3-1.クラス・フィールド変数

■クラス・フィールド変数

はじめに

この章では、Javaの基本的な構造を学びます。
各STEPでサンプルコードを記載していますので、出来ればEclipseでプロジェクトを作り試しながら読み進めて下さい。

このページで説明していることは、
ソースを読むにしても、書くにしても大切なポイントになります。
とは言っても、最初は難しいと思います。

ひとまず、読み進めていきましょう！

Step1: クラス

クラスとは、「 **データ** 」と「 **処理** 」の集まりです。
Javaのプログラムはこの「クラス」を複数組み合わせてプログラミングしていきます。
そのため、Javaのプログラミングは、大きく分けて 2 つの作業に分かれます。

> **①クラスの内部を実装（プログラミング）**

クラスには「データ」と「処理」を記述します。

・データ
そのクラスが持つ特徴・クラスに見合った情報などを指します
（例： **フィールド変数**

・処理
そのクラス内で行いたいタスク・TODO・ふるまいなどを指します
（例： **コンストラクタ** や **メソッド**

ただし、クラス内部のプログラミング時には、以下のようにいくつかルールがありますのでご注意ください。

1. `public class クラス名` と記述する。その場合、クラス名はファイル名と同じであること。
2. **「データ」** と **「処理」** を`{ }`（中括弧）で囲む

上記2のように中括弧で囲まれたプログラムコードのまとまりを一般的に **ブロック** と呼びます。

> **②データや処理を参照する**

クラスが完成したら、後述するクラスの「インスタンス（クラスのコピーのようなもの）」を生成し、
そのインスタンス経由でデータを参照し、処理（サンプルでは`System.out.println()`）を記述します。

> **サンプルコード「HelloJava.java」～データを参照して出力する** ?

```
public class HelloJava {

    // フィールド変数
    public String message = "Hello,Java World";

    // mainメソッド
    public static void main(String[] args) {

        // HelloJava(自分自身の)クラスのインスタンスを生成する
        HelloJava hj = new HelloJava();

        // フィールド変数の値を出力
        System.out.println(hj.message);
    }
}
```

> **解説①**

クラス名には、オリジナルの名前を記述することが可能ですが、
Javaではクラス名の最初の一文字は半角英大文字で記述するというルールがあります。
上記のプログラムでは、クラスに `HelloJava` という名前を付けています。

> **解説②**

クラスのブロック内には、 **データ（フィールド変数）** を宣言したり、
処理 のブロックを記述したりします。

上記のサンプルコードでは、「インスタンス」というクラスのコピー経由で`message` という名前のStringクラスのデータ（フィールド変数）を参照し、`System.out.println()`を使ってコンソールに出力しています。
データだけではなく、処理（メソッド）も参照できますが、それは後々のカリキュラムで教授します。

※例外を除いて、基本的にはインスタンス経由でデータや処理を呼び出せることを覚えておきましょう。

Step2: mainメソッド

プログラミングしたクラスを動かすため、自クラス又は他のクラス内に mainメソッド を記述します。
以下のサンプルコードで言えば、 Mainクラス に mainメソッドを記述していますね。
このmainメソッドは **プログラムのスタート地点** であり、オンリーワンのメソッドになります。

そのmainメソッドの中で「 new 」演算子を使用し、メモリ上にクラスが扱うデータ用の領域を新規に確保します。
このようにして **クラス** という 「型」 から 「具体的な実体」 を生成することを **インスタンス化** と呼び、
生成されたものを **インスタンス** と呼びます。

ややこしいことはわからん！！という方は、とりあえず、「インスタンスはクラスのコピー」という認識でOKです。
ただし、 **クラスとインスタンスはセットの関係** であり、 **インスタンスはクラスから生まれる** ということを覚えておきましょう。

サンプルクラス 「HelloJava.java」

```
public class HelloJava {

    // フィールド変数
    public String message = "Hello,Java World";

    // mainメソッド
    public static void main(String[] args) {

        // HelloJava(自分自身の)クラスのインスタンスを生成する
        HelloJava hj = new HelloJava();

        // フィールド変数の値を出力
        System.out.println(hj.message);
    }
}
```

【実行結果】

```
Hello,Java World
```

Step3: フィールド変数

フィールドまたは **フィールド変数** とは、 **クラス内の変数** のことです。
フィールドを宣言するとき、「 `public` 」や「 `private` 」などの **アクセス修飾子** を記述します。
アクセス修飾子は、省略することも可能です。
フィールドとして宣言した変数は、コンストラクタやメソッド内で変数を宣言することなく、
そのクラス内であればどこからでも値を参照したり更新したりすることができます。

サンプルクラス 「HelloJava.java」

```
// プログラムを実行しないクラス（インスタンス化されて、他から利用されるクラス）
public class HelloJava {

    // Stringクラスの、messageという名前のフィールド。
    // アクセス修飾子が「private」なので、このHelloJavaクラス内でのみ使用可能。
    private String message;

    protected void setMessage(String msg) {
        // フィールド変数にデータ(引数msg)を代入
        message = msg;
    }

    protected void printMessage() {
        // フィールド変数を出力
        System.out.println(message);
    }
}
```

サンプルクラス 「Main.java」

```
// プログラムを実行するためのクラス（mainメソッドを持つクラス）
public class Main {

    // mainメソッド
    public static void main(String[] args) {
        // HelloJavaクラスのインスタンス「hj」を生成する
        HelloJava hj = new HelloJava();

        // メッセージを設定するメソッドを呼び出す
        hj.setMessage("Javaって楽しいよね！");

        // メッセージを出力するメソッドを呼び出す
        hj.printMessage();
    }
}
```

【実行結果】

```
Javaって楽しいよね！
```

また、 **フィールド変数は、初期化を省略してもかまいません。** その場合は、デフォルト値が入ります。

Step4 : アクセス修飾子

アクセス修飾子とは、「public」、「protected」、「private」といった
修飾子（変数名、関数名の前に付与する単語）のことで、 **クラスや変数がどこからアクセス可能であるかの公開範囲** を決定します。

アクセス修飾子	概要
public	すべてのクラスからアクセスできる
protected	現在のクラスとサブクラスからアクセスできる
なし	現在のクラスと同じパッケージのクラスからアクセスできる
private	現在のクラスからだけアクセスできる

例えば、以下のようなコードがあったとします。

サンプルクラス 「ModAccess.java」

```
package testPackage;

public class ModAccess {
    public String pub_var = "publicだよ";
    protected String prot_var = "protectedだよ";
    String pack_var = "package privateだよ";
    private String pri_var = "privateだよ";

    public void test() {
        // クラス内部ではすべてのレベルのメンバーにアクセス可
        // this.(変数名)はフィールド変数のことを指すときに使用される
        System.out.println(this.pub_var);
        System.out.println(this.prot_var);
        System.out.println(this.pack_var);
        System.out.println(this.pri_var);
    }

    public static void main(String[] args) {
        // 自身のクラスのインスタンスを生成
        ModAccess m = new ModAccess();
        m.test();
    }
}
```

クラスで定義したアクセス修飾子を同一クラスから呼んだ場合は
どのアクセス修飾子にも問題なくアクセスできます。

【出力結果】

```
publicだよ
protectedだよ
package privateだよ
privateだよ
```

ModAccess.javaに書かれている変数を別のファイル（ModAccess.javaを継承）から呼んだ場合はどうでしょうか？

サンプルクラス 「ModAccessChild.java」

```
package testPackage;

// ModAccessのサブクラス(子クラス)
// 親：ModAccess、子：ModAccessChild
public class ModAccessChild extends ModAccess {
    public void test() {
        System.out.println(this.pub_var);
        System.out.println(this.prot_var);
        System.out.println(this.pack_var);
        // System.out.println(this.pri_var); ←プライベート変数にはアクセス不可
  }

    public static void main(String[] args) {
        ModAccess mc = new ModAccessChild();
        mc.test();
    }
}
```

【出力結果】

```
publicだよ
protectedだよ
package privateだよ
```

> **解説①**

ModAccess.javaを継承している（ `extends ModAccess` の部分）ので、
`private` 以外は出力できています！

また、 `private` な `pri_var` にはアクセスができないため、
実装時にはコンパイルエラーとなりエラー対象のコードに赤い下線が引かれます。

> **解説②**

継承とは？
例として、「もちもちした食感」のパンのクラス `class Pan` があるとします。
次に、パンのクラスを継承した「カレーパン」クラス `class CurryPan` を作ります。
カレーパンのクラスを作ったときに、後ろに `extends Pan` をつけるだけで継承は完了です。

次に、Panクラスにあった「もちもちした食感」をデータと考えると、
そのPanクラス(親クラス)を継承したCurryPanクラス(子クラス)も、「もちもちした食感」を持っているのです。

また、 `class CurryPan extends Pan` というのは、
日本語に変換すると、「カレーパンはパン(の一種)です」となります。

一方、クラス名を適当につけたとして `class Dog extends Pan` とすると、
「犬はパン(の一種)です」という意味不明な文になります。
すると、「もちもちした食感をもつ犬」が生まれてしまいます。

このことから、継承関係を持つ「親クラス」は「子クラス」を内包する名前にするのが良いです。
なぜなら今後自分以外の人がプログラムを見た際に混乱せずに済むからです。

> **ポイント**

コンパイルエラー時の対応策としては主に以下の3つとなります。

- 正しいコードに修正
- コメントアウト（ `//` ）することにより、コードとして認識させない
- 対象のコードを削除

アクセス修飾子のみでコンパイルエラーが発生する訳ではありませんので、
その他の実装でコンパイルエラーが発生した場合は上記3つの対応を参考にエラーを解消しましょう！

最後に、`ModAccess.java` に書かれている変数を
別のファイル（継承とかはしていない）から呼んだ場合はどうでしょうか？

サンプルクラス 「ModAccessOther.java」

```
package testPackageOther;

import testPackage.ModAccess;

// ModAccessと継承関係のない他パッケージのクラス
public class ModAccessOther {
    public static void main(String[] args) {
        ModAccess my = new ModAccess();
        System.out.println(my.pub_var);
        // publicのみアクセス可能
        // System.out.println(my.prot_var);
        // System.out.println(my.pack_var);
        // System.out.println(my.pri_var);
    }
}
```

このコードを実行すると下のように出力されます↓↓↓↓

【出力結果】

```
publicだよ
```

継承をしていないので、`public`以外は出力できません！

課題

提出課題はありませんので、一通り学習が終わったら次の章へ進んで下さい。

最終更新日時: 2022年 08月 21日(日曜日) 14:05