

# プログラミングⅢ

黒瀬 浩

[kurose@neptune.kanazawa-it.ac.jp](mailto:kurose@neptune.kanazawa-it.ac.jp)

OH: 講義の前後, 火4限21・405

居室 67・121

# 開発方法

- 1) CUI(テキストエディタと端末アプリ) での開発
- 2) 中程度機能エディタ(VSCode, Atomなど) での開発
- 3) IDE(IntelliJ Idea, Eclipse/Pleiades)での開発

教科書の演習程度であれば、2)を使うことが多いが

1)でもできるようにすること

エディタが動作しないとき

コマンド引数やキーボード入力の指定が楽

3)はプロジェクトやCLASSPASH,コンパイラの混在などの知識が必要となる

大規模開発, Androidアプリや複数のパッケージを使うときに使わないと設定を多くしなければならない場合がある

# Javaプログラミング開発手順（復習）

## JDKインストール

Javaソースをコンパイルするために必要  
JRE(Java実行環境)はJDKに含まれている

## 検索パスの設定

Windowsでは、ユーザ環境変数の編集(またはシステム環境の編集) で  
Pathにjavacのある場所を追加する(Linuxでは.bashrcなどに追加)

`echo %PATH%` で実行プログラムを探す順番を確認できる  
`where javac.exe` で実際に見つけた場所を返す

検索パスに複数のjavaコンパイラが設定されている場合、最初に見つけた  
ものが実行される

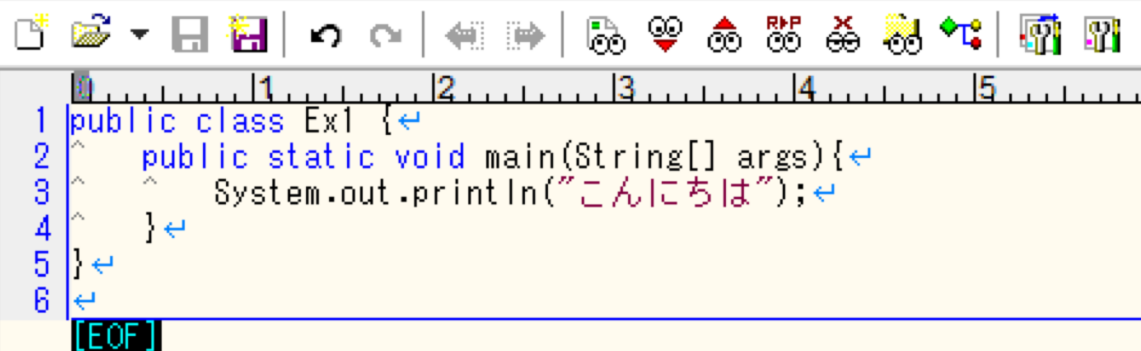
Linuxではファイル.bashrcや.profileに `PATH=` で検索パスを設定するが  
パッケージマネージャでインストールすれば特に設定する必要はない

# CUIでの開発方法

- 1) ソースファイルの作成・変更（テキストエディタ）  
ファイル拡張子は `.java`  
クラス名とファイル名を大文字小文字まで合わせる必要がある
- 2) 端末アプリ（コマンドプロンプトなど）でファイルの場所に移動  
`cd` ソースファイルを作成したディレクトリへ  
端末アプリ起動後に一度行う
- 3) コンパイル  
`javac file.java`  
クラスファイル `file.class` が作成される  
中間コード(バイトコード)とも呼ばれる
- 4) 実行  
`java file` （OSによっては `java file.class`）
- 4) デバック 動作するまで1),3),4)を繰り返す  
IDEのrunボタンはファイルの保存、コンパイル、実行を行っている  
他に `jar` コマンドで実行する方法もあるが省略

# テキストエディタと端末アプリでの開発

ファイル(F) 編集(E) 変換(C) 検索(S) ツール(T) 設定(O) ウィンドウ(W) ヘルプ(H)



```
1 public class Ex1 {  
2     public static void main(String[] args){  
3         System.out.println(\"こんにちは\");  
4     }  
5 }  
6  
[EOF]
```

サクラエディタの例

文字コードをUTF-8にする場合は  
設定->文字コードセット指定で  
UTF-8(BOMは指定しない) を  
選んでから保存する

コマンドプロンプト

```
Active code page: 65001  
C:\Users\kurose>cd Prog  
C:\Users\kurose\Prog>type Ex1.java  
public class Ex1 {  
    public static void main(String[] args){  
        System.out.println(\"こんにちは\");  
    }  
}
```

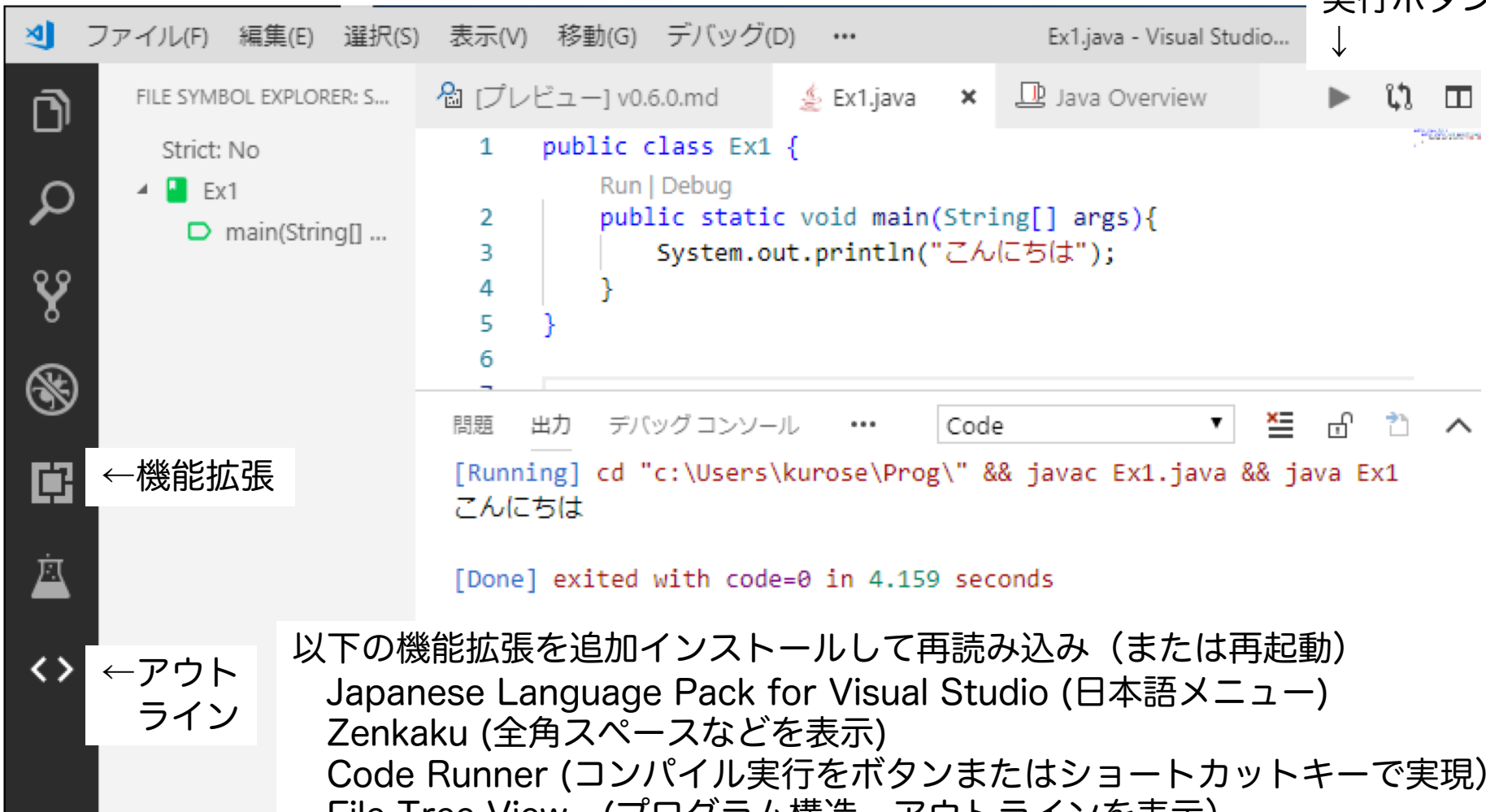
```
C:\Users\kurose\Prog>javac Ex1.java
```

```
C:\Users\kurose\Prog>java Ex1  
こんにちは
```

端末アプリ コマンドプロンプトの例  
文字コードをUTF-8にする場合は  
chcp 65001 を起動後1回実施  
ファイル保存場所にcdコマンドで移動  
  
エラーがでたらエディタで修正し保存  
してから再コンパイルする

# VSCodeでのプログラム作成例

実行ボタン  
↓



以下の機能拡張を追加インストールして再読み込み（または再起動）  
Japanese Language Pack for Visual Studio (日本語メニュー)  
Zenkaku (全角スペースなどを表示)  
Code Runner (コンパイル実行をボタンまたはショートカットキーで実現)  
File Tree View (プログラム構造, アウトラインを表示)  
Java Extension Pack (java関係)  
Java Language Support (java関係)  
など

# Java用エディタ BlueJ

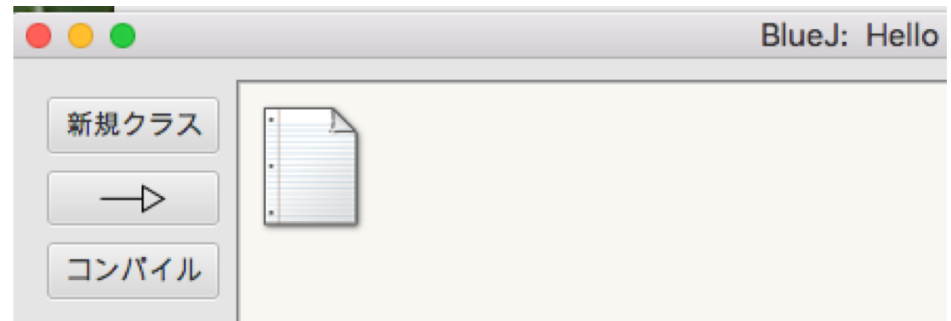
プロジェクト ⇒ 新規作成

作成するフォルダを選ぶと プロジェクト名 のフォルダが作成される

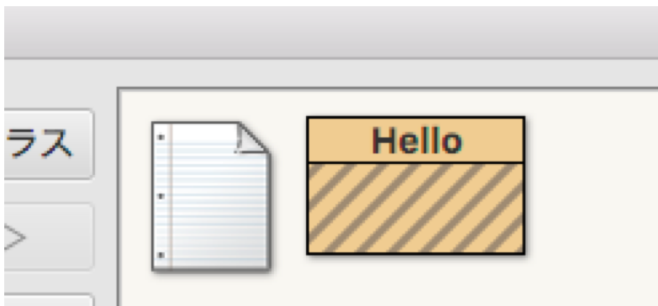
プロジェクト名 Hello

新規クラスをクリックし

クラス名 Hello を入力し OK



プロジェクトはアプリケーションごとに作成, IDEでは必須



# BlueJ エディタの表示 (Helloクラスを右クリックして)



ソースの記述  
(日本語入力の問題あるかもしれない)



コンパイル を押す

Helloクラスを右クリックし

void main() を選ぶと 実行される





# IntelliJ IDEA Community (<https://www.jetbrains.com/idea/>)

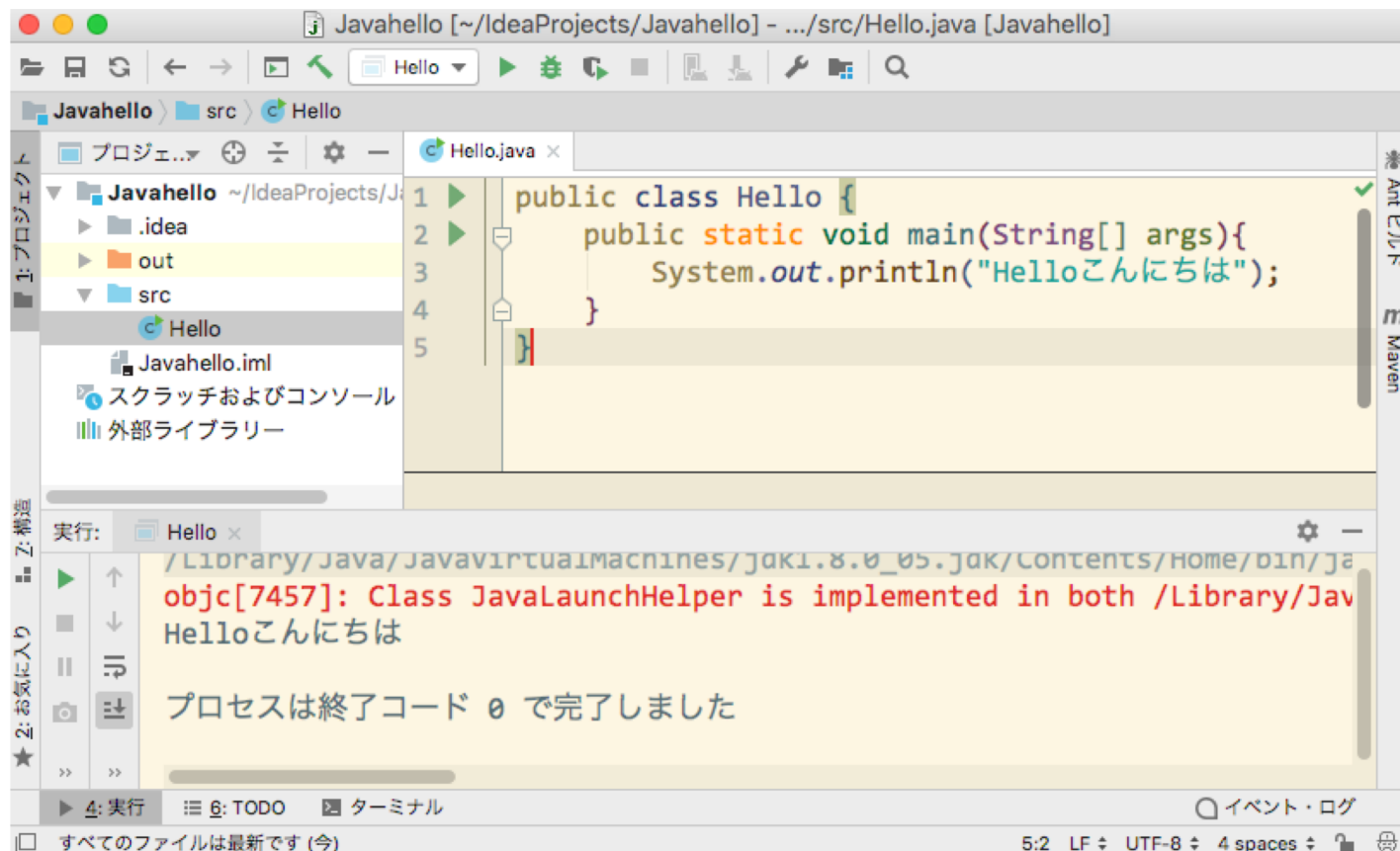
JetBrains社の開発しているIDE

各種言語の機能拡張が用意されている

Ultimate版は有償なのでCommunity版を使う

先にJDKを入れておく

日本語メニューはPleiadesプラグインで表示できる

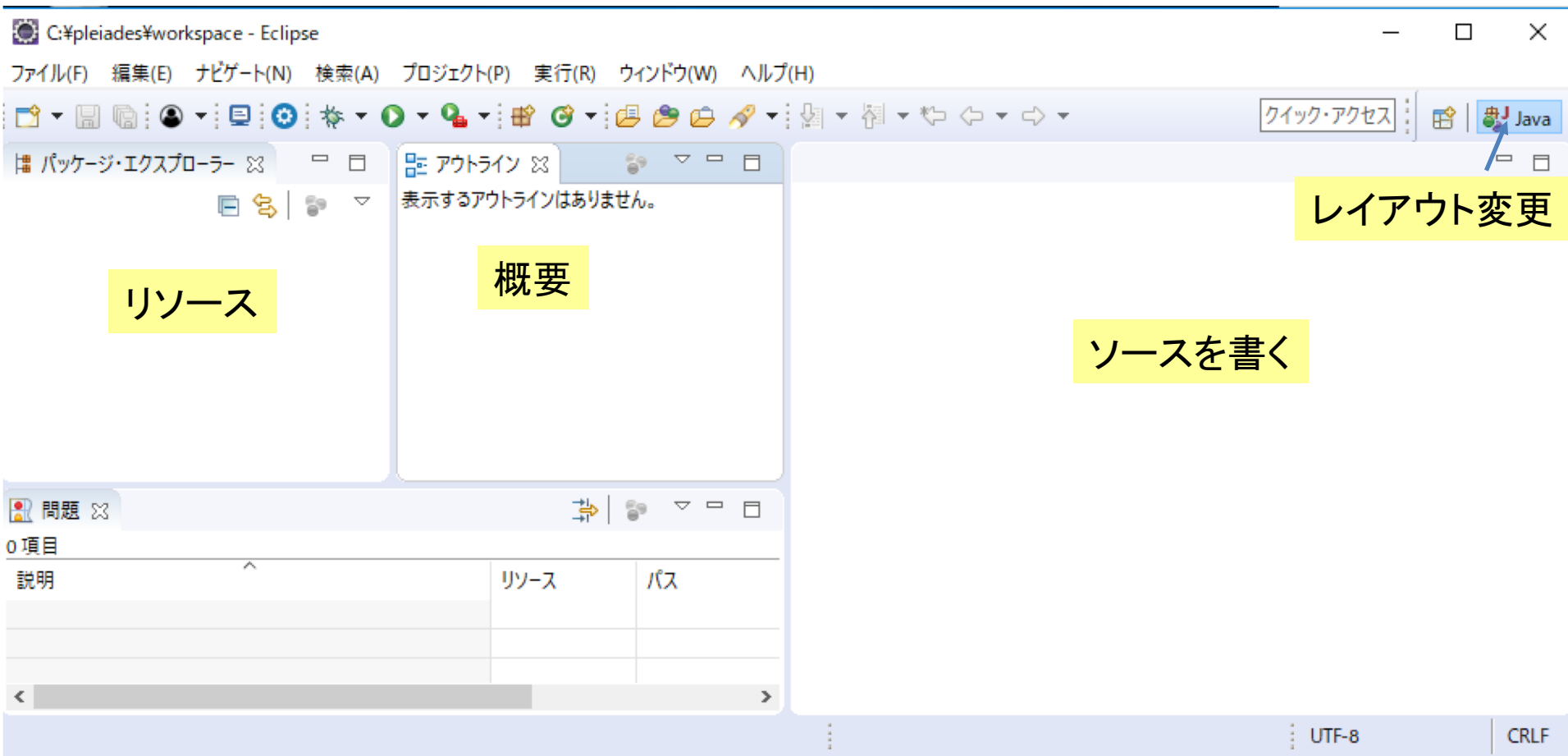


# IntelliJ IDEA Community Edition

- ダウンロードしてインストール
- Pleiades 日本語化プラグイン で検索してダウンロード
  - zipファイルを「すべて展開」して、中のsetup.exeを実行
  - 日本語化するアプリを聞かれるので、IntelliJ IDEAの場所を指定  
C:¥Program Files¥JetBrains¥IntelliJ IDEA....¥
- IntelliJ IDEA起動後の設定
  - 設定で、 proxy を検索し、自動プロキシ構成で  
<http://www.kanazawa-it.ac.jp/proxy.pac> を指定
  - あとは、キーマップ変更、プラグイン追加、カラーテーマ変更等お好きに
- IntelliJ IDEAでのプログラム作成
  - ファイル -> 新規 -> プロジェクト -> Javaモジュール で  
プロジェクト名を入れる
  - できたプロジェクトの src を右クリックし 新規->Javaクラス で  
ファイル名(=クラス名)を入れる  
エディタが開くのでソースを入れて保存  
初回の実行だけ、実行構成が必要（ソース左の実行ボタンを押す）  
一度実行構成を設定すれば、実行ボタンかSHIFT-F10でコンパイル・実行ができる

# eclipseの画面

- 起動時 workspaceの問い合わせがでたらそのまま 起動
- ようこそ画面がでたら 画面のXボタンで 閉じる



# eclipseでの開発方法

## 1) プロジェクト作成

ファイル -> 新規 -> Javaプロジェクト      プロジェクト名を入力

## 2) クラス作成

ファイル -> 新規 -> クラス      クラス名を入力

## 3) ソース作成

右側ペーンにソースコードを入力

## 4) 実行

再生ボタン(または 実行ボタン)

初回起動時は「実行構成画面」が出るので

Javaアプリケーションをダブルクリック

プロジェクト名とメイン・クラスが入っていることを確認して「実行」

「保存して起動」画面がでたら「OK」

コンソールペーンに出力が出る

## 5) 再実行

再生ボタン (または 実行 -> 前回の起動を実行)

# Eclipseの注意

Eclipseはpackage(ソフトウェアを提供する範囲の単位) を前提としているため最初の行に package Sample; 等パッケージ名を入れる

大規模開発などでソースファイルが複数のディレクトリに分かれていても関連したクラスと認識させるために使う

IDEではアプリケーションごとにプロジェクトを作成するのが普通

プロジェクトごとにディレクトリが分かれるので、共通で使うファイルはコピーする (CLASSPATHを設定すればそこも探せるようになるが)

教科書の例のように少し変更して別のファイルを作成していく場合はEclipseなどのIDEでは少し面倒

Pleiadesを入れないと日本語メニューが出ない(IntelliJ Ideaも同じ)

# 日本語表示が文字化けする場合

ソースファイルを utf-8 で作成していることを前提とする

コマンドプロンプトの場合

chcp 65001↵ の後に実行してみる

コマンドプロンプト起動ごとに上記コマンドを打つ  
(起動時に実行する方法もある)

VSCodeの場合(実行時に「出力」に出している場合)

設定の検索で terminal.integrated.sh で検索

Terminal > Integrated > Shell > Windowsを

C:¥Windows¥System32¥cmd.exe にする

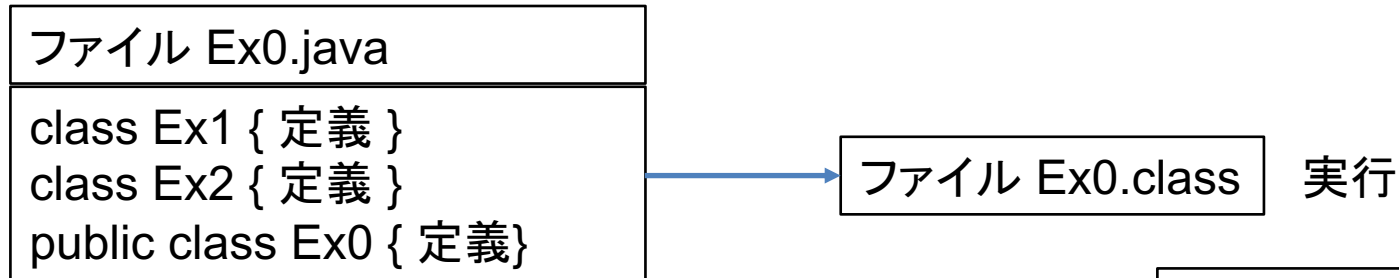
その下の方の Terminal > Integrated > Shell Args: Windowsで

setting.jsonで編集を押す 左画面の鉛筆アイコンを押す

右画面の [ ]の中に "/k", "chcp 65001" を追加して, 再起動

# Javaのファイルの構成方法

例1 1つのファイルに複数クラスを定義



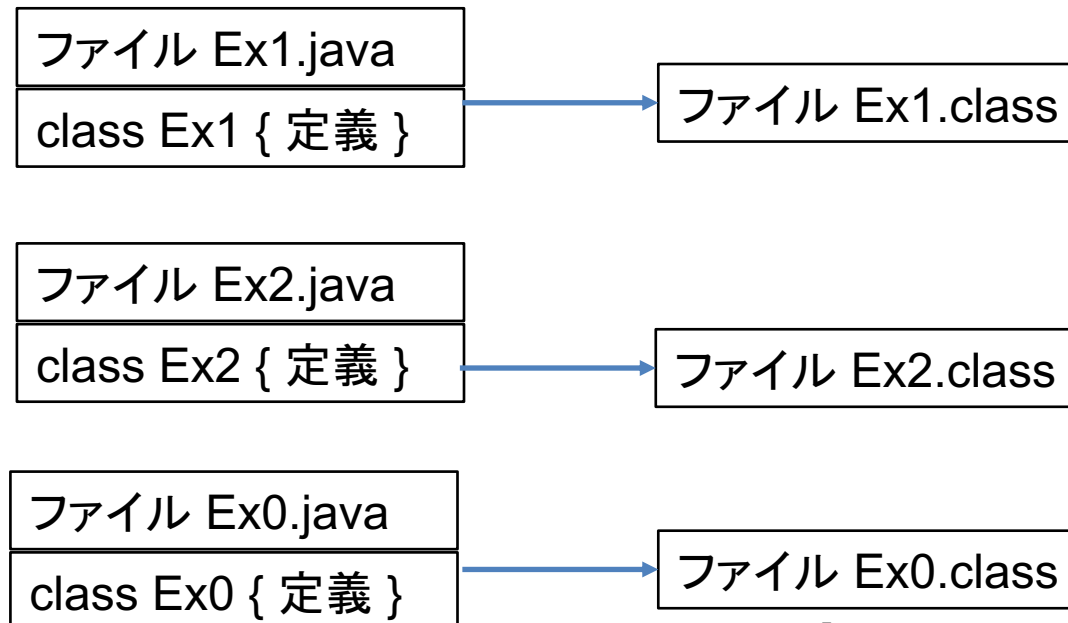
同じディレクトリにある  
クラスファイルは呼び出せる

既に作ったものを再利用

名前をいい加減につけると  
おかしい動作をすることが  
ある。どのclassは何を定義  
しているか知っておく必要が  
ある

CLASSPATHを設定すれば  
複数のディレクトリのクラス  
ファイルを扱える

実はこの方法は一般的ではない。以下が一般的



実行

# 前ページの問題点

ファイル ExA.java

```
class Ex1 { 定義 }  
class Ex2 { 定義 }  
public class ExA { 定義 }
```

ファイル ExB.java

```
class Ex1 { 定義 }  
class Ex2 { 定義 }  
public class ExB { 定義 }
```

同じ機能を複数のファイルにもたせた場合

ファイルExAのEx1を修正してもファイルExBのEx1は元のまま

クラスごとに別ファイルにしておけば、1ファイルの修正で済む

ファイル名をいい加減につけてしまうとどのような問題が起こるか？

注 あとで出てくる継承の考え方から、関連のあるクラスは同じファイルにまとめる場合もある



# 前回

new演算:クラスからインスタンスを生成する

// クラス定義

```
class クラス名 {  
    定義  
}
```

// 実体化

クラス名 インスタンス変数 = new クラス名()

クラスは値(フィールド変数)とメソッドを含められる

参照

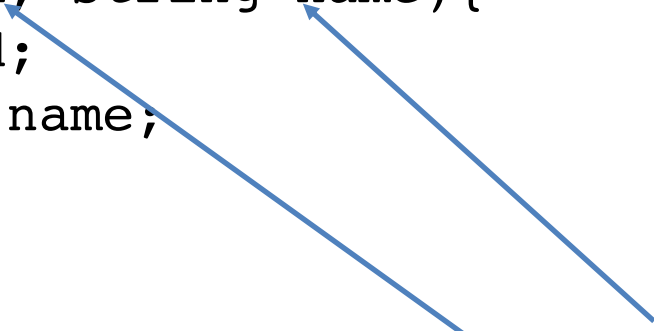
インスタンス変数.フィールド名 = 値

# コンストラクタ(constructor)

インスタンスが生成されたら実行するメソッド  
クラス名と同じ名前のメソッド  
初期化などに使う。生成時に引数で値を渡す。

List6-1改

```
class StudentCard {  
    int id;  
    String name;  
    // constructor  
    StudentCard(int id, String name){  
        this.id = id;  
        this.s.name = name;  
    }  
}  
// 生成  
StudentCard a = new StudentCard(12, "太郎");  
StudentCard b = new StudentCard(34, "花子");
```

A diagram consisting of two blue arrows. The first arrow originates from the integer value '12' in the first constructor call 'new StudentCard(12, "太郎")' and points to the parameter 'id' in the constructor definition 'StudentCard(int id, String name)'. The second arrow originates from the string value '"太郎"' in the same constructor call and points to the parameter 'String name' in the constructor definition.

クラス生成時の引数を受け取るのはコンストラクタ

# this

自分自身のオブジェクトを表す(言語によっては self だったりする)  
thisが必要な理由

List6-1改

```
class StudentCard {  
    int id;  
    String name;  
    StudentCard(int id, String name){  
        this.id = id;  
        this.name = name;  
    }  
}  
  
// 生成  
StudentCard a = new StudentCard(12, "太郎"); // (1)  
StudentCard b = new StudentCard(34, "花子"); // (2)
```

(1), (2)実行後はStudentCardクラス外では a.id, b.idが使える  
実行前はどのオブジェクト(インスタンス) か不明  
クラス内ではaかbか識別ができない

# コンストラクタのオーバーロード

StudentCardクラス内でコンストラクタを定義

// 戻り値の型は指定しない、return文も無い

```
StudentCard(int id, String name){  
    this.id = id;  
    this.name = name;  
}
```

```
}
```

// 生成

```
StudentCard a = new StudentCard(12, "太郎"); // (1)
```

```
StudentCard b = new StudentCard(34, "花子"); // (2)
```

生成時の引数の数、並び、型は一致している

```
StudentCard c = new StudentCard()
```

と呼び出したい場合は、引数の数、並び、型が一致するメソッドがない

引数なしで生成したければ、対応するメソッドを作成すれば良い

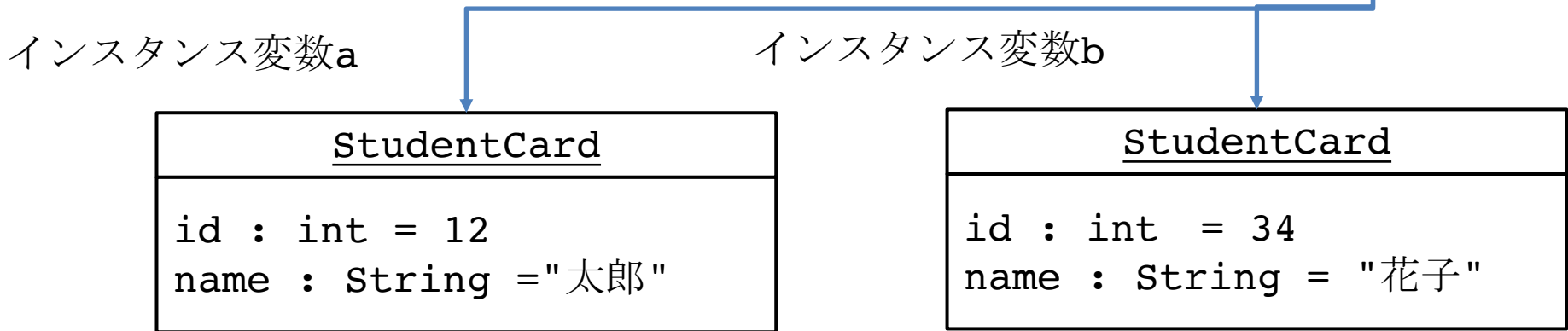
# 先週のクラス図 (をより正確にするが厳密には正確では無い)

クラス

```
class StudentCard {  
    StudentCard(int id, String name){  
        this.id = id;  
        this.name = name;  
    }  
}
```

// 生成

```
StudentCard a = new StudentCard(12, "太郎");  
StudentCard b = new StudentCard(34, "花子");
```



インスタンスはクラスと区別するために名前に下線を引く (オブジェクト図)

# インスタンスメソッド

クラスにはメソッドを含められる  
クラスから生成されたインスタンスもメソッドを保持する

```
戻り値の型  メソッド名( 引数列 ){  
    定義  
    return 戻り値;  
}
```

List-6-4抜粋

```
void printInfo(){  
    System.out.println("学籍番号:" + this.id);  
}
```

C言語の関数と変わらないが

クラス内で（フィールド）変数を参照するときにthis(自分自身)を使用

呼び出しは インスタンス変数.メソッド（引数列）  
a.printInfo();

# thisの省略

コンストラクタの引数と区別ができれば `this.` は不要

```
// this. が必要
StudentCard(int id, String name){
    this.id = id;
    this.name = name;
}
this.id = なにか
```

```
// this. がなくても可
StudentCard(int i, String n){
    id = i;
    name = n;
}
id = なにか
```

自分自身のインスタンスメソッドを呼ぶ

```
    this.メソッド名(引数列);
```

# クラス変数

インスタンス変数：      インスタンスごとに別物

クラス変数：              クラスで1つ

違いは    static をつけるかどうか

List6-5改

```
class StudentCard {  
    static int counter = 0; // クラス変数  
    int id;                // インスタンス変数  
    String name;           // インスタンス変数
```

略

クラス変数はインスタンスを作らなくても存在する

クラス変数を変更すると他のインスタンスにも影響する

クラス変数も他になければ `this.` を省略できる

クラス変数をコンストラクタで変更するとどうなるか？



# クラスメソッド

インスタンスを生成しなくても呼べるメソッド  
static を付けて定義するとクラスメソッドとなる

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

今まで使用していたmain()はクラスメソッドだった

System.out.println(); もnewしないで呼び出している

メソッドには インスタンスメソッドとクラスメソッドがある

# 演習

List6-1から6-8を実際に確認せよ

List6-9の内容を理解して確認せよ  
(List5-7をやってから)

6章の章末問題

List6-1はクラス定義のみなので実行できない

同一プロジェクトに複数ファイルを作成している場合は  
クラス定義が悪影響を及ぼす場合がある

その場合はクラス名を変える(関係する部分全部)

StudentCard ⇒ StudentCard1

- ・ 作成するプログラム
  - 1を入れると1: Janを、2なら2: Febのように月と英月名3字を出力するJava言語を使用すること
  - 英月名への変換はif～else if、switch case、配列参照いずれでも可
  - 最低限1月から6月までに対応すること(12ヶ月対応しても良い)
- ・ 入力値の与え方
  - コマンド引数で指定する か キーボードから値を読むの機能を使う
  - チャレンジ：引数やキー入力で月を複数指定可能、表示文字を配列に格納
- ・ エラーチェック
  - 入力値が範囲外ならエラーメッセージ出力を行う
- ・ 実行
  - 範囲内2つと範囲外少なくとも3回実行すること
- ・ 備考
  - EclipseなどIDEを使用している者は引数の与え方を調べること
  - 文字を整数に変換する必要がある場合は調べること
- ・ レポート
  - タイトル(プログラミングIIIレポート1),提出日(印刷日ではない) 、
  - クラス名列、氏名を上部に書く, A4 1枚(両面可能)
  - CUIの場合はソース全て表示されているエディタ画面と、
  - コンパイル・実行部分が表示されているコマンドプロンプトのハードコピー
  - CUIエディタ以外はソース全てと実行結果の画面のハードコピー
  - 字が小さくて読みにくい場合はA4用紙を横置にすること

# 次回小テスト実施

- 入門編5章
- 入門編6章
- 入門編1～4章やJava開発方法は前提知識