Lab: Objects and Classes

Problems for exercises and homework for the "Programming Fundamentals" course @ SoftUni

You can check your solutions in Judge.

I. Using the Built-in Java Classes

1. Randomize Words

You are given a **list of words in one line**. **Randomize their order** and print each word on a separate line.

Examples

Input	Output	Comments

`Welcome to SoftUni and have fun learning programming`	`learning`
	`Welcome`
	`SoftUni`
	`and`
	`fun`
	`programming`
	`have`
	`to`
		The order of the words in the output will be different after each program execution.

`Java is the best programming language`	`the`
	`programming`
	`best`
	`language`
	`is`
	`Java`	

Hints

- **Split** the input string (by space) and create an **array of words.**
- Create a random number generator - an object **`rnd`** of type **Random.**
- In a **for-loop exchange, each number** at positions 0, 1, …, **`words.Length-1`** by a number at **random. position**. To generate a random number in range use **`rnd.nextInt(words.length)`**.
- Print each word in the array on a new line.

2. Sum Big Numbers

You will receive two numbers **(0 to 10^{50}),** and print their sum.

Examples

Input	Output

923847238931983192462832102 934572893617836459843471846187346	934573817465075391826664309019448
4 100	104

Hints

Use the class **BigInteger**

Import the namespace "**java.math.BigInteger**":

```
import java.math.BigInteger;
```

Use the type **BigInteger** to read the numbers and calculate their sum:

```
BigInteger firstNumber = new BigInteger(sc.nextLine());
BigInteger secondNumber = new BigInteger(sc.nextLine());

BigInteger sum = firstNumber.add(secondNumber);
```

3. Big Factorial

You will receive **N** - the number in the range **[0 - 1000]**. Calculate the **Factorial** of **N** and print the result.

Examples

Input	Output
5	120
50	30414093201713378043612608166064768844377641568960512000000000000

II. Defining Simple Classes

4. Songs

Define a class **Song**, which holds the following information about songs: **Type List**, **Name,** and **Time**.

On the first line, you will receive the **number of songs - N**.

On the **next N-lines,** you will be receiving data in the following format: **"{typeList}_{name}_{time}".**

On the last line, you will receive "**Type List**" / "**all**". Print only the **names of the songs** which are from that **Type List** / **All songs**.

Examples

Input	Output
3 favourite_DownTown_3:14	DownTown Kiss

favourite_Kiss_4:16 favourite_Smooth Criminal_4:01 favourite	Smooth Criminal
4 favourite_DownTown_3:14 listenLater_Andalouse_3:24 favourite_In To The Night_3:58 favourite_Live It Up_3:48 listenLater	Andalouse
2 like_Replay_3:15 ban_Photoshop_3:48 all	Replay Photoshop

Solution

Define class Song with fields: **Type List**, **Name,** and **Time**:

```
public class Song {
    private String typeList;
    private String name;
    private String time;
}
```

Define getters and setters: use keys **ALT + INS** and generate Getter and Setter:

```
public String getTypeList() {
    return typeList;
}

public void setTypeList(String typeList) {
    this.typeList = typeList;
}

// TODO implement others methods
```

Read the input lines, make the collection, and store the data:

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int numSongs = Integer.parseInt(sc.nextLine());

    List<Song> songs = new ArrayList<>();

    for (int i = 0; i < numSongs; i++) {
        String[] data = sc.nextLine().split( regex: "_");

        String type = data[0];
        String name = data[1];
        String time = data[2];

        Song song = new Song();

        song.setTypeList(type);
        song.setName(name);
        song.setTime(time);

        songs.add(song);
    }
}
```

Finally, read your last line – **Type List** and print the result:

```
String typeList = sc.nextLine();

if (typeList.equals("all")) {
    for (Song song : songs) {
        System.out.println(song.getName());
    }
} else {
    for (Song song : songs) {
        if (song.getTypeList().equals(typeList)) {
            System.out.println(song.getName());
        }
    }
}
```

You can use **Stream API** to filter the collection:

```
List<Song> filterSong = songs.stream().filter(e -> e.getTypeList().equals(typeList))
        .collect(Collectors.toList());

for (Song song : filterSong) {
    System.out.println(song.getName());
}
```

5. Students

Define a class **Student**, which holds the following information about students: first name, last name, age, and hometown.

Read the list of students until you receive the "**end**" command. After that, you will receive a city name. Print only students which are from the given city, in the following format: **"{firstName} {lastName} is {age} years old"**.

Examples

Input	Output
John Smith 15 Sofia Peter Ivanov 14 Plovdiv Linda Bridge 16 Sofia Simon Stone 12 Varna end Sofia	John Smith is 15 years old Linda Bridge is 16 years old
Anthony Taylor 15 Chicago David Anderson 16 Washington Jack Lewis 14 Chicago David Lee 14 Chicago end Chicago	Anthony Taylor is 15 years old Jack Lewis is 14 years old David Lee is 14 years old

Solution

Define a class student with the following properties: **firstName**, **lastName**, **age,** and **city**:

```
public class Student {
    private String firstName;
    private String lastName;
    private int age;
    private String city;

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    // TODO: generate others Getter and Setter
}
```

Generate constructor in class Student: **ALT + INSERT**

```
public Student(String firstName, String lastName, int age, String city){
    this.firstName = firstName;
    this.lastName = lastName;
    this.age = age;
    this.city = city;
}
```

Read a list of students.

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    List<Student> students = new ArrayList<>();

    String line = sc.nextLine();

    while (!line.equals("end")) {
        String[] tokens = line.split( regex: " ");

        String firstName = tokens[0];
        String lastName = tokens[1];
        int age = Integer.parseInt(tokens[2]);
        String city = tokens[3];

        Student student = new Student(firstName, lastName, age, city);

        students.add(student);
        line = sc.nextLine();
    }
}
```

Read a city name and print only the students who are from the given city.

You can filter the students with the stream.

```
String filterCity = sc.nextLine();

for (Student student : students) {
    if(student.getCity().equals(filterCity)){
        System.out.printf("%s %s is %d years old\n", student.getFirstName(),
                student.getLastName(), student.getAge(), student.getCity());
    }
}
```

6. Students 2.0

Use the class from the previous problem. If you receive a student who already exists (first name and last name should be **unique**), overwrite the information.

Hints

Check if the given student already exists:

```
if(IsStudentExisting(students, firstName, lastName)){

}else{
    Student student =
            new Student(firstName, lastName, age, city);
    students.add(student);
}
```

```
private static boolean IsStudentExisting(List<Student> students, String firstName, String lastName) {
    for (Student student : students) {
        if(student.getFirstName().equals(firstName) && student.getLastName().equals(lastName)){
            return true;
        }
    }
    return false;
}
```

Overwrite the student information.

First, we have to find the existing student:

```
if(IsStudentExisting(students, firstName, lastName)){
    Student student = getStudent(students, firstName, lastName);
}
```

```
private static Student getStudent(List<Student> students, String firstName, String lastName) {
    Student existingStudent = null;

    for (Student student : students) {
        if(student.getFirstName().equals(firstName) && student.getLastName().equals(lastName)){
            existingStudent = student;
        }
    }
    return existingStudent;
}
```

Finally, we have to overwrite the information:

```
if(IsStudentExisting(students, firstName, lastName)){
    Student student = getStudent(students, firstName, lastName);

    student.setFirstName(firstName);
    student.setLastName(lastName);
    student.setAge(age);
    student.setCity(city);
}
```

We can use **Stream API** as well:

```
Student student = students
        .stream()
        .filter(s -> s.getFirstName().equals(firstName) && s.getLastName().equals(lastName))
        .findFirst()
        .orElse( other: null);

if (student == null) {
    Student studentToAdd = new Student();
    studentToAdd.setFirstName(firstName);
    studentToAdd.setLastName(lastName);
    studentToAdd.setAge(age);
    studentToAdd.setHometown(hometown);
    students.add(studentToAdd);
} else {
    student.setFirstName(firstName);
    student.setLastName(lastName);
    student.setAge(age);
    student.setHometown(hometown);
}
```

`findFirst` returns the first occurrence or **`null`**.

Examples

Input	Output
John Smith 15 Sofia John Smith 16 Sofia Linda Bridge 17 Sofia Simon Stone 12 Varna end Sofia	John Smith is 16 years old Linda Bridge is 17 years old
J S 3 S Peter Ivanov 14 P P J 104 S J P 61 S Simon Stone 12 Varna Simon Sone 12 Varna end Varna	Simon Stone is 12 years old Simon Sone is 12 years old