Examen Programación 1º
DAM Semipresencial
ejemplo 2
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 clase.
- **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)

Vamos a modelar el proceso de **ITV** (Inspección Técnica de Vehículos) de tal forma que utilicemos las estructuras vistas durante el tercer trimestre.

Para ello tenemos la clase `Vehiculo` que se te proporciona en la carpeta `RecursosParaExamen\ejercicio1\Vehiculo.cs` y si la examinas detalladamente define una serie de propiedades que devuelven de forma aleatoria características a evaluar del vehículo como los porcentajes de frenada e integridad estructural para los que un 100%, como es lógico, significa que están en perfectas condiciones. Además, tienes la propiedad `ComprobacionLuces` que devolverá un texto indicando un fallo en el alumbrado del vehículo o `null` si está todo correcto.

Define la clase `VehiculoConEmisiones` que herede de `Vehiculo` y añada un propiedad de acceso pública denominada `PorcentajeEmisiones` que devuelva un valor entre 0 y 30.

También se te proporciona la clase `CLiente` que se encuentra en la carpeta `RecursosParaExamen\ejercicio1\Cliente.cs` y modela los datos de un cliente de la ITV con un `Vehiculo` asociado.

Por último, también se te proporcionan las clases `ItvExcepction` y la clase `Program` con el programa principal para que puedas probar el funcionamiento de tu código y al que haremos referencia a lo largo del enunciado.

Define la clase de tipo **record** `Comprobacion` que tenga las siguientes propiedades de acceso públicas:

- `Nombre` de tipo `string` que contendrá el nombre de la comprobación.
- `Apto` de tipo `bool` que indicará si el vehículo ha pasado la comprobación.
- `Observacion` de tipo `string` que contendrá una nota u observación respecto a la comprobación realizada.

Por último, invalidaremos `ToString` que devolverá una cadena con el siguiente formato:

- Si el vehículo **NO** ha pasado la comprobación: `[X] Nombre: Observacion`. Ej. `[X] Frenada: 50%`
- Si el vehículo ha pasado la comprobación: `[V] Nombre: Observacion`. Ej. `[V] Frenada: 98%`

Define la clase `InformeItv` que ...

1. Tenga una propiedad **privada** denominada `Comprobaciones` que contendrá una lista con las comprobaciones realizadas al vehículo.
2. Tenga las siguientes propiedades de **acceso públicas**:
   - `Inspector` de tipo `string` que contendrá el nombre del inspector que realiza el informe.
   - `Cliente` de tipo `Cliente` que contendrá el cliente al que se le realiza el informe.
   - `Vehiculo` de tipo `Vehiculo` que contendrá el vehículo al que se le realiza el informe. **Debes obtenerlo del cliente**.
   - `Apto` de tipo `bool` que indicará si el vehículo es apto al pasar todas las comprobaciones. Por tanto, es una propiedad calculada que devolverá `true` si todas las comprobaciones son aptas y `false` en caso contrario.
3. Propiedad pública de **acceso y privada** de modificación de tipo bool denominada `Abierto` que indicará si el informe está abierto o cerrado. Esto es, si se le pueden añadir aún comprobaciones o no.
4. Define el constructor que reciba **únicamente** la información imprescindible.
5. Método `AñadirComprobacion` que reciba una comprobación y la añada a la lista de comprobaciones. Si el informe está cerrado debe lanzar una excepción de tipo `ItvException`.
6. Método `Cerrar()` que cierre el informe.
7. invalidación del método `ToString` que devolverá una cadena con el siguiente formato. (EL contenido de los textos es orientativo y vendrá determinado por es estado de la clase `Comprobacion`)

```
Informe ITV de 8231 KNS Opel
Cliente: Paco 88888888 H
Inspector: Carmen
    [V] Frenada: Medición 74% (Mínimo 70%)
    [V] Integridad estructural: Medición 98% (Mínimo 90%)
    [X] Luces: Luz de freno trasera izquierda
    [X] Emisiones: Medición 18% (Máximo 10%)
Resultado: NO APTO
```

> 📌 **Nota**: Ten en para componer la cadena las cadenas que se devuelven en el método `ToString` de las clases `Cliente`, `Vehiculo` y `Comprobacion`.

Define el interfaz `IReceptorInformesItvAutorizado` que parametrice un tipo de informe genérico y que define un único método `Recibe` que no retorne nada y reciba un informe del tipo parametrizado.

Define la clase `DireccionGeneralDeTrafico` que imeplemente el interfaz `IReceptorInformesItvAutorizado` para informes del tipo `InformeItv` y cuya implementación del método `Recibe` muestre por consola el texto `DGT recibiendo informe de ITV ...` y a continuación el informe recibido

```
DGT recibiendo informe de ITV  ...
Informe ITV de 8231 KNS Opel
Cliente: Paco 88888888 H
...
```

Define la clase `Itv` que modele la gestión de la ITV interactuando con las clases anteriores. Para ello, su especificación será la siguiente:

> 📌 **Nota**: Se recomienda hacerse un esquema del modelo propuesto a modo de borrador antes de comenzar a programar. Si tienes dudas de algún interfaz mira el código de la clase `Program.cs` .

1. Definiremos el **tipo enumerado privado** `Linea` con los valores `Linea1 = 1` , `Linea2 = 2` y `Linea3 = 3` que enumerarán las 3 **líneas de vehículos** para pasar la ITV de que dispone nuestro modelo.
2. Definiremos una propiedad autoimplementada **privada** `LinealesDeVehiculos` que representará las `colas` de vehículos que están esperando en cada línea de la ITV. Esta propiedad será de tipo `Dictionary` cuyas claves serán los valores del enumerado `Linea` y cuyos valores serán de tipo `Queue<InformeItv>` . Fíjate que realmente no guardamos un Vehículo en la cola sino que guardamos un `InformeItv` que es el que contiene el vehículo y será abierto por un inspector.
3. Definiremos una propiedad autoimplementada **con solo un accesor privado** denominada `Inspectores` y que será un tipo `Queue<string>` que representará la cola de inspectores disponibles para pasar la ITV.
4. Definiremos un **constructor** que inicialice la propiedad `LinealesDeVehiculos` que contenga las tres líneas de vehículos vacías. Además, inicializará la propiedad `Inspectores` con un array de inspectores que le llegará como parámetro.
5. Definiremos un método de instancia **privado** denominado `LineaMenosOcupada` que no recibirá ningún parámetro y devolverá el valor del enumerado `Linea` que corresponda a la línea de vehículos **menos ocupada**. Si hay varias un mismo número de vehículos en las líneas, devolverá la primera que encuentre.
6. Definiremos un método de instancia `Llegada` que reciba un `Cliente` y que **encole un informe de ITV** a la línea de vehículos **menos ocupada** que encontraremos usando el método anteriormente implementado. Para asignar un inspector al informe, lo **desencolaremos** de la cola de inspectores disponibles y si la cola está vacía se lanzará una excepción de tipo `ItvException` .
7. Define un **evento privado** dentro de la clase denominado `ItvPasada` que será un delegado predefinido que recibirá un `InformeItv` . Lógicamente este evento servirá para que la ITV (publicador) notifique a un suscriptor de que un vehículo ha pasado la ITV pasándole su informe. Además, como el evento es privado, para suscribirnos a él, lo haremos mediante el método `AñadeReceptorInformes` que será público y recibirá el suscriptor de tipo `IReceptorInformesItvAutorizado<...>` .(Mira el programa principal de ejemplo)
8. Definiremos un método de instancia **privado** denominado `PasaItvAlSiguienteVehiculoEnLaLinea` que recibirá una `Linea` y se encargará de pasar la Itv al siguiente vehículo en la misma siguiendo las siguientes especificaciones:
    i. Definirá las constantes `PORCENTAJE_MAXIMO_EMISIONES` , `PORCENTEJE_MINIMO_FRENADA` y `PORCENTEJE_MINIMO_INTEGRIDAD_ESTRUCTURAL` con los valores `10` , `70` y `90` respectivamente.
    ii. Si la línea está vacía se lanzará una excepción de tipo `ItvException` con el mensaje `"La linea {linea} está vacia."` .
    iii. **Desencolaremos el informe** a atender de la cola de la línea.
    iv. Mostraremos por consola el mensaje con el siguiente formato `"Pasando ITV vehículo {MATRICULA} en línea {LINEA} por {INSPECTOR} ..."` .
    v. Realizaremos las **comprobaciones del informe**. Donde las tres primeras serán son los porcentajes de **Frenada**, **Integridad Estructural** y **Emisiones**. Además, tendremos la comprobación de **Luces** que me devolverá una cadena que si es distinta de null recuerda que me indicará que la comprobación '*No es apta*'. Para cada una de ellas crearemos una instancia de la clase `Comprobacion` y la añadiremos al informe.

    > 📌 **Nota**: Ten en cuenta que solo podremos pedirle **una sola vez** el valor de la comprobación al vehículo deinforme ya que se genera aleatoriamente y si lo hiciéramos más deuna vez obtendríamos

> valores diferentes. Además, el porcentaje deemisiones solo podremos saberlo si el vehículo es un vehículo conemisiones. Los mensajes a añadir en el parámetro `Observacion` los puedesdeducir del ejemplo en el punto 7 de la clase `InformeItv` .

Tras realizar las comprobaciones debes **cerrar el informe** y volver a **encolar al inspector que lo lleva** en la cola de inspectores como disponible.

vi. Por último debes '*disparar*' o desencadenar el evento `ItvPasada` para notificar el informe de la ITV a aquellos suscriptores de informe autorizados que se suscribieran mediante `AñadeReceptorInformes`

9. Implementaremos el método público denominado `AtenderInformesEnLineas` que usaremos en el programa principal y el cual se encargará de atender los informes de ITV de todas las líneas de vehículos hasta que se queden vacías. Para ello, usaremos el método `PasaItvAlSiguienteVehiculoEnLaLinea` que acabamos de implementar en el punto anterior.

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, artista y periodo de las pinturas del Barroco ordenados ascendentemente por titulo.

Salida por consola:

```
{ Titulo = La rendición de Breda, Artista = Diego Velázquez, Periodo = Barroco }
{ Titulo = Las meninas, Artista = Diego Velázquez, Periodo = Barroco }
{ Titulo = Las tres Gracias, Artista = Pedro Pablo Rubens, Periodo = Barroco }
```

Consulta 2: Mostrar titulo, dimensiones y área de las pinturas que tenga de alto más de 3m ordenados descendentemente por altura.

Salida por consola:

```
{ Titulo = Las meninas, Dimensiones = Dimensiones { Alto = 3,18, Ancho = 2,76 }, Area = 8,7768 }
{ Titulo = La rendición de Breda, Dimensiones = Dimensiones { Alto = 3,07, Ancho = 3,67 }, Area = 11,2669 }
```

Consulta 3: Mostrar titulo, artista y géneros de las pinturas que tengan como genero "mitología" o "historia".

> 📌 **Nota**: Debes hacerlo completando el cuerpo de expresión de método `static IEnumerable<Consulta3Dto> Consulta3() => ...` proporcionado.

Salida por consola:

```
Consulta3Dto { Titulo = Las meninas, Artista = Diego Velázquez, Generos = [retrato, historia] }
Consulta3Dto { Titulo = Saturno devorando a su hijo, Artista = Francisco de Goya, Generos = [mitología] }
Consulta3Dto { Titulo = El 3 de mayo en Madrid, Artista = Francisco de Goya, Generos = [historia, guerra] }
Consulta3Dto { Titulo = La rendición de Breda, Artista = Diego Velázquez, Generos = [guerra, historia] }
Consulta3Dto { Titulo = Las tres Gracias, Artista = Pedro Pablo Rubens, Generos = [desnudo, mitología] }
```

Consulta 4: Mostrar el número de pinturas de cada periodo ordenados descendentemente por número de pinturas.

> 📌 **Nota**: No se puede usar `Select`

Salida por consola:

```
{ Periodo = Barroco, NumeroPinturas = 3 }
{ Periodo = Renacimiento, NumeroPinturas = 1 }
{ Periodo = Romanticismo, NumeroPinturas = 2 }
```

Consulta 5: Mostrar los géneros sin repeticiones usando `SelectMany`.

Salida por consola:

```
retrato, historia, cristiandad, mitología, guerra, desnudo
```

> ✋ **Importante**:
>
> - En `Museo.cs` de la propiedad estática `Museo.Pinturas` 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.