IE Algorithms & Programming
SCAN - S2
March 2018

Total duration : 1.30 hours
Authorized documents : None

- All the exercises are independent of each other, the grading scale is approximate and the whole exam sheet is on 4 pages.
- A program which is badly indented, badly commented or with inappropriate names of variables can lead to a subtraction of up to 1 point.

Exercice 1 : Code correction (5pt)

Given the following classes `Test.java` and `Container.java`.

(1.1) List the 10 coding errors contained in the following files (specifying the file and the line) and suggest a correction for each of them.

```
1  public class Container {
2      public static void main (String[] args) {
3          int capacity ;
4          double volume ;
5      }
6
7      public Container(int c) {
8          c = capacity;
9          volume = 0;
10     }
11
12     public void fill (double v){
13         if(volume+v<=capacity){
14             volume = volume + v;
15         else
16             volume = capacity;
17     }
18
19     public void empty(){
20         volume = 0;
21         return volume;
22     }
23
24     public boolean isFull(){
25         if(volume == capacity)
26             return true;
27         else
28             return false;
29     }
30
31     public void transfer(Container o){
32         if(volume+volume <= this.capacity){
33             this.volume = this.volume + o.volume;
34             o.volume = 0;
35         } else {
36             o.volume = o.volume - (this.capacity - this.volume);
37             this.volume = this.capacity;
38         }
39     }
40 }
```

```java
public class Test {
    public static void main (String[] args) {
        Container o1 = new Container(10, 5);
        while(isFull() == false){
            o1.fill(1.5);
            System.out.println("filling␣"+o1.volume+"␣/␣"+o1.capacity);
        }
        o1.empty;
        System.out.println("FINISHED␣"+o1.volume+"␣/␣"+o1.capacity);

        Container o2 = new Container(20)
        o2.fill(15);
        o1.transfer(15);
        System.out.println("o1␣:␣filling␣"+o1.volume+"␣/␣"+o1.capacity);
        System.out.println("o2␣:␣filling␣"+o2.volume+"␣/␣"+o2.capacity);
    }
}
```

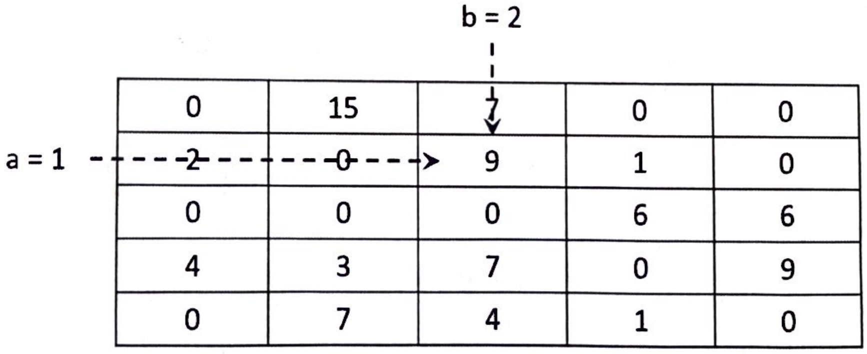
Exercice 2 : 2D array (10pts)

The Velo'V is a bike sharing system of the Lyon metropolis that exists since 2005. It is composed of a network of 300 stations covering the city. At any time, a person can take a bike at a given station and leave it at another one once arrived to the destination. In order to understand how the bikes are used and to further adapt the available network, it is necessary to analyze the information about the rides done by users between various stations.

All the rides are recorded automatically in a database. In this exercise, this database is represented by a 2D array named `travel`, that contains all the rides of the previous month :

- Line number `i` : travel identifier,
- `travel[i][0]` $= a$: station of departure a,
- `travel[i][1]` $= b$: station of arrival b.

To understand the use, a transformation is applied : this array is converted into an *adjacency matrix* between stations (figure 1), named `adj`, where `adj[a][b]` represents the number of travels from station a to station b. We can note that this matrix can contain a large number of zeroes as there may be no rides between two given stations.



0	15	7	0	0
2	0	9	1	0
0	0	0	6	6
4	3	7	0	9
0	7	4	1	0

FIGURE 1 – Example of an adjacency matrix, where `adj[1][2]` $= 9$.

Based on this adjacency matrix, one can provide an analysis allowing to determine the largest number of rides between two stations, the links between stations, etc.

(2.1) Write the `transform` *method that computes the adjacency matrix from the rides list* `travel`*.*

```
/**
 * transform the list  of travel into an adjacency matrix
 * @param travel : list of travels
 * @param nb : number of stations in the network
 * @return the adjacency matrix
 */
public static int[][] transform (int[][] travel, int nb)
```

(2.2) *Write the `computeNbTravel` method that counts the number of rides having the station `a` as a departure point, by using the matrix `adj`.*

```
/**
 * count the number of rides starting at station a
 * @param adj : the adjacency matrix
 * @param a : identifier of the station
 * @return the number of rides having as departure the station a
 */
public static int computeNbTravel (int[][] adj, int a)
```

(2.3) *Write the `maxMatrix` method that searches the coordinates `(i,j)` of the maximum value stored in the adjacency matrix `adj`.*

Remark : a particular attention should be paid to the signature of this method.

We say that the station b is a neighbor of the station a if there exists at least one ride that went from station a to station b.

(2.4) *Write a method `neighbors` that lists all the neighbor stations of the station `a`.*

We say that there is a pendulous link between two stations a and b if there was at least one ride going from a to b **AND** at least one ride going from b to a.

(2.5) *Write the `pendulous` method that verifies if two stations `a` and `b` have a pendulous link.*

We want to know if users take and return the Velo'V bike at the same station.

(2.6) *Write a method `ride` to verify that there was at least one bike ride during the last month.*

Exercice 3 : Object oriented programming (5pt)

We want to develop a Queue class in order to handle problems of waiting queues. In a queue, data can enter (*enqueue*), wait and exit (*dequeue*) in the same order they entered. It is a concept similar to the waiting queue that we can find in a supermarket or in a bank.

We use an array to store elements given by positive integers. The attributes, constructor and methods enter and exit of the Queue are given as follows :

```
public class Queue{
    // Attributes declaration
    private int[] tab;
    // position of the first element
    private int first;
    // position of the last element
    private int last;
```

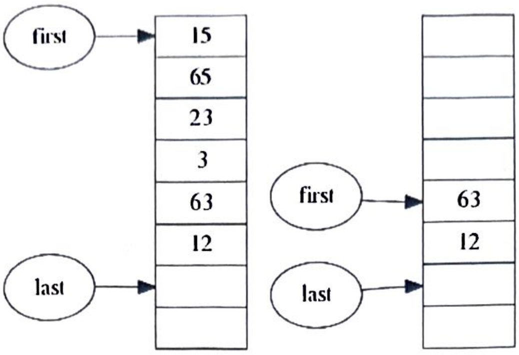


FIGURE 2 – A queue

```
    // Constructor taking as a parameter the maximum size of the array
    public Queue( int size ){
        tab = new int[size];
        last = 0;
        first = 0;
    }

    // Enter an element in the Queue then return true. If the Queue is empty, return false.
    public boolean enter( int x ){
        if (last==tab.length)
            return false;
        tab[last] = x;
        last++;
        return true;
    }

    // Exit an element from the Queue and return it. If the Queue is empty return -1.
    public int exit() {
        if (first==last)
            return -1;
        int x = tab[first];
        first++;
        return x;
    }
}
```

(3.1) *In the class `TestQueue.java`, write the program that creates a Queue `f1` of size 10 then adds 2 values to it.*

(3.2) *Complete this program to fill a Queue with random numbers taking values between 1 and 100 until the queue is full.*

(3.3) *Add a method `public int size()` to the class `Queue` allowing to return the number of elements in the queue.*

(3.4) *In the class `TestQueue.java`, create a Queue `f2` of the same size as `f1` and transfer all the elements of `f1` until `f1` and `f2` are of identical size.*

(3.5) *In the class `TestQueue.java`, we want the instruction `System.out.println(f1);` to display the contents of the Queue `f1`. What solution would you suggest for this problem?*