**)の考え方を中心に展開します。クラスで定義されたフィールドとメソッドをクラスの**メンバー**と呼びます。

インスタンスは参照型の変数を通してメソッドやフィールドを利用することができます。これを「**概念を定義したひな形から、状態と機能（属性と操作）を持つ実体を作り出す**」と、表現します。

オブジェクト指向プログラミングは広大なトピックであり、すべての概念をカバーするには一朝一夕には叶いません。このセクションでは、OOPの基本を学びます。

2.クラスとインスタンス（オブジェクト）

クラスは、データを保持する変数(フィールド)と、機能を表す操作(メソッド)を含んでいる参照型の定義体です。クラスは単なるデータ形式と機能の定義であり、それ自体にデータを含めることはできません(実体がデータを保持します)。

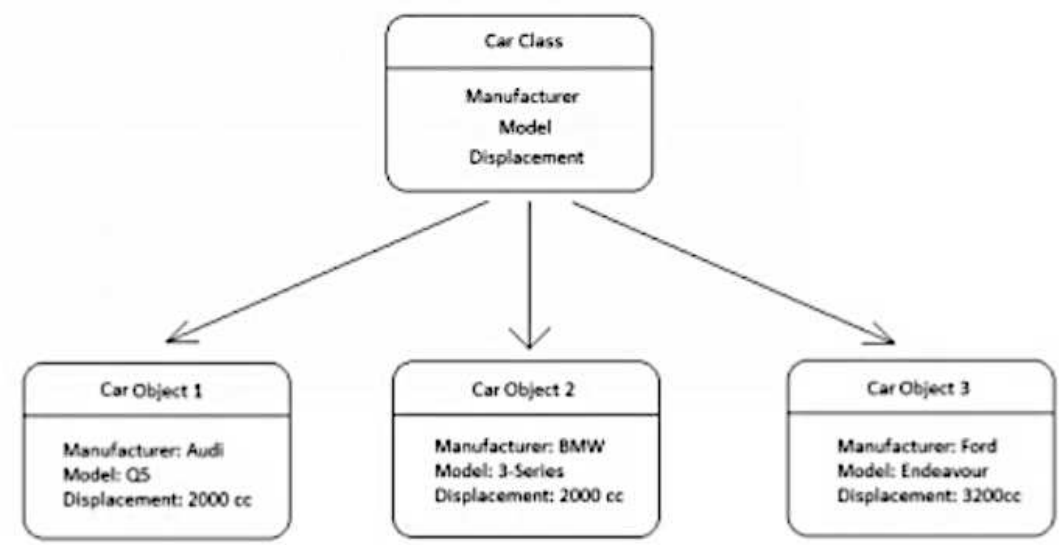
メソッドは、機能を実現すると共にクラスのフィールドにアクセスするためにも使用されます。これをセッターメソッド(**setter**)、ゲッターメソッド(**getter**)と呼びます。

インスタンスは、クラスで定義された独自のデータとメソッドのセットを持つ実行主体です。データメンバーは属性とも呼ばれます。

この概念をよりよく理解するために、例を見てみましょう。

Carというクラスを考えてみましょう。車を考えるときに頭に浮かぶのは、メーカー、モデル、エンジン排気量、燃料の種類など、特定の特性です。Carクラスの各インスタンスは、独自の属性セットを持つ異なる車です。つまりCarクラスから作られる実体としての車にはBMWやAudiやFord等いろいろあるということです。これを図で表してみましょう。

次の図のように考えます。



この図からわかるのは、まずCarと呼ばれるクラスがあるということです。**Carクラス**には、**製造元(Manufacture)**、**形式(Model)**、および**排気量(Displacement)**の3つのデータがあります。このクラスの3つのインスタンスは、Carオブジェクト1、Carオブジェクト2、Carオブジェクト3として作成されています。

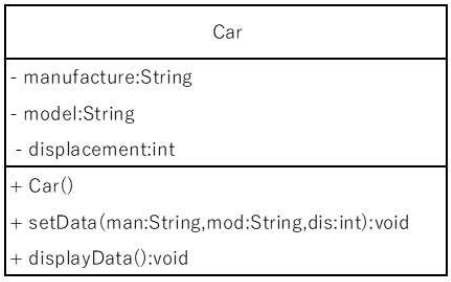
各オブジェクトは、製造元、モデル、および排気量の独自の値を持つ異なる車です。この例をプログラマーの観点から見ると、Carクラスは3つのデータを持つと認識することができます(製造者、モデル、排気量)。これらのデータにアクセスするために、setData()やdisplayData()などのメソッドを用意して、setData()で各データを設定し、displayData()で各データの値を表示させるように設計する必要がありそうです。

3. クラスの定義（設計）

このようなクラスの設計をUMLで表すと次のように表現できます。

【クラス図】

```
----------------------------------------------------------------------//クラス定義
Car
----------------------------------------------------------------//フィールド定義
-  manufacture:String
-  model:string
-  displacement:int
----------------------------------------------------------------//メソッド定義
+  Car()   //コンストラクタ
+  setData(man:String,mod:String,dis:int):void   //セッター
+  displayData():void
-------------------------------------------------------------------------------
－private
＋ public
＃ protected
```



- private , + public , # protected

4.クラスのメンバーの実装

クラスは、あらゆる関数からアクセスできるように、**public(つまり公開)に宣言**する必要があります。クラスを宣言する構文は次のとおりです。

【構文】

```
public class クラス名 {
   //フィールド定義
   //メソッド定義
}
```

一方で**クラスメンバー（フィールドとメソッド）**は、パブリック、プライベート、およびプロテクティッドの3つのアクセス修飾子の下で宣言できます。プロテクティッドは継承時に使われます。これは若干進んだトピックなので、ここではパブリックとプライベートのみを見ていきます。

パブリックメンバー：
パブリックメンバーは、プログラムのどこからでもアクセスできます。パブリックメンバーにアクセスするためのメソッドは必要ありません。

プライベートメンバー：
同じクラスのメソッドのみがプライベートデータにアクセスできます。メソッドがプライベートとして宣言されている場合、同じクラスのメソッドのみがプライベートメソッドを呼び出すことができます。これは、オブジェクト指向プログラミングにおけるデータ隠蔽の非常に重要な概念です。

【注】パブリックおよびプライベートの宣言としての制限はありませんが、通常、**データ(フィールド変数)はプライベートとして宣言**し、**メソッドはパブリックとして宣言**します。これは、オブジェクト指向の**カプセル化の概念**として「**データメンバーにアクセスする場合は必ずアクセス用のメソッド(アクセサメソッド)を利用する**」という考え方があるためです。

※カプセル化とは**情報と機能を一つにまとめて外部アクセスを制限**することです！

メソッドがプライベートに宣言されてしまうと、他のクラスからそれらのメソッドにアクセスする方法がなくなってしまいます。一方でデータメンバー(フィールド変数)がパブリックにされている場合、他のクラスからそのデータに直接アクセスできてしまいます。これではアクセサメソッドの必要もなくなってしまいますし、カプセル化の概念も崩れてしまいます。

そこで上記の例（Carクラス）を、Manufacturer、Model、Displacementがプライベートデータメンバーとして宣言し、setData(),displayData()はパブリックメソッドとして宣言することにします。

【ソースコード例】

```
public class Car {
   //フィールド定義
   private String manufacture;
   private String model;
   private int displacement;
   //メソッド定義
   public void setData(String man,String mod,int dis){
   }
   public void displayData(){
   }
}
```

メソッドには機能(操作)を定義する必要があります。今回、setDataメソッドにはインスタンス変数(フィールド)に値を設定し、displayDataメソッドには、その値を出力する機能(操作)を実装します。

【ソースコード例】setDataメソッド

```
this.manufacture = man;
this.model = mod;
this.displacement = dis;
```

※インスタンス変数に値を設定することを明示するためにthis.をつけています。今回、引数の名前とフィールド名が違うためthis.は付けなくても構いません。

【ソースコード例】displayDataメソッド

```
System.out.println("製造元："+manufacture);
System.out.println("形　式："+model);
System.out.println("排気量："+displacement);
```

5.クラスの実体化（データの操作）

以上の考え方は単なる定義であり、データに対する操作はこれまで行われていません。データを操作するにはクラスのインスタンスとなるCarクラスのオブジェクトを作り出す必要があります。オブジェクトに設定される独自のデータとメソッドのセットでオブジェクトの操作が可能になります。**クラスのオブジェクトを作り出すには、次の構文をmainメソッド内に記述します。**

【構文】
クラス名　参照型変数名 = new コンストラクタ名();

【ソースコード例】

```
Car c1 = new Car();
Car c2 = new Car();
Car c3 = new Car();
```

コンストラクタは、クラスと同じ名前を持つ特別な種類のメソッドで通常はインスタンスのデータメンバーを初期化するために使用されます。**new演算子でクラスに定義されたコンストラクタが呼び出され実体化される**と

考えて構いません。つまりコンストラクターが呼び出されるのは実体化の際の一度だけです。

コンストラクタは他のメソッドと同じように**オーバーロードできます**が、値を返すことはできません。ですから**void宣言は行いません**。つまり戻り値の宣言がされていない、**クラス名と同じ名前のメソッドはコンストラクタ**です。仮にプログラム内にコンストラクタを記述しなければ、コンパイラが自動的に何も処理を行わないコンストラクタを追加してくれます(もちろんプログラム内に記述すればそちらが優先されます)。

クラスの実体化には参照型の変数を宣言します。Car c1 = new Car();ステートメントは、Carクラス定義で定義された独自のデータメンバーとメソッドのセットを持つCar型のインスタンスc1を作成します。同様にしてc2,c3も作成されます。new演算子の後にはコンストラクタを記述します。**インスタンスのメソッドにアクセスするには、ドット（.）演算子**を次のように使用します。

【構文】
参照型変数名.メソッド名

ドット（.）演算子を使用してアクセスできるのは**パブリックメンバーのみ**です。

setData()は、インスタンスごとに個別に呼び出す必要があります。メソッドは3つのパラメーターを受け入れるため、3つの引数を指定する必要があります。

【ソースコード例】setDataメソッド

```
c1.setData（ "Audi", "Q5",2000）;
c2.setData（"BMW","3シリーズ",2000）;
c3.setData（"フォード","エンデバー",3200）;
```

displayData()は引数はありませんので次のように呼び出すことができます。

【ソースコード例】displayDataメソッド

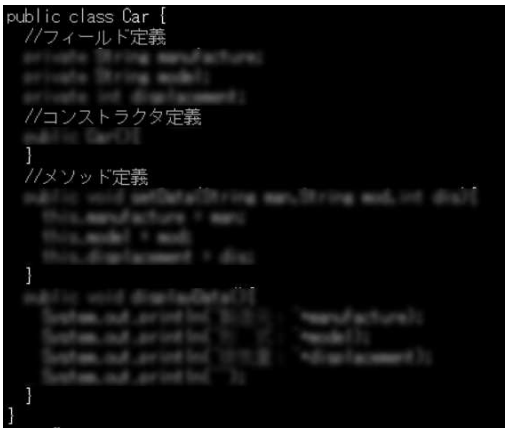
```
c1.displayData();
c2.displayData();
c3.displayData();
```

6.オブジェクト（インスタンス）の作成例

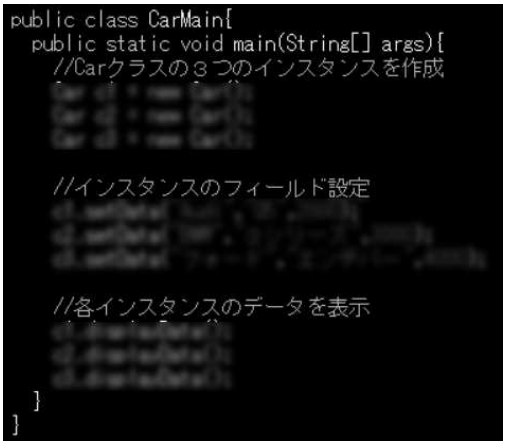
これらの概念をすべて組み合わせて、完全に動作するJavaプログラムを作成してみましょう。

【演習】
クラス定義と実行プログラムは別々に作成します。
両者のプログラムを作成し**コンパイルと実行**を行って出力を確認してください。

（Car.java）



（CarMain.java）



（実行結果）