Segundo Parcial de IIP (ETSInf). Recuperación
18 de enero de 2019. Duración: 2 horas y 30 minutos

**Nota:** El examen se evalúa sobre 10 puntos, pero su peso específico en la nota final de IIP es de **3,6 puntos**

| NOMBRE: | GRUPO: |
|---|---|

1. 6 puntos Se tiene un sistema de distribución de internet y TV en una localidad, y se desea estudiar los fallos o averías que se producen en el servicio a lo largo de un día. Para ello se dispone de la clase `Failure`, cuyos objetos representan la notificación de una avería y que contiene el identificador de la notificación, y el instante del día en que se produjo el fallo. La clase posee los siguientes métodos:

### Class Failure

***Constructor Summary***

**Constructors**

| Constructor and Description |
|---|
| `Failure(int id, TimeInstant ins)`<br>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()`<br>Devuelve el identificador del Failure. |
| `TimeInstant` | `getInstant()`<br>Devuelve el instante del Failure. |

Los instantes son de la clase `TimeInstant`, ya conocida, algunos de cuyos métodos son:

### Class TimeInstant

***Method Summary***

**All Methods** | **Static Methods** | **Instance Methods** | **Concrete Methods**

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

Se desea desarrollar una clase Java que permita estudiar la distribución de las averías notificadas a lo largo de un día.

**Se pide:** Implementar la clase tipo de datos `FailureReport` que representa el conjunto de averías producidas en la localidad a lo largo de un día, usando los siguientes atributos y métodos:

a) (0,5 puntos) Atributos:

- `MAX`: atributo de clase público constante de tipo entero, que indica el máximo número de averías que pueden guardarse, siendo 100 en este caso.
- `reported`: atributo de instancia privado de tipo array de objetos de la clase `Failure` y capacidad `MAX`, para almacenar las averías que se van notificando. Cada `Failure` se almacena en posiciones consecutivas del array, desde la 0 hasta `nFailures - 1`. Un nuevo `Failure` siempre se dispone en el array a continuación del último previamente guardado (no tiene por qué haber ninguna ordenación de ningún tipo entre los elementos del array).
- `nFailures`: atributo de instancia privado de tipo entero que indica el número de `Failures` que se llevan almacenados.

b) (0,5 puntos) Un constructor por defecto (sin parámetros) que crea el array e inicializa a 0 el número de averías almacenadas.

c) (0,75 puntos) Un método con perfil:

```
public boolean add(Failure f)
```

que añade `f` al `FailureReport` y devuelve un `boolean` que indica si la operación se ha completado con éxito. Se debe situar `f` en la primera posición libre de `reported`, a la derecha de todos las componentes ocupadas del array, siempre que haya espacio en el array, en cuyo caso se devuelve `true`. Si el array estuviera ocupado por completo, no se añade `f`, y se devuelve `false`.

d) (1,25 puntos) Un método con perfil:

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

que busca y devuelve el primer `Failure` cuyo instante de fallo se corresponda a una hora `hour`, y unos minutos entre `iniM` y `finM`, ambos inclusive. Como precondición se supone que `0 <= hour <= 23` y `0 <= iniM <= finM <= 59`. Si no encuentra ninguno, devuelve `null`.

e) (1,5 puntos) Un método con perfil:

```
public TimeInstant closest(TimeInstant t)
```

que devuelve, de entre todos los fallos guardados en el `FailureReport`, el instante más cercano a `t` en el que se ha producido algún fallo, es decir, aquél que difiere un mínimo en minutos respecto a `t`. Como precondición se supondrá que en el `FailureReport` hay al menos un `Failure`.

f) (1,5 puntos) Un método con perfil:

```
public int[] histogram()
```

que devuelve la distribución de fallos a lo largo de las 24 horas de un día, en forma de un array de `int` en el que la componente 0 contiene el número de fallos cuyo instante se encuentra entre las 00:00 y las 00:59 inclusive, la componente 1 el número de fallos entre las 01:00 y las 01:59, ... , y la última componente el número de fallos entre las 23:00 y las 23:59.

**Solución:**

```
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;
    }

    /** Precondición: 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; }
    }

    /** Precondición: En this hay al menos un Failure. */
    public TimeInstant closest(TimeInstant t) {
        TimeInstant tI = reported[0].getInstant(), minT = tI;
        int tToMinutes = t.toMinutes();
        int min = Math.abs(tI.toMinutes() - tToMinutes);
        for (int i = 1; i < nFailures; i++) {
            tI = reported[i].getInstant();
            int dif = Math.abs(tI.toMinutes() - tToMinutes);
            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 puntos Dado un entero $n \geq 0$, se desea calcular el invertido de $n$, es decir, otro entero que contenga las mismas cifras que $n$ pero en orden inverso. Para ello, se escribirán un par de métodos que se supondrán en la misma clase, de forma que uno pueda usar al otro en sus cálculos.

   **Se pide:**

   1. (1 punto) Realizar un método estático que calcule el número de cifras de un número entero dado $\geq 0$. Por ejemplo, para 2347 el método debe devolver 4, para 8 debe devolver 1, para 0 debe devolver 1.
   2. (1 punto) Usando el método anterior, realizar un método estático que calcule el invertido de un número entero dado $\geq 0$. Por ejemplo, para el 2347 calculará el 7432.

      Nótese en el ejemplo que $7432 = 7 \cdot 1000 + 4 \cdot 100 + 3 \cdot 10 + 2 \cdot 1$.

      Se podrá usar el método `Math.pow(a, b)` predefinido de Java, que devuelve un `double`: $\texttt{a}^{\texttt{b}}$.

**Solución:**

```
/** Calcula el número de cifras de n, n >= 0. */
public static int digits(int n) {
    int count = 1;
    while (n > 9) {
        n = n / 10;
        count++;
    }
    return count;
}

/** Calcula el invertido de n, n >=0. Versión 1 */
public static int reversed(int n) {
    int i = digits(n) - 1, add = 0;
    while (i >= 0) {
        add = add + (n % 10) * (int) (Math.pow(10, i));
        n = n / 10; i--;
    }
    return add;
}

/** Calcula el invertido de n, n >=0. Versión 2 */
public static int reversed(int n) {
    int i = digits(n), add = 0;
    while (i > 0) {
        add = add * 10 + (n % 10);
        n = n / 10; i--;
    }
    return add;
}
```

3. 2 puntos **Se pide:** escribir un método estático que tenga como parámetro un array `a` de `char` y que escriba en la salida estándar, línea a línea, todos los *sufijos* de la cadena de caracteres en `a`, desde el más corto en adelante. Se entiende por sufijo cualquier subcadena que comprende los caracteres desde uno dado hasta el último inclusive. Por ejemplo, si `a` es {'s','t','a','t','i','c'}, se debe escribir:

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

**Solución:**

```
/** Versión 1 */
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);
    }
}

/** Versión 2 */
public static void suffixes(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();
    }
}
```