Java Labs 1&2
The journey begins.

Antoine Tauvel, ENSEA

October 2021

1 Introduction

These basics exercises should be done in more or less 4 hours of your time. They lay down basics skills in object oriented programming, you must take them seriously. Once 80% of a group has succeeded in an exercice, the professor will give you a correction. This correction will also be available on moodle after the lecture. Do not wait for the correction to be done, these exercice are obvious and each one of you should be able to complete with few help.

There are no setup requirement for this two labs, but you really shoud use the same as in the upcoming project. We strongly recommend IntelliJ with an actual JDK (Java SE 16).

The exercise follows one another, you cannot pass one you haven't finished (it is frequent that the Class you define on exercise n is used in exercise $n+1$. If you experience any trouble, please ask your professor.

There are two type of work in this document :

This setup indicates programming question. Use your laptop.

This setup indicates reflection question. You really should use a paper and a pen.

2 Exercice 0 - homework

This exercice has to be down at home, before the first session.

- Program and test a simple "Hello world".
- Check that your code build and execute without problems.
- Modify your code so that each passing arguments are displayed on the console.

Hint : here's a simple Hello world. The arguments are inside the args variable. You can use a "for each" structure.

```
1  public class HelloWorld{
2
3  static public void main (String[] args){
4      System.out.println("Hello world");
5      }
6  }
```

Listing 1: Classic Hello world in Java

3 Exercise 1 - The Student Class

- Define a Student class that has two private String attributes : firstName and lastName. This class also have a static private attribute : totalNumberOfStudents.
- Define a constructor that takes these two arguments. The constructor updates the totalNumberOfStudents.
- Define getter methods accordingly to each attributes.
- Override the toString method
- Create a main method inside the Student Class that allows to test the creation and display of two Students.

4 Exercise 2 - Instance and Reference

Add a overriden "finalize" method to your Student class. This will cause a warning to appear because this method is deprecated since Java 9. A new object, Cleaner exist for replacement, but its usage is more complex. After this exercise, erase this method from the Student class.

```
@Override
    protected void finalize(){
        numberOfStudent--;
    }
```

Read the following code and answer these questions :

- On line 6, what does the parseInt method do ? To what object is it applied ?
- Compare the creation of an array in Java and in C (line 7). What nice feature do you note ?
- What is the lifespan of the "x" integer value. Compare to the C language.
- After line 12, how many references to "Students" objects exists ? How many instances of "Students" objects ?
- Same question after line 16.
- Why is the "sleep" command inside a try / catch block ?

Keeping in mind that System.gc() is more or less a call to the garbage collector (don't do this, the VM knows better when to garbage collect), what will be displayed on the console, if we call the program via this command line :
`$ java demoStudent 10`

Try the program

```java
import java.util.concurrent.TimeUnit;

public class demoStudents {

    public static void main (String args[]){
        Integer a= Integer.parseInt(args[0]);
        Student[] list = new Student[a];

        for (int x=0;x<10;x++){
            list[x] = new Student("Antoine","Tauvel");
        }

        System.out.println("Actual number of Students : " +
            list[0].getNumberOfStudent());

        list[1]=list[0];
        System.gc();
        try{TimeUnit.SECONDS.sleep(10);}
        catch (Exception e){
            System.out.println("Won\'t go to sleep !");
        }
        System.out.println("Actual number of Students : " +
            list[0].getNumberOfStudent());

    }
}
```

Listing 2: Instance and reference code : garbage collection

5 Exercise 3 - Custom exception and regular expression

On this exercise, we'll implement a Group Student whose name must match those of the ENSEA.

To check our group name, you'll need *regular expression*, also known as *regex*. Regex is a very powerfull tool that exist in every language. Here is an example in Java :

```java
import  java.util.regex.Pattern;
...
boolean b = Pattern.matches("[1-2]G[1-3]",name);
```

These will return True if name is 1G2 but false if name is 0G3 or 1g2...

To generate a new Exception the method's signature has to comply with this declaration :

```
public StudentGroup(String name) throws Exception{
    //... to be completed
    if (!Pattern.matches("[1-2]G[1-3]TD[1-3]TP[1-6]",name)
        {throw new Exception(name +" is not a real ENSEA name");}
    //...
    }
}
```

- Create a new Class named StudentGroup.
- At this point, this class includes just a name, and a static count of the number of group.
- The StudentGroup name must be of the form "xGyTDzTPw". If the name is not well formed, the constructor does not construct the StudentGroup but throws a new Exception. x must be between 1 and 2, y between 1 and 3, z between 1 and 4, and w between 1 and 8. Because we are nice, we don't made you check that :

  $$2.z \leq w \leq 2.z + 1$$

  .
- Override the toString method to display the group name.

Now test your code with this particular main :

```
public static void main (String[] args)
    {
        Group a = null,b=null,c= null;
        try{a = new Group("1G1TD1TP1");
            b = new Group("3G1TD1TP1");
            c = new Group("1G1TD1TP2");
            }
        catch (Exception e)
        {
            e.printStackTrace();
        }

        System.out.println(a);
        System.out.println(b);
        System.out.println(c);

    }
```

Listing 3: Try / Catch block for Group constructors.

- Try to guess the Console output before compiling your sources.
- Why are the Group reference definition outside the try / catch block ?
- Modify the code so that the c Group is also created.

6 Exercise 4 - Linked list / Array list manipulation

Please read the Javadoc of ArrayList before this session.

Two implementation of list coexist in Java : ArrayList and LinkedList. One use a dynamic size array, the other one a double link list. Both have the same access methods : they are *polymorphic*, from the list Interface.

Consequently, if you have many creation / suppression of items in your list, you'll prefer LinkedList objects, if you have many access to middle - queue objects, you'll prefer ArrayList objects.

To create an empty LinkedList or an empty ArrayList, you'll use those constructors :

```
LinkedList<Student> llist=new LinkedList<Student>();
ArrayList<Student> alist=new ArrayList<Student>();
```

- Add an ArrayList in your Group class. Initialize it inside the constructor to an empty list.
- Create a `public void addStudent(Student s);` method that add a Student to the Group.
- Create a `public void displayPresenceList(),` method that list all of the Student inside the group.
- Modify the `public String toString(void);` so that the number of Students appear.
- Test your functionality.

7 exercice 5 - Find function

Java offers an Interface name "equals" available on every Object. It's been instanciated for every common object[1]. So, in order to compare two string, we can just use something like this :

```
String name;
if (name.equals("Bonjour")){...}
```

Inside the Group class, create a `public String getName()` method.
Inside the Student class, create getter method for the first name and the last name of the student.
Inside the Promotion class, create a `Group findGroup(String name);` method that compare the every Group name in the Promotion to the name variable. If the method cannot find any Group Name, return the `null` value.
Inside the Promotion class, create a `Student findStudent(Group g, String first, String last);` method that compare inside the Group g every Student. The Student must match first name and last name. If the method cannot find any Student, return the `null` value. If listGroup is null, this method also return `null`.
Test both of these function with the following code number 4 on next page.

[1] Of course, we can create an override method to create our own comparaison norm, but that won't be necessary with Strings

```
    public static void main (String[] args)
    {
        Promotion p = new Promotion();
        Group a=null;
        try{a = new Group("1G1TD1TP1");
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }

        p.addGroup(a);

        a.addStudent(new Student("Antoine", "Tauvel"));
        a.addStudent(new Student("Sylvain", "Reynal"));
        a.addStudent(new Student("Christophe", "Barès"));

        p.presenceList();

        System.out.println(p.findGroup("1G1TD1TP1"));
        System.out.println(p.findGroup("1G1TD1TP2"));
        System.out.println(p.findStudent(p.findGroup("1G1TD1TP1"),"Antoine","Tauvel"));
        System.out.println(p.findStudent(p.findGroup("1G1TD1TP1"),"Antoine","Reynal"));
    }
```

Listing 4: Main test for String comparaison

8 exercise 5 - File manipulation

We arrive now at the "hard" part. We want to read a csv file in order to fill our "promotion" object. A csv (comma separated value) file can be viewed as the ancestor of Excel and other sheet manipulating software. They are commonly used in Big Data to exchange information other different systems. In a csv file, every line is a new entry. Fields are separated by a comma. Here is few lines from our .csv file (the complete file is made of 213 lines) :
ABBAS,Marwan,2G3TD2TP3,1A
AFOU,Amine,2G1TD1TP2,1B
AISSAOUI,Yannis,2G1TD2TP3,1B
Of course, this correspond to :

ABBAS	Marwan	2G3TD2TP3	1A
AFOU	Amine	2G1TD1TP2	1B
AISSAOUI	Yannis	2G1TD2TP3	1B

Table 1: Extrait du fichier csv

To open and read a text file, you'll need two classes : FileInputStream and BufferedReader.

Read the javaDoc about this two classes. We particularly need to use the readLine method of the FileReader class.
Read also the javaDoc about String manipulation. We'll need the `split` method that return an array containing each sub-string.
First name is betweeen 0 and the first comma.
Last name is between the first comma+1 and the second comma.
Group name is between the second comma+1 and the third comma.

Create a `public void fillPromotion(String fileName);` method. This method opens via a try / catch block a new FileReader on the *fileName* file.
For every line in the file, check whether the group exist. If it doesn't exist, create a new Group with this name.
Check if the Student is in the Group. If not, create the student.
For the test, use p.presenceList(); to display every students.
This exercise is a little bit harder, you'll find a solution beneath, but try to find your own solution.

This exercise is not quite realistic. What flows can you see in this model ? Can you summarize how to bypass it ?

```java
public void fillPromotion(String fileName){
        try {
            FileReader fr = new FileReader(fileName);
            BufferedReader br = new BufferedReader(fr);
            String li = br.readLine();
            while (li!=null)
            {
                Integer firstComma=li.indexOf(',');
                String lastName=li.substring(0,firstComma);
                Integer secondComma=li.indexOf(',',firstComma+1);
                String firstName=li.substring(firstComma+1,secondComma);
                Integer thirdComma=li.indexOf(',',secondComma+1);
                String groupName=li.substring(secondComma+1,thirdComma);

                Group g=this.findGroup(groupName);
                if (g==null) {
                    g=new Group(groupName);
                    this.addGroup(g);
                }

                Student s=this.findStudent(g,firstName,lastName);
                if (s==null){
                    s=new Student(firstName,lastName);
                    g.addStudent(s);
                }

                li=br.readLine();
            }
        }
```

Listing 5: A solution to the fillPromotion problem, not the best one (reading of a csv file).

9 exercise 7 - Inheritance and polymorphism

In this exercise, we'll imagine that exchange students and local students don't use the same validation rules at ENSEA. Local students must validate each one of the modules. Exchange students must have an average grade of 10 or above[2].

In this exercise, we'll need to build random number through this code :

```java
Random alea = new Random();
for (int i = 0; i < 6; i++) {
```

[2]Please do not speculate on this, it's just an exercise

```
    grades[i] = Math.floor((12 + alea.nextGaussian() * 3) * 10) / 10;
    grades[i]=grades[i]>20?20:grades[i];
}
```

Read the javaDoc about the java.util.Random class and Math.floor function. What are the caracteristics of each number generated ?
Comment the ternary operator used in this code.

Change the "Student" class to an abstract class. Why can't any of your former code work ?
Inside the Student class, add a `protected boolean yearValidation` and a `protected Integer grades[]`.
Inside the constructor, initialise the yearValidated value to false and the grades to random number using the above code.
Inside the Student class, create an `abstract public void checkResult();` method.

Creates two public classes, LocalStudent and ExchangeStudent both of which extends Student.
Implements in each class a constructor that just calls the super class constructor, and the checkResult function according to each kind of Students.
Modify the Promotion's fillPromotion method to create both localStudents and exchangeSudents. You can either use a random number or use the fourth indication on the csv file. We'll need for this exercice approximatly 20% of exchange Students.
Finally, use the following code listing to test both of your class. Pay great attention to line 10, which illustrate polymorphism : we apply the same method (checkResult) to every object defined as a Student, whether it is really a LocalStudent or an ExchangeStudent.

```java
public static void main (String[] args)
 {
     Promotion p = new Promotion();
     p.fillPromotion("./src/liste2_2020_2021.csv");
     p.presenceList();

     System.out.println("Failed Students : ");
     for(Group g : p.listGroup){
         for(Student s : g.getList()){
             s.checkResult();
             if (!s.getValidatedYear()){
                 System.out.println (g.getName()+" "+s);
             }
         }
     }

 }
```

Listing 6: Usage of polymorphism

10 Conclusion

This end your first journey through basic Java programs. All this program are in console mode, we'll look at graphical user interface on our next labs, which will have the form of a small project.

11 List of listing

List of source codes