IFC33W - PROGRAMACIÓN

EXAMEN DICIEMBRE 2025

SE VALORARÁ POSITIVAMENTE:
1. Los algoritmos y las estructuras de datos utilizados para la solución así como su justificación.
SE PENALIZARÁ:
2. Los errores en los requisitos y los algoritmos, teniendo en cuenta que un error leve descuenta 1 punto y un error grave 2 puntos. • Se considera un error leve aquel, que pese a su existencia, permite una solución al problema. • Un error grave es aquel que imposibilita una correcta y exacta solución al problema. • La no utilización del método, de cada ejercicio, en un **programa principal será un leve**.
Los ejercicios valen 2,5 puntos y se debe respetar el tiempo máximo establecido y las normas de permanencia. Recomendable utilizar, como método de resolución, el visto en clase pero se corregirá sobre Java.

El peso del examen en la nota total es del 50% y RA relacionada es la RA1 - CPRO-1: Utilizar los conceptos básicos de programación y algoritmia.

Nombre y apellidos: **27/11/2025** **Firma:**

Para gestionar una red de estaciones meteorológicas situadas en Mallorca, se requiere llevar un registro de los nombres de las estaciones y las temperaturas registradas. Se necesita controlar qué estaciones están activas (con lecturas válidas) y cuáles están fuera de servicio.

En una primera propuesta se modela un listado de estaciones como un array de String y la lista de temperaturas como un array de números enteros, con el siguiente significado: el número que está en la posición i del array de temperaturas se corresponde con la temperatura de la estación cuyo nombre está en la posición i del array de String.

CONSIDERACIONES PARA LOS EJERCICIOS:

Podemos declarar las variables que representan las siguientes listas de elementos:
- estaciones_0 sin estaciones.
- estaciones_1 con la estación: "EST-PALMA".
- estaciones_3 con las estaciones: "EST-MARRATXÍ", "EST-MURO", "EST-PUIGMAJOR".

De este modo:

```
String[] estaciones_0 = {};
String[] estaciones_1 = {"EST-MARRATXÍ"};
String[] estaciones_2 = {"EST-MARRATXÍ", "EST-MURO"};
String[] estaciones_3 = {"EST-MARRATXÍ", "EST-MURO", "EST-PUIGMAJOR"};
```

Así vemos que podemos declarar la variable temp_3_0 que representa la lista de temperaturas (que se corresponde con la lista de estaciones estaciones_3), inicialmente con valores especiales (nótese que el valor -999 significa estación fuera de servicio):

```
int[] temp_3_0 = {-999, -999, -999};
```

Así pues, el contenido de dicha variable después de registrar lecturas de 25°C en EST-MARRATXÍ y 18°C en EST- PUIGMAJOR sería:
temp_3_0 = {25, -999, 18};
(En este caso EST- MURO está marcada como fuera de servicio con -999)

1. Implementa un método llamado **obtenerTemperatura** que recibe como parámetros: un String con el nombre de la estación, un array de Strings con los nombres de las estaciones, y un array de enteros con las temperaturas. El método debe devolver la temperatura de la estación indicada. Si la estación no existe, devuelve -999.

CONSIDERACIONES:
- Debes hacer uso del método buscar que ya está implementado de este modo:

```
static int buscar(String estacion, String[] estaciones) {
    for (int i = 0; i < estaciones.length; i++) {
        if (estaciones[i].equals(estacion)) {
            return i;
        }
    }
    return -1;
}
```

- Sabemos que la longitud del array de temperaturas es igual a la longitud del array de String de las estaciones.

Ejemplos de uso:

obtenerTemperatura("EST-MARRATXÍ", {"EST-MARRATXÍ", "EST-MURO", "EST-*PUIGMAJOR*"}, {25, -999, 18}) → 25

obtenerTemperatura("EST-PUIGMAJOR", {"EST-MARRATXÍ", "EST-MURO", "EST-PUIGMAJOR"}, {25, -999, 18}) → -18

obtenerTemperatura("EST-ESTE", {"EST-NORTE", "EST-SUR", "EST-CENTRO"}, {25, -999, 18})→ -999 (no existe)

2. Implementa el método llamado **lasEstacionesActivas** que recibe como parámetros: un array de enteros con las temperaturas y un array de String con las estaciones, y que devuelve un array de String, que contiene como resultado: cada elemento que se forma con el nombre de una estación y su temperatura, separadas por un espacio en blanco.

CONSIDERACIONES:
- Solo deberá contener elementos cuya temperatura sea diferente de -999 (es decir, únicamente debemos tener en cuenta las estaciones activas).
- Sabemos que la longitud del array de temperaturas es igual a la longitud del array de String de las estaciones.

Ejemplos de uso:

lasEstacionesActivas({25, -999, 18}, {"EST-PALMA", "EST-MARRATXÍ", "EST-MURO"})
→ {"EST-PALMA 25", "EST-MURO 18"}
lasEstacionesActivas({-999, -999, -999}, {"EST-PALMA", "EST-MARRATXÍ", "EST-PUIGMAJOR"})→ {}
lasEstacionesActivas({22, 19, 25}, {"EST-PALMA", "EST-MARRATXÍ", "EST-PUIGMAJOR"})
→ {"EST-PALMA 22", "EST- MARRATXÍ 19", "EST- PUIGMAJOR 25"}

3. Implementa un método llamado **actualizar** que recibe como parámetros: un array de enteros con las temperaturas, un array de String con las estaciones, un String con una estación específica y un entero con la nueva temperatura.
El método debe actualizar el array de temperaturas con el nuevo valor para la estación indicada y devolver un booleano indicando si la actualización se ha realizado correctamente.

CONSIDERACIONES:
- Se supone que el método **buscar** está implementado.
- El método devuelve true si la estación existe y se ha actualizado, o false si la estación no existe y no se ha podido actualizar.
- Sabemos que la longitud del array de temperaturas es igual a la longitud del array de String de las estaciones.

Ejemplos de uso:

actualizar({25, -999, 18}, {"EST-NORTE", "EST-SUR", "EST-CENTRO"}, "EST-SUR", 21) → true (*el array queda:* {25, 21, 18})

actualizar({25, 21, 18}, {"EST-NORTE", "EST-SUR", "EST-CENTRO"}, "EST-NORTE", 28) → true (*el array queda:* {28, 21, 18})

actualizar({25, 21, 18}, {"EST-NORTE", "EST-SUR", "EST-CENTRO"}, "EST-ESTE", 15) → false *(la estación no existe, el array no cambia)*

4. Implementa un método llamado lasActivasOrden que funciona igual que **lasEstacionesActivas** del ejercicio 2, pero construyendo el resultado en orden decreciente de las temperaturas. Es decir, el método debe recibir como parámetros: un array de enteros con las temperaturas y un array de Strings con las estaciones, y debe devolver un array de Strings con las estaciones activas y sus temperaturas en orden decreciente (primero la estación con mayor temperatura).

CONSIDERACIONES:

- Se supone que el método **buscar** está implementado y solo deberá contener elementos cuya temperatura sea diferente de -999.
- Sabemos que la longitud del array de temperaturas es igual a la longitud del array de String de las estaciones.

- Debes hacer uso del método posMayorTemp ya está implementado de este modo:

```
static int posMayorTemp(int[] valor) {
    int maxPos = 0;
    for (int i = 1; i < valor.length; i++) {
        if (valor[i] > valor[maxPos]) {
            maxPos = i;
        }
    }
    return maxPos;
}
```

- Para marcar una temperatura como procesada podemos utilizar un -1 para que no se vuelva a elegir como máximo.

Ejemplos de uso:

lasActivasOrden({25, -999, 18}, {"EST-PALMA", "EST-MURO", "EST-AEROPUERTO"})
→ {"EST-PALMA 25", "EST- AEROPUERTO 18"}

lasActivasOrden({15, 30, 22}, {"EST-PALMA", "EST-MURO", "EST-AEROPUERTO"})
→ {"EST-MURO 30", "EST- AEROPUERTO 22", "EST-PALMA 15"}

5. Para evitar interferencias entre sensores, las estaciones meteorológicas deben estar separadas por una distancia mínima. Se representa la ubicación de las estaciones en una cuadrícula (mapa) usando un array bidimensional de caracteres.

- Un punto sin estación se representa con un guion ('-').
- Las estaciones Norte, Sur y Centro se representan con 'N', 'S' y 'C' respectivamente.
- Puede haber hasta tres estaciones en el mapa, dos, una o ninguna.

Implementa el método llamado sinInterferenciasManhattan que recibe como parámetros un array bidimensional de caracteres (char[][]) que representa el mapa, y un entero que representa la distancia mínima necesaria para evitar interferencias.

El método debe devolver:
- true si hay una o ninguna estación, o si la distancia entre TODAS las estaciones es mayor o igual a la distancia mínima.
- false si hay dos o tres estaciones y la distancia entre al menos dos de ellas es menor que la distancia mínima.

CONSIDERACIONES:
- La distancia entre dos puntos se calcula usando la fórmula de distancia Manhattan: distancia = |x1-x2| + |y1-y2|
(suma de las diferencias absolutas de las coordenadas)
- Para calcular el valor absoluto puedes usar: si a < 0 entonces -a, si no a.

Representación de ejemplo:

```
    0  1  2  3
0   -  -  N  -
1   -  S  -  -
2   C  -  -  -
```

Donde N está en posición (0,2), S en (1,1) y C en (2,0).
Distancia Manhattan entre N(0,2) y S(1,1) = |0-1| + |2-1| = 1 + 1 = 2
Distancia Manhattan entre N(0,2) y C(2,0) = |0-2| + |2-0| = 2 + 2 = 4
Distancia Manhattan entre S(1,1) y C(2,0) = |1-2| + |1-0| = 1 + 1 = 2

Ejemplos de uso:

```
char[][] mapa1 = {
    {'-', '-', 'N', '-'},
    {'-', 'S', '-', '-'},
    {'C', '-', '-', '-'}
};
```

sinInterferenciasManhattan(mapa1, 3) → false (distancias: 2, 4, 2 - hay menores que 3)
sinInterferenciasManhattan(mapa1, 2) → true (todas las distancias son >= 2)

```
char[][] mapa2 = {
    {'N', '-', '-', '-'},
    {'-', '-', '-', '-'},
    {'-', '-', '-', 'S'}
};
```

sinInterferenciasManhattan(mapa2, 5) → true (distancia N-S = |0-2| + |0-3| = 5)
sinInterferenciasManhattan(mapa2, 6) → false (distancia 5 < 6)

```
char[][] mapa3 = {
    {'-', 'N', '-'},
    {'-', '-', '-'},
    {'-', '-', '-'}
};
```

sinInterferenciasManhattan(mapa3, 10) → true (solo hay una estación)