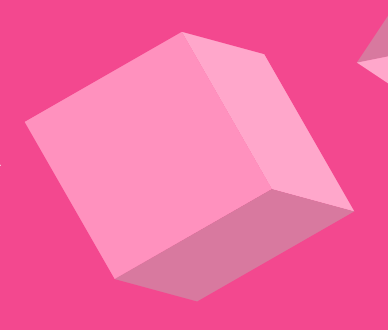
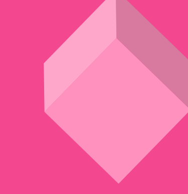
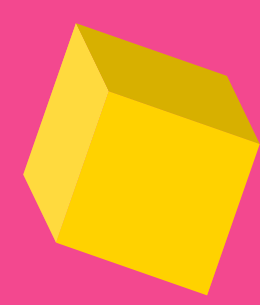
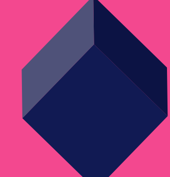
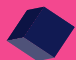


Javaプログラミング

(第1回 理論勉強会)

伊藤 創太 作成 (2012)

目次

-基本編-

1．はじめに（なぜプログラミング？）

2．超基礎（やりながら確認）

3．データ構造とアルゴリズム

4．計算量について

-実用編-

1．平均・分散

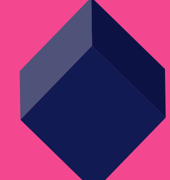
2．和と差

3．OD表の作成

4．メソッド

+α．オブジェクト指向

基本編



なぜプログラミング？

・高度な計算のため
（最適化，シミュレーション，…）
・膨大なデータを処理するため
（統計解析，可視化，…）
・手戻りをなくすため
（共通のデータ処理・集計手法）
・…

基本事項 (やりながら確認)

Eclipseでのプログラムの作成手順

■[ファイル]-[新規]-[Javaプロジェクト]

・プロジェクト名は自由（例：”Startup2015”）

・Workspaceにプロジェクトフォルダが作成される

■[ファイル]-[新規]-[クラス]

・クラス名を定義（推奨：”Main”）

・public static void main(String[] args) にチェック

```
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

	}

}
```

基本事項 (やりながら確認)

Javaプログラミングの超基礎

1．変数の宣言（**型**：String，int，long，double,…）

2．セミコロン（”;”）で指示を終了

3．計算してみよう（**演算子**：+, -, *, /, %,…）

//演算子の前後はスペースを挟む（例：x * 2 + y）

4．結果を表示（**System.out.prntln()** メソッド）

5．制御構文（if ~ else, for, while）

6．条件文（**比較演算子**：==, !=, >, <, >=, <=,…）

//定義の”=“とはっきりと区別する．※文字列の比較はequals()を使う

7．デバッグ（debug）

※どの教科書・Webサイトにも載っているので
　あとでもう一度確認しておくこと。

やってみよう１

問題

1. コンソールに“Hello! World”と表示させてください.
2. for文を使って1～100までの数字を表示させてください.
3. for文を使って”1,2,3,…,99,100”と1行で表示させてください（※print()を使います）.
4. for文を使って1~100のうち素数のみを表示させてください.
5. while文を使って，以下の式を満たす最小の x, y をそれぞれ求めてください.

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

データ構造とアルゴリズム

プログラムの実装

データ構造
データをどのようにコンピュータのメモリに並べるかという方式・形式

アルゴリズム
データを操作する手続き

アルゴリズムの理解
→プログラミング
適切なアルゴリズム・データ構造の選択
→計算の高速化・汎用化

配列

・1つの変数に複数の値を格納
・必ず[0]から始まるインデックスを持つ

インデックス（順序番号）

インデックス	値
[0]	Chikamatsu
[1]	Shoji
[2]	Maeda
[3]	Miki
[4]	Umezawa

コード例：

```
//5要素の配列を用意
String[] name = new String[5];
name[0] = “Chikamatsu”;
name[1] = “Shoji”;
name[2] = “Maeda”;
name[3] = “Miki”;
name[4] = “Umezawa”;
```

取り出し方

name[4] → “Umezawa”

Q1. for構文を使って順番に表示させてみよう。

Arraylist（リスト）とHashmap（マップ）

・データを管理するデータ構造
・データの要素数を事前に定義しなくても良い（拡張可能）

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
:	:

取り出し方
(Arraylist名).get(2) → “アメリカ”

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
(Hashmap名).get(“Japan”) → “日本”

値の探索と並び替え（ソート）

・最も基本的で重要なアルゴリズム

・先ほど用意した配列に対して、

[0]	Chikamatsu
[1]	Shoji
[2]	Maeda
[3]	Miki
[4]	Umezawa

Q2. 探索：自分の名前だけ取り出せますか（何番目にあるか）？

Q3. ソート：Arrays.sort(name) とやれば簡単にできるけれど…

線形探索

・順番に配列の中身を調べていって，答えが見つかれば終了

"Miki" は何番目か？

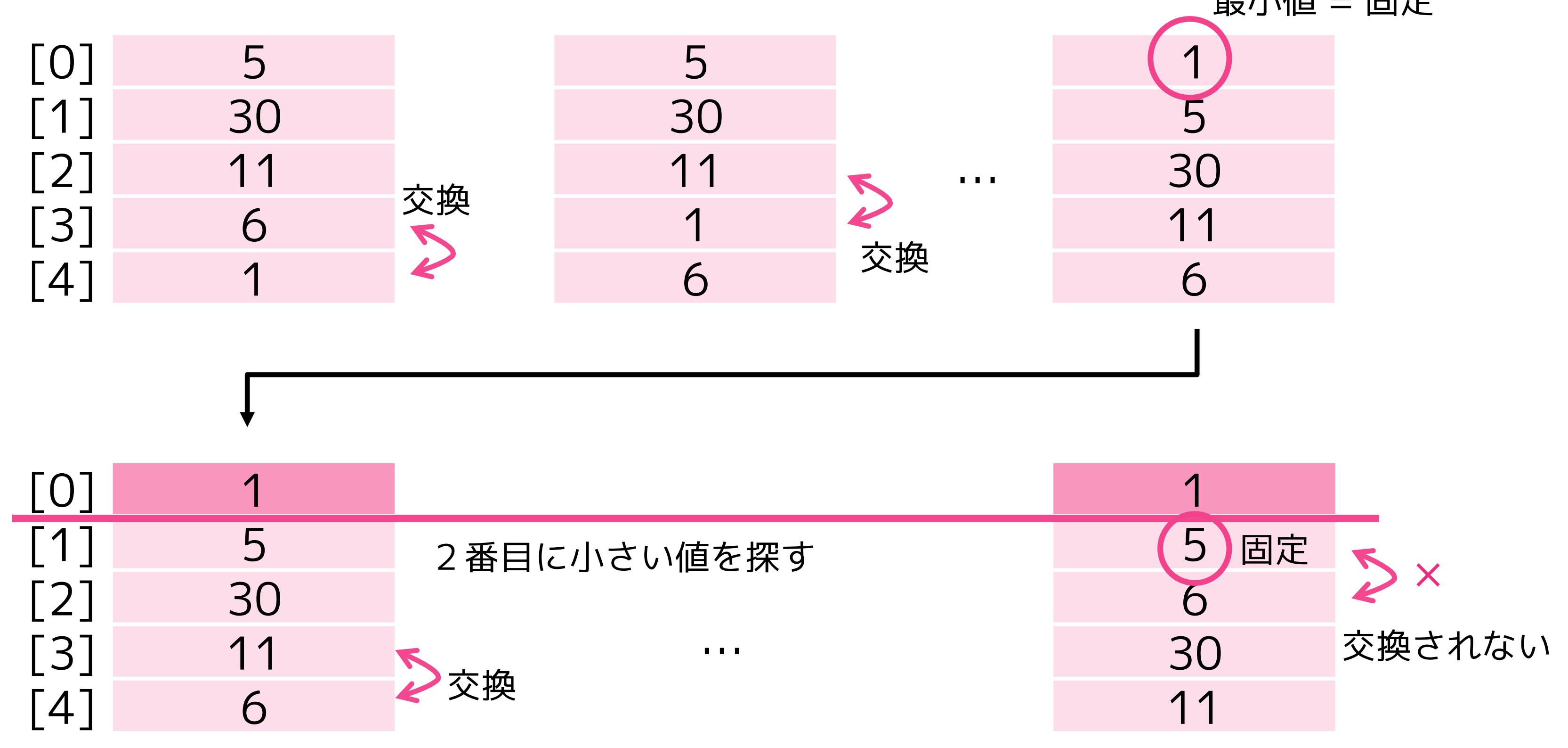
[0]	Chikamatsu	・・・ちがう！
[1]	Shoji	・・・ちがう！
[2]	Maeda	・・・ちがう！
[3]	Miki	・・・**あった！（探索終了）**
[4]	Umezawa	

→ （3＋1）＝ 4番目 が答え

バブルソート

・上の要素と比較し、上の方が大きければ交換

・最小値から順番に決定して（浮かんで）いく



やってみよう２

問題

・配列：`int[] a = {14,59,1,20,37,90,22}`

について，

1. **バブルソートアルゴリズム**を作って並び替えたのち，
2. **線形探索**で「22」が何番目に来たかを求めてください．

（１）配列を用意

[0]	14
[1]	59
[2]	1
[3]	20
[4]	37
[5]	90
[6]	22

（２）ソート

[0]	1
[1]	14
[2]	20
[3]	22
[4]	37
[5]	59
[6]	90

（３）線形探索

アルゴリズムと計算量

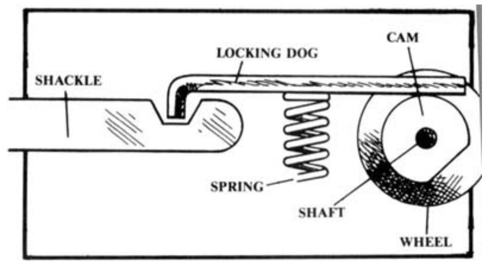
・n 桁の暗証番号を線形探索で解く場合
・n = 4 であれば，10000通り
・1パタン1秒とすると最悪 2.8時間
・n = 10 だと…？：317年

10^n 秒 ⟶ $O(10^n)$

O記法（ビッグオー記法）

・セサミロックデコーダー*を使えば：
・n = 4 → 40秒，n = 10 → 100秒

$10n$ 秒 ⟶ $O(n)$



*1桁ずつ正解かどうかわかる

計算量の評価 – O記法

データ数nが増えると、オーダーの違いによる差は大きくなる

n:	2	4	8	16	100	1万	100万	1億
1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
$log_2 n$	0.7	1.4	2.1	2.8	4.6	9.2	13.8	18.4
logn	1.0	2.0	3.0	4.0	6.6	13.3	19.9	26.6
n	2.0	4.0	8.0	16.0	100.0	10000.0	1.0×10^6	1.0×10^8
nlogn	1.4	5.6	16.6	44.4	460.5	92103.4	1.4×10^7	1.8×10^{10}
n^2	4.0	16.0	64.0	256.0	10000.0	1.0×10^8	1.0×10^{12}	1.0×10^{16}
n!	2.0	24.0	40320.0	2.0×10^{13}	#	#	#	#

計算時間の計測

currentTimeMillis()メソッド

・1970年1月1日からの経過時間をミリ秒単位で取得
・計算開始・終了時の取得時間の差で、計算時間を計測
・プログラム全体や、部分ごとの計算時間計測も可能
・1000ミリ秒 = 1秒
・long型を使用

```
long start = System.currentTimeMillis();
```

start･･･1340418208076（2012年6月23日11:30の場合）

計算時間の計測

currentTimeMillis()メソッド

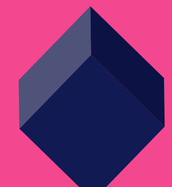
コード例：

```
public class Main {
  public static void main(String[] args) {
    long start = System.currentTimeMillis();          //計算開始時の時刻を取得

    for (int i = 0; i < 1000000; i++){     //計算の内容
      System.out.println(i + 1);           //ここでは1から100万までの数字を表示させる
    }

   long finish = System.currentTimeMillis();          //計算終了時の時刻を取得
   double duration = (finish - start) / 1000;         //ミリ秒から秒に変換
   System.out.println("計算時間:" + duration + "秒");    //計算時間を表示
  }
}
```

実行結果：

```
1
:
999999
1000000
計算時間:16.0秒
```

実用編



合計・平均・標準偏差

データの集計 – 合計を求める

・データを集計したり、計算したりしたいことがあるよね
・大きいデータだと、Excelで開けないこともあるよね
・簡単な計算はサクッとできるようになりたい

アウトプットデータ：

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

合計・平均・標準偏差

データの集計 – 合計を求める

・ここでは、合計の求め方だけ、例を示す

```
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");
    }
  }
}
```

合計・平均・標準偏差

BufferedReader

・ファイルの入力

・readline()で1行ごとに読み込む

PrintWriter

・ファイルの出力

・printlnで1行ごとに書き込む

・String型で入出力が行われる

・tryとcatchで例外処理を書かなければならない

・close()で閉じるのを忘れずに

valueOf

・数値→文字列、文字列→数値の変換

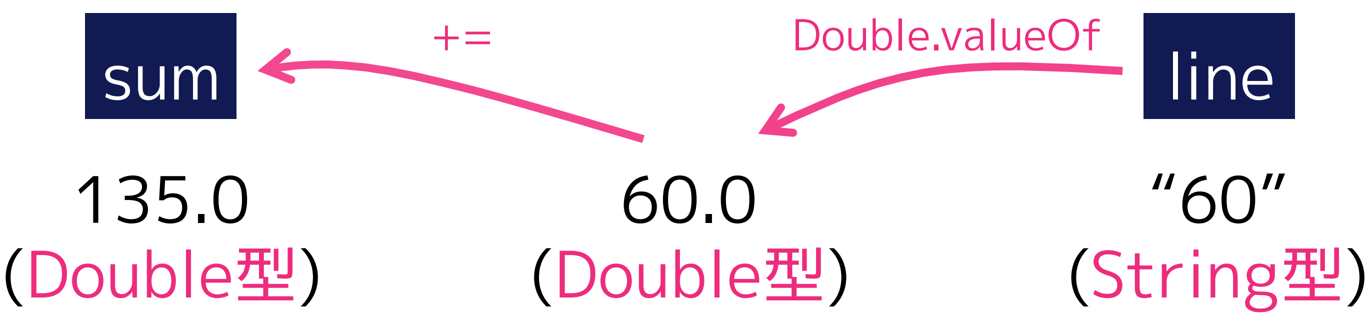
・例1：String.valueOf(数値)

・・・数値を文字列(String)に変換

・例2：Double.valueOf(文字列)

・・・文字列(String)を数値(Double)に変換

11行目 `sum += Double.valueOf(line);`



課題1

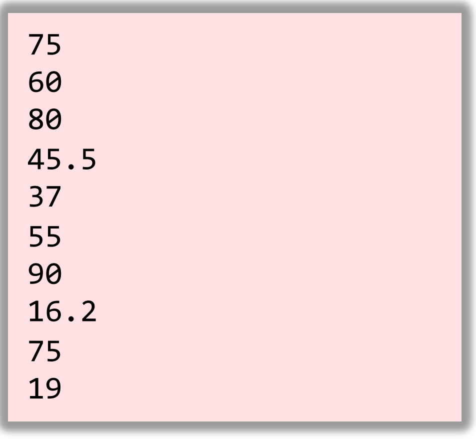
データの集計 – 合計・平均・標準偏差

- 合計と平均と標準偏差を求めるプログラムを作れ
- どんな変数を用意するべきか
- 標準偏差

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\bar{x} - x_i)^2}$$

- ヒント：Math.sqrt

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
```

課題1'

データの集計 – 合計・平均・標準偏差

・課題1の計算に要した時間を計測して、コンソールに表示させよ。

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
```

データの集計 – 足し合わせ

・データの和や差をとりたいことがあるよね
・大きいデータだと、Excelで開けないこともあるよね
・簡単な計算はサクッとできるようになりたい

インプットデータ：

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

→ こうしたい

アウトプットデータ：

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

複数列のcsvファイルの読み込み – split

・特定の文字で区切って配列にしてくれる

例
```
line = br.readLine()  //1行分読み込み
String[] splitline = line.split(",");
```

line —Split→ splitline

（String型）
"151215,1513205"

（String[]型、String型の配列）

"151215"	"1513205"
splitline[0]	splitline[1]

0からはじまるので注意

課題2

データの集計 – 複数データの和と差

・1列目と2列目の和と差を各行について求めよ

・アウトプットデータの1行目に和、2行目に差を出力

・各行の和と差の値を格納する方法が必要

・ヒント：Math.abs、println(何か + “,” + 何か)

input2.csv

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

→

output2.csv

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

課題2'

データの抽出

・1列目と2列目の和と差を各行について求めよ

・アウトプットデータの1行目に和、2行目に差を出力

・ただし、3列目がNGの場合は出力させないこと

・ヒント：eqauls()

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
```

データの集計 – カウント

・属性ごとに集計したいことがあるよね
・大きいデータだと、Excelで開けないこともあるよね
・簡単な計算はサクッとできるようになりたい

インプットデータ：

```
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
```

→ 出発ゾーンごとに発生交通量を集計したい

アウトプットデータ：

```
0,34887185
1,19043082
2,12088410
3,12743172
4,3425779
5,2642548
```

※データは実際の東京都市圏PTデータを加工したもの
ゾーンは、0:東京、1:神奈川、2:埼玉、3:千葉、4:茨城、5:東京都市圏外

(補足)PTデータの拡大係数について

層別拡大

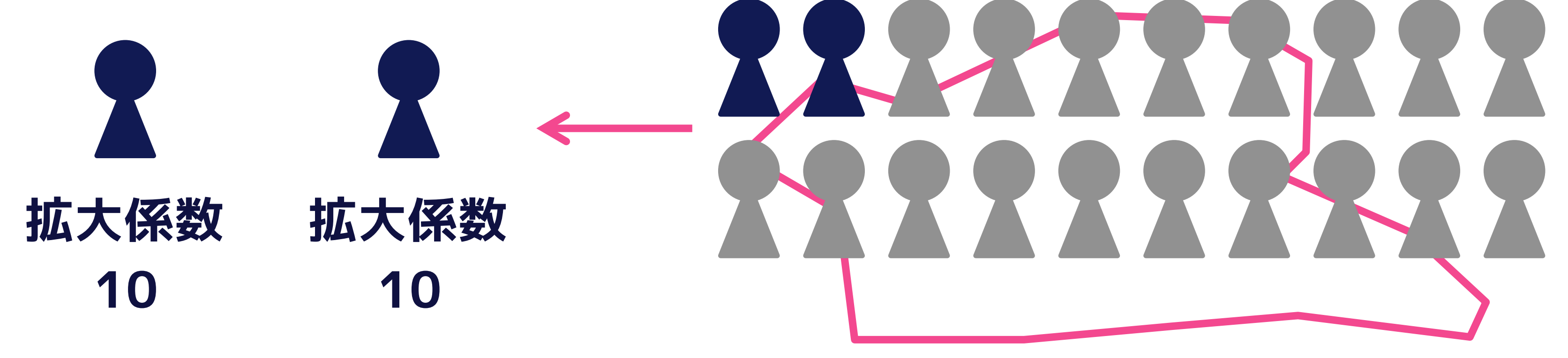
・居住地・年齢階層・性別で層別に拡大

・抽出データから実際のトリップ数に拡大する

・何人分のトリップを代表しているかという意味

男性、20〜25歳、地域Aの取得サンプル・・・2人

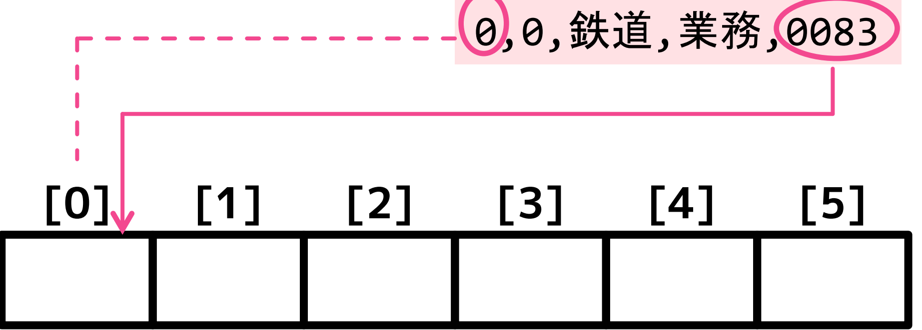
男性、20〜25歳、地域Aの居住者・・・20人



データの集計 – 発生交通量の集計（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();
```

配列



generation

データの集計 – 発生交通量の集計（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");
    }
  }
}
```

課題3

データの集計 – OD表を作ろう

- 出発・到着ゾーンごとに拡大係数を合計せよ
- 表の行方向を出発、列方向を到着とする
- 余力があれば代表交通手段別のODも求めよ
- ヒント：int[][] od = new int[6][6]でOD行列を初期化

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	?	?	?	?	?	?

メソッド

・戻り値・引数を指定する

・よく使うデータの挿入や抽出、計算などはメソッドに。

```
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
  }
}
```

課題4

メソッド – 2地点の緯度経度から距離を計算

・4つの引数（Aの緯度、Aの経度、Bの緯度、Bの経度）から直線距離を求めるメソッドを作って計算

・最も簡単には地球を球体とみなして、地球の半径と緯度差から南北距離、経度差から東西距離を出して・・・

・より正確な距離を求める計算式もいろいろあるので実装してみてください

ヒント：Math.toRadians、Math.sqrt

(答え合わせ)

東京駅(35.681143,139.767208)から横浜駅(35.466193,139.622498)の距離 → 27280m

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
・データの要素数を事前に定義しなくても良い（拡張可能）

Arraylist：順番と要素で管理

Hashmap：キーと要素で管理

キー	要素
Japan	日本
Canada	カナダ
USA	アメリカ
Germany	ドイツ
China	中国
：	：

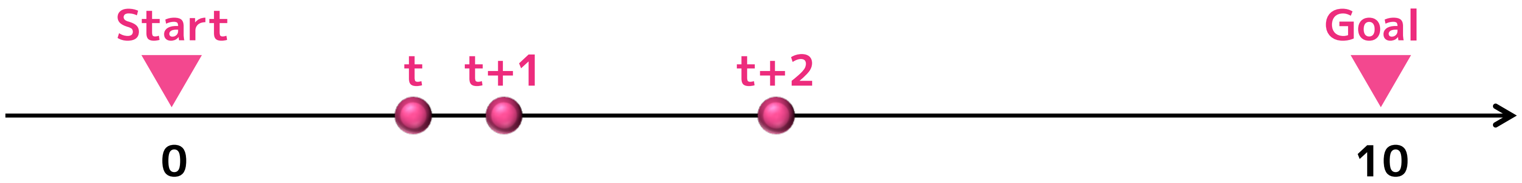
取り出し方
(Arraylist名).get(2) → “アメリカ”

取り出し方
(Hashmap名).get(“Japan”) → “日本”

データの格納

Arraylistの例：1次元でのランダムウォーク

・ゴールまで1ステップに0～1の間でランダムに進む




```
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
   }
}
```

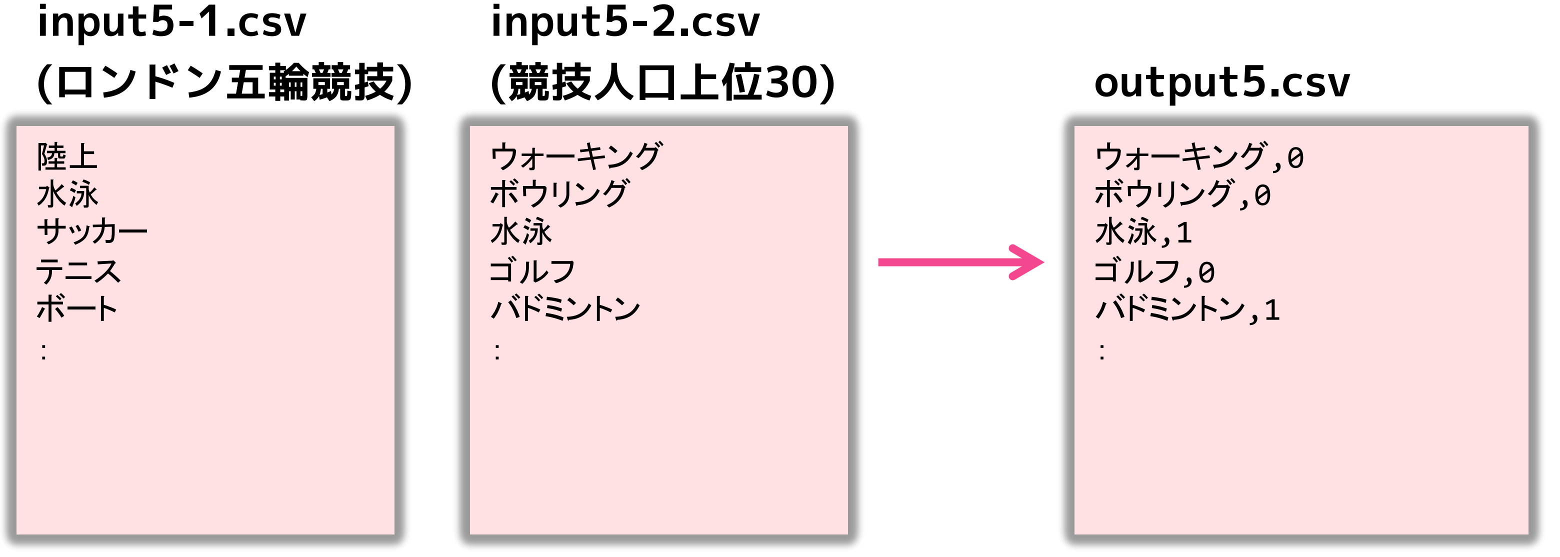
課題5

データの集計 – 一致データの探索

・競技人口上位30のスポーツがロンドン五輪競技に含まれるかどうかを調べる

・データはArrayListで格納しておく

・ヒント：文字列比較はequals()を使う



オブジェクト指向

クラスとインスタンス（“学生”をクラスの例で考えると）

・オブジェクトの型を**クラス**として定義する

・ひとつひとつの実体(学生1人1人)が**インスタンス**

・同クラスのインスタンスは共通の性質(学生である)を持つ

・**フィールド**：クラスで共通の状態を表す値
（学籍番号、名前、性別、出身など）

・**メソッド**：クラスで共通の機能
（勉強する、就活する、遊ぶなど）

クラスとインスタンスなどの話は全てできないので、
各自勉強してフォローするように。

オブジェクト指向

例

“学生”クラス

“学生”インスタンス
名前：“Yばし”
学年：1
学籍番号：0815
出身：“関東”
性別：“男“

“学生”インスタンス
名前：“Tきー”
学年：2
学籍番号：1116
出身：“九州”
性別：“男“

“学生”インスタンス
名前：“Eぽよ”
学年：2
学籍番号：1123
出身：“近畿”
性別：“女“

研究室のプログラムの例だと、

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
ノードID：0
緯度：35.75
経度：36.64
最短経路探索済：0

Nodeインスタンス
ノードID：1
緯度：35.99
経度：36.32
最短経路探索済：0

Nodeインスタンス
ノードID：2
緯度：35.88
経度：36.09
最短経路探索済：0

オブジェクト指向

クラスの実装（例として鉄道駅データを扱う）

```
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;
  }
}
```

クラスは「～.java」という一つのファイルになる

例）

- Mapmatching20111115 — マップマッチングのプログラム（プロジェクト）
  - src
    - (デフォルト・パッケージ)
      - Dijkstra.java — ダイクストラ法を実装したクラス
      - Heap.java — ヒープ構造を実装したクラス
      - Henkan.java
      - Length.java
      - Link.java — リンクデータを格納するクラス
      - Location.java — GPSのロケーションデータを格納するクラス
      - Main.java — メインのクラス（必ず必要）

インスタンスの作り方・取り出し方

ここではArrayListの中にStationクラスのインスタンスを加えていく

```java
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);
      }
   }
}
```

ArrayList
stalist

[0]	[1]	[2]	
name:"東京" lat:35.6813 lon:139.7661	name:"上野" lat:35.7137 lon:139.7770	name:"新橋" lat:35.6661 lon:139.7585	・・・

Stationのクラスの型

課題6

最寄り駅の探索

・input6-2.csvの位置データ1つ1つに対して、最寄り駅を求めて出力する。

・Stationクラスを自分で定義する

・ヒント：課題4のメソッドを使って・・・

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
:
```