Java lab 2

The answers to the exercices below are due before *20 October 2008*. The procedure to send the exercices is:

1. Create a directory named `yourname-lab2`.
2. Copy all the `.java` source files to this directory.
3. Issue following command in a shell : `gtar czvf yourname-lab2.tar.gz yourname-lab2/` to create a tarball.
4. Send this tarball to `pablo@sifflez.org` with subject `yourname-lab2`.

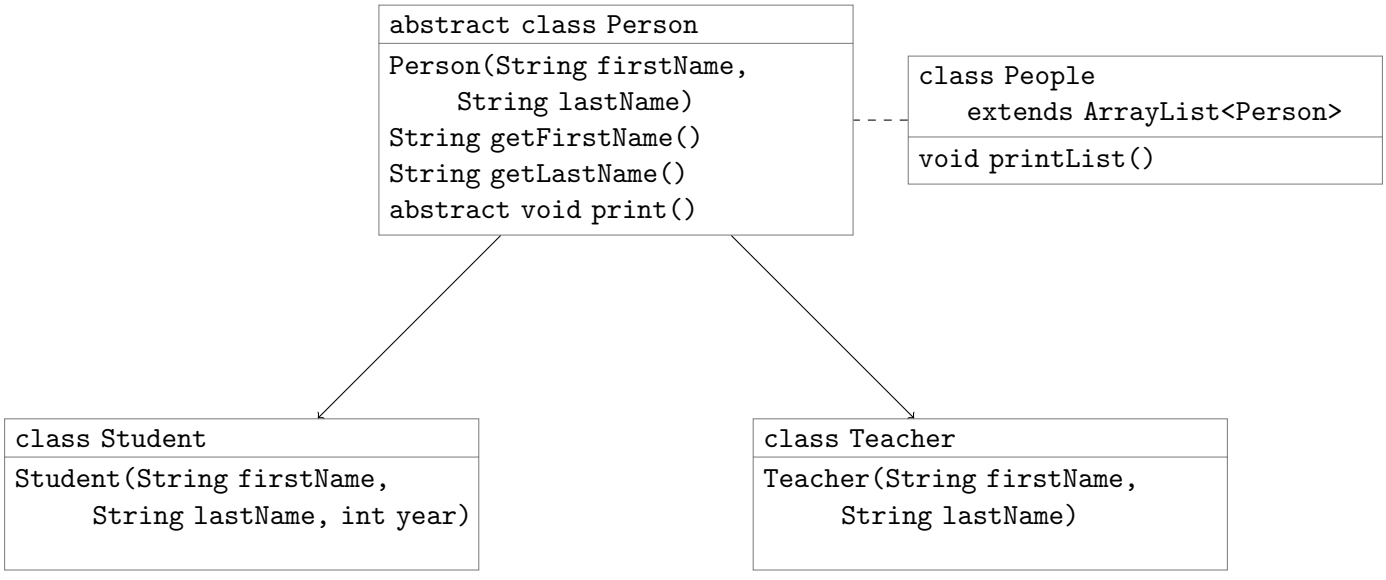
I) Goal

This lab is divided in two sections:

- In the first section you will write a java program that keeps a list of students and teachers, sorts this persons' list by name and prints it.
- In the second section you will add to this program the possibility to keep track of courses, each course will have a teacher and many students. You will add a class that tries to find a schedule so that no teacher has to teach two courses the same day, and no student has to follow two courses the same day.

II) People database

a) Classes hierarchy



Exercice 1 [**class Person**]
Write an abstract class Person that implements above methods. To store the first and last name use two private instance String variables.

Exercice 2 [**classes Teacher and Student**]
Write Teacher and Student classes that extend Person. Students should have in addition to a first and last name, a field for their current year in school. Teacher and Student class should implement method print which prints the person name, the person type (teacher or student) and in the case of students their current year.

Exercice 3 [**class People**]
Write a class People that extends java.util.ArrayList<Person> class. This class will keep the list of students and teachers. It will provide a method printList, which calls print() for each Person it contains. Add a main method to this class that adds some teachers and students to People, and call printList.

b) Sorting persons

Now we want to sort the persons in People, so that we can print them in alphabetical order. To achieve this we will use method java.util.Collections.sort() and interface java.util.Comparable. Read the documentation for interface Comparable at `http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Comparable.html` and for method sort at `http://java.sun.com/j2se/1.4.2/docs/api/java/util/Collections.html#sort(java.util.List)`.

Exercice 4 [**interface Comparable<Person>**]
We want to specify a total order for objects of type Person. To do this we will modify class Person so that it implements Comparable<Person>. We want to order persons using lexicographical order by their last name and if they have the same last name by their first name.

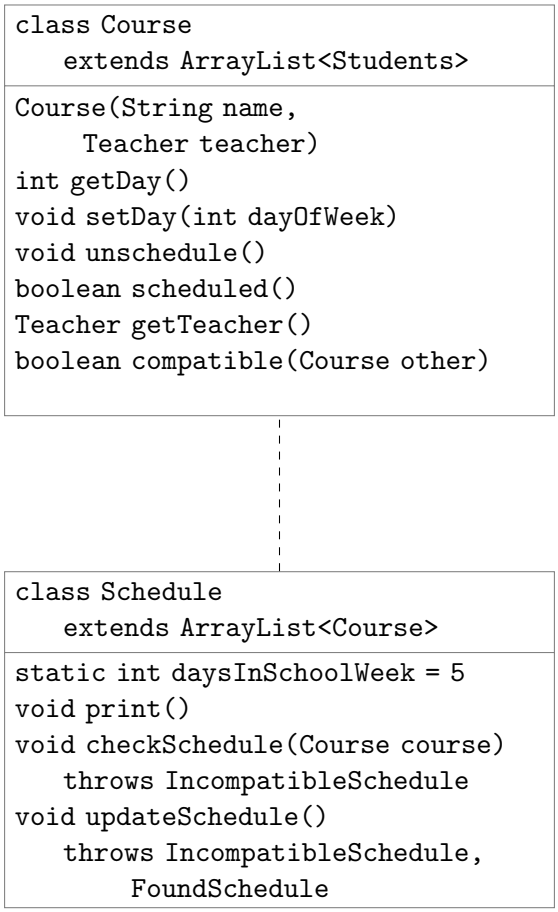
For example this would be a valid ordering:

```
first   last

Zoe     Amele
Clement Dupont
Cloe    Dupont
Eric    Ziegler
```

Exercice 5 [**Sort People**]
Add a method sort to class people that sorts its Persons, using static method java.util.Collections.sort(); Test your implementation by calling printList.

III) Courses scheduler



a) Courses

Exercice 6 [**Class Course**]
Create a class course with the members listed above. Each Course has a single teacher (provided in the constructor) and many students (added with method add inherited from ArrayList).

- Each course is either *not scheduled* or *scheduled for a specific day of the week, represented by an int.*
- Method setDay, allows to schedule a Course for a specific day.
- Method unschedule, set the Course to *not scheduled.*
- Method scheduled, returns true if the course is scheduled.
- Method compatible, check if current course is compatible with another course. ie, they have different teachers and they have no students in common.

b) Scheduler

Finally we are going to write a class Schedule that tries to find a compatible schedule for a list of courses.

Exercice 7 [**Exceptions**]
Create two new exception classes called IncompatibleSchedule and FoundSchedule.

Exercice 8 [**Schedule**]
Write a class Schedule like above but only implement method print(). The method print should print all the Courses in Schedule, and the day they are scheduled or `not scheduled` if they are not scheduled. Add a list of courses to a Schedule and call print(), all the courses should be marked as `not scheduled`.

Exercice 9 [**checkSchedule**]
Implement method checkSchedule. This method takes a Course:

- if all the others courses scheduled the same day are compatible, it does nothing.
- if one of them is not compatible, it throws exception IncompatibleSchedule.

Exercice 10 [**updateSchedule**]
Implement method updateSchedule, using method checkSchedule. This method tries to find a schedule for all the Courses, if this schedule is found it throws ScheduleFound, if no compatible schedule exists it throws IncompatibleSchedule.

- To implement this method you will use a backtracking algorithm.
  1. If all the courses are scheduled, throw ScheduleFound.
  2. If there still are unscheduled courses, take one of them.
  3. For each of the available days:
     (a) Try to schedule if for day d.
     (b) call updateSchedule() recursively to try to schedule the other courses.
     (c) if there is a schedule incompatibility, try another day.
  4. if there is no possible day on which to schedule the course, throw IncompatibleSchedule.

Exercice 11 [**Test**]
Add some students, teachers and courses and test the scheduling algorithm. To avoid adding a lot of courses, you can set daysInSchoolWeek to a small value.