

Programación III – Práctica Final

Curso 2025/2026

Autor:

Sebastián Martínez Abarca

1-1979-0679

Nombre del proyecto:

Programa

Universidad de Salamanca – Facultad de Ciencias

Introducción:

El “Programa” es una aplicación desarrollada en Java que gestiona un banco de preguntas tipo test y realiza exámenes con esas preguntas para poder obtener una serie de resultados de ese examen, el proyecto se basa completamente en la arquitectura MVC (Modelo-Vista-Controlador, lo que facilita la exportación a interfaces gráficas como TTSS.

Las funciones principales del proyecto son:

- CRUD completo de preguntas
- Persistencia automática mediante serialización
- Exportación e importación en formato JSON
- Modo examen con cálculo de nota, penalización y tiempo
- Creador automático de preguntas con Gemini

Arquitectura MVC

El diseño separa correctamente la lógica de negocio, la interacción con el usuario y el control del flujo.

```
public static void main(String[] args) {
    IRepository repository = new BinaryRepository();
    QuestionBackupIO backupIO = new JSONQuestionBackupIO();
    ArrayList<QuestionCreator> creators = new ArrayList<QuestionCreator>();

    String questionCreatorModelId = "Modelo demo local";
    QuestionCreator creator = new GeminiQuestionCreator(questionCreatorModelId);
    creators.add(creator);

    Model model = new Model(repository, backupIO, creators);
    BaseView view = new InteractiveView();
    Controller controller = new Controller(model, view);
    view.setController(controller);

    controller.start();
    controller.end();
}
```

Aquí se demuestra:

- Carga del repositorio
- Carga del creador de IA
- Inyección de dependencias
- Ensamblaje limpio del MVC

Modelo

Creacion de preguntas:

```
public Question createQuestion(String author, Set<String> topics, String statement,
List<Option> options)
    throws RepositoryException {
    HashSet<String> topicSet = new HashSet<String>();
    for (String t : topics) {
        topicSet.add(t.toUpperCase());
    }
    Question q = new Question(UUID.randomUUID(), author, topicSet, statement,
options);
    return repository.addQuestion(q);
}
```

- Normaliza temas
- Crea objeto de dominio
- Genera UUID
- Delegación al repositorio

Obtener preguntas ordenadas:

```
public List<Question> getAllQuestionsOrdered() throws RepositoryException {
    List<Question> list = repository.getAllQuestions();
    Collections.sort(list, new Comparator<Question>() {
        public int compare(Question o1, Question o2) {
            return o1.getId().compareTo(o2.getId());
        }
    });
    return list;
}
```

Filtrar por tema:

```
public List<Question> getQuestionsByTopic(String topic) throws RepositoryException {
    String t = topic.toUpperCase();
    List<Question> list = repository.getAllQuestions();
    List<Question> filtered = new ArrayList<Question>();
    for (int i = 0; i < list.size(); i++) {
        Question q = list.get(i);
        if (q.getTopics().contains(t)) {
            filtered.add(q);
        }
    }
    Collections.sort(filtered, new Comparator<Question>() {
        public int compare(Question o1, Question o2) {
            return o1.getId().compareTo(o2.getId());
        }
    });
    return filtered;
}
```

Modificar pregunta:

```
public Question modifyQuestion(Question original, String author, Set<String> topics,
String statement,
                                List<Option> options) throws RepositoryException {
    HashSet<String> topicSet = new HashSet<String>();
    for (String t : topics) {
        topicSet.add(t.toUpperCase());
    }
    Question updated = new Question(original.getId(), author, topicSet, statement,
options);
    return repository.modifyQuestion(updated);
}
```

Eliminar pregunta:

```
public void deleteQuestion(Question q) throws RepositoryException {
    repository.removeQuestion(q);
}
```

Exportar preguntas a JSON:

```
public void exportQuestions(String fileName) throws QuestionBackupIOException,
RepositoryException {
    List<Question> list = repository.getAllQuestions();
    backupHandler.exportQuestions(list, fileName);
}
```

Importar preguntas sin duplicar UUID:

```
public int importQuestions(String fileName) throws QuestionBackupIOException,
RepositoryException {
    List<Question> existing = repository.getAllQuestions();
    HashSet<UUID> ids = new HashSet<UUID>();
    for (int i = 0; i < existing.size(); i++) {
        ids.add(existing.get(i).getId());
    }
    List<Question> imported = backupHandler.importQuestions(fileName);
    int added = 0;
    for (int i = 0; i < imported.size(); i++) {
        Question q = imported.get(i);
        if (!ids.contains(q.getId())) {
            repository.addQuestion(q);
            ids.add(q.getId());
            added++;
        }
    }
    return added;
}
```

Configuración del examen:

```
public Exam configureExam(String topic, int numQuestions) throws RepositoryException {
    List<Question> base;
    if ("ALL".equals(topic)) {
        base = getAllQuestionsOrdered();
    } else {
        base = getQuestionsByTopic(topic);
    }
    if (numQuestions <= 0 || numQuestions > base.size()) {
        this.currentExam = new Exam(base);
    } else {
        List<Question> selected = new ArrayList<Question>();
        for (int i = 0; i < numQuestions && i < base.size(); i++) {
            selected.add(base.get(i));
        }
        this.currentExam = new Exam(selected);
    }
    return this.currentExam;
}
```

Corrección de respuestas:

```
public String answerCurrentExamQuestion(Integer optionIndex) {
    if (currentExam == null) {
        return "No hay examen configurado.";
    }
    return currentExam.answerCurrentQuestion(optionIndex);
}
```

Resumen del examen:

```
public String getExamResultSummary() {
    if (currentExam == null) {
        return "No hay examen.";
    }
    StringBuilder sb = new StringBuilder();
    sb.append("\n=== Resultados del examen ===\n");
    sb.append("Preguntas correctas:");
    sb.append(currentExam.getCorrectCount()).append("\n");
    sb.append("Preguntas incorrectas:");
    sb.append(currentExam.getWrongCount()).append("\n");
    sb.append("Preguntas no contestadas:");
    sb.append(currentExam.getSkippedCount()).append("\n");
    sb.append("Nota sobre 10: ").append(String.format("%.2f",
currentExam.getScoreOverTen()));
    sb.append("\n");
    sb.append("Tiempo empleado (segundos):");
    sb.append(currentExam.getSecondsElapsed()).append("\n");
    return sb.toString();
}
```

El Controlador

El controlador conecta la vista con el modelo y no tiene lógica.

La Vista

La vista es responsable de toda entrada/salida por consola (ejemplo):

```
private void menuCrud() {  
    boolean back = false;  
    while (!back) {  
        System.out.println();  
        System.out.println("== CRUD Preguntas ==");  
        System.out.println("1. Crear nueva pregunta");  
        System.out.println("2. Listar todas las preguntas");  
        System.out.println("3. Listar preguntas por tema");  
        System.out.println("0. Volver");  
        int option = readInt("Elige una opcion: ");  
        switch (option) {  
            case 1:  
                createQuestionFlow();  
                break;  
            case 2:  
                listQuestionsFlow(false);  
                break;  
            case 3:  
                listQuestionsFlow(true);  
                break;  
            case 0:  
                back = true;  
                break;  
            default:  
                showErrorMessage("Opcion no valida.");  
        }  
    }  
}
```

```
}
```

Persistencia

Carga automática desde questions.bin

```
package model;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.List;

public class BinaryRepository implements IRepository {

    private ArrayList<Question> questions;
    private File file;

    public BinaryRepository() {
        this.questions = new ArrayList<Question>();
        String home = System.getProperty("user.home");
        this.file = new File(home, "questions.bin");
    }

    public int load() throws RepositoryException {
        if (!file.exists()) {
            this.questions = new ArrayList<Question>();
            return 0;
        }
        try {
            FileInputStream fis = new FileInputStream(file);
            ObjectInputStream ois = new ObjectInputStream(fis);
            Object obj = ois.readObject();
            ois.close();
            if (obj instanceof ArrayList) {
                this.questions = (ArrayList<Question>) obj;
            } else {
                this.questions = new ArrayList<Question>();
            }
            return this.questions.size();
        } catch (Exception e) {
```

```

        throw new RepositoryException("Error al leer fichero binario", e);
    }
}

public int save() throws RepositoryException {
    try {
        FileOutputStream fos = new FileOutputStream(file);
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(questions);
        oos.close();
        return questions.size();
    } catch (Exception e) {
        throw new RepositoryException("Error al guardar fichero binario", e);
    }
}

public Question addQuestion(Question q) throws RepositoryException {
    questions.add(q);
    return q;
}

public void removeQuestion(Question q) throws RepositoryException {
    questions.remove(q);
}

public Question modifyQuestion(Question q) throws RepositoryException {
    for (int i = 0; i < questions.size(); i++) {
        Question current = questions.get(i);
        if (current.getId().equals(q.getId())) {
            questions.set(i, q);
            return q;
        }
    }
    throw new RepositoryException("Pregunta no encontrada");
}

public List<Question> getAllQuestions() throws RepositoryException {
    return new ArrayList<Question>(questions);
}
}

```

Exportación/Importación JSON

Se encuentra en JSONQuestionBackupIO, se utilizó un sistema de búsqueda completa para evitar problemas a la hora de la importación.

Modo Examen

Se utilizaron lógicas de cálculo de notas y penalizaciones

Creador automático de preguntas

```
package model;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.UUID;

public class GeminiQuestionCreator implements QuestionCreator {

    private String modelId;

    public GeminiQuestionCreator(String modelId) {
        this.modelId = modelId;
    }

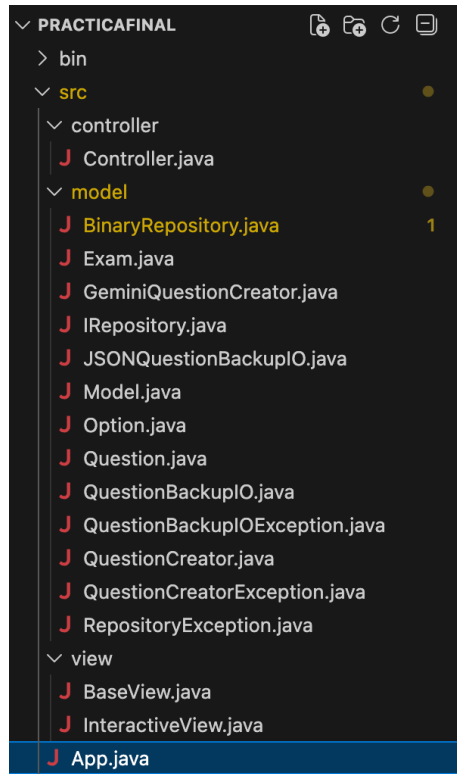
    public Question createQuestion(String topic) throws QuestionCreatorException {
        try {
            HashSet<String> topics = new HashSet<String>();
            topics.add(topic.toUpperCase());
            String author = "Gemini-" + modelId;
            String statement = "Pregunta generada automaticamente sobre el tema: " +
topic;

            List<Option> options = new ArrayList<Option>();
            options.add(new Option("Opcion 1", "Razon de ejemplo 1", true));
            options.add(new Option("Opcion 2", "Razon de ejemplo 2", false));
            options.add(new Option("Opcion 3", "Razon de ejemplo 3", false));
            options.add(new Option("Opcion 4", "Razon de ejemplo 4", false));
            return new Question(UUID.randomUUID(), author, topics, statement,
options);
        } catch (Exception e) {
            throw new QuestionCreatorException("Error creando pregunta automatica",
e);
        }
    }

    public String getQuestionCreatorDescription() {
        return "Gemini Question Creator (" + modelId + ")";
    }
}
```

Imágenes de funcionamiento:

Archivos en práctica



Menú principal

```
○ sebastianmartinez@Sebastians-MacBook-Air PracticaFINAL % /usr/bin/env /Library/Java/JavaVirtualMachines/openjdk-17.jdk/Contents/Home/bin/java -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/sebastianmartinez/Library/Application\ Support/Code/User/workspaceStorage/fai125a4236e9a0a6ecdae050a0a712b/redhat.java/jdt_ws/PracticaFINAL_34bd8699/bin App
Banco de preguntas cargado: 5 preguntas.

==== Programa ====
1. Gestion de preguntas (CRUD)
2. Exportar / Importar preguntas (JSON)
3. Crear pregunta automatica
4. Modo examen
0. Salir
Elige una opcion: █
```

CRUD de preguntas

```
== CRUD Preguntas ==
1. Crear nueva pregunta
2. Listar todas las preguntas
3. Listar preguntas por tema
0. Volver
Elige una opcion: █
```

(Opcion de listado de preguntas)

```
== CRUD Preguntas ==
1. Crear nueva pregunta
2. Listar todas las preguntas
3. Listar preguntas por tema
0. Volver
Elige una opcion: 2
1. ¿Qué estructura permite repetir un bloque de código?
2. Pregunta generada automaticamente sobre el tema: Perros
3. ¿Qué es una clase en Java?
4. ¿Para qué se utiliza la sentencia SELECT en SQL?
5. ¿Qué principio de la P00 permite ocultar detalles internos?
6. ¿Para qué se utiliza la sentencia SELECT en SQL?
7. ¿Qué estructura permite repetir un bloque de código?
8. ¿Qué principio de la P00 permite ocultar detalles internos?
9. ¿Qué es una clase en Java?
Elige numero de pregunta para ver detalle (0 para volver): █
```

Listado de preguntas

Elige numero de pregunta para ver detalle (0 para volver): 1

Id: a0d337fa-6b87-4f98-aa27-24daecb0a233

Autor: Sistema

Temas: [ALGORITMOS]

Enunciado: ¿Qué estructura permite repetir un bloque de código?

1) if [incorrecta]

Razon: if es para decisiones.

2) switch [incorrecta]

Razon: switch es selección múltiple.

3) while [correcta]

Razon: while repite hasta que una condición deje de cumplirse.

4) break [incorrecta]

Razon: break termina un ciclo.

1. Modificar pregunta

2. Eliminar pregunta

0. Volver

Elige una opcion:

Opción del menú (2)

Exportación

```
==== Programa ====
1. Gestion de preguntas (CRUD)
2. Exportar / Importar preguntas (JSON)
3. Crear pregunta automatica
4. Modo examen
0. Salir
Elige una opcion: 2

== Backup JSON ==
1. Exportar preguntas a JSON
2. Importar preguntas desde JSON
0. Volver
Elige una opcion: 2
Nombre de fichero JSON (sin ruta): preguntas_examinator_programa
Leyendo JSON desde: /Users/sebastianmartinez/Library/Containers/com.apple.CloudPhotosConfiguration/Data/Downloads/preguntas_examinator_programa.json
Importadas 4 preguntas nuevas.

== Backup JSON ==
1. Exportar preguntas a JSON
2. Importar preguntas desde JSON
0. Volver
Elige una opcion: █
```

Importación

```
{ } PreguntasV1.json U
0. Salir
Elige una opcion: 2

== Backup JSON ==
1. Exportar preguntas a JSON
2. Importar preguntas desde JSON
0. Volver
Elige una opcion: 1
Nombre de fichero JSON (sin ruta): PreguntasV1
Exportacion completada.

== Backup JSON ==
1. Exportar preguntas a JSON
2. Importar preguntas desde JSON
0. Volver
Elige una opcion: █
```

> OUTLINE

> TIMELINE

> JAVA PROJECTS

Creación de preguntas automaticas

```
== Creacion automatica de preguntas ==
1. Gemini Question Creator (Modelo demo local)
Elige un creador (0 para volver): 1
Tema para la pregunta: Perros
Pregunta generada:

Id: e6afc034-3757-4b3a-9b7a-5a35a1d6b150
Autor: Gemini-Modelo demo local
Temas: [PERROS]
Enunciado: Pregunta generada automaticamente sobre el tema: Perros
1) Opcion 1 [correcta]
   Razon: Razon de ejemplo 1
2) Opcion 2 [incorrecta]
   Razon: Razon de ejemplo 2
3) Opcion 3 [incorrecta]
   Razon: Razon de ejemplo 3
4) Opcion 4 [incorrecta]
   Razon: Razon de ejemplo 4
Anadir al banco de preguntas? (s/n): s
Pregunta automatica anadida al banco.
```

Resultados de examen en modo examen

```
=== Resultados del examen ===
Preguntas correctas: 1
Preguntas incorrectas: 1
Preguntas no contestadas: 0
Nota sobre 10: 3.33
Tiempo empleado (segundos): 6

==== Programa ====
1. Gestion de preguntas (CRUD)
2. Exportar / Importar preguntas (JSON)
3. Crear pregunta automatica
4. Modo examen
0. Salir
Elige una opcion: █
```

Conclusión y problemáticas

En las problemáticas:

- Se podría incluir una interfaz gráfica
- Incluir más modelos de IA
- Añadir una mezcla aleatoria de preguntas

En conclusión, el proyecto cumple con las funciones esenciales del enunciado del proyecto, se mantiene la arquitectura del modelo MVC, facilitando la incorporación en nuevas vistas o sistemas de almacenamiento.