Examen
Programación 1º DAM
Ejemplo
Ordinario/Extraordinario

I.E.S. Doctor Balmis

GENERALITAT VALENCIANA
Conselleria de Educación, Cultura y Deporte

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

Nombre	

Instrucciones para la realización de la prueba:

- Duración de la prueba **3 h 40 m** desde el comienzo de la misma.
- **DESCARGAR ENUNCIADO**

Ejercicio 1: (Tiempo estimado orientativo 30 min.)

Codifica un programa que permita almacenar las mediciones de dióxido de nitrógeno (`NO2`) que pasen de un umbral establecido de **10 micro gramos por metro cúbico** para determinadas ciudades. Para guardar esta información se utilizará una **tabla dentada de doubles**. Correspondiéndose cada array en la tabla con las mediciones de una ciudad.

Para ello, los pasos a seguir serán los siguientes:

1. Simularemos la estación de medición usando un método `MedicionNO2` , que devolverá el `double` de la medición realizada **con un retardo de un segundo** (para obtener la medición dentro de un rango lógico se podrá usar la siguiente línea `double medicion = semilla.NextDouble() * 39.74 + 0.84` , siendo semilla un objeto de tipo `Random`).
2. Almacenaremos las distintas mediciones mediante el método `ControlNO2` , al que le llegará **el array correspondiente a las mediciones de una ciudad** y el valor umbral. Este añadirá **una** nueva medición siempre que pase del umbral retornando el mismo array.
3. El programa principal se encargará de llamar a los métodos necesarios para tomar y almacenar, si procede, **12 mediciones** por cada una de las ciudades (Alicante, Castellón y Valencia)
4. Se creará el método necesario para mostrar la tabla dentada como se ve en el ejemplo de salida más abajo.

> 📌 **Nota**: El nombre de la ciudad obviamente no se guardará en la tabla de mediciones.

```
Alicante
16,89 - 36,35 - 37,77 - 36,23 - 27,83 - 26,79 - 27,13 - 28,75 - 37,43
Castellón
39,00 - 23,83 - 19,20 - 26,61 - 28,28 - 18,32 - 34,63 - 33,02 - 25,67 - 23,85 - 39,37 - 37,30
Valencia
26,93 - 13,14 - 18,45 - 27,98 - 19,25 - 17,45 - 36,89 - 30,52 - 32,81 - 37,30
```

Ejercicio 2: (Tiempo estimado orientativo 45 min.)

Usando el fichero de texto que se pasa como recurso `registros.xml` . Debes crear una aplicación con los métodos necesarios para optimizar la modularidad, de forma que a partir de la lectura del fichero y **usando expresiones regulares** para el tratamiento de la información, muestre por pantalla la siguiente salida:

```
La ciudad de Alicante con una medición de 35,92 ha alcanzado el nivel más alto de contaminación
La ciudad de Castellon con una medición de 37,34 ha alcanzado el nivel más alto de contaminació
La ciudad de Valencia con una medición de 41,68 ha alcanzado el nivel más alto de contaminación
```

Debes de capturar **en el programa principal**, las **excepciones** de no existencia de archivo o la producida porque el archivo ya está siendo usado en otra aplicación (acceso).

> 💡 **Pistas:** Dependiendo de como decidas implementar la solución...
>
> 1. Puede facilitarte el trabajo eliminar los saltos de línea, caracteres `\r` y `\n` en la cadena de búsqueda o consumo.
> 2. Fíjate que en la notación para los decimales es un `','` y al mostrarse es igual. Por lo que si decides definir un grupo para localizar este tipo de entrada como cadena. Deberás tener el caracter `','` en cuenta para la expresión regular.

Ejercicio 3: (Tiempo estimado orientativo 65 min.)

Este ejercicio tiene como objetivo crear un sistema de medición que registre mediciones de diferentes tipos de dispositivos. Para ello, se deben implementar dos tipos de medidores: "***Pluviometro98***" y "***MedidorNo2***". El "***Pluviometro98***" realizará mediciones de lluvia en `l/m2` (*litros por metro cuadrado*), mientras que el "***MedidorNo2***" realizará mediciones de contaminación de dióxido de nitrógeno en `µgr/m3` (*microgramos por metro cúbico*).

Para comenzar, se debe crear:

- Una interfaz llamada `IMedidor` con las siguientes características:

    - Una propiedad de solo lectura de tipo diccionario llamada `Mediciones`. Este diccionario se utilizará para almacenar las mediciones realizadas, utilizando un objeto de tipo `DateTime` como clave y un valor de tipo `double` para representar la medición en sí.
    - Otra propiedad de solo lectura llamada `UnidadesMedicion`. Esta propiedad devolverá las unidades de medición correspondientes al tipo de medidor, es decir, `l/m2` para el "***Pluviometro98***" y `µgr/m3` para el "***MedidorNo2***".
    - Otra propiedad de solo lectura llamada `Tipo`. Esta propiedad devolverá el tipo de medidor representado por el nombre de la concreción o subclase que definamos para el tipo de medidor.
- Una clase abstracta `Medidor` que actuará como una base común para los diferentes tipos de medidores. Esta clase **debe implementar la interfaz** `IMedidor` y proporcionar una implementación básica de las propiedades y métodos comunes que permitan que no se repita código en las clases hijas y que proporciono un funcionamiento correcto.
  La clase `Medidor` debe tener las siguientes características:
    i. Una propiedad de solo lectura llamada `IDMedidor` de tipo `string`. Este identificador se utilizará para distinguir cada medidor de manera única.
    ii. Quedarán como abstractas `UnidadesMedicion` y `Tipo`.
    iii. Se instanciará el diccionario de `Mediciones` en el constructor.
    iv. Además, debe tener un método abstracto llamado `Medicion` sin parámetros y void, que realizará las mediciones específicas de cada tipo de medidor.
    v. Invalidación del método `ToString` para que devuelva la información común a todos los medidores. Con un formato similar al siguiente:

    ```
    El <Tipo> con ID: <IDMedidor>
    Ha realizado las siguientes mediciones:
    En fecha dd/mm/yyyy hh:mm:ss -> <medicion_1> <UnidadesMedicion>
    En fecha dd/mm/yyyy hh:mm:ss -> <medicion_2> <UnidadesMedicion>
    ...
    ```

    Donde iremos mostrando las diferentes mediciones del diccionario *medicion_1*, *medicion_2*, ..., *medicion_n* redondeadas a **dos decimales** con sus respectivas fechas.
- Por último, se deben implementar la clases `Pluviometro98` y `MedidorNo2`, que heredarán de la clase `Medidor` y proporcionarán sus propias implementaciones (concreciones). Para la medición del **Medidor de No2** podremos usar el medidor propuesto en los anteriores ejercicios, mientras que el pluviómetro usará la siguiente generación aleatoria
  `double medicion = semilla.NextDouble() * 300`

Una vez implementadas todas las clases y la interfaz, se debe crear una clase llamada `CentralMedicion` que actuará como un contenedor de medidores. Esta clase tendrá una lista de

medidores y permitirá agregar nuevos medidores a través del método `AñadeMedidor` al que le llegará un `Medidor` y tendrá un interfaz de tipo '*fluido*' como se aprecia en el programa principal. Además, definirá **un campo de tipo cadena y privado** `nombre` que permitirá **identificar la central**. También tendrá un método llamado `Mide` que realizará la medición **en todos los medidores agregados**. Se deberá invalidar `ToString` para conseguir la siguiente salida...

> 📌 **Nota**: La siguiente salida es un ejemplo de ejecución del programa principal que se te proporciona donde: Se crea una instancia de la clase `CentralMedicion` y se agregan instancias de `Pluviometro98` y `MedidorNo2` a través del método `AñadeMedidor` . Luego, se llama al método `Mide` para realizar las mediciones en todos los medidores agregados.

```
AlicanteCentro
El MedidorNo2 con ID: 1234A
Ha realizado las siguientes mediciones:
En fecha 30/05/2023 22:23:42 -> 31,06 µgr/m3
En fecha 30/05/2023 22:23:42 -> 40,31 µgr/m3
En fecha 30/05/2023 22:23:42 -> 1,42 µgr/m3
En fecha 30/05/2023 22:23:42 -> 13,47 µgr/m3

El MedidorNo2 con ID: 1235A
Ha realizado las siguientes mediciones:
En fecha 30/05/2023 22:23:42 -> 34,78 µgr/m3
En fecha 30/05/2023 22:23:42 -> 8,05 µgr/m3
En fecha 30/05/2023 22:23:42 -> 27,68 µgr/m3
En fecha 30/05/2023 22:23:42 -> 29,69 µgr/m3

El Pluviometro98 con ID: AF98X1
Ha realizado las siguientes mediciones:
En fecha 30/05/2023 22:23:42 -> 259,91 l/m2
En fecha 30/05/2023 22:23:42 -> 241,04 l/m2
En fecha 30/05/2023 22:23:42 -> 184,44 l/m2
En fecha 30/05/2023 22:23:42 -> 65,17 l/m2
```

Ejercicio 4: (Tiempo estimado orientativo 20 min.)

Usando el fuente del `ejercicio5.cs` que se pasa como recurso. Implementa las consultas que se proponen para conseguir el siguiente código:

- **Consulta 1**: Muestra las ciudades que hayan tenido una medición con fecha `12/03/2023`

  ```
  Alicante
  Valencia
  ```

- **Consulta 2**: Mostrar el nombre de las ciudade que hayan superado una medición mayor a 30 µg/m3

  > 📌 **Nota**: Puedes usar `Exists` para el filtrado de `where` pero no sería necesario.

```
Valencia
Castellón
```

- **Consulta 3**: Muestra todas las temperaturas de cada ciudad agrupadas por esta, ordenadas y sin repeticiones

```
{ Ciudad = Alicante, Temperaturas = 7,98 - 9,2 - 10,2 - 11,9 - 12,9 - 16,5 - 16,8 - 17,4 - 19,24 - 22,2 - 22,3 - 23,2 - 24,23 - 25,1 - 28,7 - 29,5 }
{ ciudad = Valencia, Temperaturas = 7,98 - 19,2 - 21,98 - 29,24 - 34,3 - 39,2 }
{ Ciudad = Castellón, Temperaturas = 6,4 - 9,8 - 10,4 - 12,5 - 13,8 - 14,2 - 16,1 - 19,7 - 26,1 - 28,7 - 32,9 }
```

- **Consulta 4:** Muestra la media de todas las mediciones de todas las ciudadesque hayan tenido una medición con fecha "12/02/2023"

```
21,22
```

- **Consulta 5:** Muestra la temperatura máxima de cada una de las ciudades agrupadas por fechas

```
{ fecha = 12/02/2023 0:00:00, temperaturas = 39,2 }
{ fecha = 11/02/2023 0:00:00, temperaturas = 28,7 }
{ fecha = 22/03/2023 0:00:00, temperaturas = 32,9 }
{ fecha = 03/04/2023 0:00:00, temperaturas = 25,1 }
{ fecha = 11/05/2023 0:00:00, temperaturas = 29,5 }
```