Examen Programación 1º DAM
Semipresencial
Fecha
2ª Evaluación

IES DOCTOR BALMIS
GENERALITAT VALENCIANA
Conselleria de Educación, Cultura y Deporte
Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

Nombre	

Ejercicio 1 curso 2024-25 (4,5 puntos)

El Restaurante "El Bug del Código"

Eres un desarrollador contratado para crear un sistema que genere menús para "El Bug del Código". El chef, con su particular sentido del humor, te pide tres opciones: "***Menú Sorpresa***", "***Menú sin Gluten ni Ajo***" y "***Menú Reducido***".

En los recursos para el examen, en el **workspace del ejercicio1**, se te provee con las clases `Plato`, `Entrante`, `Principal`, `Postre`, `Bebida`, `ValorNutricional`, el **enum** `Alergeno` y la clase estática `BancoDePlatos` que contiene arrays de platos predefinidos. Dedica unos minutos a analizar la estructura de las clases y los métodos que contienen.

1. **Crea la clase abstracta** `Menu`:
   - Debe tener una **propiedad de tipo array de platos** con un `set` protected y un `get` privado.
   - Propiedad abstracta `TipoMenu` de solo lectura que usaremos en el `ToString()` y que me devuelva una **cadena** con el nombre de los posibles tipos de menú.
   - Define los métodos `Precio()` que calcula el precio total del menú, `ValorNutricional()` que calcula la suma del valor nutricional de los platos y `Alergenos()` que devuelve la combinación de todos los alergenos presentes en el menú.
   - Invalida el método `ToString()` para mostrar el menú de forma legible, incluyendo el nombre de cada plato, sus ingredientes, precio y alérgenos, así como el precio total, el valor nutricional total y los alérgenos del menú completo.
2. **Crea la clase** `MenuSorpresa`: Hereda de `Menu` y genera un menú aleatorio con un entrante, un principal, un postre y una bebida obtenidos de los platos predefinidos en `BancoDePlatos`.
3. **Crea la clase** `MenuSinGlutenNiAjo`: Hereda de `Menu` y genera un menú que **no contenga el alérgeno *'gluten'***, **ni el ingrediente *'ajo'*** en ninguno de sus platos. Si no es posible generar un menú con estas restricciones, lanza una excepción personalizada del tipo `MenuException` con el mensaje "*No se puede generar un menú sin gluten ni ajo*". Para ello, puedes añadir **el primer plato** que no contenga gluten ni ajo de cada tipo de plato. Si no encuentra.
4. **Crea la clase** `MenuReducido`: Hereda de `Menu` y genera **un menú reducido** con un plato único que se recibirá como parámetro en un **único** constructor de la clase. Además, tendrá un postre y una bebida aleatorios.
   El plato único podrá ser solo un **entrante** o un **principal**, pero no un postre ni una bebida. Si el plato no es un entrante ni un principal, lanza una excepción personalizada del tipo `MenuException` con el mensaje "*El plato para crear un menú reducido debe ser un entrante o un principal*".

5. **Completa** el programa principal que se te proporciona para que:
    - Crear las instancias del menú que se proponen en las opciones dentro del switch de expresión.
    - Controlar las excepciones de tipo `MenuException` que se produzcan dentro del while y mostrar el mensaje de error correspondiente y recuperando la ejecución normal del bucle.

Ejemplo de Salida y criterios de evaluación en la siguiente página.

```
1. Menú sorpresa
2. Menú sin gluten ni ajo
3. Menú reducido con principal "Paella Valenciana"
4. Menú reducido con entrante "Ensalada César"
5. Salir
Seleccione una opción: 4

Menu Reducido

Entrante
Nombre: Ensalada César
Ingredientes: Lechuga, pollo, picatostes, salsa César
Precio: 8,5
Alergenos: Gluten, Lacteos
Postre
Nombre: Flan de huevo
Ingredientes: Huevos, leche, azúcar, caramelo
Precio: 4
Alergenos: Lacteos, Huevos
Bebida
Nombre: Agua mineral
Ingredientes: Agua
Precio: 1,5
Alergenos: Ninguno

Precio total: 14
Alérgenos: Gluten, Lacteos, Huevos
Valor nutricional: Calorías 650, Grasas 40, Azúcares 35, Proteínas 25
```

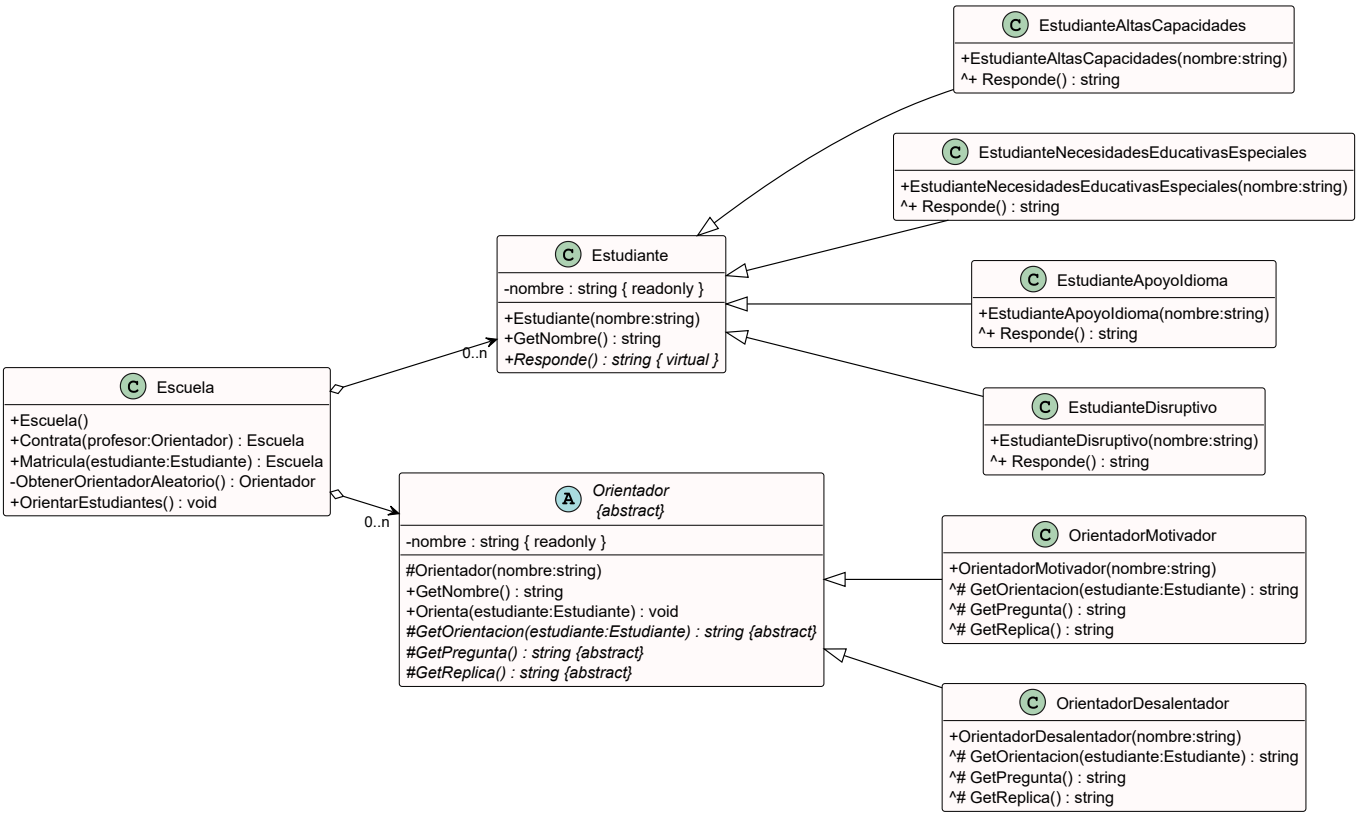
Criterios de calificación:

- (✅ **0,25 pts**) Definición de `Menu`, campos, accesores/mutadores y método abstracto.
- (✅ **0,25 pts**) Implementación del método `Precio`.
- (✅ **0,5 pts**) Implementación del método `ValorNutricional`.
- (✅ **0,5 pts**) Implementación del método `Alergenos`.
- (✅ **0,25 pts**) Implementación del método `ToString`.
- (✅ **0,25 pts**) Implementación de la clase `MenuSorpresa`.
- (✅ **1 pts**) Implementación de la clase `MenuSinGlutenNiAjo`.
- (✅ **0,75 pts**) Implementación de la clase `MenuReducido`.
- (✅ **0,25 pts**) Completa el programa principal según las especificaciones.
- (✅ **0,5 pts**) Compila y ejecuta correctamente.

Ejercicio 2 curso 2023-24 (7 puntos)

Vamos modelizar mediante programación orientada a objetos un ona serie de objetos relativos al departemento de orientación en una escuela.

Para ello, vamos a definirlas clases expresadas en el diagrama teniendo en cuenta los roles entre ellas, la accesibilidad de los campos y métodos, así como nombres de métodos y **accesores/mutadores descritos en el mismo que deberás implementar como propiedades siempre que sea posible**.



Las especificaciones de la implementación serán las siguientes:

Definiremos una clase `Estudiante` cuyo método `Responde` por defecto siempre responda *"Estoy estudiando."*

Definiremos las concreciones de estudiante `EstudianteAltasCapacidades`, `EstudianteNecesidadesEducativasEspeciales`, `EstudianteApotoIdioma` y `EstudianteDisruptivo` serán concreciones de la clase `Estudiante` y que invalidarán el método `Responde` para que devuelva *"Estoy participando activamente en actividades avanzadas."*, *"Estoy trabajando con el apoyo de un educador especial."*, *"Estoy mejorando mis habilidades en el idioma."* y *"Estoy trabajando en mejorar mi comportamiento en clase."* respectivamente.

Por otro lado, definiremos la asbtracción `Orientador` que se concretará en las clases:

1. `OrientadorDesalentador` que **siempre preguntará** *"¿Hay algo que te esté preocupando en la escuela?"*, siempre replicará *"Entiendo, a veces las cosas pueden parecer difíciles."* y que según el estudiante que atienda **devolverá la la orientación** *"no te exijas demasiado, podrías agobiarte."* si el estudiante es de **altas capacidades**, *"evitar retos académicos, podrían ser abrumadores."* si el estudiante tiene **necesidades educativas especiales**, *"no te esfuerces tanto en mejorar el idioma, quizás no lo necesitas tanto."* si el

estudiante necesita **apoyo en el idioma** y *"no te preocupes demasiado por tu comportamiento, es probable que no cambie mucho."* si el estudiante es **disruptivo**.

2. `OrientadorMotivador` que **siempre preguntará** *"Cuéntame, ¿qué te motiva a seguir aprendiendo?"*, siempre replicará *"Entiendo, es genial que tengas esa motivación."* y que según el estudiante que atienda **devolverá la la orientación** *"explorar proyectos más desafiantes."* si el estudiante es de **altas capacidades**, *"superar cualquier obstáculo educativo que encuentres."* si el estudiante tiene **necesidades educativas especiales**, *"sumergirte en prácticas diarias para mejorar tus habilidades lingüísticas."* si el estudiante necesita **apoyo en el idioma** y *"transformar tus desafíos en oportunidades de crecimiento personal."* si el estudiante es **disruptivo**.

Además el `Orientador` en el método `Orienta` simulará una conversación con el estudiante a modo de proceso de orientación. Que deberá seguir el siguiente formato:

```
Orientado a <Nombre Estudiante> por parte de <Nombre Orientador> ----

[<Nombre Orientador>]: Hola <Nombre Estudiante>, soy tu orientador/a <Nombre Orientador> ¿Cómo estás hoy?
[<Nombre Orientador>]: <Pregunta Orientador>
[<Nombre Estudiante>]: <Respuesta Estudiante>
[<Nombre Orientador>]: <Réplica Orientador>
[<Nombre Orientador>]: Mi consejo para ti es <Orientación>
```

Por ejemplo, el método `Orientacion` de un objeto orientadora Juana que es motivadora, tendría la siguiente conversación con María que es un objeto estudiante de apoyo en el idioma:

```
Orientado a María por parte de Juana ----

[Juana]: Hola María, soy tu orientador/a Juana ¿Cómo estás hoy?
[Juana]: Cuéntame, ¿qué te motiva a seguir aprendiendo?
[María]: Estoy mejorando mis habilidades en el idioma.
[Juana]: Entiendo, es genial que tengas esa motivación.
[Juana]: Mi consejo para ti es sumergirte en prácticas diarias para mejorar tus habilidades lingüísticas.
```

Vamos a definir la clase `Escuela` que será encargada de juntar orientadores y estudiantes y coordinar el proceso de orientación. La clase tendrá los siguientes métodos:

- `Contrata`: que contratará un orientador para la escuela.
- `Matricula`: que matriculará un estudiante en la escuela. Si no hay profesores contratados, lanzará una excepción `EscuelaException` con el mensaje *"No hay profesores contratados. Contrata profesores antes de matricular estudiantes."*.
- `OrientarEstudiantes` se encargará de orientar a los estudiantes de la escuela. Para ello, recorrerá los estudiantes matriculados y realizará la orientación con un orientador aleatorio de entre los contratados.

Por último, en el programa principal, crea un pequeño test de funcionalidad de las clases, donde al menos exista un orientador/a de cada tipo y mínimo un estudiante de cada concreción incluyendo tres estudiantes sin ningún tipo de concreción.

Criterios de calificación:

📌 **Nota:** Cualquier error de sintaxis en la definición de las clases o en el programa principal podrá suponer la calificación con **0 pts** puntos del apartado correspondiente.

- (✅ **0,5 pts**) Sigue la indentación (sangría), Llaves y nombres de identificadores de acuerdo a los convenios del lenguaje. Penaliza **-0,25** por cada error en este punto.
- (✅ **1 pts**) Cumple con TODOS las especificaciones del diagrama nombres, tipos, restricciones de acceso, etc. Penaliza **-0,25** por cada error en este punto.
- (✅ **0,5 pts**) Define cada clase en un archivo separado. Penaliza **-0,25** por cada error en este punto.
- (✅ **0,5 pts**) Sigue las especificaciones en cuanto a literales de texto y formato de visualización de la especificación. Penaliza **-0,1** por cada error en este punto.
- (✅ **1 pts**) Implementa correctamente `Estudiante` y sus concreciones.
- (✅ **1 pts**) Implementa correctamente `Orientador`.
- (✅ **1 pts**) Implementa las concreciones de `Orientador`.
- (✅ **0,5 pts**) Genera y lanza correctamente la excepción personalizada.
- (✅ **1 pts**) Definición del test de funcionalidad en el programa principal controlando la excepción personalizada.

Ejercicio 3 propuesta nueva (3 puntos)

Estás desarrollando el **módulo de cobros de una tienda online**. Dependiendo del monto de la compra y del estado del sistema, un pago puede tener tres resultados posibles, y cada uno lleva información diferente.

Debes implementar el **patrón Sum Type** para modelar el resultado de una transacción y una clase que procese dichos pagos.

Especificaciones:

1. Crea un abstract record class llamado `ResultadoPago`.
2. Definir las Variantes (Los Casos): Crea tres **records** que hereden de ResultadoPago:
   - **Aprobado**: Debe contener un `Guid` CodigoTransaccion.
   - **Rechazado**: Debe contener un `string` Motivo.
   - **EnRevision**: (Para montos sospechosos) Debe contener un `int` `NivelRiesgo` (ej. de 1 a 5).
3. Crea una **clase estática** `PasarelaDePago` con un método estático:
   `public static ResultadoPago Procesar(decimal monto)`
   Implementa la siguiente lógica simulada:
   - **Si el monto es negativo**: Retorna Rechazado con motivo "`Monto inválido`".
   - **Si el monto es mayor a 5000**: Retorna EnRevision con nivel de riesgo 5.
   - **Si el monto es mayor a 1000**: Retorna EnRevision con nivel de riesgo 1.
   - **En cualquier otro caso**: Retorna Aprobado con un Guid nuevo generadado (`Guid.NewGuid()`).
4. En el método Main:
   - Crea una lista de montos de prueba: -50, 100, 2500 y 6000.
   - Itera sobre la lista procesando cada pago.
   - Utiliza una expresión switch (Pattern Matching) sobre el resultado para imprimir un mensaje distinto en consola según el tipo, accediendo a sus propiedades internas:

- **Aprobado**: "`Pago exitoso. ID: {CodigoTransaccion}`"
- **Rechazado**: "`Error en el pago: {Motivo}`"
- **EnRevision**: "`El pago requiere auditoría. Riesgo: {NivelRiesgo}`"

> **Pistas (Por si te atascas)**
>
> 1. Recuerda usar la sintaxis `public record Nombre(...) : Base;` para las variantes.
> 2. En el switch del Main, utiliza el patrón de declaración para extraer los datos (ej: `ResultadoPago.Aprobado exito =>` ...).

Criterios de calificación:

- (✅ **1 pts**) Definir el tipo `ResultadoPago` y sus variantes correctamente.
- (✅ **1 pts**) Implementar la clase estática `PasarelaDePago` y su método `Procesar` según la lógica especificada.
- (✅ **1 pts**) Implementar el método Main con la lógica de prueba y el uso del switch con pattern matching.

Ejercicio 4 propuesta nueva (2,5 puntos)

Estás desarrollando un módulo para un sistema de gestión de almacenes (WMS). Un escáner de mano antiguo envía datos al servidor en una cadena de texto continua y "*sucia*", mezclando mensajes de error, ruido del sistema y los códigos de los productos escaneados.

Tu tarea es limpiar esta entrada y extraer estructuradamente la información de los productos válidos.

Especificaciones:

Un código de producto válido sigue estrictamente estas reglas:

1. **Código de Área**: Comienza con exactamente **dos letras mayúsculas** (Ej: WH, XY).
2. **Separador**: Un guion medio (-).
3. **Identificador**: Termina con una secuencia de **3 o 4 dígitos** (Ej: 100, 2024).

Ejemplos Válidos: `WH-100`, `IT-2024`, `ZZ-999`.
Ejemplos Inválidos: `wh-100` (minúsculas), `A-100` (una sola letra), `IT-12` (muy corto).

Escribe un programa de consola en C# que realice lo siguiente:

1. **Definición de Patrón**: Crea una expresión regular que identifique los códigos válidos.
2. **Grupos Etiquetados (Named Groups)**: La expresión regular debe capturar dos grupos con nombres específicos:
   - `Area`: Para las letras mayúsculas.
   - `ID`: Para los dígitos numéricos.
3. **Procesamiento**: Recorre la cadena de entrada proporcionada.

4. **Salida**: Para cada coincidencia, imprime en consola una línea con el siguiente formato interpolado... `[Área: <valor>] -> Producto #<valor>`

En el **programa principal del ejercicio4** que se te proporciona en los recursos, se te proporciona la siguiente cadena de prueba:

```
string rawData = """
    Start: [SYS-WARN];
    Scanned: WH-2024;
    Error: wh-99;
    Retry: LOG-101;
    Scanned: IT-55555 (Too long);
    Valid: QA-888. End.
    """;
```

Que mostrará la siguiente salida por pantalla:

```
[Área: WH] -> Producto #2024
[Área: QA] -> Producto #888
```

Criterios de calificación:

- (✅ **1 pts**) El código funciona correctamente y compila sin errores.
- (✅ **1 pts**) El patrón está definido correctamente.
- (✅ **0,5 pts**) Uso obligatorio de sintaxis de Grupos Nombrados (`?<Nombre>...`).