7-7.抽象クラスとインターフェース

抽象クラス

抽象クラスは以下のように定義します。

```
//抽象クラス(スーパークラス)
abstract class A {
    //抽象メソッド
    abstract void SampleMethod();

    //非抽象メソッド(具象メソッド)
    void SampleMethod(){
        System.out.println("非抽象メソッド");
    };
}

//サブクラス
class B extends A {
    //オーバーライド
    public void SampleMethod() {
        System.out.println("aaa");
    }
}
```

クラスの定義時、普段使うpublicやprivate、protectedなどの代わりに、`abstract`という修飾子を付けます。

抽象クラスには抽象メソッドを定義することができ、こちらにも`abstract`を付けます。
また、抽象メソッドは処理の中身は書きません。
データ型やクラス名、引数の設定までを行い、引数を設定したカッコの後はすぐにセミコロンで閉じる`();`の形になります。
処理内容は未定で、後で設定するもの、というイメージを持っておきましょう。

また、抽象メソッドでない普通のメソッド(非抽象メソッド)も定義することができ、こちらは普通のクラスのメソッドのように処理内容を書きますし、オーバーライド無しで呼び出すことが可能です。

抽象クラスは、 **他のクラスから継承して使用します。**
継承は、 `class クラス名 extends 抽象クラス` という形式で行います。
この時、 **継承される親クラスをスーパークラス、継承する子クラスをサブクラス** とも呼びます。

抽象クラスの注意点

インスタンス化はできない

抽象クラスを直接インスタンス化することは出来ません。

```
    public static void main(String[] args) {
       A a = new A(); //コンパイルエラーとなる
    }
```

抽象クラスは、メソッドの処理内容などが無い、それだけでは未完成のクラスです。
なのでそれをインスタンス化しても使い道が無いというイメージを持っておきましょう。

抽象メソッドを必ずオーバーライドする必要がある

先程記載したように、抽象クラス内の抽象メソッドは、未完成なメソッドです。
それを継承先のサブクラスでオーバーライドすることで、中身が確定し、使用できるようになります。
オーバーライドしていないと、コンパイルエラーとなります。

逆に言うと、プログラム設計時の意図としては、必ずオーバーライドを強制させたいメソッドを抽象メソッドとして定義しています。
強制すると、プログラマが忘れていたり記述ミスをした時に気づくことが出来ますね。

?

複数継承することは出来ない

これは抽象クラスに限らず、普通のクラスも同じ仕様となりますが、継承は1つのクラスしか行うことができません。
`extends`の後のクラス名記載は1つだけと覚えておきましょう。

インターフェース

インターフェースは以下のように定義します。

```
interface A {
    //定数の定義
    int NUM1 = 1;
    int NUM2 = 2;

    //メソッド定義
    void x();

    //デフォルトメソッド(具象メソッド)
    default public void y() {
        System.out.println(NUM1);
    }
}

class Plus implements A {
    //オーバーライド
    public void x() {
        System.out.println(NUM1 + NUM2);
    }
}
```

インターフェースは、`interface インターフェース名`と定義します。
他のクラスに実装する時は、クラス定義時に`implements インターフェース名`を付けます。

インターフェースでは、 **定数とメソッドのみが定義できます。**

メソッドは、抽象メソッドと同様に処理内容を書かず、実装先のクラスでオーバーライドして処理内容を作ります。

また、メソッド先頭に`default`を付けることで、普通のメソッドと同じように仕様できるデフォルトメソッドを定義できます。

インターフェースの注意点

変数宣言は自動で定数となる

```
    //定数の定義
    int NUM1 = 1;
    int NUM2 = 2;
```

サンプルコードの上記の部分は **変数** の定義のように見えますが、インターフェース内で定義した変数には暗黙的に`public static final`が付与されます。
すなわち、 **定数**となります。

複数実装可能

インターフェースは複数の`implements(実装)`が可能です。
複数実装する場合は、`implements`の後に、コンマで区切りながらインターフェース名を指定します。

```
interface A {
    //省略
}

interface B {
    //省略
}

class Plus implements A, B {
    //省略
}
```

インターフェースへの継承は可能

`implements(実装)`と`extends(継承)`は、少し違う概念です。
通常のクラスにインターフェースを`extends`することはできません。
しかし、インターフェース同士での継承は可能です。

```
interface A {
    //省略
}

//インターフェースAを継承したインターフェースB
interface B extends A {
    //省略
}
```

共通の注意点

finalにできない

抽象クラスやインターフェースは、`final`にすることが出来ません。

```
// 以下はどちらもコンパイルエラー
final interface A {
}

abstract final class B {
}
```

`final`がついたクラスやインターフェースは、継承や実装が出来なくなります。
抽象クラスやインターフェースは、継承や実装、中身のメソッドを書き換えるオーバーライドなどを行う前提で作られているものです。
なので、finalを付けられない仕様になっています。

以上、これら抽象クラス・インターフェースの基本知識をもとにして、
次の章では継承全般の仕組みについて詳しく解説していきます！

課題

満点を取れるまで小テストを受験して下さい。

制限時間: 30 分

評定方法: 最高評点

合格点: 100 / 100

受験件数: 128