TALLER 2

TECNICAS DE PROGRAMACIÓN

INGENIERIA DE SISTEMAS

UNIVERSIDAD DE ANTIOQUIA
1803

Desarrollo de manera individual, Fecha de Entrega Github 15 de octubre de 2025 23:59

Punto 1. Imagina que trabajas en la modernización de los sistemas de transporte público masivo en una de las principales ciudades de Colombia (como el TransMilenio, Metro, o MÍO). El sistema debe ser flexible para incorporar diferentes tipos de vehículos y tarifas.

Objetivo del Ejercicio: Comprender y aplicar la Interface para establecer un contrato común y el Polimorfismo para gestionar diferentes implementaciones de ese contrato, creando un sistema de tarifas adaptable.

Fase 1: Creación de la Interfaz (El Contrato Común): La entidad de transporte necesita garantizar que cualquier vehículo en el sistema pueda cobrar una tarifa y mostrar su ruta. Esto es un contrato que todos deben cumplir, sin importar si son buses articulados o teleféricos.

Define la Interfaz: Crea una interfaz llamada SistemaTarifario. Esta interfaz debe tener al menos dos métodos públicos:

> calcularTarifa(distancia): Un método que acepta un valor (distancia en kilómetros o estaciones) y devuelve un valor numérico (la tarifa a cobrar).
>
> mostrarRuta(): Un método que no acepta argumentos y muestra un mensaje con la ruta actual del vehículo.

Fase 2: Implementación de Clases (Las Implementaciones Específicas) Ahora, implementaremos diferentes tipos de vehículos, cada uno con su propia lógica de tarifa, pero todos obligados a cumplir el contrato de la interfaz SistemaTarifario.

> Clase BusArticulado: Implementa la interfaz SistemaTarifario.
>
> En calcularTarifa(distancia): La tarifa es fija e independiente de la distancia, por ejemplo, $2950 COP. Ignora el valor de distancia para este ejemplo.
>
> En mostrarRuta(): Muestra "Ruta troncal (estándar de $2950 COP)".
>
> Clase Teleferico: (Para zonas de difícil acceso) Implementa la interfaz SistemaTarifario.
>
> En calcularTarifa(distancia): La tarifa es variable. Por ejemplo, un costo base de $1000 COP más $500 COP por kilómetro o estación de distancia.

En mostrarRuta(): Muestra "Ruta de conexión veredal (tarifa variable, base $1000 COP)".

Fase 3: Aplicación del Polimorfismo (La Central de Control)

En la central de control del transporte, necesitamos procesar el cobro y la información de la ruta sin saber de antemano qué tipo de vehículo estamos manejando (Bus o Teleférico). Solo sabemos que es un objeto que cumple el contrato SistemaTarifario.

Crea una Función Polimórfica: Crea una función o método llamado procesarViaje(vehiculo, km) que acepte como argumento un objeto de tipo SistemaTarifario (¡aquí está la clave del polimorfismo!).

Ejecuta el Viaje: Dentro de esta función, llama a los métodos del objeto pasado:

vehiculo.mostrarRuta()

Imprime el resultado de vehiculo.calcularTarifa(km)

Prueba Final:

Crea una instancia de BusArticulado.

Crea una instancia de Teleferico.

Llama a procesarViaje con cada una de las instancias, por ejemplo:

procesarViaje(miBus, 5)

procesarViaje(miTeleferico, 2)

ENTREGABLES:

1. Código de la implementación
2. Respuesta a las siguientes preguntas
    a. ¿Qué pasaría si no hubiéramos usado la interfaz SistemaTarifario? ¿Cuántas funciones procesarViaje habríamos tenido que crear?
    b. Si mañana se incorpora un TrenLigero con una nueva lógica de tarifa, ¿qué archivos o clases tendrías que modificar en el sistema de control (Fase 3)?
    c. ¿Cuál es la diferencia fundamental en la intención entre la interfaz y las clases concretas? (La interfaz define qué se debe hacer, las clases concretas definen cómo se hace.)

TALLER 2

TECNICAS DE PROGRAMACIÓN

INGENIERIA DE SISTEMAS

UNIVERSIDAD DE ANTIOQUIA
1803

Punto 2. Con el tema que usted eligió realice lo siguiente.

Objetivo del Ejercicio: Diseñar y construir una aplicación con interfaz gráfica en Swing utilizando el patrón de diseño **Modelo-Vista-Controlador (MVC)**, a partir del tema seleccionado por el estudiante.

Estructura MVC y Requisitos en Java

El estudiante debe crear al menos **tres clases principales** que representen cada parte del patrón:

1. Modelo (Model) El Modelo es responsable de la lógica de negocio, la gestión de los datos y el estado de la aplicación. No debe tener ninguna dependencia de la Vista ni del Controlador. Ejemplo:

Clase	**Responsabilidad**	**Requisitos**
Modelo (Ej: LibroModelo)	Contener la información (atributos) y la lógica para manipularla.	- Debe tener atributos privados (ej: titulo, autor, isbn). - Debe incluir un constructor para inicializar los datos. - Debe incluir **métodos *getter*** y ***setter*** públicos para acceder a los datos.
DAO (Opcional, pero recomendado para MVC)	Clase que simula la *persistencia de datos* (ej: una ArrayList en memoria).	- Una lista (ej: ArrayList) para guardar múltiples objetos del [Tema]Modelo. - Métodos para **agregar** (agregarElemento), **consultar** (obtenerTodos), y **actualizar** (actualizarElemento).

2. Vista (View)

La Vista es lo que interactúa con el usuario (muestra datos y captura entradas). No debe tener lógica de negocio. Simplemente solicita datos al usuario y muestra la información que recibe del Controlador.

Clase	**Responsabilidad**	**Requisitos**
Vista (Ej: LibroVista)	Manejar la **entrada/salida** por consola (usando Scanner para la entrada).	- Métodos para **mostrar el menú** de opciones. - Métodos para **solicitar datos** al usuario (ej: solicitarTitulo()). - Métodos para **imprimir** los resultados o mensajes de estado (ej: mostrarDetalle(), mostrarListaCompleta()).

TALLER 2

TECNICAS DE PROGRAMACIÓN

INGENIERIA DE SISTEMAS

UNIVERSIDAD DE ANTIOQUIA
1803

3. Controlador (Controller)

El Controlador actúa como el **intermediario**. Recibe las peticiones del usuario (vía la Vista), llama a la lógica del Modelo para procesar la petición, y luego le indica a la Vista qué información debe mostrar.

Clase	**Responsabilidad**	**Requisitos**
Controlador (Ej: LibroControlador)	Conectar el Modelo y la Vista. **Contiene la lógica de flujo** de la aplicación.	- Debe tener **instancias** del Modelo y la Vista (Composición). - Un método principal (ej: iniciarAplicacion()) que maneja el **bucle de la aplicación** (mostrar menú, leer opción, llamar a los métodos del Modelo, y luego a los métodos de la Vista para mostrar resultados).

Entregable

- *Implemente un flujo que demuestre la interacción de las tres capas.*

    1. **Bucle Principal:** En el Controlador, implementa un bucle while que llama al método de la Vista para mostrar el menú, lee la opción del usuario y ejecuta la acción correspondiente.
    2. **Flujo de Creación (Ej: "Agregar un Libro"):**
        - El **Controlador** llama a la **Vista** para que pida al usuario el título, autor, etc.
        - El **Controlador** toma esos datos y crea un nuevo objeto **Modelo** (new LibroModelo(...)).
        - El **Controlador** llama al método agregarElemento() del **Modelo** (LibroDAO) para guardarlo.
        - El **Controlador** llama a la **Vista** para mostrar el mensaje de éxito.
    3. **Flujo de Consulta (Ej: "Ver todos los Libros"):**
        - El **Controlador** llama al método obtenerTodos() del **Modelo**.
        - El **Controlador** recibe la lista de objetos **Modelo**.
        - El **Controlador** llama al método mostrarListaCompleta() de la **Vista**, pasándole la lista recibida.

- ***Responda las siguientes preguntas***

TALLER 2

TECNICAS DE PROGRAMACIÓN

INGENIERIA DE SISTEMAS

1. Si quisiera cambiar la aplicación para que funcione con una **base de datos real** en lugar de una ArrayList, ¿qué capa se modificaría **principalmente**?
2. Si quisiera cambiar la aplicación para usar una **interfaz gráfica** (GUI) creada en otro tipo de lenguaje más elaborado ¿qué capa se modificaría **principalmente y que reutilizar de lo implementado**?
3. ¿Podría el **Modelo** llamar directamente a un método de la **Vista** para mostrar un error? ¿Por qué sí o por qué no?

BONUS

1. **Implementa gestión de Excepciones**
2. **Implementa formateo de datos**