UNIVERSIDAD POPULAR DEL CESAR
ESPECIALIZACION DE INGENIEIRA DE SOFTWARE
GUÍA DE ACTIVIDADES PARA LA CONSTRUCCIÓN DE CONOCIMIENTO

NOMBRE DE LA ASIGNATURA	Patrones De Diseño de Software							
NOMBRE DE LA ACTIVIDAD	Patrones de diseño creacionales							
TIPO DE ACTIVIDAD	Sincrónica		Asincrónica	x	Individual	x	Grupal	

TEMÁTICA REQUERIDA PARA LA ACTIVIDAD	**OBJETIVOS**
Unidad 3. Patrones estructurales	Evaluar la capacidad para diseñar un API aplicando patrones de diseño estructurales, con el propósito de crear una solución escalable, flexible y mantenible.

COMPETENCIAS	**INSUMOS PARA EL DESARROLLO DE LA ACTIVIDAD / REFERENCIAS BIBLIOGRÁFICAS**
• Aplicación de principios de diseño SOLID	• Presentaciones unidad 1, 2, 3 • Código de ejemplo desarrollado en aula • Bibliografía recomendada

CONOCIMIENTOS PREVIOS REQUERIDOS

Conceptos fundamentales de POO, UML, Principios SOLID, Patrones de diseño estructurales

ESPECIFICACIONES DE LA ACTIVIDAD

Caso Práctico: "Proyecto StreamFlow"

La startup **StreamFlow** ha crecido exponencialmente. Originalmente era una app simple para subir videos cortos, pero ahora quiere competir con plataformas de streaming profesional (películas, música y series).

El sistema actual es un "monolito de espagueti". El equipo de producto ha lanzado nuevos requerimientos críticos para la próxima semana, pero el código actual es rígido, el rendimiento es pobre al cargar catálogos grandes y la integración con librerías de terceros es una pesadilla.

Tu grupo debe actuar como el equipo de diseño y Refactorización para salvar el proyecto aplicando Patrones Estructurales.
.

1. **Desafíos para resolver:**

Los estudiantes deben resolver 5 desafíos específicos. No se les dice explícitamente qué patrón usar en el enunciado; deben deducirlo.

- **Desafío 1: "El Caos de los Formatos"**

  Problema: El sistema usa una librería interna antigua (VideoPlayerLegacy) que solo reproduce archivos .mkv. La empresa acaba de comprar una licencia para una librería moderna (ThirdPartyVideoLib) que soporta 4K y streaming,

pero su interfaz es totalmente incompatible con la interfaz IVideoPlayer que usa nuestra aplicación. Además, marketing quiere integrar una API de videos de YouTube.

Requerimiento: Lograr que la aplicación principal reproduzca videos de la librería antigua, la nueva y YouTube usando una única interfaz común sin modificar el código de las librerías externas.

- **Desafío 2: "Listas de Reproducción Infinitas"**

  Problema: Los usuarios quieren crear "Super Playlists". Una lista de reproducción debe poder contener:

  1. Videos individuales.
  2. Canciones individuales.
  3. Otras listas de reproducción completas (que a su vez pueden tener videos o más listas).

  Requerimiento: El sistema debe tratar a un video individual y a una lista de reproducción anidada de la misma forma (ej. poder llamar a play() o getDuration() en una carpeta y que esta calcule la duración de todo su contenido recursivamente).

- **Desafío 3: "Suscripciones y Decoraciones"**

  Problema: El modelo de negocio cambió. Ahora el video base es gratis, pero se aplican modificaciones según la suscripción:

  1. Usuario Free: El video se interrumpe con anuncios.
  2. Usuario Básico: El video tiene marca de agua.
  3. Usuario Premium: Video en HD + Subtítulos.

  Requerimiento: Necesitamos agregar estas características al objeto Video en tiempo de ejecución y en cualquier orden (ej. un video podría tener Anuncios Y Marca de Agua, o solo Subtítulos). La herencia (crear VideoConAnuncioYMarcaDeAgua) está prohibida por explosión de clases.

- **Desafío 4: "La Carga Pesada"**

  Problema: Al entrar al catálogo, el sistema intenta cargar el objeto VideoHighRes completo para miles de películas. Esto incluye cargar el buffer de memoria del archivo de 4GB. El servidor se cae por falta de memoria RAM.

  Requerimiento: Al listar el catálogo, se debe mostrar el objeto video, pero no se debe cargar el archivo pesado en memoria hasta que el usuario realmente haga clic en "Play". Mientras tanto, debe mostrar una imagen ligera o un placeholder.

- **Desafío 5: "El Complejo Subsistema de Codificación"**

  Problema: Cuando un usuario sube un video, ocurren 10 pasos complejos: (Analizar archivo, Limpiar ruido audio, Corregir color, Comprimir a MP4, Generar Thumbnails, Subir a S3, Actualizar DB, Notificar usuario). El controlador del frontend tiene 200 líneas de código solo llamando a estas clases desordenadas.

  Requerimiento: Proveer una interfaz simple VideoConverter.process(file) que oculte toda esta complejidad al programador del frontend.

- **Desafío 6: "Expansión Multi-Plataforma" (Desafío de Arquitectura)**

Situación Actual: StreamFlow ha decidido lanzar aplicaciones nativas para Windows, Linux y SmartTVs. Actualmente, el sistema de renderizado de la interfaz gráfica es un lío de herencia. Tenemos clases como: WindowsMovieView, LinuxMovieView, TVMovieView, WindowsLiveStreamView, LinuxLiveStreamView,TVLiveStreamView

El Problema: Estamos sufriendo una explosión cartesiana de clases. Tenemos dos dimensiones que cambian independientemente:

- El tipo de vista (Vista de Película, Vista de Transmisión en Vivo, Vista de Tráiler).
- La plataforma sobre la que corre (Windows con DirectX, Linux con OpenGL, TV con WebOS).

Si queremos agregar un nuevo tipo de vista (ej. "Vista de Realidad Virtual") o una nueva plataforma (ej. "Android"), tendríamos que crear docenas de nuevas subclases combinadas.

Requerimiento: Refactoriza la arquitectura para **desacoplar** la abstracción (el tipo de vista funcional) de su implementación (el motor gráfico de la plataforma). Debes permitir que el desarrollo de las *Vistas* y el desarrollo de los *Drivers de Plataforma* evolucionen por separado. Un cambio en el motor de Windows no debería afectar a la clase MovieView.

2. Dinámica del Taller:

- Dividir a los estudiantes en grupos de 2 "Diseñadores".
- Deben dibujar un diagrama UML proponiendo un esquema de la solución para cada desafío. No necesariamente debe ser un diagrama de clases detallado, pero si un bosquejo donde se represente el problema actual y la propuesta de solución
- Deben justificar la elección del patrón estructural seleccionado, qué principio SOLID están protegiendo (ventajas), los restos para su implementación.

3. Entregables Esperados

- **Diagrama:** Mostrando la propuesta de solucion a implementar.
- **Sustentación Breve:** (10 min max) Explicando ventajas y desventajas encontradasde la solucion propuesta.

4. Criterios de Evaluación

Criterio	Puntos	Descripción
Selección del Patrón	40%	¿Eligieron el patrón estructural correcto para el problema?
Implementación conceptual	30%	¿la propuesta de solución funcionara? ¿Usa interfaces/clases abstractas correctamente?
UML y Comunicación	20%	Claridad en la explicación y diagramas.

RECOMENDACIONES / OBSERVACIONES	Para el diseño UML del Diagrama de clases se sugiere utilizar cualquiera de las siguientes herramientas: StartUML, PlantUML, Draw.io o Visual Paradigm. Puede utilizar el lenguaje de programación de su preferencia, preferiblemente Java, es opcional el desarrollo de interfaces Graficas de Usuario (front-end)

Elaboro: Ing. Jairo Seoanes, Msc Ingeniería de Sistemas y Computación