Java入門：クラスの拡張

モデリングとシミュレーション

2020年度

```
public class AbstractStudent {
1.
2.      //クラス内のフィールド
3.      //name と studentIDは一度定めると変更できない
4.      protected final String name; //名前
5.      protected final int studentID; //学生番号

6.      public AbstractStudent(String name, int studentID) {
7.          this.name = name;
8.          this.studentID = studentID;
9.      }
10.         // 取得メソッドと設定メソッド
11.     public int getStudentID() {
12.         return studentID;
13.     }

14.     public String getName() {
15.         return name;
16.     }
17.}
```

フィールド

コンストラクタ

メソッド

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

AbstractStudentsクラスの拡張

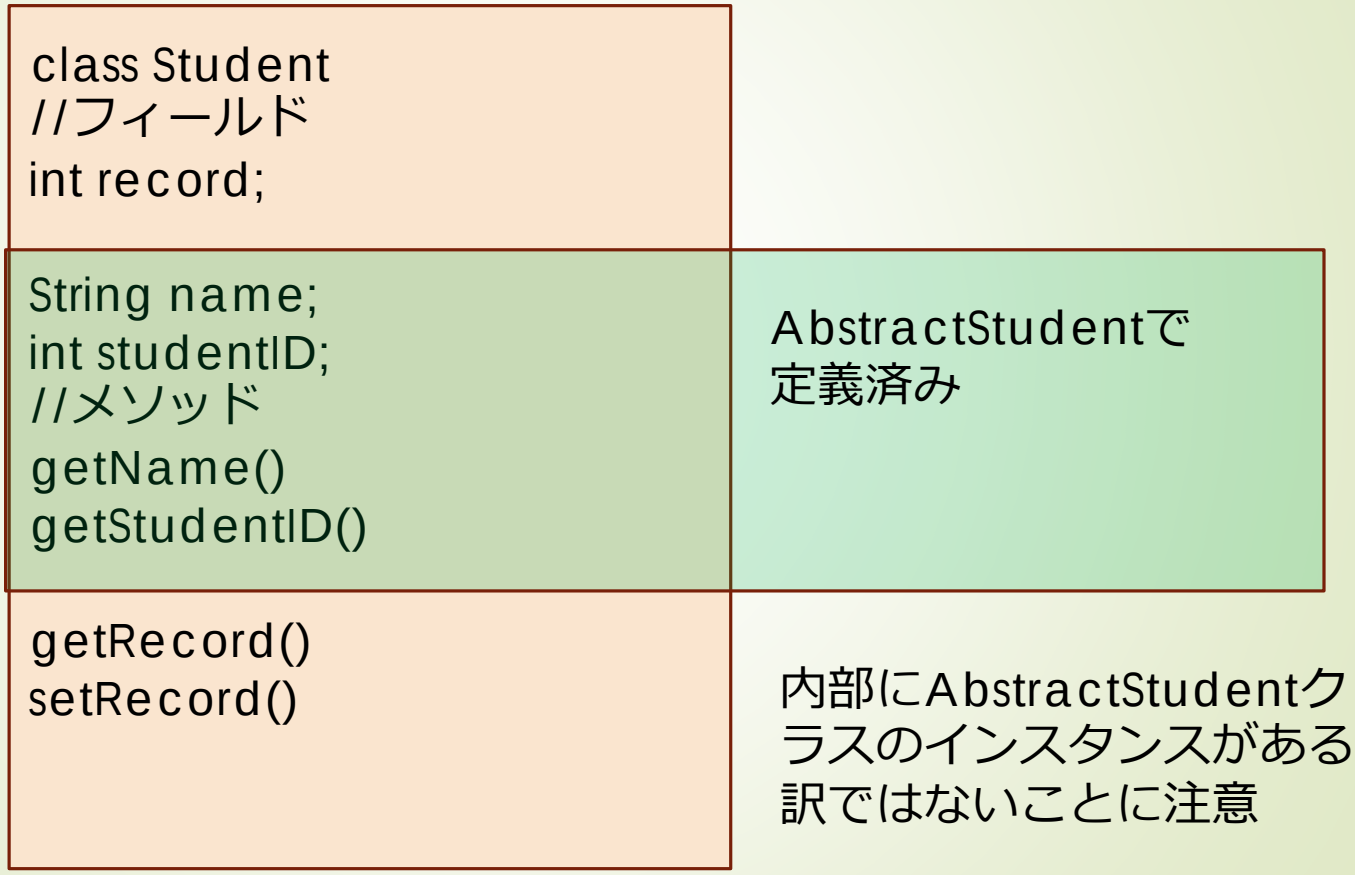
```
1.  public class Student extends AbstractStudent {

2.      private int record = 0;    //点数

3.      /**
4.       * @param name 名前
5.       * @param studentID 学生番号
6.       */
7.      public Student(String name, int studentID) {
8.          super(name,studentID);
9.      }

10.     public int getRecord() {
11.         return record;
12.     }
13.     public void setRecord(int record) {
14.         record = Math.max(0,record);
15.         record = Math.min(100,record);
16.         this.record = record;
17.     }
18.}
```

フィールドの追加

AbstractStudentsクラスのコンストラクタを使用

メソッドの追加

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

- メソッドの利用

```
student.setRecord(100);
int r = student.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);
```

```
1.  package studentSample;

2.  public class StudentMain {

3.      /**
4.       * StudentRecordクラスを実行するためのmain
5.       *
6.       * @param args the command line arguments
7.       */
8.      public static void main(String[] args) {
9.          //データの生成
10.         String names[] = {
11.             "Aoyama", "Asou", "Baba", "Edo", "Funaki",
12.             "Goto", "Gunji", "Ikeuchi", "Ito", "Mori"
13.         };
14.         int records[] = {90, 70, 88, 95, 100, 60, 45, 80, 95, 55};

15.         Student students[] = new Student[names.length];
16.         for (int i = 0; i < names.length; i++) {
17.             students[i] = new Student(names[i], i);
18.             students[i].setRecord(records[i]);
19.         }
20.     }
21.}
```

継承クラスを親クラスとして利用

```
AbstractStudent student = new Student(name,id);
```

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