プログラミングIII

黒瀬　浩

kurose@neptune.kanazawa-it.ac.jp

OH: 講義の前後，月4限21・405

Java基本

Javaのソースファイル名は　クラス名.java　　注
クラス名は，大文字で始める（のでファイル名も同じ）

コンパイル　　javac クラス名.java↵(クラス名.classができる)
実行　　java クラス名↵

```
public class Ex1 {
    public static void main(String[] args){
        System.out.println("Hello");
    }
}
```

クラスEx01

メソッドmain

Javaではクラス内にメソッド(操作，機能，振る舞い）を格納できる
publicはclassを修飾
puiblic, staticはmainを修飾
mainの型(戻り値の型)はvoid, 引数はStringクラスの配列，受ける変数はargs

注　今後，1ファイルに複数のクラスがある場合が出てくる

メソッドの書き方

```
public static メソッド名( 引数の型・名前の列　) {
  メソッドの内容
}
```

public static はメソッド名を修飾している。他の指定もある.
01-01のサンプルでは　実は　main は関数名ではなくメソッド名だった
mainメソッドはそのクラスが実行されたら最初に実行する特別なメソッド

以下はイメージの説明用（このままでは動かない）

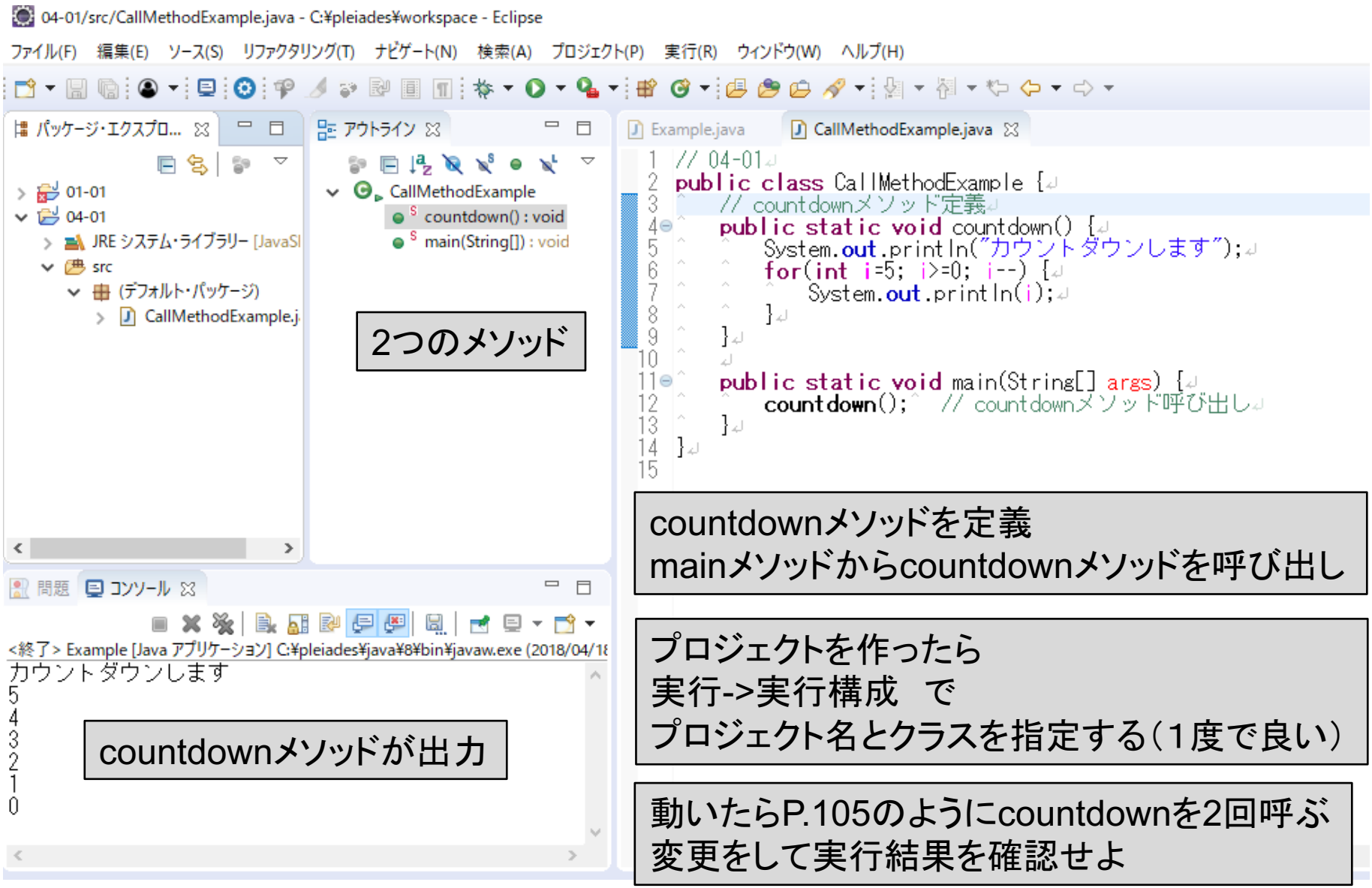
C風(関数の羅列)

```
int main(){
   int x = sub1();
}
int sub1(){
    return(1);
}
```

Java風(クラス内にメソッドを書く)

```
public class Xxx {
    public static void main() {
        int x =sub1();
    }
    public static int sub1() {
        return(1);
    }
}
```

list4-1 メソッドの呼び出し(Eclipseでの実行例)



list4-1 メソッドの呼び出し(VSCodeでの実行例)

機能拡張File Tree View
を入れ(要再起動) すれば
Eclipseで出ていたコード
概要を表示することができる
クリックすればそのメソッド
へ移動する
表示されていなければ
← の <>をクリック

引数(argument)を渡す

list4-4抜粋

メソッド定義

```
public static void countdown( int start ){ メソッドの実体  }
```

メソッドcountdownの引数は　1つ、int型、受け取る変数はstart

呼び出し

```
countdown(3);// int型　3 をメソッドへ渡す
countdown(10);       // int型　10をメソッドへ渡す
```

引数の数、それぞれの型は同じ(違っても良いことは後述)

countdownメソッドは　void型　なので　戻り値はない

特別なメソッドmain() プログラム開始点

main(String[] args) はString型の配列を変数argsで受け取る

java ファイル名　1 2 3 と実行すれば

args = {'1', '2', '3'}

すなわち、args[0]='1', args[1]='2', args[3]='3' となる

コマンド引数は無いかもしれない、複数かもしれない

配列の要素数を調べるには　配列名.length

コマンド引数は文字列となっているので数値にしたければ

型変換が必要

キャストは基本型でしか使えないので

```
int a1 = Integer.parseInt( args[0] );
```

Integerの初めが大文字なので、クラスっぽい

Integerクラス内のparseIntメソッドは　文字列を整数化する

キーボード(標準入力)からの入力 (standard input)

pythonでは input関数で入力を取得する
Cでは getc() やscanf() で入力を取得する

JavaではScannerが用意されている

```
Scanner in = new Scanner( System.in );
int i = in.nextInt();
```

Scannerはnewしないと使えない(配列もそうだった)
newはクラスっぽいものから何か作るようだ

2行目では
作られた in のメソッドnextIntを引数なしで呼ぶと変数iに値が帰るようだ

入力されたものを
整数化する　　nextInt()
倍精度実数化する　　nextDouble()
文字列化する　　next()

戻り値(return value)

メソッドが値を戻したければ、メソッド宣言で指定する

```
public static 戻り値の型 メソッド名( 引数リスト ){ 実体 }
```

戻り値は　return 文で返す(Cと同様)

```
        return 10;              // int型メソッドの場合
```

呼び出し

```
        int a = メソッド名(引数列);
```

メソッドの型と戻り値の型は一致させる
受ける変数の型が一致しなければ代入時に記憶域の小さい方に合わせられる

型変換が必要な場合はキャストや型変換メソッドを使う

オーバーロード(overload)

実はメソッドの引数列は１種類でなくても良い

呼び出し側

```
メソッド()      // 引数なし
メソッド(1)     // 引数1つ、整数型
メソッド('a') // 引数1つ、文字型
メソッド(1,2) // 引数2つ、整数型、整数型
```

メソッド定義側

```
public class クラス名 {
  public static void メソッド(){ 定義1 }
  public static void メソッド(int a){ 定義2 }
  public static void メソッド(String s){ 定義3 }
  public static void メソッド(int a, int b){ 定義4 }
}
```

メソッドの引数リストの並びに応じて同じ名前のメソッドを定義できる

似ていること

+演算子は　　数字同士だと加算、文字同士だと連結

オーバーロードできない場合

オーバーロードは引数リスト(引数の数と並びの型）で区別するので
区別できない場合は利用できない

型、引数並びが同じ

```
public static void  メソッド(int i){ 内容 }
public static void  メソッド(int j){ 内容 }
```

戻り値の型が違う

```
public static void  メソッド(int i){ 内容 }
public static int   メソッド(int i){ 内容 }
```

オブジェクト指向(Object Oriented)

C言語など(手続きを重視)

どの変数に何を入れて、どうするかを記述

オブジェクト指向

オブジェクト(物体、もの)を中心に考える

クラス：ものの雛形、設計図，似た性質のものをまとめたもの

クラスからオブジェクトを作る（実体化：インスタンス化）

学生は学籍番号、氏名を持つ。学生のクラスを考える

```
//  学生クラスから実体生成   注
        a  =  学生(12,  "金沢大郎");        //  学生クラスから生成された変数a
        b  =  学生(13,  "野々市花子");      //  学生クラスから生成された変数b
```

注 この記法は正しくない。説明用として使用している。

クラス定義（宣言）

記法　修飾子　class クラス名 {
　　定義
　}

例

```
public class Ex1 {
　　public static void main(String[] args){
　　　　System.out.println("Hello");
　　}
}
```

public: 修飾子
class名: Ex1

Ex1クラス内にmainメソッドを定義

publicはクラス外部に公開することを示す

学生を表す簡単なクラス

学生は学籍番号と氏名を持つ（こととする）

クラス内にメソッドがある例は出てきたが，ここではフィールドのみの例で説明する

```
class StudentCard {
        int id;                 // 学籍番号
        String name;            // 氏名
}
```

クラスには、メソッド以外にも値を含められる
値をフィールド、属性(attribute)，プロパティなどと呼ぶ

実体化

```
        StudentCard a = new StudentCard()
//      クラス・型   変数名   = new クラス名(引数)
```

変数aはStudentCard型、newしている

今までにnewしたもの

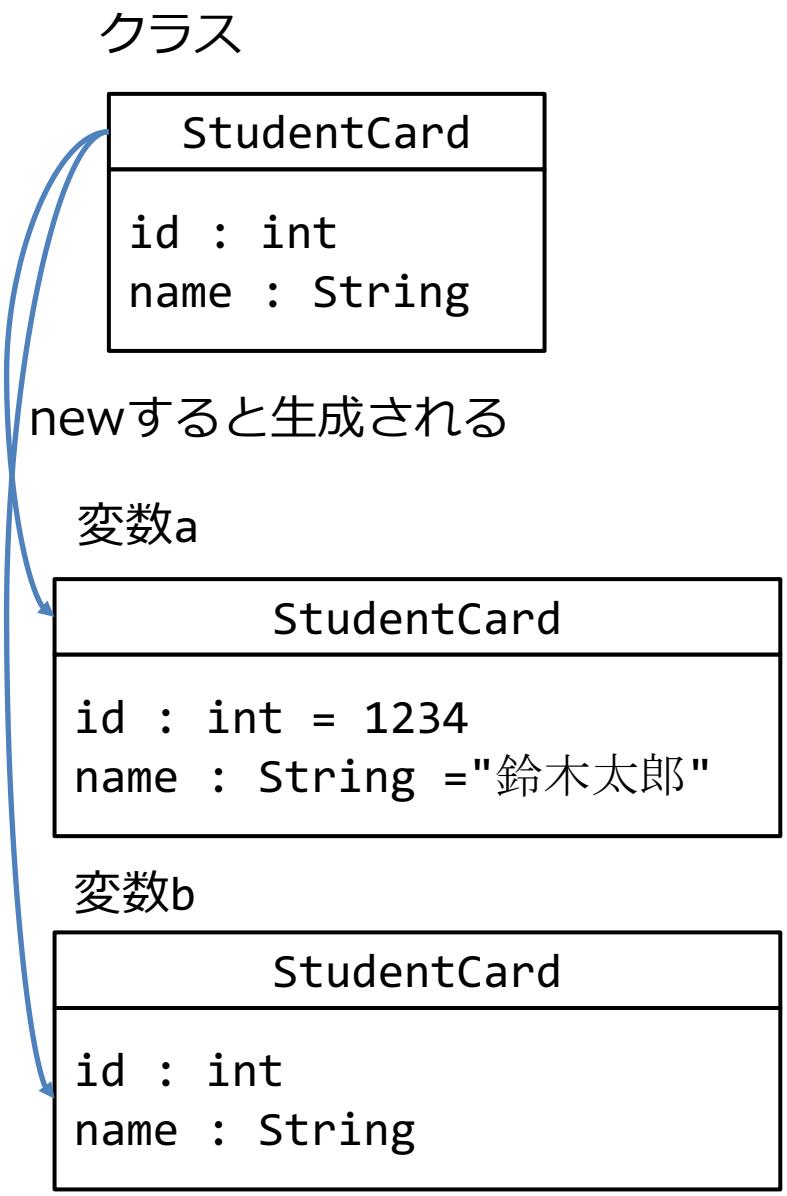
```
        Scanner in = new Scanner( System.in );
        int i = in.nextInt();                   // 整数
```

インスタンス変数

```
class StudentCard {
        int id;         // 学籍番号
        String name; // 氏名
}
// 実体化
StudentCard a = new StudentCard()
StudentCard b = new StudentCard()
a.id = 1234;
a.name = "鈴木太郎";
```

同じクラスから変数a,bを実体化（インスタンス化）した。a,bはインスタンス変数

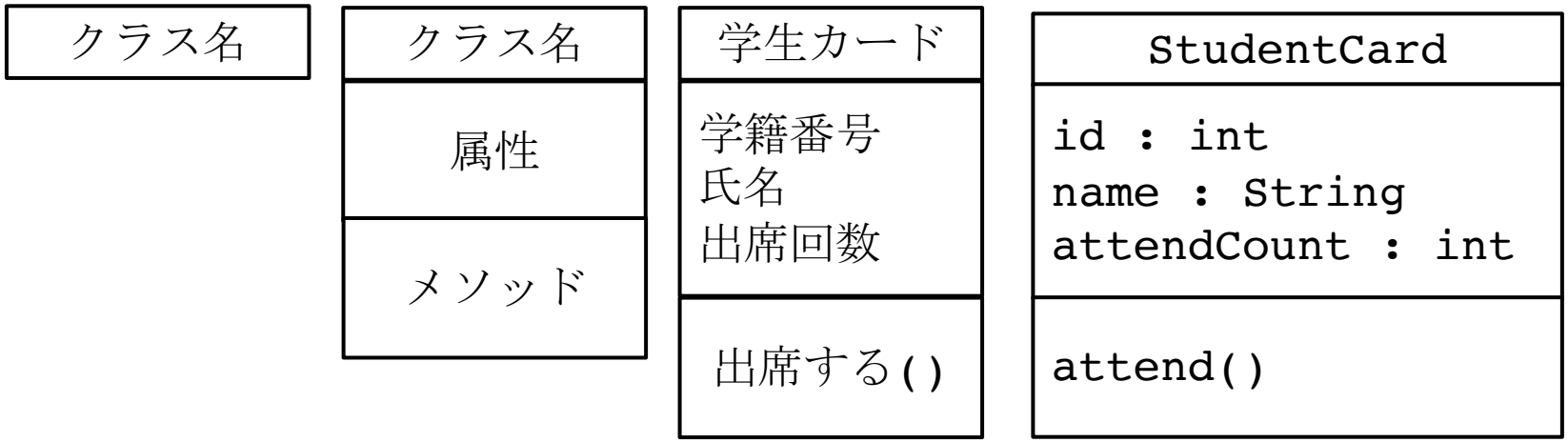
インスタンス変数名.フィールド変数名
で参照、代入可能
bは代入していないので値は不定(null)



オブジェクト指向のクラス

System.out.println("Hello");
Systemオブジェクト内のoutオブジェクト内のprintlnメソッドに"Hello"を引数として渡し呼び出す

クラス図(詳細はソフトウェア工学1で学習する)



左はクラス名のみ：　設計初期（クラス分割作業）
2番目はクラス図の一般形：　クラス名、属性(フィールド),メソッドの箱を書く
3番目は設計中にクラスの内容を固めていく段階
4番目は実装(プログラミング)に向けた記述

オブジェクト指向ではソフトウェアで実現する世界をクラスで表現する

インスタンスをたくさん作る

授業では、受講者がたくさんいる。
各受講者は学籍番号と氏名を持つ。

先のStudentCardクラスは一度定義しておけば、科目が変わっても使える
履修者が年や科目によって変わっても対応できるプログラムを考える

・作成するインスタンスの数を変える
・インスタンスは配列にする

```
StudentCard[] cards = new StudentCard[3];
```

左辺　インスタンス変数 cardsはStudentCardクラスの配列
右辺　要素数3でStudentCardの器を作る

```
cards[0] = new StudentCard(); // 実体の作成
cards[0].id = 5;                    // cardsの0個目の変数idに代入
```

値を設定しないとnull となっている

参照

Javaにはポインタはないが意識は必要

```
int a = 10;           // 変数aに整数10を代入
String s = "Hello"; //変数sは文字列"hello"の位置(アドレス)を指す
```

基本型(char, boolean, byte, short, int, long, float, double)以外は位置を指すポインタで内部的には参照する

参照型の型名は大文字で始める(推奨)　　例 String

参照型変数の値を設定しないと参照先なし(null)となり，たどれない

アクセスしようとすると　null pointer exception エラーとなる

クラス名：　大文字で始めるということは、基本型ではないということ

演習

list5-1, 5-2, 5-3, 5-4, 5-5, 5-6を確認せよ

ファイル名(クラス名)は各自が管理すること

5-1を作ったら、コピーして5-2を作成・確認する

手入力すること、エラーが出たらどこが悪いか考えてから直す

できた人は、5-4章を読んで理解した上で list5-6を動かせ
さらに、自分の好きに拡張せよ

5章の章末問題は次回までに実施すること

確認問題

1) クラス内に入れられるものは大きく分けて２種類ある。それは何か

2) クラスとインスタンスの違いは何か

3) インスタンスはプログラム実行開始から存在しているか

4) 1つのファイルに複数のクラスを定義できるか

5) 4)が可能なら，Java実行時，最初に動くメソッドはどのように見つけるか

6) 同じクラスから別々の属性を持つものをつくるにはどのようにするか

7) 自分とは別のクラスの属性を変更できるか

8) 7)が可能ならどのように指定するか

9) 教科書4章まででインスタンスを生成している例を挙げよ

10) インスタンスは日本語でなんと言うか

11) メソッドと同等な語句を挙げよ（複数ある）

12)　フィールドと同等な語句を挙げよ（複数ある）