PRÁCTICA FINAL:

EXAMINATOR 3000

Pablo Sánchez Coria

DNI: 70924571T

Asignatura: Programación 3

1. INTRODUCCIÓN

El objetivo de esta práctica es desarrollar una aplicación de consola llamada Examinator 3000 que permita gestionar un banco de preguntas tipo test y realizar exámenes a partir de dichas preguntas.

La aplicación se ha implementado siguiendo una arquitectura MVC (Modelo–Vista–Controlador):

- Modelo (model): contiene las clases de dominio (Question, Option, Exam), el repositorio de preguntas (IRepository, BinaryRepository), el sistema de copias de seguridad (QuestionBackupIO, JSONQuestionBackupIO) y el soporte para generación automática de preguntas (QuestionCreator, GeminiQuestionCreator). Además de los archivos para controlar las excepciones.
- Vista (view): interfaz de texto que interactúa con el usuario (BaseView, InteractiveView).
- Controlador (controller): coordina la comunicación entre modelo y vista (Controller).
- Aplicación (App): punto de entrada, donde se instancian los componentes y se arranca el programa.

Luego, a partir de esta estructura, he ido desarrollando e implementando las demás funcionalidades pedidas:

1. Gestión completa del banco de preguntas (CRUD: crear, listar, modificar y eliminar).
2. Persistencia de las preguntas en un fichero binario y copias de seguridad en JSON (importar/exportar).
3. Modo examen, con selección de tema, número de preguntas, feedback y cálculo de la nota.
4. Generación automática de preguntas usando Gemini.

Esto lo he ido desarrollando en orden **para poder probar cada parte por separado**, asegurar que funcionaba correctamente antes de añadir la siguiente y mantener el código lo más claro y mantenible posible.

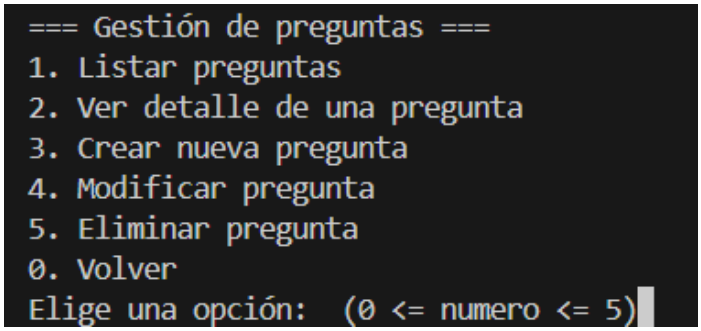
En las siguientes secciones se describe cada funcionalidad y se muestran capturas de la ejecución.

```
===== Examinator 3000 =====
1. Gestión de preguntas
2. Importar / Exportar preguntas
3. Modo examen
4. Crear pregunta automática (Gemini)
0. Salir
Elige una opción:  (0 <= numero <= 4)
```

Funcionalidad 1: Gestión de preguntas (CRUD)

Un CRUD (créate, read, update y delete) consiste en la gestión de preguntas que permite:

- Listar todas las preguntas ordenadas por fecha de creación.
- Ver el detalle de una pregunta concreta.
- Crear nuevas preguntas manualmente.
- Modificar una pregunta existente (excepto su identificador).
- Eliminar preguntas del banco.



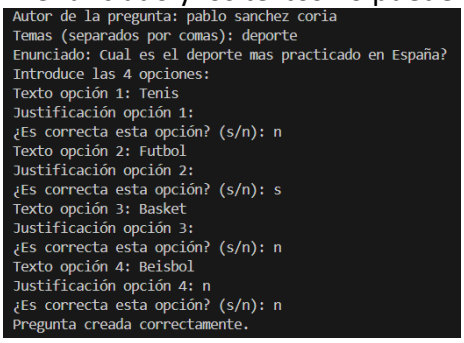
3) Crear nueva pregunta:

Al crear una pregunta, la vista solicita:

- Autor de la pregunta.
- Temas (introducidos como cadena y normalizados a mayúsculas en el modelo).
- Enunciado de la pregunta.
- Cuatro opciones de respuesta, cada una con:
  - Texto de la opción.
  - Justificación (rationale).
  - Marcado de si es correcta o no.

El modelo genera automáticamente un UUID para cada pregunta (Question.id) y realiza validaciones:

- Debe haber exactamente 4 opciones.
- Debe haber una sola opción correcta.
- El enunciado y los textos no pueden ser vacíos.



1)Listar preguntas:

El listado muestra cada pregunta en una sola línea con:

- Su UUID.
- El enunciado.

```
=== Preguntas ===
8e0596cf-fa60-4b04-acf8-aee32ea49514 - ¿Cuál de las siguientes afirmaciones describe con mayor precis
ión el transporte activo secundario a través de la membrana celular?
85814117-6314-4340-ba63-d563e2c8cbc1 - Cual es el deporte mas practicado en Espa¤a?
```

2) Ver detalle de una pregunta:

Desde ese listado, el usuario puede introducir un UUID y ver el detalle completo:

- Autor.
- Temas.
- Enunciado.
- Las 4 opciones con indicación de [CORRECTA] / [INCORRECTA] y su justificación.

```
Introduce el UUID de la pregunta: 85814117-6314-4340-ba63-d563e2c8cbc1

ID: 85814117-6314-4340-ba63-d563e2c8cbc1
Autor: pablo sanchez coria
Temas: [DEPORTE]
Enunciado: Cual es el deporte mas practicado en Espa¤a?
1. Tenis [INCORRECTA]
    Justificación:
2. Futbol [CORRECTA]
    Justificación:
3. Basket [INCORRECTA]
    Justificación:
4. Beisbol [INCORRECTA]
    Justificación: n
```

4) / 5) Modificar y Eliminar:

El usuario puede:

- Modificar cualquier campo excepto el identificador interno (UUID).
- Eliminar la pregunta del banco.

La lógica de modificación/eliminación se implementa en el controlador, que llama a los métodos del repositorio (modifyQuestion, removeQuestion) y muestra mensajes de error si se produce alguna excepción de repositorio.

```
Introduce el UUID de la pregunta: 85814117-6314-4340-ba63-d563e2c8cbc1
Pulsa Enter para dejar el valor actual.
Autor [pablo sanchez coria]:
Temas (comas) [DEPORTE]:
Enunciado [Cual es el deporte mas practicado en Espa¤a?]:
Opción 1: Tenis
Nuevo texto (Enter para mantener):
Nueva justificación (Enter para mantener):
¿Es correcta? (s/n) (Enter para mantener): n
Opción 2: Futbol
Nuevo texto (Enter para mantener):
Nueva justificación (Enter para mantener):
¿Es correcta? (s/n) (Enter para mantener): s
Opción 3: Basket
Nuevo texto (Enter para mantener):
Nueva justificación (Enter para mantener):
¿Es correcta? (s/n) (Enter para mantener):
Opción 4: Beisbol
Nuevo texto (Enter para mantener):
Nueva justificación (Enter para mantener):
¿Es correcta? (s/n) (Enter para mantener):
Pregunta modificada correctamente.
```

```
Introduce el UUID de la pregunta: 456b0e84-5ef9-4b53-abf0-7c34eafec158
¿Seguro que quieres eliminarla? (s/n): s
Pregunta eliminada correctamente.
```

Funcionalidad 2: Persistencia y copias de seguridad (binario + JSON)

Repositorio binario (BinaryRepository):

El modelo utiliza un objeto IRepository cuya implementación concreta es BinaryRepository.

- Al iniciar la aplicación se intenta leer el fichero binario questions.bin del home del usuario.
- Si el fichero existe, se deserializa la lista de preguntas y se carga en memoria.
- Al finalizar la aplicación, se vuelven a guardar todas las preguntas en el mismo fichero.

Esto permite que las preguntas se mantenga entre ejecuciones.

Exportación/Importación JSON:

```
=== Copias de seguridad ===
1. Exportar preguntas a JSON
2. Importar preguntas desde JSON
0. Volver
Elige una opción:  (0 <= numero <= 2)
```

Desde el menú de “Importar / Exportar preguntas” la aplicación permite:

1. Exportar preguntas a JSON

    - El usuario introduce un nombre de fichero (sin ruta ni extensión).
    - El modelo delega en JSONQuestionBackupIO para serializar todas las preguntas a <nombre>.json en el home del usuario.
    - Se informa cuántas preguntas se han exportado.

```
Elige una opción:  (0 <= numero <= 2)1
Nombre del fichero (sin ruta, sin extensión): preguntass
Preguntas exportadas a preguntass.json (8 preguntas).
```

2. Importar preguntas desde JSON

    - Se lee el fichero indicado.
    - Se cargan las preguntas y se insertan en el repositorio solo si su UUID no existe ya, siguiendo la política de evitar duplicados.
    - Se normalizan los temas y se valida cada pregunta antes de guardarla.
    - Se muestra cuántas preguntas nuevas se han importado.

```
Elige una opción:  (0 <= numero <= 2)2
Nombre del fichero (sin ruta, sin extensión): preguntas
Importadas 3 preguntas desde preguntas.json
```

Funcionalidad 3: Modo examen

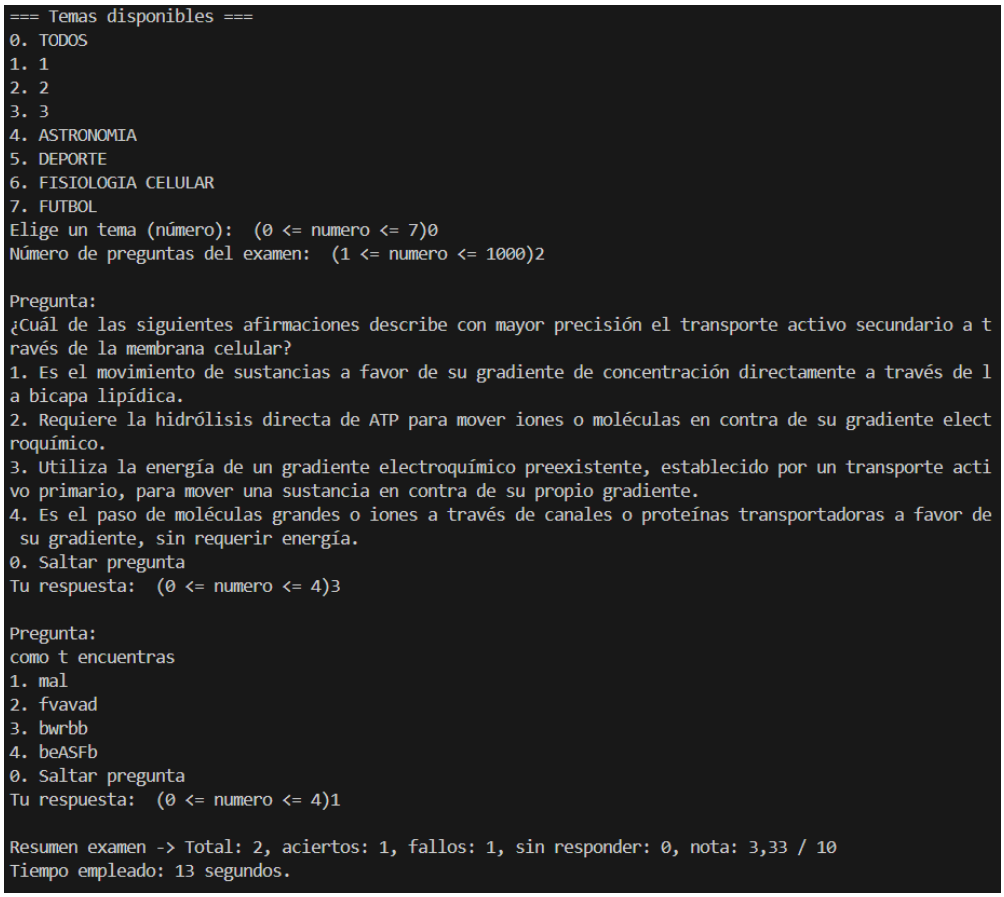
El modo examen permite al usuario hacer un test eligiendo primero un tema entre los que se obtienen dinámicamente del banco de preguntas, o la opción de Todos. Después introduce el número de preguntas del examen y el modelo crea un objeto Exam con una selección aleatoria de preguntas que cumplan ese criterio.

Durante el examen, el sistema va mostrando las preguntas una a una con su enunciado y las 4 opciones numeradas, sin indicar cuál es la correcta. El usuario puede responder indicando el número de opción o saltar la pregunta, que quedará como no contestada. La vista se apoya en el controlador, que llama al modelo mediante los métodos obtenerPreguntaActualExamen, responderPreguntaActual, saltarPreguntaActual y examenHaTerminado.

Al finalizar, el modelo calcula los aciertos, fallos, sin responder y la nota sobre 10 usando la fórmula:

nota = (aciertos − fallos / 3) × (10 / nº total de preguntas)

De este modo, cada fallo penaliza 1/3 del valor de una pregunta. Finalmente, la vista muestra un resumen con todos estos datos y el tiempo empleado.



Funcionalidad 4 : Generación automática de preguntas con Gemini

La aplicación permite crear preguntas automáticamente a partir de un tema utilizando un QuestionCreator basado en Gemini.

- En App se procesa el parámetro de línea de comandos -question-creator junto con el identificador de modelo (por ejemplo, gemini-2.5-flash) y la API key.
- Si se proporcionan correctamente, se crea un objeto GeminiQuestionCreator que se añade a la lista de QuestionCreator del modelo.
- En la vista aparece una nueva opción “Crear pregunta automática (Gemini)”.
- El usuario introduce un tema, se selecciona el creador Gemini y el sistema genera una pregunta completa (enunciado, cuatro opciones y justificaciones) siguiendo la estructura del DTO QuestionDTO.
- Esto es gracias al prompt que le pasamos a Gemini.

La integración se realiza usando:

- GenAiConfig y GenAiFacade para configurar el modelo y llamar a la API.
- SimpleSchemas y Schema para mapear la respuesta de Gemini al objeto QuestionDTO.

En caso de error de red, de cuota o de formato, se captura la excepción y se muestra un mensaje al usuario, sin detener la aplicación.

```
=== Question creators disponibles ===
1. Gemini (gemini-2.5-flash)
Elige un question creator:  (1 <= numero <= 1)1
Tema de la pregunta: informatica
Pregunta creada automáticamente.
```

```
ID: 969452d4-09f6-46e0-99b2-17e783d2588b
Autor: Gemini
Temas: [INFORMATICA]
Enunciado: ¿Cuál de los siguientes componentes del CPU es responsable de ejecutar las operaciones ari
tméticas y lógicas de un programa?
1. Unidad de Control (UC) [INCORRECTA]
    Justificación: La Unidad de Control se encarga de dirigir y coordinar las operaciones del CPU, in
terpretando las instrucciones y generando las señales de control, pero no las ejecuta directamente.
2. Unidad Aritmético-Lógica (ALU) [CORRECTA]
    Justificación: La ALU es el componente del CPU que realiza todas las operaciones aritméticas (sum
a, resta, multiplicación, división) y lógicas (AND, OR, NOT) especificadas por las instrucciones del
programa.
3. Registros [INCORRECTA]
    Justificación: Los registros son pequeñas unidades de almacenamiento de muy alta velocidad dentro
 del CPU que se utilizan para guardar datos e instrucciones temporalmente durante el procesamiento, p
ero no ejecutan operaciones.
4. Memoria Caché [INCORRECTA]
    Justificación: La memoria caché es una memoria de acceso rápido que almacena copias de datos e in
strucciones de uso frecuente para acelerar el acceso, pero no es responsable de la ejecución de opera
ciones.
```

Problemáticas, aspectos a mejorar y posibles líneas futuras

Durante el desarrollo de la práctica han surgido varios problemas y decisiones de diseño:

- Me ha costado saber como integrar y conectar alguna cosa en el modelo MVC
- Modo examen y penalización:
  Se separó la lógica de negocio del examen en la clase Exam, dejando en la vista solo la presentación por consola. También se tuvo que ajustar el cálculo de la nota para respetar la penalización de 1/3 por pregunta fallada.
- Integración con Gemini:
  La configuración de la API key y del modelo, así como la gestión de errores, ha sido una de las partes más complicadas la verdad.

Como posibles mejoras futuras:

- Guardar también los resultados de los exámenes realizados para tener un histórico por estudiante.
- Ampliar el uso de Gemini para generar no solo preguntas nuevas, sino también explicaciones más detalladas o variantes de preguntas a partir de las existentes.