OPEN STREAM

212.オブジェクト指向(3)
抽象クラスとインタフェース・多態性

Make IT your Success

目次

この講義について

目的：

オブジェクト指向の考え方や、基本的な用法

- 抽象クラス
- インタフェース
- ポリモフィズム

について学ぶ。

ゴール：

オブジェクト指向の基本概念を理解し、抽象クラス、インタフェース、ポリモフィズムを利用したプログラムが書くことが出来る。

多態性

カプセル化、継承に続き、
抽象クラス・インタフェース・多態性を解説する。
・・・抽象化・多態性

① カプセル化
② 継承
③ 多態性

抽象クラス

抽象クラスとは

クラス設計で親クラス・子クラスを考えたとき

■メソッドの詳細な内容をかけない場合がある。

・親クラスではメソッドを書かず、子クラスで書く
→子クラスのメソッドの実装忘れが生じるかも

・親クラスは処理を書かないメソッドを用意し子クラスでオーバーライドする
→何もしないメソッドと見分けがつきにくい

■クラスがインスタンス化出来ると、問題がある場合がある。

これらをオブジェクト指向の継承で表現したい時に利用する。

抽象クラス

- 抽象クラスとはそれ自身でインスタンス化できないクラス
- 抽象メソッドは具体的な処理内容を持たないメソッド
  抽象メソッドはサブクラスが具体的な処理内容を記述（実装）しなければならない。

抽象クラス

抽象クラスと抽象メソッド

書き方：

```
// 抽象クラス
アクセス修飾子 abstract class 抽象クラス名 {
    // フィールド等

    // 抽象メソッド
    アクセス修飾子 abstract 戻り値の型 抽象メソッド名(引数の型 引数名, ...);
}

// 抽象クラスの継承
アクセス修飾子 class クラス名 extends 抽象クラス名 {
    // フィールド等

    // 抽象メソッドの実装
    アクセス修飾子 戻り値の型 メソッド名(引数の型 引数名, ...){
        // 処理
    };
}
```

抽象クラスの留意事項（注意事項）

- 抽象クラスはインスタンス化（実体）出来ない。
- 抽象メソッドを定義したクラスは抽象クラスにしなければならない。
- 抽象クラスにフィールド変数、非抽象メソッドを定義してよい。
- 抽象クラスには抽象メソッドを定義しなくても良い。

不変クラス

インスタンス生成後にインスタンスの中身が変更できないクラス
変数に値を再代入した場合は、再代入した値のインスタンスが生成されて参照が変わる。

不変クラスには、String、基本データ型のラッパークラス、BigInteger、BigDecimalがある。

不変クラスの規則

- オブジェクトを修正するメソッドをもたない
- どのメソッドもオーバーライドされないことを保障する
- 全てのフィールドをfinalにする
- 全てのフィールドをprivateにする
- 参照型フィールド変数に対する唯一のアクセス権をもつ

Memo ……Immutable(不変)

不変クラスの様にフィールドを変更不可能なオブジェクトをImmutable(不変)なオブジェクトとよぶ。

Immutableオブジェクトは参照のコピーのみでオブジェクトの複製ができる。

また、Immutable Programing は変数やオブジェクトのフィールドが変わらないので、テストなどが容易になる。

インタフェース

こちらもクラスを抽象化する仕組み。

機能の概要(抽象メソッド)を持つ。クラスがインタフェースを実装して機能の概要(抽象メソッド)を具体的な処理(具象メソッド)に置き換える。

そうすることで開発効率・保守性を向上する仕組み。

書き方：

```
// インタフェース
アクセス修飾子 interface インタフェース名 {
        // 定数
        // ※インタフェースのフィールドは定数なので、static fainalは基本的に省略する
        //      またインタフェースの定数は必ずpublicになるため、publicも省略する
        型 定数名 = 初期値;

        // 抽象メソッド
        // ※インタフェースのメソッドは抽象メソッドなので、abstractは基本的に省略する
        //      またインタフェースの抽象メソッドは必ずpublicになるため、publicも省略する
        戻り値の型 抽象メソッド名(引数の型 引数名, …);
}

// インタフェースの実装クラス
アクセス修飾子 class クラス名 implements インタフェース名 {
        // メソッドの具体的な処理内容を記述（抽象メソッドの実装）
}
```

インタフェースの実装クラスは、インタフェースを介して他のクラスから利用される。

つまり、インタフェースで定義される物(定数、抽象メソッド)はある種の契約(定義)といえる。

そのため、原則的にインタフェースで定義する定数、抽象メソッドは公開される(publicになる)

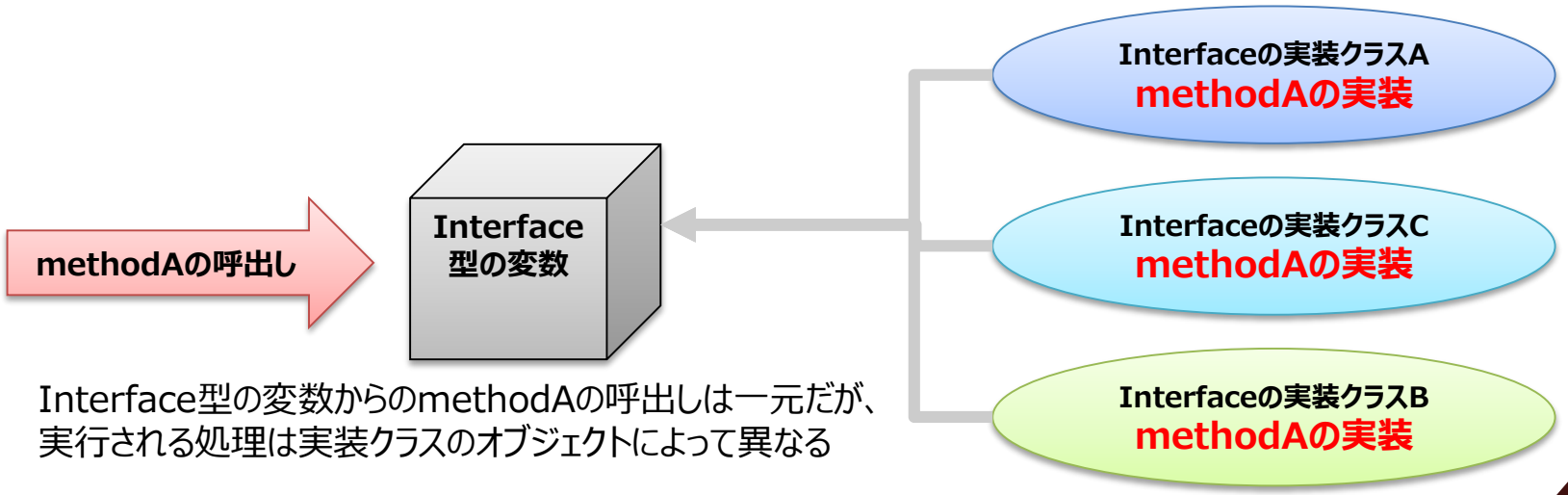
インタフェースの留意事項（注意事項）

- インタフェースはインスタンス化（実体）出来ない。
- インタフェースには定数と抽象メソッドしか定義できません。
  インタフェースが持つフィールドは必ず定数
  → 自動的にpublic static final になる。
  インタフェースが持つメソッドは必ず抽象メソッド
  → 自動的にpublic abstract になる。
- クラスはインタフェースを複数実装することが出来る。

多態性（ポリモーフィズム）

特定のインターフェースや親クラスを通じてメソッドにアクセスする時に生じる性質で、
アクセス自体は統一的に行われますが、そのメソッドが実行する具体的な処理は、使用されている具体的なインスタンス（オブジェクト）により異なります。

例）
親クラス型の変数に子クラス型のインスタンスを代入しているとする。
実際には子クラス型ではあるけれど親クラス型の変数であるので、
親クラスのメンバは呼び出すことは出来ますが、子クラスのメンバを呼び出すことは出来ません。
ここで子クラスで親クラスのメソッドをオーバーライドしていて、そのメソッドを呼び出したらどうなるか？
（親クラス・子クラスの両方にあるメソッド）
親クラス型の変数から子クラスのメソッドが呼び出す事ができる



Interface型の変数からのmethodAの呼出しは一元だが、
実行される処理は実装クラスのオブジェクトによって異なる

インタフェース

多態性（ポリモーフィズム）のメリット

- プログラムが複雑化しない(コードの再利用性)

  あるプログラムを考えるとき、インタフェース型(親クラス型)だけで考えて記述することが出来る。
  多態性がなかったら子クラス型毎にプログラムを記述しなければならない。

- 新しいクラスが容易に利用できる(オブジェクトの拡張性)

  新たなクラスを追加する場合、既存のインターフェースまたは親クラスを実装または継承していれば、既存のコードを変更することなく、新たなクラスのオブジェクトを扱うことができます。

- クラスの実装内容を意識せずに実装を行う事ができる(抽象化)

  オブジェクトの具体的なクラスを意識せずに、インターフェースや親クラスを操作する事だけを考えることで、システム全体をより抽象的に考えることができます。