Perhatikan permasalahan yang ada pada tutorial ini baik-baik. Permasalahan yang disajikan pada tutorial ini akan menjadi **landasan penting** untuk mengikuti perkuliahan SDA dan juga mengerjakan tugas lain yang diberikan saat sesi lab.

Problem 1. Implementasi Interface

Diberikan definisi interface **BentukDuaDimensi** berikut:

```
public interface BentukDuaDimensi {
     double getLuas();
     double getKeliling();
}
```

Implementasikan kelas Circle yang merepresentasikan sebuah lingkaran dengan spesifikasi sebagai berikut:

- Kelas Circle mengimplementasikan **BentukDuaDimensi**.
- Kelas Circle mempunyai sebuah atribut, yaitu **radius (double)**.
- Object-object yang berasal dari kelas Circle **dapat dibandingkan**. Object-object Circle akan diurutkan **berdasarkan radius**, dari yang paling pendek hingga yang paling panjang.

Problem 2. Reading Input, Tokenizing, Sorting

Ketika kuliah DDP, Anda sering sekali menggunakan **Scanner** untuk menerima input standar (*standard input*). Untuk input yang sangat banyak, penggunaan **Scanner** tidak efisien karena Scanner menggunakan *proses Parsing* yang cukup memakan waktu. Sekarang, saatnya Anda belajar menggunakan Java API yang lain, yaitu **InputStreamReader** yang dibungkus dengan **BufferedReader** agar proses penerimaan input bisa lebih cepat. Method yang digunakan hanya satu, yaitu **readLine()** yang membaca stream satu baris dan mengembalikan dalam bentuk String. Untuk melakukan tokenisasi String, Anda bisa menggunakan method **split()** yang ada pada object String atau **StringTokenizer**.

Buatlah sebuah program yang menerima sejumlah informasi mahasiswa **<nama, npm, ipk>**. Input mempunyai format berikut:

<#baris>
<$nama_1$, npm_1, ipk_1>
<$nama_2$, npm_2, ipk_2>
...
<$nama_n$, npm_n, ipk_n>

Baris pertama berisi sebuah integer yang menyatakan banyaknya mahasiswa yang informasinya akan dimasukkan, yaitu sebanyak **n**. Baris berikutnya hingga akhir berisi informasi mahasiswa sebanyak **n** yang sudah dinyatakan pada baris pertama.

Contoh:

```
5
Ani 1234 3.8
Andi 1223 3.5
Toni 1213 2.7
Rudi 1244 3.5
Budi 1245 3.7
```

Diberikan input seperti di atas, program akan melakukan komputasi dan akhirnya menampilkan kembali informasi **<nama, npm, ipk>** secara terurut sesuai aturan:

- Urutkan berdasarkan **ipk**, dari yang paling **besar** ke yang paling **kecil**
- Jika **ipk** sama, urutkan berdasarkan **npm** (yang bersifat unik) dari yang paling kecil ke yang paling besar.

Contoh Output:

```
Ani 1234 3.8
Budi 1245 3.7
Andi 1223 3.5
Rudi 1244 3.5
Toni 1213 2.7
```

Sementara ini, Anda bisa menggunakan **collections.sort()** untuk mengurutkan data yang disimpan di dalam sebuah collections, seperti **ArrayList**. Di kuliah berikutnya, Anda akan belajar cara mengurutkan data yang lebih efisien.

Problem 3. Reading Input until EOF

Modifikasi kode yang sudah Anda buat untuk menyelesaikan Problem 2! Pada Problem 2, informasi banyaknya mahasiswa dituliskan pada baris pertama sehingga Anda bisa dengan nyaman menggunakan struktur **for loop**. Sekarang, input tidak mengandung informasi banyaknya baris secara eksplisit. Program Anda harus bisa membaca informasi mahasiswa **hingga akhir input (*end of file*)**.

Ketika Anda memasukkan input di command prompt windows, tanda EOF direpresentasikan dengan **Ctrl+Z**. Jika Anda menggunakan Linux, gunakan **Ctrl+D.**

Contoh Input:

```
Ani 1234 3.8
Andi 1223 3.5
Toni 1213 2.7
Rudi 1244 3.5
Budi 1245 3.7
Ari 1227 3.5
Cepy 1244 3.0
```

Contoh Output:

```
Ani 1234 3.8
Budi 1245 3.7
Andi 1223 3.5
Ari 1227 3.5
Rudi 1244 3.5
Cepy 1244 3.0
Toni 1213 2.7
```

Problem 4. StringBuilder

Ketika kuliah DDP, kalian sering menggunakan operator **(+)** untuk melakukan konkatenasi beberapa String (menggabungkan String). Jika operasi konkatenasi

tidak begitu banyak, cara seperti itu tidak masalah. Namun, jika program yang kita buat melibatkan banyak operasi konkatenasi, cara seperti itu **bisa tidak efisien**. **StringBuilder** hadir dimulai pada JDK 5 agar operasi konkatenasi lebih efisien.

Mengapa bisa lebih efisien? String bersifat immutable sehingga setiap kali operasi konkatenasi dilakukan, operator **(+)** akan menghasilkan object String baru. Jelas, cara ini tidak efisien. StringBuilder bisa dianggap memiliki sifat seolah-olah mutable sehingga operasi konkatenasi String hanya cukup menambahkan karakter-karakter baru pada ujung *internal char array* dari String tersebut.

Contoh konversi konkatenasi menggunakan **(+)** dengan **StringBuilder**:

```
String s = a + b + c;

String s = new StringBuilder().append(a).append(b).append(c).toString();
```

Implementasikan method **toString()** pada kelas **Folder** sehingga menghasilkan String dengan format:

```
[judul-1, author-1]
[judul-2, author-2]
...
[judul-n, author-n]
```

Template pengisian ada di file **Problem4.java**.

Problem 5. Problem Solving using Data Structure

Sejauh ini, Anda baru belajar menggunakan ADT List dengan implementasi **ArrayList**. Jadi, tidak masalah jika Anda menggunakan **ArrayList** untuk menyelesaikan permasalahan ini. Di kuliah berikutnya, Anda akan belajar menggunakan ADT atau jenis struktur data lain yang **jauh lebih efisien** untuk menyelesaikan permasalahan yang diberikan.

Sebuah toko baju menjual 5 jenis ukuran baju, yaitu **S**, **M**, **L**, **XL**, dan **XXL**. Anda diminta untuk melakukan **rekapitulasi berapa banyak masing-masing jenis baju tersebut dibeli**.

Format masukan:

```
<nama pembeli-1, jenis baju-1>
<nama pembeli-2, jenis baju-2>
...
<nama pembeli-n, jenis baju-n>
```

Masukan berakhir dengan tanda End-of-File (**EOF)**.

Contoh masukan:

```
Andi S
Budi M
Toni S
Rani XXL
Rudi M
Ani M
Roni L
```

Contoh keluaran:

```
S 2
M 3
L 1
XL 0
XXL 1
```

Output pasti hanya terdiri dari 5 baris, dimana setiap baris berkaitan dengan masing-masing jenis ukuran baju. **"S 2"** artinya adalah baju berjenis ukuran **S** telah dibeli sebanyak 2 unit.

Problem 6. Problem Solving using Data Structure

Modifikasi program yang sudah Anda buat untuk menyelesaikan Problem 5 di atas sehingga format output menjadi:

```
S 2 Andi Toni
```

```
M 3 Ani Budi Rudi
L 1 Roni
XL 0
XXL 1 Rani
```

Jadi, selain bisa menampilkan berapa banyak unit baju yang dibeli untuk setiap jenis ukuran, program juga bisa menampilkan siapa saja nama-nama orang yang membeli baju tersebut. **Nama-nama tersebut diurutkan secara alfabetis**.

Problem 7. ADT Matrix

Implementasikan ADT Matrix pada file **Problem7.java**. ADT Matrix merepresentasikan sebuah object matriks pada matematika dengan operasi-operasi (*methods*) yang dapat diterapkan padanya.

Problem 8. Minesweeper

Have you ever played Minesweeper? It's a cute little game which comes within a certain Operating System which name we can't really remember. Well, the goal of the game is to find where are all the mines within a MxN field. To help you, the game shows a number in a square which tells you how many mines there are adjacent to that square. For instance, supose the following 4x4 field with 2 mines (which are represented by an * character):

```
*...
....
.*..
....
```

If we would represent the same field placing the hint numbers described above, we would end up with:

```
*100
2210
1*10
1110
```

As you may have already noticed, each square may have at most 8 adjacent squares.

The Input
The input will consist of an arbitrary number of fields. The first line of each field contains two integers n and m (0 < n,m <= 100) which stands for the number of lines and columns of the field respectively. The next n lines contains exactly m characters and represent the field. Each safe

square is represented by an "." character (without the quotes) and each mine square is represented by an "*" character (also without the quotes). The first field line where n = m = 0 represents the **end of input** and should not be processed.

The Output

For each field, you must print the following message in a line alone:

```
Field #x:
```

Where x stands for the number of the field (starting from 1). The next n lines should contain the field with the "." characters replaced by the number of adjacent mines to that square. There must be an empty line between field outputs.

Sample Input

```
4 4
*...
....
.*..
....
3 5
**...
.....
.*...
0 0
```

Sample Output

```
Field #1:
*100
2210
1*10
1110

Field #2:
**100
33200
1*100
```

Tips: buat beberapa class, termasuk kelas **Field** !