About the exam

You may NOT use any IDE while you are solving the exam.

You may [download JUnit's libraries required for testing by clicking here]

And the test files here!

- I suggest that you create your solution for the exercise in the order it appears in the text.

- Run the tests like this. (If you're using Linux, use `:` instead of `;`)

```
javac -cp .;junit-4.13.jar tests/EmployerTest.java
java -cp .;junit-4.13.jar;hamcrest-core-1.3.jar org.junit.runner.JUnitCore te
sts.EmployerTest
```

Also, make sure that you create good quality code or you will not gett a full mark.

Code should compile without errors!

Submitting your solution:

- During submission, you have to create a single **ZIP** file. Canvas does not accept any other archive formats.
- Make sure you include all `.java` source files that you've created. All other files (`.class` files, `.jar` files for the tester) are not necessary.

Exercise:

- Create an `exam.findsuitablejob.Contract` class representing a contract, has the following fields:
  - `degree` of type String , `numberOfEmployees` of type int , `contractPeriod` of type int.
  - It also has a static field: `goalSalary`. this should set to 4000 initially.
- Make all fields of `Contract` class accessible to the descendants of `Contract`, but not to any other class.
- Create getters for all of them, and a setter only for `goalSalary`.
- `Contract` has two constructors:
  1. A constructor that takes the `degree`, `numberOfEmployees` and `contractPeriod` as parameters and sets the corresponding fields. In it, make sure that the following conditions hold:
     - `contractPeriod` is at least 6 and not more than 12 months, (for this create two constants(outside the constructor) that hold values 6 and 12 of type 'int'.
     - `numberOfEmployees` is at least 15 and not more than 40 .
     - If either one is incorrect, throw an `IllegalArgumentException`.
     - To make it easier to check those conditions, create a static method `isBetween(number, min, max)` that returns whether `number` is between `min` and `max`.
  2. A constructor that takes no parameters. It calls the previously defined constructor with the following values:
     - `degree`: "Master"
     - `numberOfEmployees` : 20
     - `contractPeriod` : 12
- Override `toString()` method and give it a suitable implementation.

Create an enum `exam.findsuitablejob.Position` which includes the four positions: FREELANCER, UNIVERSITY,HOSPITAL and ITCOMPANY.

- Each of these enum values have one `int` that indicate the associated `salary`, as following: `FREELANCER` has 7000, 'UNIVERSITY' has 4000 , 'HOSPITAL' has 5000 and 'ITCOMPANY' has 2700.
- Create getter for `salary`.
- Set a constructor for `Position` enum that takes `salary` as argument and set its value.
- Create the interface `exam.findsuitablejob.Employable` with the method `getHiredAt()` that takes a `Position` and the method `getSalaryIncrease()` that takes `increaseRate` of type `double` as a parameter.

Create the class `exam.findsuitablejob.Employee` that implements `Employable` with fields: `employeeName` of type `String` and `salary` of type `int` and set it to 0.

The employee's constructor has `private` access modifier, it takes its `employeeName` and `salary` and set their values. both fields have getters, but no setters.
the constructor should throw an `IllegalArgumentException` if `employeeName` is null.

- Write a static method `make` of type `Employee` which takes `data` of type `String` as argument. Data is in the form `name,salary`. If the line does not have this structure or if the salary is smaller than zero, the method returns null. Otherwise it convert the salary to 'int' and Catch the proper Exception (`Integer.parseInt ()`returns a `java.lang.NumberFormatException` exception). The 'make' method is responsible to return an Employee initialized with the name and salary you just extracted from the `data`.
- Override `equals()` which return equality of 'Employee's object depending on 'name' and 'salary'.
- Override `hashcode()` and 'toString()' methods.
- The implementation of `getHiredAt()` method should increase the 'salary' by the enum value. while the implementation of `getSalaryIncrease()` should throw an `IllegalArgumentException` if the `increaseRate` is less than (1.1), anyway, this method should return the result of multiplying given argument value by the `salary`.
- The class `Employee` ALSO implements `Comparable<Employee>` interfaces , give an implementation to `compareTo()` method in such way that it make the comparision according to the `salary` value.

Create `exam.findsuitablejob.EmpAgency`, a child class of `Contract`.

- It has a field called `employees`, a list of `Employee' objects.
  Its only constructor takes `degree`, `contractPeriod` and an arbitrary number of `Employee`s and stores them in its fields.
- Create method `load()` which takes `filePath` of type `String` and return an Array of 'Employee' objects. It will read from an input file line by line, each line will be passed to `Employee.make()` to create a new employee. If a line is 'null' you should skip it. Make sure to catch 'IOException' since you are reading a file.
- Create method `hire()` that takes a 'Position' as argument and it go through 'employees' list (using Iterator). Then the list of hired employees is sorted in ascending order depending on 'salary' value.
  If an employee's salary after being hired, is equal or less than the 'goalSalary', then hire him/her with 'getHiredAt()' method and remove the employee from `employees` and update `numberOfEmployees`. Remember to use `Iterator` to achieve that.
- Create method `getMaxIncrease()` which takes an argument of type 'double' and throw `IllegalStateException` if the number of employees is 0; otherwise it will calculate the increase of the salary of each employee in `employees` list depending on the given argument, and it will return maximum increase value among them ('maxIncrease' of type double).
- Override `toString()` in `EmpAgency` which prints employees.