Sistema: “Plataforma de Aprendizaje en Línea”

La facultad quiere una plataforma que gestione:

- Cursos, estudiantes y docentes
- Comunicación en tiempo real
- Módulos de ejercicios y evaluaciones
- Reportes y análisis de datos

Objetivos de aprendizaje

- Comprender el problema que resuelve cada patrón.
- Implementar en Java las estructuras de clases necesarias.
- Practicar principios de diseño: separación de responsabilidades, extensibilidad y mantenimiento.

Actividades obligatorias por patrón

1. Chain of Responsibility

Enunciado: Implementar un sistema de solicitud de tutorías donde las solicitudes son revisadas por distintos niveles: Asistente, Profesor, Coordinador.

- **Ayuda:** Crear interfaz Handler con métodos setNext(Handler next) y handle(Solicitud s). Clases concretas: Asistente, Profesor, Coordinador.

---

2. Command

Enunciado: Implementar un sistema donde un alumno puede enviar comandos para registrarse en cursos, abandonar un curso o solicitar certificado.

- **Ayuda:** Crear interfaz Command con método execute(). Implementar InscribirseCursoCommand, AbandonarCursoCommand, SolicitarCertificadoCommand. Crear un Invoker para ejecutar los comandos.

---

3. Iterator

Enunciado: Recorrer la lista de cursos inscritos por un alumno.

- **Ayuda:** Crear clase Alumno con lista de Curso. Implementar CursoIterator con métodos hasNext() y next().

---

4. Mediator

Enunciado: Gestionar la comunicación entre alumnos y profesores a través de un sistema de mensajería interna.

- **Ayuda:** Crear interfaz ChatMediator con método enviar(String msg, Usuario u). Clases Alumno y Profesor se registran en ChatRoom.

---

5. Memento

Enunciado: Permitir que un alumno guarde el progreso de un examen en línea y pueda restaurarlo más tarde.

- **Ayuda:** Crear clase Examen con métodos save() y restore(Memento m). Crear clase Memento que guarde el estado de las respuestas.

---

6. Observer

Enunciado: Notificar a los alumnos cuando hay cambios en horarios de clases o nuevos avisos de cursos.

- **Ayuda:** Crear interfaz Observer con método update(String msg). Clase Curso mantiene lista de observadores (Alumno) y notifica cambios.

---

7. State

Enunciado: Gestionar el estado de inscripción de un alumno en un curso: Inscrito, EnEspera, Cancelado.

- **Ayuda:** Crear interfaz EstadoInscripcion con método cambiarEstado(). Crear clases concretas Inscrito, EnEspera, Cancelado. La clase Inscripcion mantiene el estado actual.

---

8. Strategy

Enunciado: Calcular la nota final de un alumno usando distintas estrategias: promedio simple, ponderado o con examen extra.

- **Ayuda:** Crear interfaz CalculoNota con método calcular(List<Integer> notas). Implementar PromedioSimple, PromedioPonderado, ExamenExtra. La clase Alumno debe usar una estrategia configurable.

---

9. Template Method

Enunciado: Generar reportes de desempeño académico siguiendo pasos comunes: encabezado, contenido y pie, pero permitiendo personalizar el contenido según tipo de curso.

- **Ayuda:** Crear clase abstracta ReporteAcademico con método generarReporte(). Subclases ReporteCurso y ReporteAlumno implementan pasos específicos.

---

10. Visitor

Enunciado: Aplicar descuentos o becas a distintos tipos de alumnos sin modificar las clases de los alumnos.

- **Ayuda:** Crear interfaz Visitor con métodos visitar(AlumnoRegular a), visitar(AlumnoBecado a). Cada alumno tiene método aceptar(Visitor v). Implementar clase AplicarBeca que calcula beneficios.

Requerimientos técnicos

- Proyecto en **Java 8+**.
- Puede usarse Maven o Gradle.
- Organizar el código en **paquetes por funcionalidad** (ej. mediator, observer, etc.).
- Deben incluir un único Main con ejemplos que muestren el funcionamiento de **cada patrón** ejecutando el código que corresponda.

Entregables

1. Código fuente completo (repositorio Git).
2. En la clase Main explique por medio de comentarios **qué patrón está implementado y en qué parte del código**.