Examen Programación 1º
DAM Semipresencial
Ejemplo 1
3ª Evaluación

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 **2 h 45 m** desde el comienzo de la misma.
- Lee atentamente el enunciado y no formules tus preguntas en voz alta, levanta la mano y espera a que el profesor te diga que tienes que hacer.
- Si te atascas en un apartado déjalo y pasa al siguiente. Cada apartado tiene su puntuación, independientemente del funcionamiento del programa final y el orden propuesto en el enunciado.
- Penalizará con **0,1 décimas** en cada uno de los siguientes casos:
  - Llaves mal puestas e instrucciones mal tabuladas dentro de un bloque.
  - No seguir las reglas de nomenclatura de C#.
  - Identificador de variable o método que a juicio del profesor no esté bien puesto porque no sigue las reglas descritas en classe.
- **No se puede usar ningún tipo de dispositivo USB, ni código de ejercicios o ejemplos realizados en clase. Solo los pdf de apuntes que se te proporcionarán para la ocasión**.
- En caso de pillar a algún alumno copiando o usando material no autorizado se le retirará el examen y su calificación en la evaluación será un suspenso.

Pregunta 1 (7 puntos)

partiendo de los recursos para el examen en la carpeta `ejercicio1` donde dispones programa principal `Program.cs` para **probar tu implementación** y que te puede servir de orientación si tienes dudas en alguna de las especificaciones. Vamos a implementar un **programa que gestione el inventario de una farmacia**. Para ello, vamos a definir una abstracción de almacén genérico con las siguientes especificaciones:

1. Definir la interfaz `IProducto` cons las propiedades `Nombre`, `Precio` y `Proveedor` de solo lectura y los tipos de datos adecuados.
2. Definir la clase `DatosAlmacenaje` con las propiedades `Unidades` de tipo entero y lectura/escritura y `Ubicacion` de tipo entero y solo lectura. El constructor de la clase recibe la ubicación del producto en el almacén y establece las unidades a 0.
3. Definir la **clase parametrizada** `Almacen` donde el tipo parametrizado debe cumplir la restricción de que implemente la interfaz `IProducto` es decir **el tipo parametrizado representará algún tipo de producto**. La clase debe tener las siguientes características:
   - Una propiedad **privada** `Inventario` de tipo `Dictionary<T, DatosAlmacenaje>` y solo lectura donde `T` es el tipo parametrizado de la clase.

     > 📌 **Nota**: Fíjate que las **claves** del diccionario son los productos y los **valores** son los datos de almacenaje del mismo, esto es, las unidades y su ubicación.

   - Una propiedad **privada** `Ubicaciones` de tipo `bool[]` y solo lectura que representará si las ubicaciones fijas en el almacén están ocuparas o no. Teniendo en cuenta que:
     - **Solo podrá haber un producto por ubicación**.
     - El **índice del array representará la ubicación** que guardamos en `DatosAlmacenaje` y el valor `true` si está ocupada y `false` si está libre.
       En el ejemplo siguiente, las ubicaciones 0, 1 y 2 están ocupadas y las demás libres:

       ```
                      0  1  2  3  4  5  6  7  8  9  ...
       Ubicaciones -> [T][T][T][F][F][F][F][F][F][F] ...
       ```

   - Un constructor que recibe el número de ubicaciones disponibles en el almacén e instanciara en memoria las colecciones que hemos definido como propiedades de solo lectura.
   - Un método `Productos` que recibe un delegado con forma de predicado predicado y devuelve **una lista** de productos que cumplen dicho predicado que se aplicará sobre los datos de almacenaje del mismo. Esto es, me devolverá aquellos productos que cumplan un determinado stock o ubicación.
   - Un método `EstaEnAlmacen` que recibe un producto y devuelve si está en el almacén o no.
   - Un método `Alta` que recibe un producto y lo da de alta en el almacén. Teniendo en cuenta que:
     - Si el producto **ya está** en el almacén se lanzará una excepción de tipo `InvalidOperationException` con el mensaje `"El producto ya está dado de alta en el almacén"`.
     - Buscaremos la primera ubicación libre en el almacén y si no hay lanzaremos una excepción de tipo `InvalidOperationException` con el mensaje `"No hay ubicaciones libres en el almacén"`.
     - Si todo es correcto, añadiremos el producto al diccionario de inventario y marcaremos la ubicación como ocupada en el array de ubicaciones.
   - Un método `Baja` que recibe un producto y lo da de baja del almacén. Teniendo en cuenta que:
     - Si el producto **no está** en el almacén se lanzará una excepción de tipo `InvalidOperationException` con el mensaje `"El producto no está en el almacén"`.

    - Marcaremos la ubicación como libre en el array de ubicaciones y eliminaremos el producto del diccionario de inventario.
- Un método `AgregaStock` que recibe un producto y un número de unidades y agrega stock al producto. Ten en cuenta que, no se puede agregar stock a un producto que no está en el almacén o en caso contrario se lanzará una excepción igual a la del método `Baja`.
- Un método `RetiraStock` que recibe un producto y un número de unidades y retira tantas unidades como se indique por parámetro. El método **devuelve el número de unidades retiradas**. Ten en cuenta que, no se puede retirar stock a un producto que no está en el almacén o en caso contrario se lanzará una excepción igual a la del método `Baja`. Además, no podemos retirar más unidades de las que tiene el producto en stock y por tanto el número de unidades no puede quedar negativo. Por ejemplo, si un producto tiene 10 unidades y queremos retirar 15, solo podremos retirar 10 unidades que será lo que retorne el método y se quedará el stock en 0.
- Un método `ResumenInventario` que devuelve un string con un resumen del inventario en el siguiente formato:

```
Nombre               Ubicación  Unidades    Precio
-------------------  --------- --------- ---------
Paracetamol                  0        10      5,99
Ibuprofeno                   1         0      7,49
Amoxicilina                  2        10      9,99
-------------------------------------------------
Valor total: 159,80
```

> 📌 **Nota**: Nombre ocupa 20 caracteres, Ubicación 10, Unidades 10 y Precio 10.

4. Define la clase `Medicamento` que implementa la interfaz `IProducto` y es un **record** (registro).

> 📌 **Nota**: Si no sabes definir un record y lo defines como clase normal, recuerda que al usarse como clave en un diccionario en ese caso debes sobrescribir los métodos `Equals` y `GetHashCode`.

5. Define la clase `Farmacia` con las siguientes características:
    - Una propiedad **privada** `Nombre` de tipo `string` y solo lectura.
    - Una propiedad **privada** `Almacen` de tipo almacén de medicamentos.
    - Un constructor que recibe el nombre de la farmacia y lo almacena en la propiedad correspondiente y crea un almacén de medicamentos con 100 ubicaciones disponibles.
    - Los métodos `Alta`, `Baja`, `Reponer` y `Retirar` como ***'cuerpo de expresión'*** que delegan en el almacén de medicamentos.
    - Un método `MedicamentosAgotados` que devuelve una lista de medicamentos que tienen 0 unidades en stock.
    - Un método `MostrarInventario` que devuelve un string con el nombre de la farmacia y un resumen del inventario del almacén.

Criterios de evaluación:

> 📌 **Nota:** Cualquier error de sintaxis en la definición de las clases o métodos que genere errores de compilación, así como el código a medio implementar podrá suponer la calificación con **0 pts** puntos del apartado correspondiente.

Pregunta 2 (3 puntos)

A partir de los recursos para el examen en la carpeta `ejercicio2` completa el código del programa principal `Program.cs` en los métodos estáticos correspondiente las 6 consultas que se piden y que describimos a continuación:

- **Consulta 1**: Mostrar titulo, año, país y valoración de las películas españolas del año 2014 ordenados descendentemente por valoración.
  Salida por consola:

```
{ Titulo = Ocho apellidos vascos, Año = 2014, Pais = España, Valoracion = 6,95 }
{ Titulo = El Niño, Año = 2014, Pais = España, Valoracion = 6 }
```

- **Consulta 2**: Mostrar titulo, año, país y duración de las películas con duración entre 100 y 120 minutos, es decir, con duración mayor o igual de 100 y menor o igual de 120.
  Salida por consola:

```
{ Titulo = El pasajero, Año = 2018, Pais = Estados Unidos, Duracion = 105 }
{ Titulo = Venganza bajo cero, Año = 2019, Pais = Reino Unido, Duracion = 118 }
{ Titulo = El rey león, Año = 2019, Pais = Estados Unidos, Duracion = 118 }
{ Titulo = Toy Story 4, Año = 2019, Pais = Estados Unidos, Duracion = 100 }
```

- **Consulta 3**: Mostrar titulo, año y géneros" de las películas que el genero sea de "*drama*", "*romance*" o "*fantástico*".

  > 📌 **Nota**: Debes hacerlo completando el cuerpo de expresión de método `static IEnumerable<Consulta3Dto> Consulta3() => ...` proporcionado en el fuente.
  > Debes completar también el **método de extensión** `ToJSON` para generar el fichero `consulta3.json` con la serialización a **JSON** del resultado de la misma con las propiedades en minúscula.

  Salida por consola:

```
Consulta3Dto { Titulo = El Niño, Año = 2014, Generos = [intriga, accion, drama, policiaca] }
Consulta3Dto { Titulo = Ocho apellidos vascos, Año = 2014, Generos = [comedia, romance, crimen] }
Consulta3Dto { Titulo = Aquaman, Año = 2018, Generos = [accion, fantastico] }
Consulta3Dto { Titulo = El rey león, Año = 2019, Generos = [animacion, aventuras, drama] }
```

- **Consulta 4**: Mostrar titulo, país y reparto de las películas en las que conste un reparto con 2 actores.
  Salida por consola:

```
{ Titulo = Todos los caminos, Pais = España, Reparto = [Dani Rovira, Clara Lago] }
{ Titulo = El pasajero, Pais = Estados Unidos, Reparto = [Liam Neeson, Patrick Wilson] }
```

- **Consulta 5**: Calcular la duración media de las películas de cada país.

  > 📌 **Nota**: Se debe mostrar `"Pais: DuracionMedia"`

  Salida por consola:

```
España: 107,5
Estados Unidos: 111,2
Reino Unido: 118
```

- **Consulta 6**: Muestra sin repeticiones todos géneros de las películas en cartelera usando SelectMany.
  Salida por consola:

```
[intriga, crimen, accion, drama, policiaca, comedia, romance, documental, fantastico, animacion, aventuras
```

> ✋ **Importante**:
>
> - En `Cartelera.cs` de la propiedad estática `Cartelera.Peliculas` con la secuencia con la que realizar las consultas. **Analiza detenidamente su contenido y estructura** para saber realizar las mismas.
> - Cada consulta debe implementarse en una única expresión.
> - Puedes usar `Select`, `SelectMany`, `Where`, `OrderBy`, `OrderByDescending`, `GroupBy`, `Count` y `Distinct`.
> - Puedes definir usar clases anónimas para realizar las mapeos o proyecciones salvo que se especifique otra cosa.