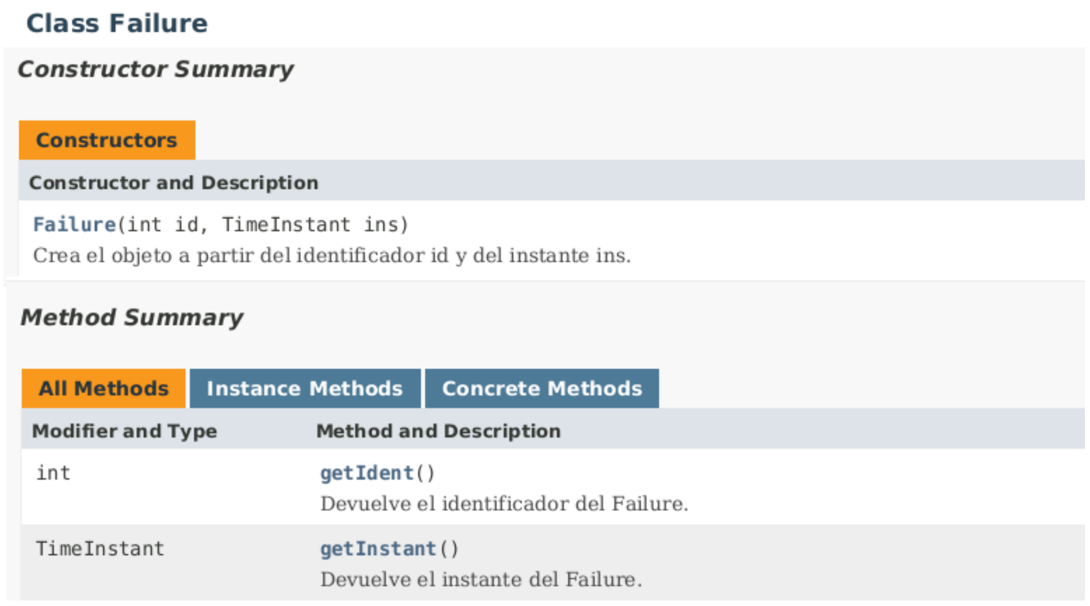
IIP – ETSINF – Recovery of the second mid term exam
January 18th, 2019 – duration 2 hours and 30 minutes
Notice: This exam is evaluated over 10 points, but its weight in the final grade of IIP is **3,6 points**

NAME:	GROUP:

1. $\boxed{\text{6 points}}$ We have a system for distributing Internet and TV in a town, and we need to study the failures in the service provided to citizens within a day. The class `Failure` is available for creating and manipulating objects representing notifications (failures and errors reported by clients). This objects have an identifier of the notification and a time-stamp (only hours and minutes) when the notification was reported. The class `Failure` have the following methods:



Class Failure

Constructor Summary

Constructors

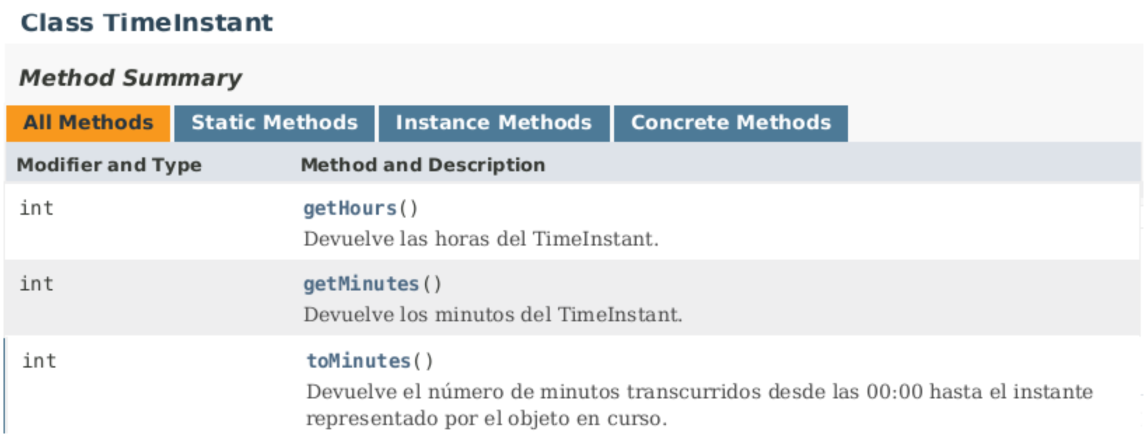
Constructor and Description
`Failure(int id, TimeInstant ins)` Crea el objeto a partir del identificador id y del instante ins.

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
`int`	`getIdent()` Devuelve el identificador del Failure.
`TimeInstant`	`getInstant()` Devuelve el instante del Failure.

The time instants are objects of the class `TimeInstant`, a class you already know, but the following figure describe the methods you need:



Class TimeInstant

Method Summary

All Methods | Static Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
`int`	`getHours()` Devuelve las horas del TimeInstant.
`int`	`getMinutes()` Devuelve los minutos del TimeInstant.
`int`	`toMinutes()` Devuelve el número de minutos transcurridos desde las 00:00 hasta el instante representado por el objeto en curso.

You have to write the code of a Java class for studying the distribution of failures notified within a day.

To do: Write the data class `FailureReport` for representing all the failures that can occur in one town or city within a day. The following attributes and methods must be implemented:

a) (0,5 points) Attributes:
   - `MAX`: public and constant class attribute of type integer for indicating the maximum number of failures that can be stored in an object of the class `FailureReport`. Its value must be 100.
   - `reported`: private attribute (instance variable), an array for storing `Failures` with an maximum capacity equal to `MAX`. The reported failures will be stored in this array and must be stored consecutively starting at position 0. The order must be the same in that failures are stored, it is not necessary to maintain the chronological order.
   - `nFailures`: private attribute (instance variable), an integer indicating the current number of failures stored in the array.

b) (0,5 points) A default constructor (with no parameters) for creating the array and initialising the `nFailures`.

c) (0,75 points) A method with the following profile:

   `public boolean add(Failure f)`

   for adding `f` to the `FailureReport` and returns a `boolean` indicating if the adding operation was successful. `f` must be stored in the first available position, i.e., after the last inserted failure in the array `reported`. If the array is full the method must return `false`, otherwise must return `true` because the element can be added.

d) (1,25 points) A method with profile:

   `public Failure search(int hour, int iniM, int finM)`

   for searching the first object the class `Failure` with a time instant with an hour equal to the specified hour and the value of the minutes in the range $[\texttt{iniM}, \texttt{finM}]$. As a precondition, the method assumes that the values of hour are in the range $[0, 23]$ and the values of `iniM` and `finM` are in the range $[0, 59]$ and `iniM` is lower than `finM`. If no object that fulfils the conditions is found the method must return `null`.

e) (1,5 points) A method with the following profile:

   `public TimeInstant closest(TimeInstant t)`

   the returns the closest instant to `t` in that a failure was produced, i.e., the time-stamp of the failure that is closest to `t`. As a precondition you can assume that there is one failure in the `FailureReport` at least.

f) (1,5 points) A method with the following profile:

   `public int[] histogram()`

   that returns the distribution of failures along the 24 hours of the day. This method must return an array of integers whose size must be 24, then, in the position 0 it must return the count of failures reported between 00:00 and 00:59, in the position 1 the count of failures reported between 01:00 and 01:59, and so on.

Solution:

```
public class FailureReport {

    public static final int MAX = 100;
    private Failure[] reported;
    private int nFailures;

    public FailureReport() {
        reported = new Failure[MAX];
        nFailures = 0;
    }

    public boolean add(Failure f) {
        if (nFailures == MAX) { return false; }
        reported[nFailures] = f;
        nFailures++;
        return true;
    }

    /** Precondition: 0 <= hour <= 23, 0 <= iniM <= finM <= 59. */
    public Failure search(int hour, int iniM, int finM) {
        int i = 0; boolean found = false;
        while (i < nFailures && !found) {
            TimeInstant t = reported[i].getInstant();
            int h = t.getHours(), m = t.getMinutes();
            if (h == hour && iniM <= m && m <= finM) { found = true; }
            else { i++; }
        }
        if (found) { return reported[i]; }
        else { return null; }
    }

    /** Precondition: there is one Failure at least. */
    public TimeInstant closest(TimeInstant t) {
        TimeInstant tI = reported[0].getInstant(), minT = tI;
        int min = Math.abs(tI.toMinutes() - t.toMinutes());
        for (int i = 1; i < nFailures; i++) {
            tI = reported[i].getInstant();
            int dif = Math.abs(tI.toMinutes() - t.toMinutes());
            if (dif < min) { min = dif; minT = tI; }
        }
        return minT;
    }

    public int[] histogram() {
        int[] result = new int[24];
        for (int i = 0; i < nFailures; i++) {
            int h = reported[i].getInstant().getHours();
            result[h]++;
        }
        return result;
    }
}
```

2. 2 points Given an integer $n \geq 0$, we need to compute the reverse of n, i.e., another integer with the same figures of n but in the reverse order. Two methods must be written for this purpose. It is supposed both methods are in the same class, so that one can use the other.

To do:

1. (1 point) Write an static method for computing the number of figures (digits) of an integer greater than zero. For instance, for the integer 2347 must return 4, for the integer 8 must return 1, and for 0 must return 1 too.
2. (1 point) Using the previous method, write an static method for computing the reverse of an integer greater than zero. For instance, the reverse of 2347 is 7432.

   Notice that $7432 = 7 \cdot 1000 + 4 \cdot 100 + 3 \cdot 10 + 2 \cdot 1$.

Solution:

```
    /** Counts the number of figures of n, n >= 0. */
    public static int digits( int n )
    {
        int count = 1;
        while (n > 9) {
            n = n / 10;
            count++;
        }
        return count;
    }

    /** Computes the reverse of n, n >=0. */
    public static int reverse( int n )
    {
        int i = digits(n) - 1;
        int add = 0;
        while( i >= 0 ) {
            add = add + (n % 10) * (int) (Math.pow(10, i));
            n = n / 10;
            i--;
        }
        return add;
    }
    public static int reverse_alternative( int n )
    {
        int rev = 0;
        while( n > 0 ) { rev = 10 * rev + (n%10); n/=10; }
        return rev;
    }
```

3. 2 points **To do:** write an static method that receives a char array `a` as parameter, and that writes on the standard output, line by line, all the suffixes of the string of characters contained in `a`, starting from the shortest one to the largest one. For instance, if `a` contains {'s','t','a','t','i','c'}, the output must be:

```
c
ic
tic
atic
tatic
static
```

Solution:

```
public static void suffixes(char[] a) {
    String s = "";
    for (int i = a.length - 1; i >= 0; i--) {
        s = a[i] + s;
        System.out.println(s);
    }
}

public static void suffixes_alternative(char[] a) {
    for (int i = a.length - 1; i >= 0; i--) {
        for (int j = i; j < a.length; j++) {
            System.out.print(a[j]);
        }
        System.out.println();
    }
}
```