

Java入門：クラス の利用

モデリングとシミュレーション

1

2019年度

今日のテーマ

- Javaのクラス構造を理解
 - クラスとインスタンス
 - クラスの拡張・継承
- 整列プログラムを例に

クラスとは

- モノの抽象化されたもの
 - 一般名詞に相当
 - 具体的モノの一般化・抽象化
 - 具体的モノ→インスタンス
- データの塊とそれらの操作
 - データ：フィールド
 - 操作：メソッド

クラスとその利用

```
public class クラス名 {  
    フィールド  
    コンストラクタ  
    メソッド  
}
```

■ クラスの定義

■ コンストラクタ：初期化

- クラス名と同じ名前のメソッド

■ インスタンス生成

```
クラス名 インスタンス名 = new クラスコンストラクタ
```

■ メソッド利用

```
インスタンス名.メソッド
```

- メソッドは戻り値を持つことも可能

```
public class AbstractStudent {
```

```
    //クラス内のフィールド
```

フィールド

```
    //name と studentIDは一度定めると変更できない
```

```
    protected final String name; //名前
```

```
    protected final int studentID; //学生番号
```

```
    public AbstractStudent(String name, int studentID) {
```

```
        this.name = name;
```

```
        this.studentID = studentID;
```

```
    }
```

コンストラクタ

```
    // 取得メソッドと設定メソッド
```

メソッド

```
    public int getStudentID() {
```

```
        return studentID;
```

```
    }
```

```
    public String getName() {
```

```
        return name;
```

```
    }
```

```
}
```

アクセス制限

- ▶ フィールド、メソッドへのアクセス制限
 - ▶ public : どこからでも利用可能
 - ▶ private : 同一クラス内からのみ
 - ▶ protected : 同一・継承クラスからのみ
 - ▶ 指定なし : 同一package内からのみ

static キーワード

- ▶ **static**とすると、アプリケーション起動時から利用できる
 - ▶ インスタンス生成は不要
- ▶ 使い方
 - ▶ 定数
 - ▶ 述語的な機能：サブルーチン、数学関数など
- ▶ 例：Mathクラス内で定義
 - ▶ `Math.random()`
 - ▶ `Math.PI`

finalキーワード

- ▶ 一度設定すると変更できない変数
 - ▶ 定義時に値を確定する
 - ▶ コンストラクタで一度だけ設定する
- ▶ 注意：クラスインスタンスの場合
 - ▶ インスタンスが変更できないこと
 - ▶ インスタンス内のフィールドは変更可能
 - ▶ 例: `final List<Integer> list`

変数のスコープ (scope)

変数の有効範囲

- 空間的スコープ
 - 変数が定義された場所から、当該プログラムブロック内
 - 例外：クラスフィールドは、どこに書いても良い
- 時間的スコープ
 - 変数が定義されたときから、当該プログラムブロック終了まで
 - `static`と付くと、アプリケーションの起動時から終了時まで

一般名詞の階層とクラスの階層

■ 一般名詞の階層

■ 例：生物→動物→哺乳類→ヒト

■ 抽象性⇔具体性

■ 抽象クラス

■ 抽象度、一般性の高いクラス

■ 継承クラス

■ 基となるクラスに具体性を付与

クラスの拡張・継承

- より具体的な属性や機能の追加
- 属性の追加
 - フィールドの追加
- 機能の追加
 - メソッドの追加
- 機能の具体化
 - 抽象メソッドの実装

クラスの拡張・継承

- フィールドやメソッドを追加する
 - 親のクラスのフィールドやメソッドは存在している
 - private ならば直接利用できない
 - protected ならば、拡張クラスからは利用可能

クラスの拡張・継承

- 継承クラスのコンストラクタ
 - 親クラスのコンストラクタをsuperとして利用する
- 継承クラスのメソッド
 - 同名の親クラスのメソッドを上書き可能
 - 親クラスのメソッドの利用も可能

```
public class Student extends AbstractStudent {
```

```
    private int record = 0; //点数
```

フィールドの追加

```
    /**
```

```
     * @param name 名前
```

```
     * @param studentID 学生番号
```

```
     */
```

```
    public Student(String name, int studentID) {
```

```
        super(name, studentID);
```

```
    }
```

AbstractStudentsクラスの
コンストラクタを使用

```
    public int getRecord() {
```

```
        return record;
```

```
    }
```

メソッドの追加

```
    public void setRecord(int record) {
```

```
        record = Math.max(0, record);
```

```
        record = Math.min(100, record);
```

```
        this.record = record;
```

```
    }
```

```
}
```

クラス拡張のイメージ

```
class Student
//フィールド
int record;
```

```
String name;
int studentID;
//メソッド
getName()
getStudentID()
```

```
getRecord()
setRecord()
```

AbstractStudentで
定義済み

内部にAbstractStudentク
ラスのインスタンスがある
訳ではないことに注意

クラスの利用

■ クラスインスタンスの生成

```
Student s = new Student(name,id);
```

■ メソッドの利用

```
s.setRecord(100);  
int r = s.getRecord();
```

■ クラスインスタンスの配列を作ることができる

```
Student students[] = new Student[10];  
students[0] = new Student(name,0);
```

```
package studentSample;
```

```
public class StudentMain {
```

```
    /**
```

```
     * StudentRecordクラスを実行するためのmain
```

```
     *
```

```
     * @param args the command line arguments
```

```
     */
```

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

```
        //データの生成
```

```
        String names[] = {
```

```
            "Aoyama", "Asou", "Baba", "Edo", "Funaki",
```

```
            "Goto", "Gunji", "Ikeuchi", "Ito", "Mori"
```

```
        };
```

```
        int records[] = {90, 70, 88, 95, 100, 60, 45, 80, 95, 55};
```

```
        Student students[] = new Student[names.length];
```

```
        for (int i = 0; i < names.length; i++) {
```

```
            students[i] = new Student(names[i], i);
```

```
            students[i].setRecord(records[i]);
```

```
        }
```

```
    }
```

```
}
```

泡立ち法

最も簡単な整列アルゴリズム

➡ 大きさ n の配列 A

```
for (i = n - 1; i > 0; i--) {  
    for (j = 0; j < i; j++) {  
        if ( A[j] > A[j+1] ){  
            j 番目とj+1 番目の要素を入替  
        }  
    }  
}
```

比較は何回実施される

- ▶ 内側のループ
 - ▶ 各*i*に対して*i*回
- ▶ 外側のループ
 - ▶ *i*の値を*n* - 1から1まで
- ▶ 比較の総回数

$$\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$$

比較

- ▶ 何を比較して整列するか？
 - ▶ Studentクラスの場合には、recordフィールドの値
- ▶ 整列アルゴリズムは一般的方法
 - ▶ 対象毎にコードを書くことは無駄

次回

- 基本的クラスの利用
 - リスト、集合、写像
- 抽象クラス、インターフェイス
- 型パラメタ