Assignment 2

General instructions (Read carefully!)

Programming component

- You are provided some starter code that you should fill in as requested. Add your code only where you are instructed to do so. You can add some helper methods. Do not modify the code in any other way and in particular, do not change the methods or constructors that are already given to you, do not import extra code and do not touch the method headers. The format that you see on the provided code is the only format accepted for programming questions. **Any failure to comply with these rules will result in an automatic 0.**

- Public tests cases are available on ed-Lessons. You can run them on your code at any time. If your code fails those tests, it means that there is a mistake somewhere. Even if your code passes those tests, it may still contain some errors. We will grade your code with a more challenging, private set of test cases. We therefore highly encourage you to modify that tester class, expand it and share it with other students on the discussion board. Do not include it in your submission.

- Your code should be properly commented and indented.

- **Do not change or alter the name of the files you must submit, or the method headers in these files**. Files with the wrong name will not be graded. Make sure you are not changing file names by duplicating them. For example, main (2).java will not be graded.

- **Do not add any package or import statement that is not already provided**

- Please submit only the individual files requested.

- **You will automatically get 0 if the files you submitted on ed-Lessons do not compile, since you can ensure yourself that they do. Note that public test cases do not cover every situation and your code may crash when tested on a method**

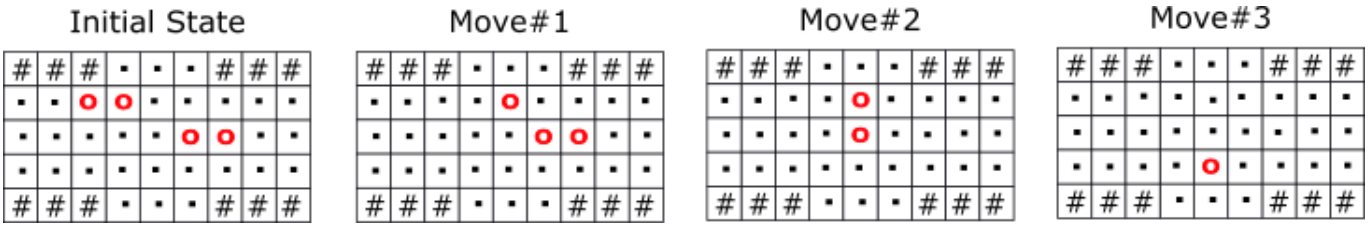
that is not checked by the public tests. This is why you need to add your own test cases and compile and run your code from command line on linux.

Homework

Exercise 1. *Complete Search*

Let's play a fun game :). We provide a two-dimensional board with some holes on it, where each hole can contain a small ball. During the game, and in each turn, you will be allowed to jump (horizontally or vertically) over an adjacent ball into the empty hole next to the jumped ball in line with it. Once the jump is performed, the ball (which was jumped over) is then removed from the board. The idea of the game is to apply the minimum number of moves (i.e., jumps) to end in a state with a minimum number of balls.

Let's analyze one example to make sure that the game is clear. Figure 1 shows the initial state of the game, and a sequence of moves (i.e., horizontal or vertical jumps over adjacent balls) that allows for a state (i.e., the last state) with the minimum number of balls in the board. In particular, we performed three moves to end with one ball on the board. Please notice that in this representation, "." (a period) denotes an empty hole, and "o" (small vowel o) a ball, and "#" (number sign) indicates a part of the board without a hole. For this question, you can safely assume that all the boards have always the same shape.



For this question, your task is to develop an algorithm (i.e., a complete search one) that determines the minimum number of balls that remains in the board after applying (recursively) the above-mentioned movement. You will also need to report the minimum number of moves required to reach that number of balls. You will need to submit your `A2_Q1.java` source file to the `Assignment 2 => Q1 - Complete Search` lesson in Ed-Lessons. Please complete the function `game` which receive a 2-D array of Strings representing a board (as the one shown in Figure 1). The function `game` must return an array of integers (`int[]`) of two positions, where the first position (i.e., index 0) reports the minimum number of balls that remains in the board and the second position (i.e., index 1) reports the minimum number of moves performed to reach that number of balls.

Exercise 2. *Dynamic Programming*

Dr. Robert Bruce Banner (yes, go and google the name in case you do not know who I am talking about :p) contacted me in order to help him with his weight training program. In particular, he is interested in creating a (training) program that will allow him to lift $1000kg$. Dr Banner has several (positive integers less or equal than 1000) weight plates, possibly of different weights,

and its goal is to add some of the plates to a bar so that it can train with a weight as close as possible to $1000kg$. Given the (possible) case where two numbers are equally close to 1000 (e.g., 995 and 1005), Dr Banner will pick the greater one (in this case 1005).

Let's see an example to make sure that the question is clear. Imagine that Dr Banner has 4 plates with the following weights $[900, 500, 498, 4]$, then your program must return as answer 1002, because this is the combined weight closest to $1000kg$. In particular, the combined weight is composed by the three last plates $(500 + 498 + 4 = 1002)$. In this example, you can find another combination (i.e., 998) that is equally close to $1000kg$. This combination is composed by the second and third plates $(500 + 498 = 998)$. Your algorithm must return 1002 as the answer given that Dr Banner will pick the greater one between the two equally close solutions.

Dr Banner is an excellent coder and he is really good in Dynamic Programming; however, he would be busy in a trip to planet Titan (yes, go and google the planet if you do not know what I am talking about) and he asked me for help. I told him that I had great Comp251 students that will be more than happy to help him. For this question, you will need to complete the function `weight`, which receives as a parameter a list of positive integers (where each integer is less than or equal than 1000), denoting the weight of each plate. You can safely assume that the length of the list is between 1 and 1000. The function must return one integer, the combined weight closest to 1000.

HINT: Please code a nice program, believe me, you do not want to make Dr Banner mad :P

Exercise 3. *Greedy*

In this exercise, you will plan your homework with a greedy algorithm. The input is a list of homeworks defined by two arrays: deadlines and weights (the relative importance of the homework towards your final grade). These arrays have the same size and they contain integers between 1 and 100. The index of each entry in the arrays represents a single homework, for example, `Homework 2` is defined as a homework with deadline `deadlines[2]` and weight `weights[2]`. Each homework takes exactly one hour to complete.

Your task is to output a `homeworkPlan`: an array of length equal to the last deadline. Each entry in the array represents a one-hour timeslot, starting at 0 and ending at 'last deadline - 1'. For each time slot, `homeworkPlan` indicates the homework which you plan to do during that slot. You can only complete a single homework in one 1-hour slot. The homeworks are due at the beginning of a time slot, in other words if an assignment's deadline is `x`, then the last time slot when you can do it is `x - 1`. For example, if the homework is due at t=14, then you can complete it before or during the slot t=13. If your solution plans to do `Homework 2` first, then you should have `homeworkPlan[0]=2` in the output. Note that sometimes you will be given too much homework to complete in time, and that is okay.

Your homework plan should maximize the sum of the weights of completed assignments (regardless of the order on which they are completed). Please notice that an "-1" value is allowed in between the homework plan.

Let's see an example to make sure that the task is clear. Imagine the following deadlines and weights.

```
   int[] weights = new int[] {23, 60, 14, 25, 7};
int[] deadlines = new int[] {3, 1, 2, 1, 3};
```

where homework 0 has a weight of 23 and a deadline of 3, homework 1 has a weight of 60 and a deadline of 1, and so on so on. Then, the solution of your algorithm is an array of length equal to the last deadline (in this case 3). In particular, the solution is `{1,2,0}`, meaning that the first day you will do homework 1 (with a weight of 60), second day you will do homework 2 (with a weight of 14) and third day you will do homework 0 (with a weight o 23). Please notice that this array maximize the sum of weights of the completed assignments. Notice that you did not do assignment 4 [even if it has a good weight] because its deadline is 1 and you made the decision to do assignment 1 [which have a better weight] in that slot.

To organize your schedule, we give you a class `HW_Sched.java`, which defines an `Assignment` object, with a number (its index in the input array), a weight and a deadline.

The input arrays are unsorted. As part of the greedy algorithm, the template we provide sorts the homeworks using Java's `Collections.sort()`. This sort function uses Java's `compare()` method, which takes two objects as input, compares them, and outputs the order they should appear in. The template will ask you to override this `compare()` method, which will alter the way in which Assignments will be ordered. You have to determine what comparison criterion you want to use. Given two assignments A1 and A2, the method should output:

- 0, if the two items are equivalent
- 1, if a1 should appear after a2 in the sorted list
- -1, if a2 should appear after a1 in the sorted list

The `compare` method (called by `Collections.sort()`) should be the only tool you use to re-organize lists and arrays in this problem. You will then implement the rest of the `SelectAssignments()` method.

Exercise 4. ***Divide and Conquer***

During our Comp251 lectures, we defined the Fibonacci sequence as follows:
$fib_1 = 1$
$fib_2 = 2$
$fib_n = fib_{n-2} + fib_{n-1}$
Generating the following sequence $1, 1, 2, 3, 5, 8, 13, 21 \ldots$

We can also define the Fibonacci sequence using letters. For example, if:
$fib_1 = X$
$fib_2 = Y$
$fib_n = fib_{n-2} + fib_{n-1}$
we will get now the sequence $X, Y, XY, YXY, XYYXY, YXYXYYXY, XYYXYYXYXYYXY \ldots$, by considering "+" as string concatenation,

For this question, you will need to implement a divide and conquer algorithm than given N and K as parameters, your function returns the K-th letter in the N-th string in the fibonacci sequence. For example, if $N = 7$ and $K = 7$, your algorithm must return X (please notice that the N=7 string in the sequence is "XYYXYY**X**YXYYXY", and the K=7 letter in that string is "X", which is bold). You can safely assume that K is at most the length of the N^{th} string

HINT 1: Please check if there is a relation between the two sequences shown in this question
HINT 2: The integers could be very big, then please use BIGINTEGERS
HINT 3: Please do not even try concatenating the strings, this approach will never succeed

What To Submit?

Attached to this assignment are java template files. You have to submit only this java files. Please DO NOT zip (or rar) your files, and do not submit any other files.