Javaプログラマ育成コース
第三回　オブジェクト指向とJava

ソフトシンク株式会社
代表取締役
周　順彩
zhousc@soft-think.com

SOFT-THINK ソフトシンク株式会社
Copyright (C) 2018 Soft Think Corporation. All Rights Reserved

目次

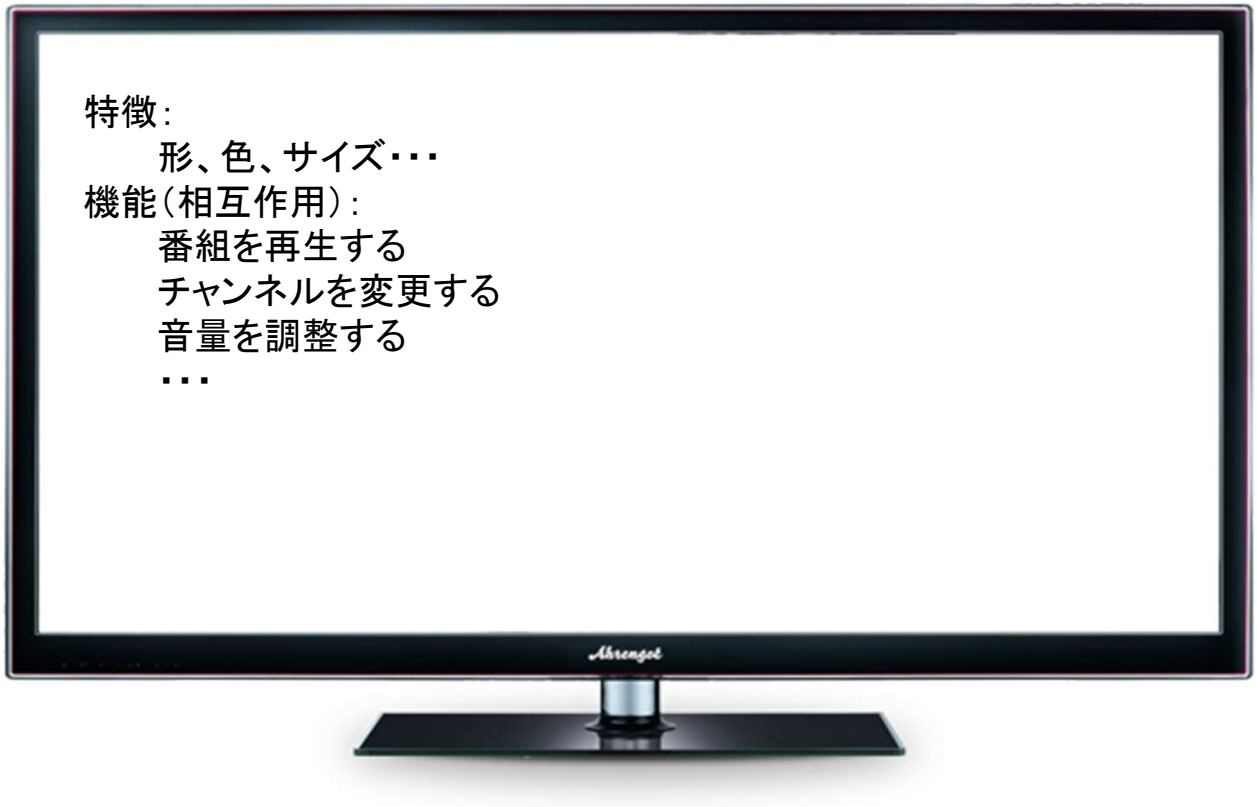
- オブジェクト指向とは？
- オブジェクト指向の3大要素－カプセル化（情報隠蔽）
- オブジェクト指向の3大要素－継承
- オブジェクト指向の3大要素－ポリモーフィズム（多態性）
- オブジェクト指向プログラミング（OOP）
- Javaのオブジェクト指向の実装－クラス
- Javaのオブジェクト指向の実装－抽象クラス
- Javaのオブジェクト指向の実装－インターフェース
- 演習
- 内部クラス
- 練習課題
- 次回の予習タスク

Copyright (C) 2018 Soft Think Corporation. All Rights Reserved

オブジェクト指向とは？

◆ オブジェクト指向（object-oriented ）とは

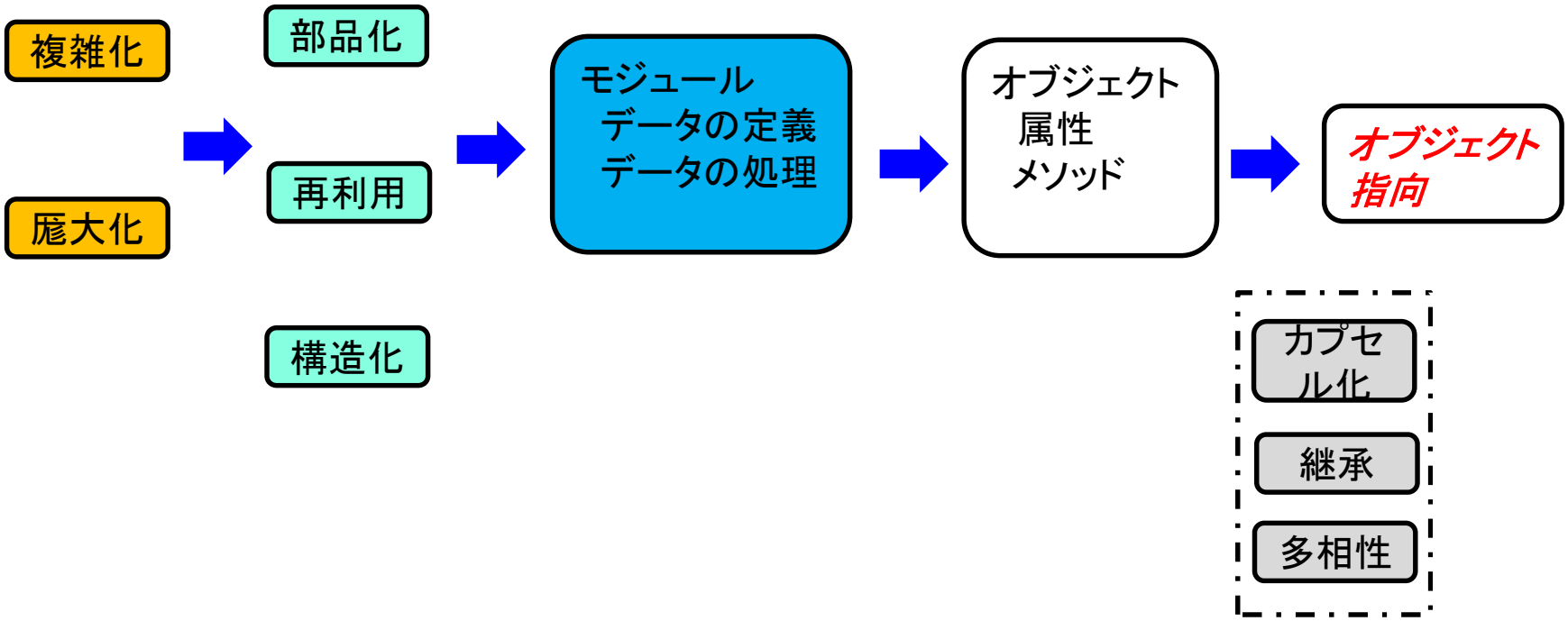
システム開発において、対象となる現実世界の物事をすべて抽象化したモデルのオブジェクトとし、そのオブジェクトの持つ特徴とオブジェクト間の相互作用を着目してシステム設計、開発する手法である。



Copyright (C) 2018 Soft Think Corporation. All Rights Reserved

オブジェクト指向とは？

◆ なぜオブジェクト指向？

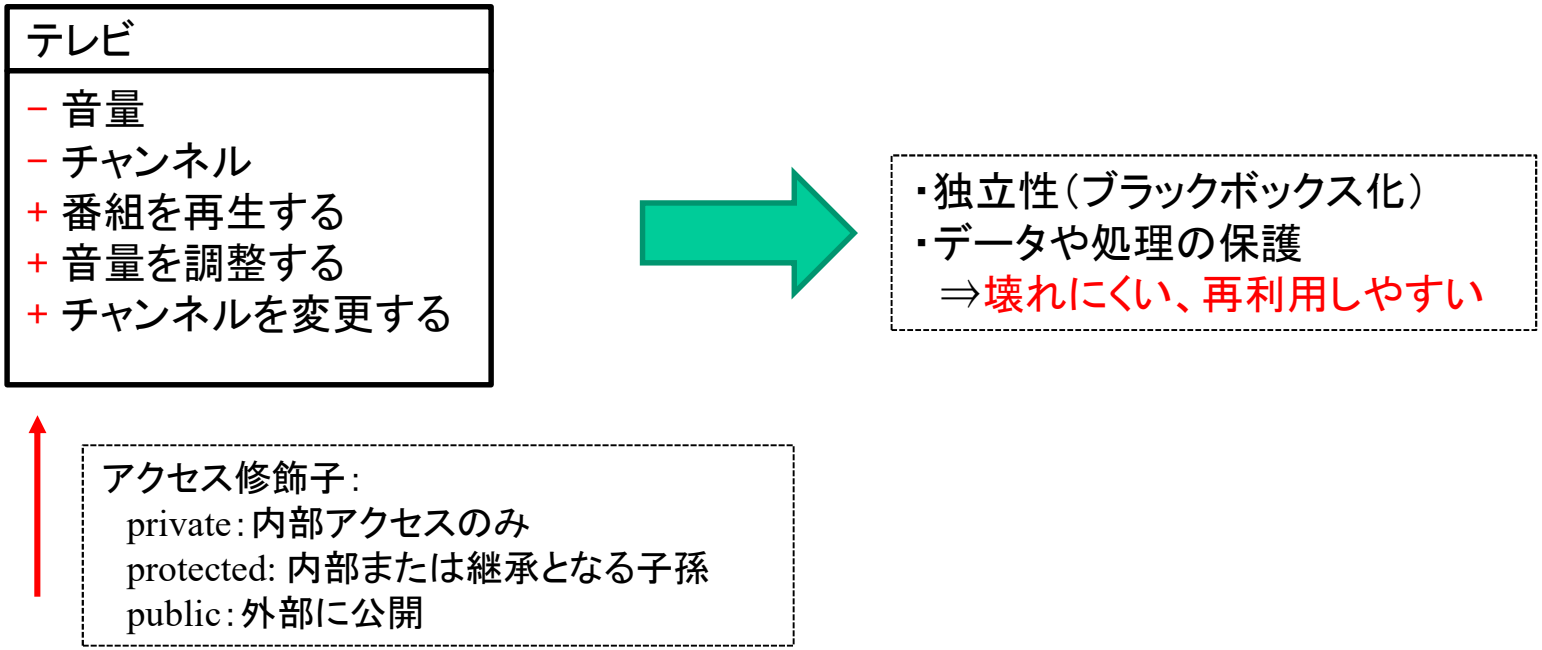


Copyright (C) 2018 Soft Think Corporation. All Rights Reserved

オブジェクト指向3大要素ーカプセル化

◆ カプセル化（情報隠蔽）

オブジェクトの独立性を高めるものとして、ほかのオブジェクトから直接利用される必要のないデータや処理を隠蔽すること。それを利用する場合は、外部から操作できる処理を経由して行う。

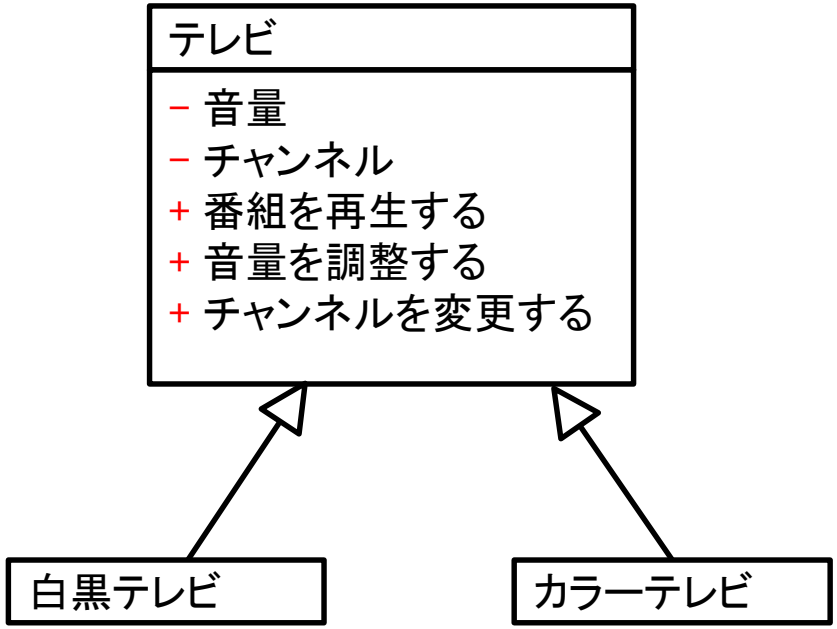


SOFT-THINK ソフトシンク株式会社
Copyright (C) 2018 Soft Think Corporation. All Rights Reserved

オブジェクト指向3大要素ー継承

◆ 継承

あるオブジェクト（A）がほかのオブジェクト（B）の特性（属性、メソッド）を引き継ぐこと。
両者の間は継承関係（A is B）を持つこととなる。



Copyright (C) 2018 Soft Think Corporation. All Rights Reserved

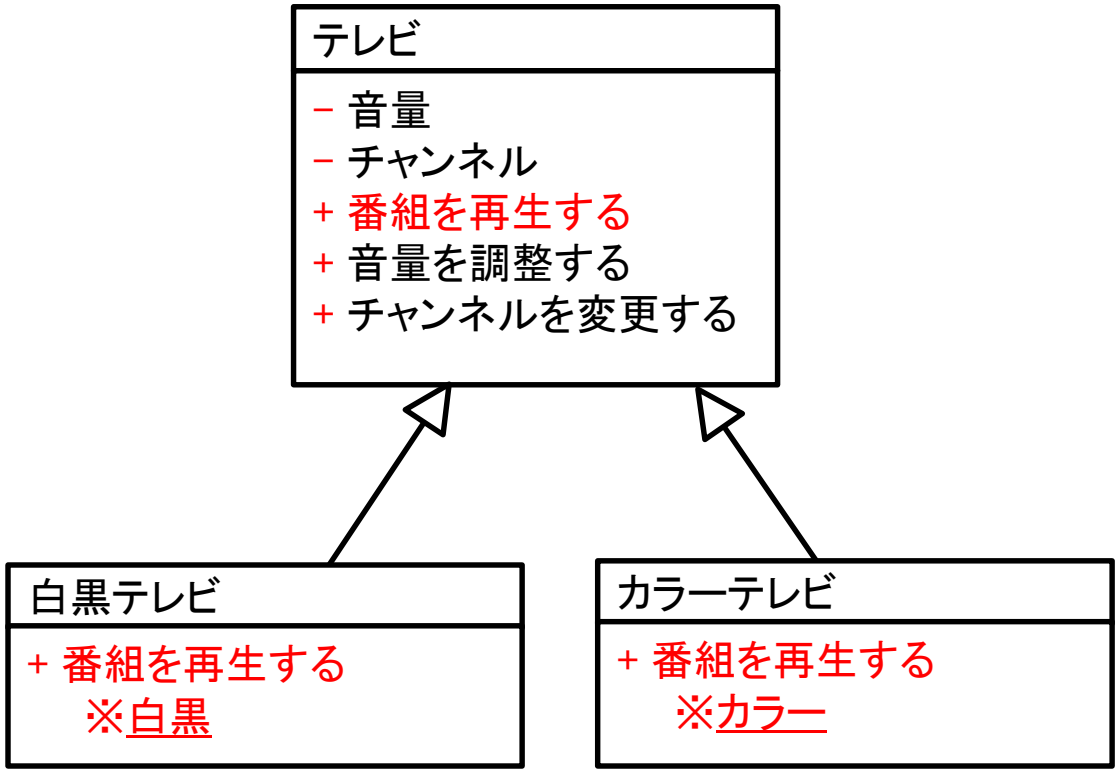
オブジェクト指向３大要素－ポリモーフィズム

◆ ポリモーフィズム（polymorphism）

同名のメソッドや型などをオブジェクトの種類に応じて使い分けることができる性質のこと。

狭義：**オーバーライド**

広義：**オーバーロード**



Copyright (C) 2018 Soft Think Corporation. All Rights Reserved

オブジェクト指向プログラミング

■ オブジェクト指向プログラミング

■ オブジェクト指向の概念や手法を取り入れたプログラミングのパラダイムである。

➢ オブジェクト指向プログラミング言語

- ◆ Simula 1962年
- ◆ Smalltalk 1972年
- ◆ C++ 1979
- ◆ Objective-C 1983年
- ◆ Python 1990年
- ◆ Ruby 1993年
- ◆ Perl 1994年
- ◆ Java 1995年
- ◆ Object Pascal (Delphi) 1995年
- ◆ Javascript 1996年
- ◆ PHP 2000年
- ◆ C# 2000年
- ◆ VB.NET 2002年

Copyright (C) 2018 Soft Think Corporation. All Rights Reserved

Javaのオブジェクト指向の実装ークラス

◆ クラス

Javaのオブジェクト定義。

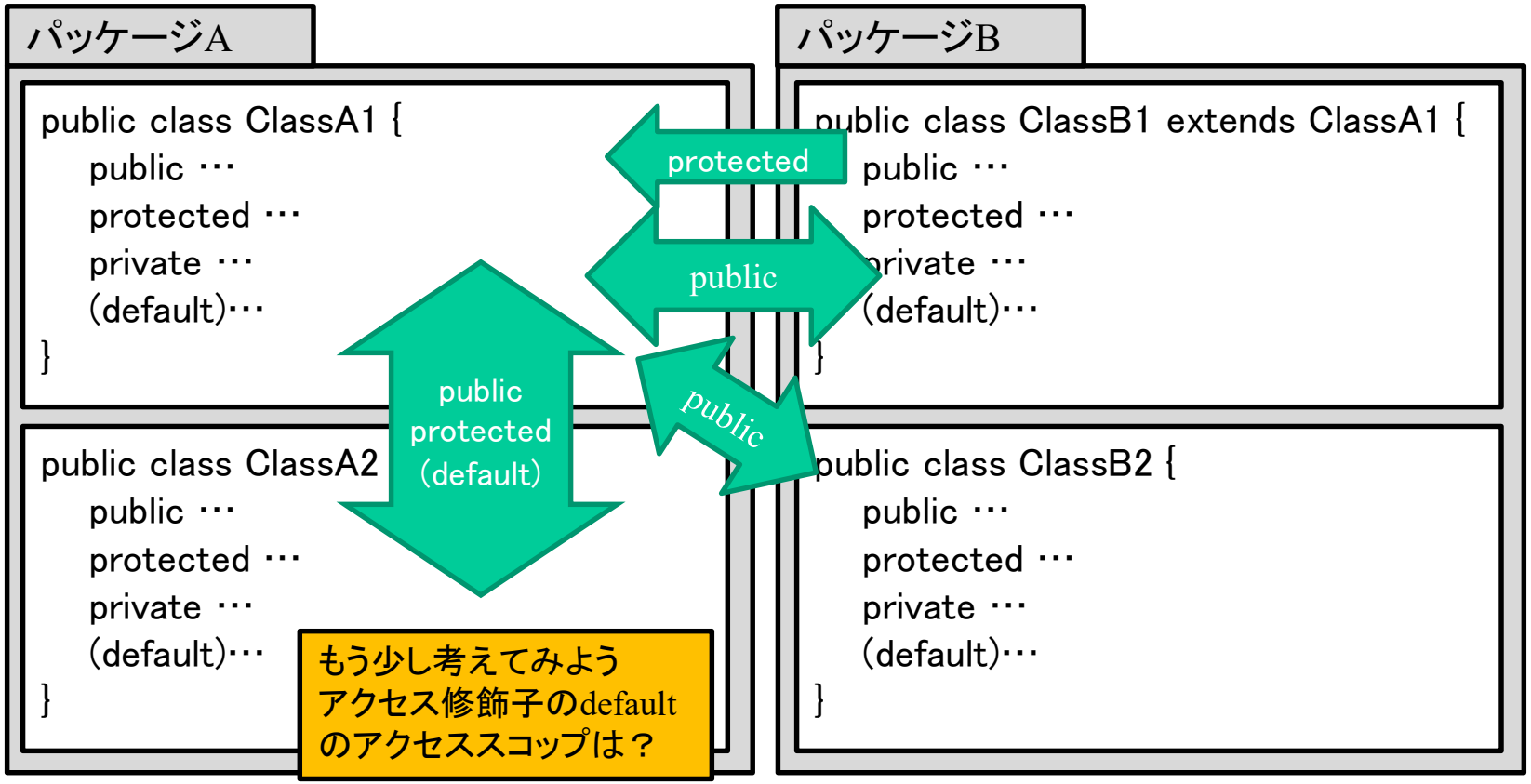
```
アクセス修飾子 class クラス名 extends 親クラス implements インターフェース {
    アクセス修飾子 [static] [final] 型 変数名 = 初期値;
    // 初期値は任意、初期値が設定されない場合は、型によってデフォルトの初期値を設定
    ・・・
    アクセス修飾子 クラス名() {
        // コンストラクタ
    }
    アクセス修飾子 クラス名(パラメータリスト) {
        // パラメータ付きコンストラクタ
    }
    @Override
    アクセス修飾子 戻り値型 メソッド名(パラメータリスト) {
        // 親クラスのメソッドをオーバーライド
    }
    アクセス修飾子 [static] 戻り値型 メソッド名(パラメータリスト) {
        // staticをつけると、インスタンスに紐づけないクラスメソッドとなる
    }
    ・・・
}
```


Copyright (C) 2018 Soft Think Corporation. All Rights Reserved

Javaのオブジェクト指向の実装－アクセス修飾子

◆ 種類

public/protected/private/(default:左記のいずれも指定しない場合。パッケージアクセスともいう)



Copyright (C) 2018 Soft Think Corporation. All Rights Reserved

Javaのオブジェクト指向の実装－final

◆ finalの用途

修飾先	説明
クラス	そのクラスを継承することができなくなる。
メンバー変数	定義時またはコンストラクタにて、一度のみ初期化する必要がある。 定義時に付与したら、それ以降どのタイミングも再付与不可。
クラス変数 (static final)	定数。定義時のみ値付与可能。
メソッド	そのメソッドはサブクラスでオーバーライドできなくなる。
パラメータ	パラメータへの再付与は不可。
ローカル変数	定義時を含め、値の付与は一度のみ。

Copyright (C) 2018 Soft Think Corporation. All Rights Reserved

Javaのオブジェクト指向の実装－抽象クラス

◆ 抽象クラス

継承されることを前提（必須）としたクラス。
new演算子でインスタンスを作成不可。
一般的に中身が無いメソッド(abstract)が1つ以上含まれている(implementsしているインターフェースのメソッドも含め）。

```
アクセス修飾子 abstract class クラス名 extends 親クラス implements インターフェース {
    アクセス修飾子 型 変数名 = 初期値;
    ・・・
    アクセス修飾子 クラス名() {
        // コンストラクタ
    }
    アクセス修飾子 クラス名(パラメータリスト) {
        // コンストラクタ
    }
    アクセス修飾子 abstract 戻り値型 抽象メソッド名(パラメータリスト); //中身ない
    ・・・
}
```


Copyright (C) 2018 Soft Think Corporation. All Rights Reserved

Javaのオブジェクト指向の実装－インターフェース

◆ インターフェース

外部に公開している抽象メソッド、default実装メソッド(JDK1.8から)、privateメソッド(JDK1.9から)と定数だけ定義するもの。一般的にクラスの抽象化したもの。
フィールドやメソッドを一切定義しないインターフェースはマーカーインターフェース。
抽象メソッドが1つだけ定義されているインターフェースは関数型インターフェース。

```
アクセス修飾子 interfaceインターフェース名 extends 親インターフェース {
    [public static final] 型 定数名 = 初期値;
    ・・・
    [public abstract] 戻り値型 メソッド名(パラメータリスト);
    [public/private] static 戻り値型 メソッド名(パラメータリスト) {
        // staticメソッド。アクセス修飾子は継承不可のため、protectedのみ指定不可。
        // privateの場合は、インターフェースのほかのstaticメソッド、
        // privateメソッド、defaultメソッドからのみ利用する想定。
    }
    default [public] 戻り値型 メソッド名(パラメータリスト) {
        ;
    }
    ・・・
}
// 赤字の部分は省略可、一般的には省略。
```


Copyright (C) 2018 Soft Think Corporation. All Rights Reserved

内部クラス

◆ 内部クラス(INNERクラス)

クラスの内部に定義するクラス。

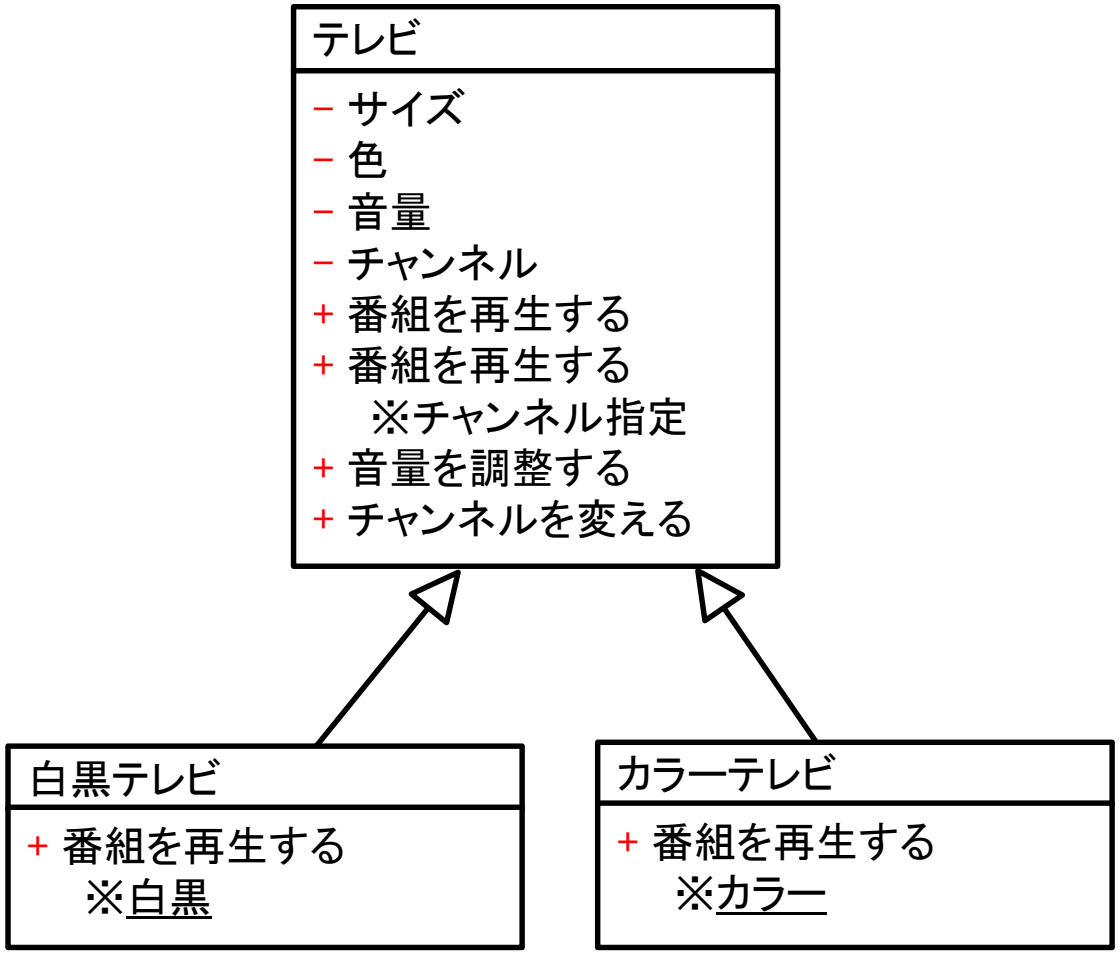
- 内部クラスはクラスをメンバーと同じように扱うことができ、private、protected、publicなどアクセス修飾子によってクラスのカプセル化が可能となる。
- 内部クラスからは同じクラス内のフィールド変数、メソッドを参照することができます。
- 内部クラスはメソッドの中でも定義できる。その場合、アクセススコープはメソッド内となる。
- static内部クラスは通常クラスと同じようなもの。ポイントはカプセル化できること。

➢ 内部クラスの利用

```
外部クラス名 外部クラスのオブジェクト名 = new 外部クラス名();
// 内部クラス
外部クラス名.内部クラス名 内部クラスのオブジェクト名 = 外部クラスのオブジェクト名.new 内部クラス名();
// static内部クラス
外部クラス名.内部クラス名 内部クラスのオブジェクト名 = new 外部クラス名.内部クラス名();
```


Copyright (C) 2018 Soft Think Corporation. All Rights Reserved

演習：テレビ



Copyright (C) 2018 Soft Think Corporation. All Rights Reserved

演習ポイント

◆ Javaにおけるオブジェクト指向

- クラス定義
- メンバー変数(protected, private)、メソッド(public) ⇒カプセル化
- 抽象クラス、クラス継承⇒継承
- finalメンバー変数、メンバー変数の初期化
- コンストラクタ、コンストラクタパラメータ
- メソッドオーバーロード、メソッドオーバライド⇒多相性
- テンプレートメソッド

Copyright (C) 2018 Soft Think Corporation. All Rights Reserved

練習課題

◆ 加算器の拡張

- 四則演算できるようにする
- オブジェクト指向で再定義する

Copyright (C) 2018 Soft Think Corporation. All Rights Reserved

次回の予習タスク

- JDKライブラリの利用

Copyright (C) 2018 Soft Think Corporation. All Rights Reserved