3-4.mainメソッド・インスタンス・メソッドの呼び出し

提出する 評定を受ける

■mainメソッド・インスタンス・メソッドの呼び出し

はじめに

この課題が終われば、もう簡単なコードは読めるはずです！
構造的な話で言えば、これまでやってきたこと以外で、Javaのコードは成り立っていません。
それくらい大切な部分です。焦らず慣れて行きましょう。

Step1: mainメソッド

mainメソッドはJavaのプログラムにおける全ての始まりです↓

```
...
public static void main(String[] args) {
    ...
}
```

> **mainメソッドの決まり事①**

mainメソッドは特別なメソッドです。
一番最初にmainメソッドの処理が実行されるため、
「プログラムのエントリポイント（入り口）」と考えるとよいでしょう。

> **mainメソッドの決まり事②**

mainメソッドは一つのクラスに一つしか記述することができません。

> **mainメソッドの決まり事③**

mainメソッドは、「 `public static void main(String[] args)` 」という形式で記述します。
一字一句間違えないように記述しましょう。

また、mainメソッドは`String`の配列、 `args`という名前の **引数** を持っています。
明確な使い所はありますが、今は頭の片隅にでも入れておきましょう。

Step2 : インスタンス

インスタンスとは、クラスの **実体** のことです。
クラスは、`int`や`boolean`と同じように、ひとつの **型** であり、クラスのままでは使うことができません。
使うためには、 **インスタンス化** という作業が必要になります。

?

クラスは、クラスを構成する概念や情報は存在するけれど、具体的な状態ではありません。
それ故、クラスは何かの **設計図** と例えられることが多いですが、
その **フワッとしたイメージを形にする作業がインスタンス化** です。

ざっくりですが、例えば、
CADというツールから **家の製図** を作成すれば、
作成された図面を元に大工さんが家を組み立てて **家** が完成します。

「皆さんが広告やチラシで目にする家の間取り＝クラス」 で、
「実際に居住可能になった家＝インスタンス化された実体」
といった感じでしょうか。

この概念の正解はひとつとは限らないため、この説明がしっくりくる方はより内容を理解し、
逆にしっくりこない方は、似たような考え方を用いてイメージを膨らませることが重要となります！

偏らずに色々な考え方ができると内容の理解も多少スムーズになります！
（Step3でも似たような考え方で説明していきます！

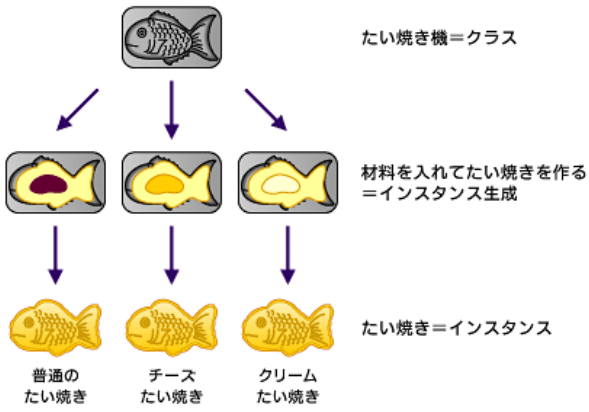
Step3: クラス型

例えば`Taiyaki`というクラスの変数であれば

```
Taiyaki tai = new Taiyaki();
```

のように記述して、new演算子によりTaiyakiクラスを呼び出して`tai`という名前で使用可能な状態にします。
このように、 **そのクラスを呼び出して使用できるようにすること** を、 クラスのインスタンス化（実体化）と呼びます。

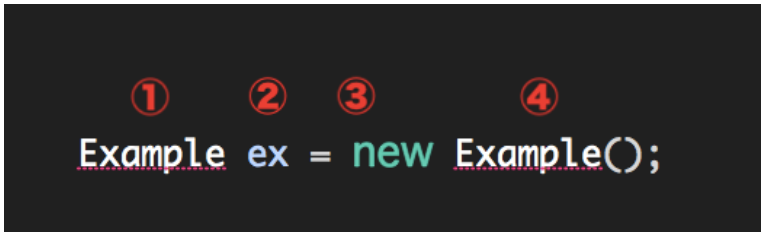
また、生成されたもの（実体）を **インスタンス** と呼びます。
たい焼きの型（クラス）をインスタンス化してたい焼き（実体）を作るみたいなイメージです。



生成されたインスタンスをもとにして、そのクラスに宣言してあるフィールドを参照したり、
定義してあるメソッドを呼び出したりすることができるようになります。
インスタンスの生成は、 **他のクラス内やmainメソッド内など、さまざまな場所で行うことができます。**

> **例**

例えば下記に示すものは、Exampleクラスのインスタンスを生成する記述です。



① クラス名
② インスタンス名（変数名）
③ 「=」で結んで、new演算子

④ コンストラクタ名

の順で記述します。

上記のインスタンスを生成した例ですが、最初のクラス名「Example」と
最後のコンストラクタ名「Example()」が **同じ** です。

これは、前のページのコンストラクタの説明にもあるように、
コンストラクタ名＝クラス名 であるためです。

```
クラス名 インスタンス名 = new コンストラクタ(引数の値);
```

また、インスタンスを生成するクラスのコンストラクタによっては引数の値を与えてインスタンスを生成しなければなりません。
これはインスタンスを生成するクラスのコンストラクタの作り方によって左右されます。

Step4: メソッドの呼び出し（メソッドの実行）

メソッドの呼び出しとは、 **作成したクラスのインスタンスを生成してから**
「インスタンス名.メソッド名()」**という形式でメソッドの呼び出しを行うこと** です。
例えば下記に示すものは、インスタンス名taiがeatという名前のメソッドを呼び出す例です。

Taiyaki.java

```
public class Taiyaki {

    public static void main(String[] args) {
        // Taiyakiクラスのインスタンスtaiを生成
        Taiyaki tai = new Taiyaki();

        // eatメソッドを呼び出す
        tai.eat();
    }

    public void eat() {
        System.out.println("たい焼き食べた");
    }
}
```

実行結果

```
たい焼き食べた
```

クラスのメソッドによっては、引数の値を与えてメソッドの呼び出しを行わなければなりません。
その場合の記述方法は以下のようになります。

```
インスタンス名.メソッド名(引数の値);
```

Step5: カプセル化

カプセル化とは、作成したクラスのデータ（フィールド）や処理（メソッド）を他のクラスから隠蔽することで、
作成したクラスに対する **アクセス方法を限定する** プログラミング技法です。

隠ぺいなどと難しい表現をしていますが、その実体は `private` です。

必要性は？

「カプセル化」することによって、作成したクラスが何をしてくれるクラスなのかが **明確** になり、他のクラスから使用しやすくなります。

また、 **オブジェクト指向** においてはそのようにプログラミングするのが一般的です。
具体的には、通常はフィールドのアクセス修飾子をprivateにして、メソッドのアクセス修飾子をpublicにしておきます。
そのようにすれば大概はカプセル化になっています。

アクセス修飾子の復習

アクセス修飾子とは、`public`、`protected`、`private`といった修飾子のことで、
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、 **猫** をクラスとして考えてプログラミングしていきます。

Cat クラス

```
public class Cat {

    private String name;

    public int age;

    public Cat(String n, int a) {
        name = n;
        age = a;
    }

    public void showName() {
        System.out.println("名前は、" + name + "です。");
    }

    public void showAge() {
        System.out.println("年齢は、" + age + "才です。");
    }
}
```

Main クラス

```
public class Main {

    public static void main(String[] args) {
        Cat pepe = new Cat("ペペ", 3);  // ←こちらで3歳と設定していますが、、、
        pepe.age = 0;  // ←ageはカプセル化されてないために、このように不正な代入が行われる可能性がある！
        pepe.showName();
        pepe.showAge();
    }
}
```

【実行結果】

```
名前は、ペペです。
年齢は、0才です。
```

解説

このCatクラスは、フィールドに猫の名前と年齢を持っています。
コンストラクタ では、引数に猫の名前と年齢の値を受け取り、それをもとにインスタンスを生成します。

Step6: フィールド変数の設定

名前の設定

名前（ `name` フィールド）は、`private`のアクセス修飾子がついているため、
このCatクラス内からしかアクセスできません。

Catクラス内では、nameに対する代入はコンストラクタでのみ行っています。

よって、Catクラスを使用する他のクラスがCat内の名前（ `name` ）を設定できるのは、
Catのコンストラクタを呼び出すとき（インスタンスを生成するとき）のみに制限されています。

年齢の設定

これに対して、年齢（ age フィールド）はpublicであるため、
Catクラス以外の他のクラスからも自由に値を参照したり変更したりすることができる ようになってしまっています。

このため、インスタンス生成時に年齢を設定したあと、
別の任意のタイミングで年齢の値を書き換えることができてしまいます。

上記の例では、
Mainクラスのmainメソッド内で、Catクラスのインスタンスpepeをまず3才として生成しています。

ここまでは正常ですが、その後、`pepe.age = 0;`の代入文により、年齢の値を不正に **0** にしてしまっています。

このageフィールドのように、カプセル化がされていないと如何様にも年齢の値の書き換えが可能ですし、
また、そのようにされてしまう危険性があります。

せっかく猫の歳を設定するためのコンストラクタを用意して、
年齢を **正しく設定させる操作を提供していても** 、
年齢の値を勝手に書き換えられてしまっては、元も子もありあません。

しっかりとカプセル化されていれば、このようなことは無くなり一本筋の通ったプログラムになります。

さらに加えると、上記の例の場合、
値がint型の範囲ならばエラーにならないため、-100(歳)というような年齢を
入れられてしまう可能性もあります。

値の格納を必ず「メソッドを介して行う」ように徹底 すれば、
チェックロジックを入れることができるため、
そのような不正な値の操作を回避することもできます。

【チェックロジックのサンプル】

仮に直接 `age` に値を設定してOKとした場合に、
`age` に設定する値をチェックするロジックになります。

Cat.java

```
public class Cat {

    private String name;

    public int age;

    public Cat(String n, int a) {
        name = n;
        age = a;
    }

    public void showName() {
        System.out.println("名前は、" + name + "です。");
    }

    public void showAge() {
        System.out.println("年齢は、" + age + "才です。");
    }

    // 引数のageが 0以上かどうかの真偽値を返すメソッドを追加
    public boolean validateNumber(int age) {
        // 当然ですが、(age > -1); でも可です
        return (age >= 0);
    }
}
```

Main.java

```
public class Main {

    public static void main(String[] args) {

        Cat pepe = new Cat("ペペ", 3);
        int age = -10; // ←不正な値を代入

        if (pepe.validateNumber(age)) {
            pepe.age = age;
        } else {
            System.out.println("設定する年齢がマイナスです！");
        }
    }
}
```

実行結果

```
設定する年齢がマイナスです！
```

課題

インポートした3-4のフォルダの中にプロジェクトがありますので、
指示通りにコーディングして、ファイルを提出して下さい。

評定概要

学生から秘匿	No
参加者	75
提出	51
要評定	3