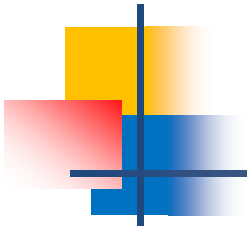
情報工学実験　実験テーマ２



# クラスの作り方

～Collectionsの違い：List,Stack,Map～

2020年10月23日

電気情報工学科　　田島　孝治

# 基本課題

- 次のデータを計算機上で表現し・・・

| Number | Name | Score1 | Score2 | Score3 | Score4 |
|---|---|---|---|---|---|
| 1 | Michael | 75 | 86 | 89 | 31 |
| 2 | Emma | 74 | 63 | 70 | 48 |
| 3 | Hannah | 73 | 45 | 82 | 31 |
| 4 | Emily | 40 | 30 | 49 | 48 |
| 5 | Daniel | 46 | 59 | 47 | 70 |

  - 名列番号順(Number)に表示(ArrayList)
  - 名列番号の逆順に表示(Stack)
  - 名前順(Name)にソートして表示(HashMap)

を実現せよ

# 発展課題

- 先ほどのデータに対し・・
  - 名前と平均点を表示できるようにせよ
  - 平均点でソートして表示せよ
  - 名前がEで始まる学生だけを表示せよ
    - 文字と文字列の検索を行う

# 講義

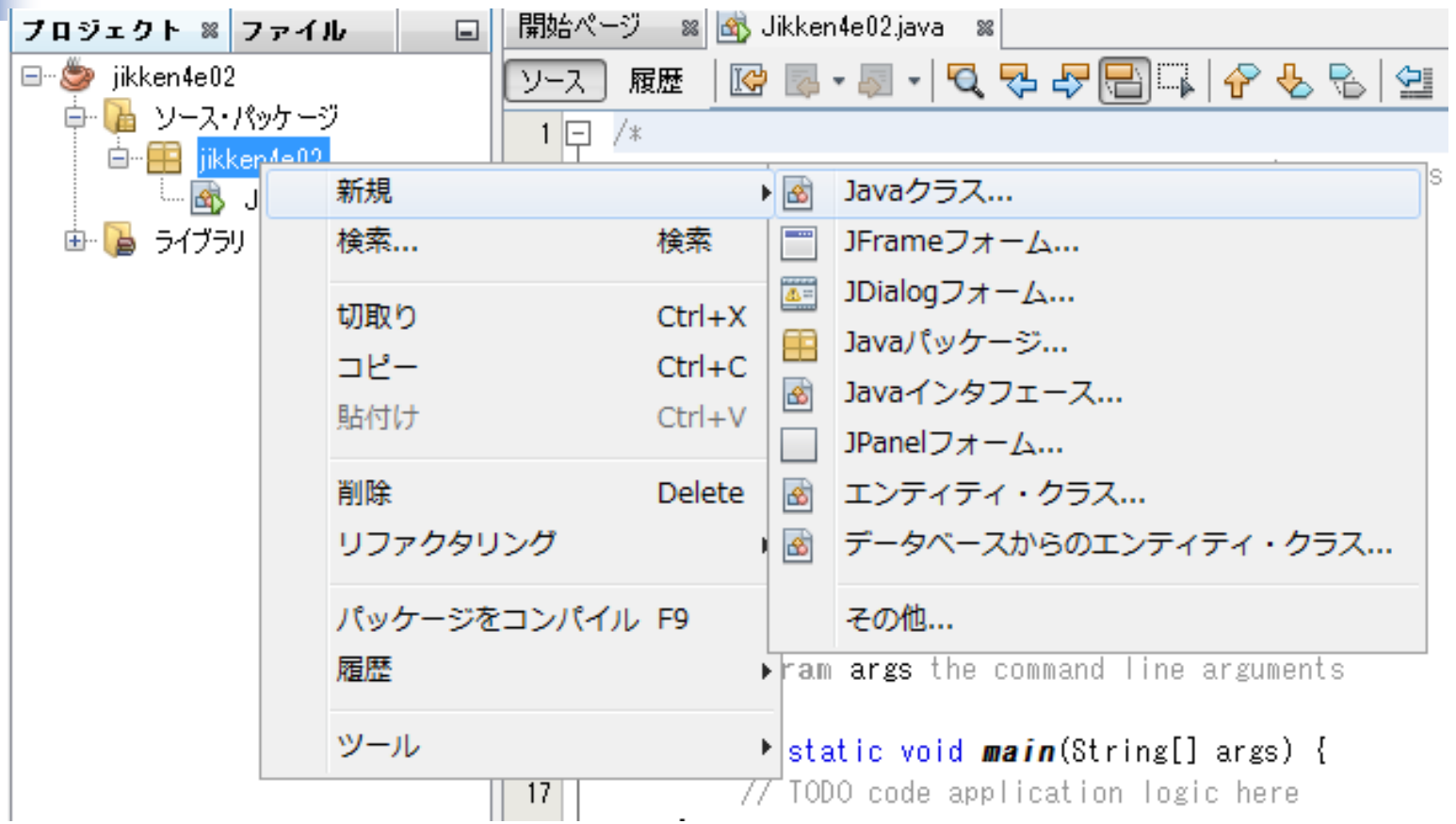
# クラス

- 変数と機能(メソッド)をまとめて管理する概念
- クラスに属するメソッド、変数をメンバという
- 変数へのアクセスはメソッドを通して行う
  - 例:1～5までの数しか入らない変数に数を代入
    → 代入する値が範囲内かどうかチェックしてから代入するメソッドを作れば実現できる
- 変数、メソッドにはアクセスできる範囲がある
  - 変数やメソッドの「スコープ」とも呼ぶ
  - アクセス修飾子により決定される

公開範囲

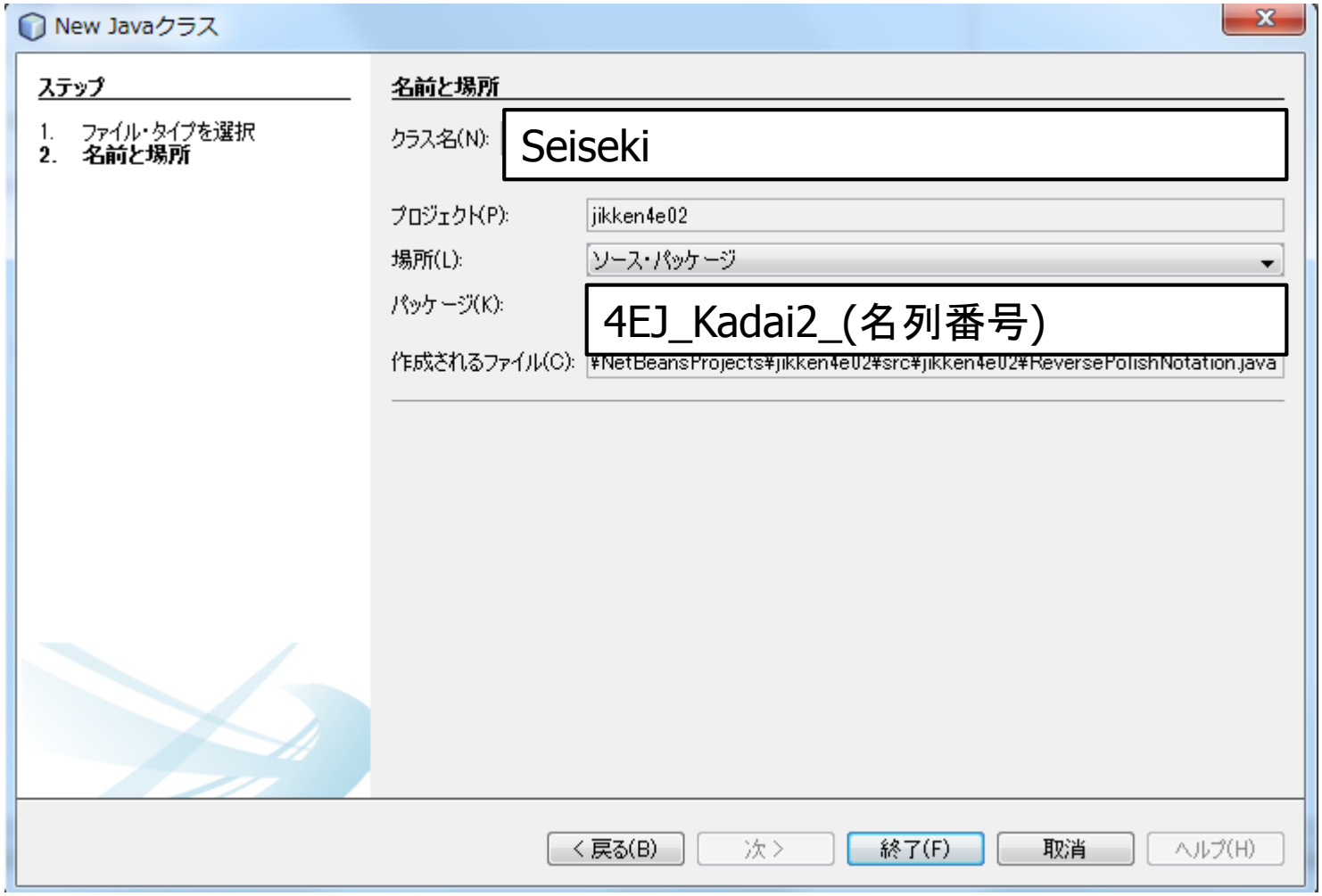
public > デフォルト(何も付けない) > protected > private

# クラスを作ってみよう




- パッケージの名前を右クリック
  - 新規→Javaクラスを選択

# クラスの名前を入力




- 今回は「Seiseki」としました

# 作られたクラスのひな型

どこからでもアクセスできることを示している

クラスを定義していることを示している

```
public class Seiseki {

}
```

この中をこれから作っていきます

- 中身が何もないクラスができました

# クラスに要素(メンバ)を持たせよう

- 今回持たせる要素
  - 6つの変数
    - 名列番号とスコア(int)
    - 氏名(String)
  - 3つのメソッド
    - 名列番号と名前を設定する機能
    - スコアを設定する機能
    - これらを文字列として取り出す機能

⇒クラス内のみで扱う(private)

⇒ クラスの外から使う(public)

# クラスに要素を追加(名列番号)

```
public class Seiseki {
    private int number; ←― 変数

    public void setNumber(int n){
        number = n;
    }

    public int getNumber(){
        return number;
    }
}
```

- メソッドを定義する場合

| アクセス修飾子　戻り値の型　メソッド名(引数) |
|---|

# クラスに要素を追加(氏名)

```
public class Seiseki {
    private int number;
    private String name;
```
←変数を追加

```
    (numberの分は省略)

    public void setName(String str){
        name = str;
    }

    public String getName(){
        return name;
    }
}
```

# メソッドが多すぎない？

- 全ての変数にgetとsetをつけたらあまりprivateで定義する意味はありません
- 次の手立てを考えるべきです
  - まとめて複数の値をセットする
  - 入力するタイミングを決めて入力する（変なタイミングで更新できない方がより良い）
  - 個別に取り出さず、クラスに出力機能を設ける
  - 他の方法で取り出せるようにする（配列等）

# 要素をまとめて代入／出力

```
public class Seiseki {
    private int number;
    private String name;

    public void setID(int num,String str){
        name = str;
        number = num;
    }
    public void getString(){
        return number+" "+name;
    }
    public void printID(){
        System.out.println(number+name);
    }
}
```

# 得点についても適宜実装

- 得点についても同様に実装します
- まとめて4教科分データを渡せると良いです

- 次のページ以降はこのクラスの使い方です

# さっそくこのクラスを使ってみよう

- メインのプログラム（最初に作られたファイル）に記入していきます

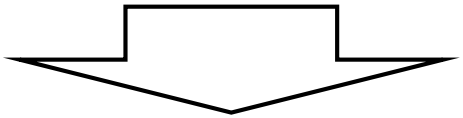
```
public static void main(String[] args) {
    Seiseki student1 = new Seiseki();

    student1.setID(1,"Michael");

    System.out.println(student1.getString());
}
```

自分で作ったクラスを初期化して利用している

何と表示されましたか？

# クラスの初期化とコンストラクタ

- クラスの初期化の際にあらかじめデータを渡しておきたい
- メンバ変数の初期化をあらかじめ行っておきたい



- コンストラクタを活用
  - 初期化の際の処理をまとめて記入できる
  - 初期化時に引数を渡すこともできる

# コンストラクタの追加

```
public class Seiseki {
   private int number;
   private String name;

   public Seiseki(int num,String str){
       name = str;
          number = num;
   }
   （以前に作った部分は省略）
}
```

- コンストラクタの特徴
  - 戻り値を書かない
  - メソッドの名前はクラスと一緒

# コンストラクタを使ってみよう

- メインのプログラムに記入

```
public static void main(String[] args) {
   （先ほど作ったプログラムがあってもよい）

    Seiseki student2 =
                new Seiseki (2,"Emma");

    System.out.println(student2.getString());
}
```

# printlnは何を表示するの？

- クラスが標準で持っているtoStringメソッドの戻り値を表示する
  - クラスであれば何かしらの文字列が帰ってくる
  - この名前のメソッドを実装し、表示内容を定める
- クラスにtoStringメソッドを追加（上書き）

```
public class Seiseki {
（既に書いた部分はそのままにしておきます）
 @Override                          既にあるメソッドを上書くことを明示
 public String toString(){
    return "IDは"+number;
 }                                  文字列と数字は＋で結合できる
}
```

# 値を入れていこう

- ArrayList(FIFO)にいれる

```
ArrayList<Seiseki> list = new ArrayList<>();
list.add(student1);
```

- Stack(FILO,LIFO)にいれる

```
Stack<Seiseki> stack = new ArrayList<>();
stack.push(student1);
```

- HashMapに入れる

```
HashMap<String,Seiseki> map=new HashMap<>();
map.put(student1.getName(),student1);
```

# 値を取り出す機能

- ArrayList

```
Seiseki data = list.get(0);
とするか拡張forループで
for(Seiseki sdata:list){ … }
```

- Stack(FILO,LIFO)

```
Seiseki data = stack.pop();
```

- HashMap

```
Seiseki data = map.get("Michael");
```

  - 名前(key)が分からなかったら･･･

```
ArrayList<String> keys = new ArrayList(map.keySet());
```

# 基本課題に挑戦

- ここまでで基本課題の説明は終わりです
- 点数を追加すること、リストやスタックを使うことに挑戦すれば課題は比較的簡単に実現できると思います
- 基本課題ができた人から発展課題に挑戦してください

# 発展課題のためのメモ

# プログラムのヒント1

- 文字列と文字
  - Javaでは文字列と文字は別のクラス（型）です
- 文字列（String）クラス
  - ダブルクォーテションで囲むことで表現
  - 部分文字列の取り出しや、一致の判定はメソッドを使う
  - 数値に変換するのも別クラスのメソッドを使う
- 文字（char）型
  - シングルクォテーションで囲むことで表現
  - 一文字しか表せず、==等で一致を判断

# 文字列(String)の取り扱い

- 初期化

```
String str1 = "123Hello";
```

- 切り出し(一部を取り出す)

```
String str2 = str1.subString(0,2);
```

- 文字列の判定(一致の判定)

```
if(str1.equals("Bye")){ ... }
```

- 文字列から数字(int)に変換
  - Integerクラスを使います

```
int value = Integer.parseInt(str2);
```

# 文字(char)型の取り扱い

- 文字の定義

```
char ch1 = '1';
```

  - 文字列から文字を取り出す場合

```
char c = str1.charAt(0);
```

- 文字の判定(一致の判定)

```
if(ch1 == '3'){ ··· }
```

- 文字(char)を数値にする

```
int value1 = ch1 - '0';
```

# 以上で説明は終わりです

- 課題に取り組んでください
- 完成したプログラムに適当な数式を入れて結果をレポートとしてアップしてください
- ソースコード（プロジェクト）も、アップロードしてください