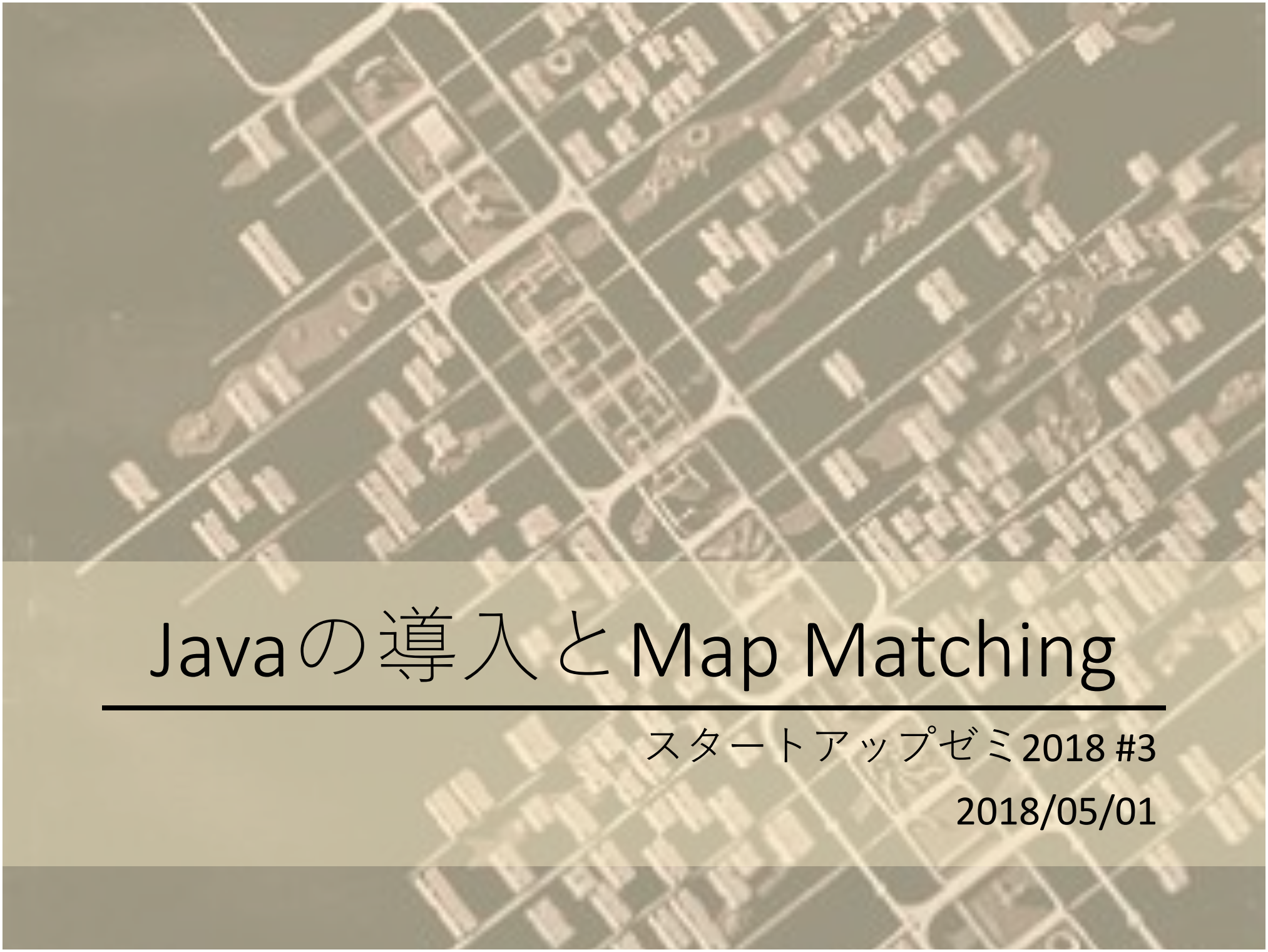


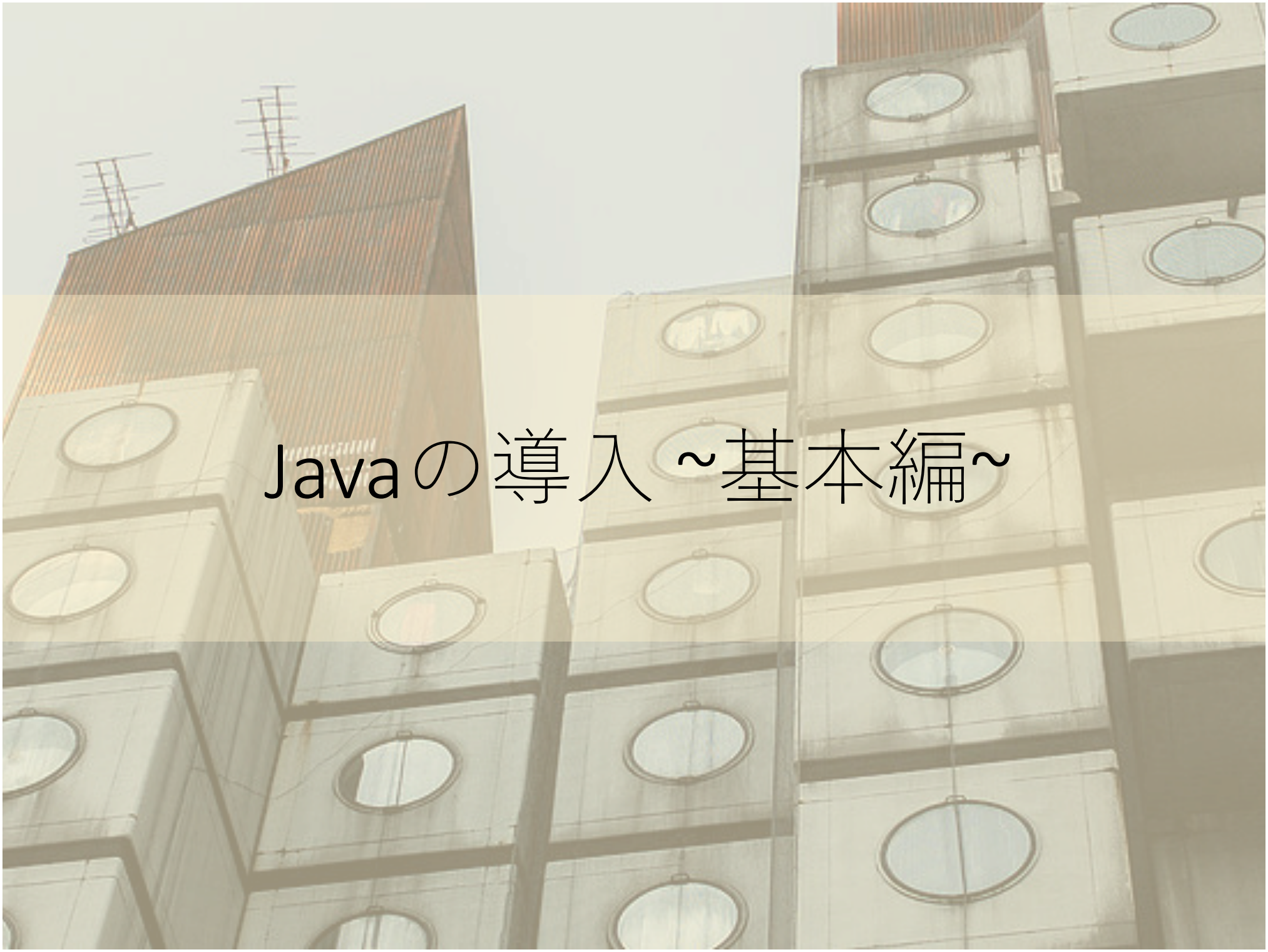
# Javaの導入とMap Matching

スタートアップゼミ2018 #3

2018/05/01

# 目次

- Javaの導入 ~基本編~
  1. JavaとEclipseのインストール
  2. Javaの基本
- Javaの導入 ~実践編~
  1. データの読み込みと書き出し
  2. データの格納
  3. クラスとメソッド
- Map Matching


Javaの導入 ~基本編~

# 1.JavaとEclipseのインストール

- JDK

  JDK(Java Development Kit)が
  Javaプログラム開発には必須．(Macは搭載済)

- Eclipse

  主に用いられるIDE(開発環境)

別紙配布資料を参照してください．

(Eclipseは特に注意しながら進めてください!!)

# 2.Javaの基本

Javaは，C系のプログラミング言語と似た言語形式を持っており，汎用性の高い言語です．

コンパイラ言語(高速処理!!)であり，データ形式を最初に固定する形式をとります．

スクリプト言語のPythonに比べると，覚えることがやや多く難しいですが，研究室の過去のコードはJavaで書いてあるものも多いので，少なくとも基本は押さえておきましょう！

# 2.Javaの基本

プログラムの作成手順

■[ファイル]-[新規]-[Javaプロジェクト]

-プロジェクト名(ex.Startup2018)を入力-[完了]

■[ファイル]-[新規]-[クラス]

-クラス名(Mainなど)を入力-[完了]

※public static void main(String[] args)にチェックを入れること

```
public class Main {
	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO 自動生成されたメソッド・スタブ

	}

}
```

# 2.Javaの基本

Javaプログラミングの超基礎[自習！]

1. 変数の宣言(型：String, int, long, double,…)
2. 各処理の最後にはセミコロン”;”
3. 算術演算子による計算
4. 制御構文(if~else, for, while)
5. 比較演算子

# 2.Javaの基本

例題(基本の確認)

1. コンソールに”Behavior in Networks!”と表示させる.
2. 1~120までの数字を表示する. (for文)
3. ”1,2,3,…,119,120”と1行で表示する. (for文)
4. 1 ~ 120のうち素数のみ表示する. (for文)
5. 以下の式を満たす最小のx,yを求める. (while文)

$$\left(\sum_{i=1}^{x} i\right) > 100 \quad , \quad (y!) > 1{,}000{,}000$$


Javaの導入 ~実践編~

# 1.データの読み込みと書き出し

## 読み込み

- BufferedReaderで，ファイルの入力
- readline()で1行ごとに読み込む
- 複数列の場合はsplit(“,”)を使う

## 書き出し

- PrintWriterで，ファイルの出力
- println()で1行ごとに書き込む

※入出力は，String型で行われる
※try-catchで例外処理が必要
※ BufferedReaderもPrintWriterも使った後はclose()で閉じる

# 1.データの読み込みと書き出し

String型では，計算処理はできないので．．

## 型の変換：valueOf

ex.1)数値→文字列

String.valueOf(数値)

ex.2)文字列→数値

Double.valueOf(文字列)

# 1.データの読み込みと書き出し

練習1

input1.csvの各行を読み込み，合計・平均・標準偏差を求め，output1.txtファイルを作成する．

**input1.csv**

```
75
60
80
45.5
37
55
90
16.2
75
19
```

**output1.txt**

```
合計:552.7
平均:55.27
標準偏差:25.58203
```

# 1.データの読み込みと書き出し

## 練習1の参考(合計のみ)

```
import java.io.*;                                   //java.ioパッケージを使う

public class Main {
  public static void main(String[] args) {
    try {                                           //データをうまく入出力できるとき
      String inputfile = “./input/input1.csv”;          //インプットファイル名
      BufferedReader br = new BufferedReader(new FileReader(inputfile));
      String line = null;                                   //1行ごとに読み込む変数を用意
      double sum = 0;                                       //合計値を足していく変数を用意
      while ((line = br.readLine()) != null) {          //最終行になるまで読み込む
        sum += Double.valueOf(line);                //各行の内容をint形式にしてsumに足す
      }
      br.close();

      String outputfile = “./output/output1.txt”;       //アウトプットファイル名
      PrintWriter pw = new PrintWriter(new FileWriter(outputfile));
      pw.println(“合計:” + String.valueOf(sum));        //sumをString形式にして書き込み
      pw.close();
    }
    catch( IOException e ) {                    //データを入出力ができなかったとき
      System.out.println("データ入出力失敗");
    }
  }
}
```

# 1.データの読み込みと書き出し

練習2

input2.csvの各行を読み込み，1列目と2行目の和と差を求め，output2.csvとして，1列目に和を，2列目に差を出力せよ．

ヒント：Math.abs

**input2.csv**

```
151215,1513205
4564258,151
672842,5446
3542415,6545
84542,1215
```

**output2.csv**

```
1664420,1361990
4564409,4564258
678288,667396
3548960,3535870
85757,83327
```

# 1.データの読み込みと書き出し

練習2’

練習2と同様だが，input2-2には3列目がある．
3列目がNGの場合には出力させないこと．
ヒント：文字列比較は”equals()”

**input2-2.csv**

```
151215,1513205,OK
4564258,151,OK
672842,5446,OK
3542415,6545,NG
84542,1215,OK
```

**output2-2.csv**

```
1664420,1361990
4564409,4564258
678288,667396
85757,83327
```

# 2.データの格納

## 配列

１つの変数に複数の値を格納する

インデックスが[0]から始まることに注意

## 練習3

## input3.csvからOD表を作成せよ.

ヒント：文字列比較は”equals()”

**input3.csv**

```
出発,到着,手段,目的,拡大係数
0,0,鉄道,業務,0083
0,0,鉄道,業務,0083
0,2,鉄道,業務,0083
2,0,鉄道,帰宅,0083
0,0,徒歩,買い物,0037
0,0,徒歩,帰宅,0037
0,0,鉄道,通勤,0047
:
4,4,自転車,帰宅,0092
```

**output3.csv**

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | ? | ? | ? | ? | ? | ? |
| 1 | ? | ? | ? | ? | ? | ? |
| 2 | ? | ? | ? | ? | ? | ? |
| 3 | ? | ? | ? | ? | ? | ? |
| 4 | ? | ? | ? | ? | ? | ? |
| 5 | ? | ? | ? | ? | ? | ? |

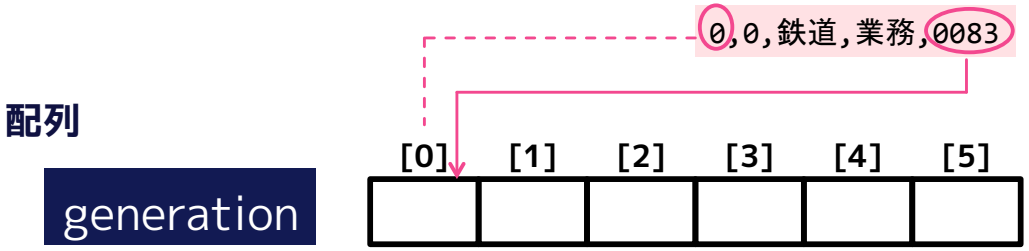
# 2.データの格納

## 練習3の参考(前半)

```
import java.io.*;                                   //java.ioパッケージを使う

public class Main {
  public static void main(String[] args) {
    try {                                           //データをうまく入出力できるとき
      String inputfile = "./input/input3.csv";          //インプットファイル名
      BufferedReader br = new BufferedReader(new FileReader(inputfile));
      String line = null;                                   //1行ごとに読み込む変数を用意
      int[] generation = new int[6];                //要素6つの配列用意、各ゾーンの発生量が入る
      while ((line = br.readLine()) != null) {          //最終行になるまで読み込む
        String[] splitline = line.split(",");           //カンマ区切りで分割
        generation[Integer.valueOf(splitline[0])] += Integer.valueOf(splitline[4]);

                                        //ゾーン(1列目)に対して拡大係数(5列目)をint型にして足す
      }
      br.close();
```



# 2.データの格納

練習3の参考(後半)

```
      String outputfile = “./output/output3.txt”;      //アウトプットファイル名
      PrintWriter pw = new PrintWriter(new FileWriter(outputfile));
      for (int i = 0; i < 6; i++) {          //ゾーン0～5の集計結果を出力したい
        pw.println(i + “:” + String.valueOf(generation[i]));
                                  //ゾーンiの発生交通量を書き出し
      }
      pw.close();
    }
    catch( IOException e ) {                  //データを入出力ができなかったとき
      System.out.println("データ入出力失敗");
    }
  }
}
```

# 2.データの格納

## Arraylist(リスト)とHashmap(マップ)

・データを管理するデータ構造

・要素数の事前の定義が不要

**Arraylist**：順番と要素で管理

| | |
|---|---|
| 0 | 日本 |
| 1 | カナダ |
| 2 | アメリカ |
| 3 | ドイツ |
| 4 | 中国 |
| : | : |

**Hashmap**：キーと要素で管理

| | |
|---|---|
| Japan | 日本 |
| Canada | カナダ |
| USA | アメリカ |
| Germany | ドイツ |
| China | 中国 |
| : | : |

**取り出し方**

(Arraylist名).get(2) → “アメリカ”

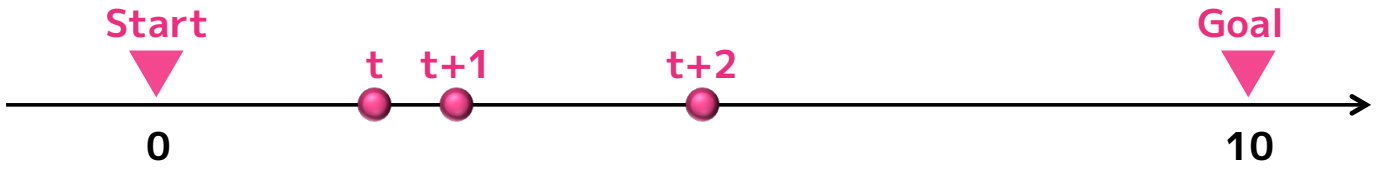
**取り出し方**

(Hashmap名).get(“Japan”) → “日本”

# 2.データの格納

Ex.)1次元ランダムウォーク(Arraylistの例)

ゴールまで毎ステップ0~1の範囲でランダムに進む




```
import java.util.ArrayList;                    //ArrayListパッケージのインポートが必要
public class Main {
   public static void main(String[] args) {
      double now = 0.0;                        //現在位置を表す変数
      ArrayList<Double> place = new ArrayList<Double>();
                                    //placeに時系列の位置を格納する、この時点では空っぽ
      while(now < 10){              //現在位置が10を超えるまで計算を繰り返す
         now = now + Math.random();            //現在位置更新、Math.random()は0～1の乱数
         place.add(now);                       //placeに要素を追加する
      }
                                    //この時点でゴールまでの時系列データが書き込まれた
      for (int i = 0; i < place.size(); i++){          //.size()で要素数を取得
         System.out.println(place.get(i));             //コンソールにi番目の要素を表示
      }
      System.out.println("ステップ数:" + place.size());   //要素数(ステップ数)を表示
   }
}
```

# 3.クラスとメソッド

## クラスとインスタンス

クラス：関連情報を一つにまとめたもの

一つ一つの実体がインスタンスであり，同じクラスのインスタンスは共通の性質を持つ.

フィールド(状態/性質を定義する)とメソッド(機能を定義する)

**Nodeクラス・・・プログラム中でノードデータを格納する**

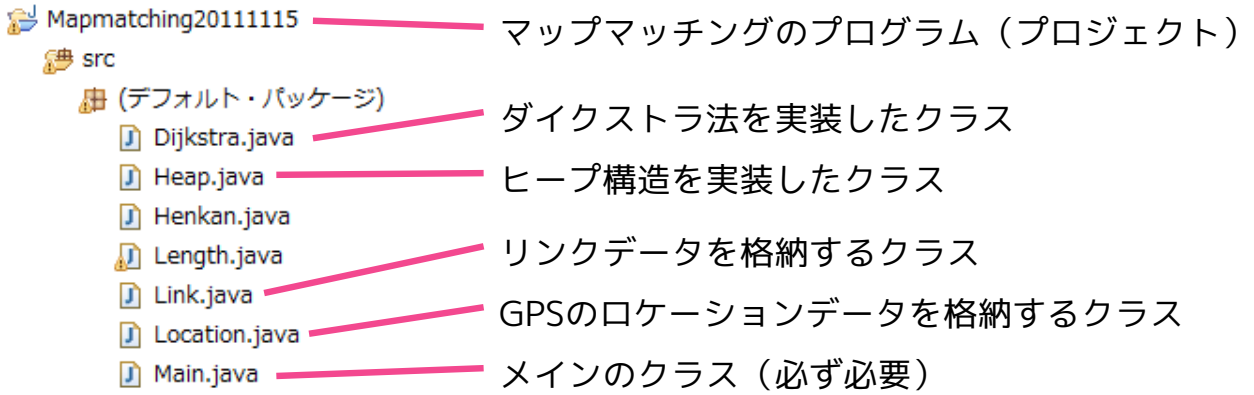
| **Nodeインスタンス** | **Nodeインスタンス** | **Nodeインスタンス** |
|---|---|---|
| ノードID：0 | ノードID：1 | ノードID：2 |
| 緯度：35.75 | 緯度：35.99 | 緯度：35.88 |
| 経度：36.64 | 経度：36.32 | 経度：36.09 |
| 最短経路探索済：0 | 最短経路探索済：0 | 最短経路探索済：0 |

# 3.クラスとメソッド

## クラスの実装例

```
public class Station {              //駅データを格納するクラス
  String name;                      //駅の名前
  double lat;                       //駅の緯度
  double lon;                       //駅の経度
  Station(String n, double x, double y){              //駅の情報を設定する
    name = n;
    lat = x;
    lon = y;
  }
}
```

各クラスは”~.java”というファイルになる.



# 3.クラスとメソッド

## インスタンスの例

```
import java.util.ArrayList;                    //ArrayListのインポート
public class Main {
   public static void main(String[] args) {
      ArrayList<Station> stalist = new ArrayList<Station>();       //ArrayList作成
      stalist.add(new Station(“東京”,35.6813,139.7661));
      stalist.add(new Station("上野",35.7137,139.7770));
      stalist.add(new Station("新橋",35.6661,139.7585));
                                    //作ったArrayListのstalistにデータを3つ入れてみた
      for (int i = 0; i < stalist.size(); i++){          //格納データの駅名を全て表示
         System.out.println(stalist.get(i).name);
      }
   }
}
```

# 3.クラスとメソッド

## メソッド

戻り値・引数を指定する(Rでいうところの関数のようなもの)

よく使うデータ操作はメソッドにすると楽

Ex.)掛け算のメソッド

```
public class Main {
  public static void main(String args[]){
    double a = 1.5;
    double b = 1.5;
    double ans = product(a,b);   //aとbをproductメソッドを使って計算
    System.out.println(ans);     //答えを表示
  }

          //ここからがメソッド、かけ算をするメソッドを作る
  private static double product(double x, double y){
          //返り値の型がdouble、引数にdoubleの変数xとyをとるという意味
    return (x * y);   //returnで返り値を表す
  }
}
```

# 3.クラスとメソッド

練習4

2地点A,Bの緯度経度からA-B間の距離を計算するメソッドを作成せよ.

ヒント: 引数は4つ(Aの緯度, Aの経度, Bの緯度, Bの経度)

緯度経度と距離の関係はぐぐる！(何個か方法があります)

東京駅(35.681143, 139.767208)と
横浜駅(35.466193, 139.622498)との距離27280mで確認する

Math.toRadians, Math.sqrtなど. . .

# 3.クラスとメソッド

練習5

input6-2.csvの位置データ一つ一つに対し，
input6-1.csvを用いて最寄駅を求めよ．

ヒント: Stationクラスを定義する

練習4のメソッドを使う

**input6-1.csv**
**(山手線内駅データ)**

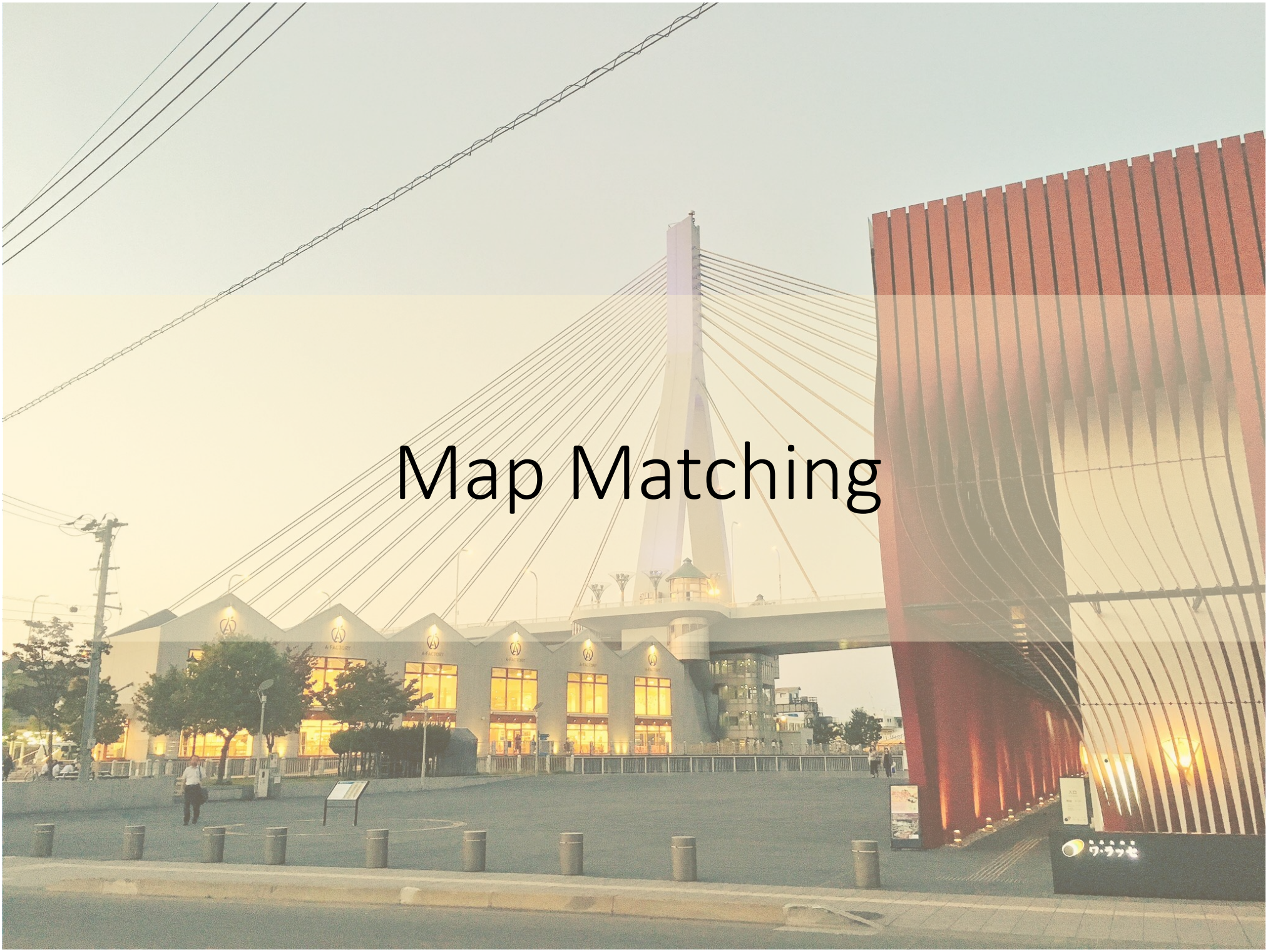
```
東京,35.6813,139.7661
上野,35.7137,139.7770
有楽町,35.6754,139.7638
新橋,35.6661,139.7585
浜松町,35.6553,139.7571
```

**input6-2.csv**
**(位置データ)**

```
35.6921,139.7515
35.6435,139.7204
35.6242,139.7495
35.7122,139.7354
35.6513,139.7264
:
```

**output6.csv**

```
35.6921,139.7515, 飯田橋
35.6435,139.7204, 恵比寿
35.6242,139.7495, 品川
35.7122,139.7354, 飯田橋
35.6513,139.7264, 恵比寿
:
```


Map Matching

# Map Matching

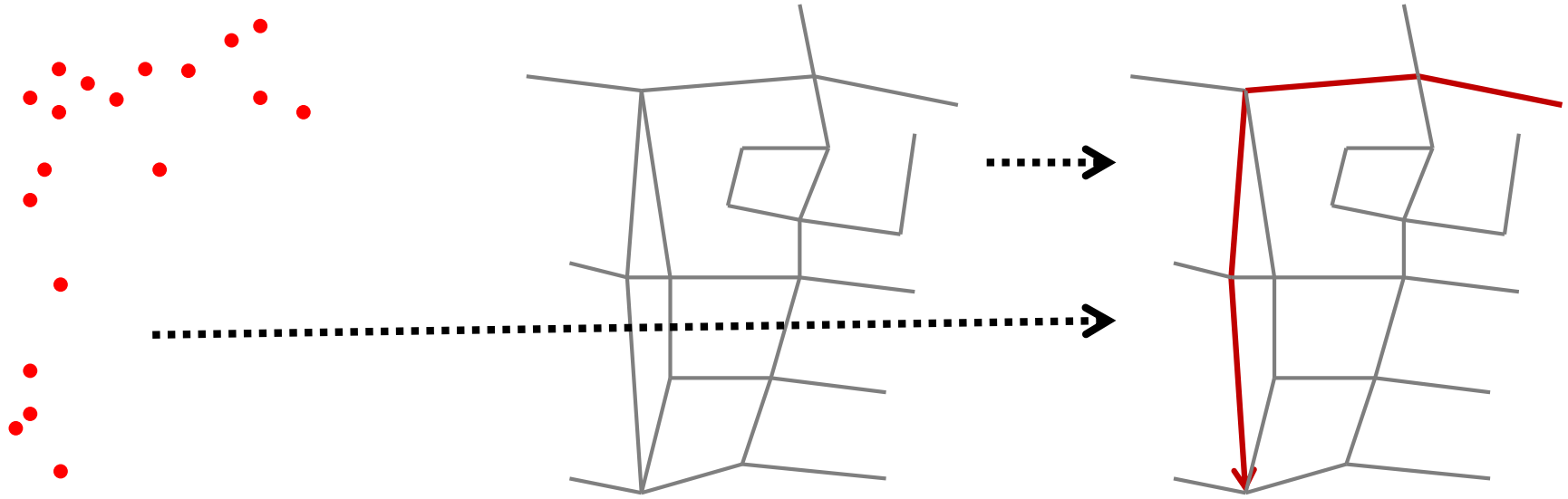
## Map Matching

位置データを用いた，ネットワーク上での経路特定手法

ネットワークと位置データの関係を定量化し，通過リンクを特定

多くの手法が存在(Point to Point/ Point to Curve…//幾何解析/位相幾何解析…)

**観測データ**　　**ネットワークデータ**　　**経路**



# 宿題

配布したPPデータ(位置情報データ)とネットワークデータを用いて経路データを作成してください.

また，解答例のアルゴリズムの持つ問題点を探し，これの解決策を提案してください.