Trabajo Práctico 2: Armado de recital (Grupo Mu)

Integrantes:

- Mangalaviti Sebastian 45.233.238
- Dario Aguirre 44.355.010
- Facundo Caceres 45.747.823
- Franco Benvenuto 44.760.004

Docentes:

- Federico Gasior
- Hernán Lanzillotta
- Lucas Videla

Comisión :

- 02 - Martes Turno Noche

ÍNDICE:

Introducción:

El siguiente informe describe el desarrollo del Trabajo Práctico N°2, El objetivo del mismo es la aplicación de los conocimientos sobre la Programación Orientada a Objetos (POO), utilizando los conceptos utilizados en clase. El mismo resolverá un problema sobre la organización de un recital por parte de una discográfica.

A partir de la consigna se desarrollará un programa el cual tendrá la capacidad de :

- Analizar la información sobre los artistas y las canciones pertenecientes al recital
- Determinar roles o artistas faltantes para las canciones del recital
- Realizar contrataciones para los puestos faltantes del recital ya sea de artistas que están en el mismo o artistas invitados
- Integrar reglas mediante Prolog para complementar el modelo orientado a objetos

La solución aplica el uso de clases, objetos, colecciones y archivos JSON. buscando que el sistema sea extensible a futuro

Objetivo General:

Desarrollar un sistema orientado a objetos que modele el problema de armar una “banda temporal” para un recital, utilizando artistas base y artistas externos, de forma tal que sea posible cubrir todas las canciones requeridas con el menor costo posible.

Desarrollo:

En esta sección se describe el diseño e implementación del sistema desarrollado para resolver el problema planteado en la consigna: la organización de un recital a partir de artistas base y artistas contratados, minimizando costos y respetando restricciones de roles y disponibilidad.

Organización en paquetes y capas:

El proyecto se estructura principalmente en cuarto paquetes principales:

- .APP

En esta parte del trabajo se contiene la clase “MAIN”, desde aqui se inicializar los servicios,se cargan los archivos de entrada y se invocan las operaciones principales

- .MODEL

Dentro del “Model” se incluyen las clases principales del modelo de dominio, estas son, Artista,Canción,Recital y Asignación.Estas clases representan directamente los conceptos definidos en el glosario de la consigna.

- .SERVICE

Contiene la clase “Productora”, la cual actúa como servicio de negocio, está centraliza la lógica para:

1. Analizar roles faltantes.
2. Decidir contrataciones.
3. Registrar entrenamientos.
4. Mantener el estado del recital.

- .UTIL

Incluye clases auxiliares como “LectorJSON” encargada de la escritura y lectura de archivos JSON, “ServiciosProlog” responsable de la comunicación con el servicio del motor Prolog.

Modelo de dominio:

- **Artista** : Modela cada músico disponible para el recital, los mismos se resuelven a partir de los archivos “artistas.json” y “artistas-discográfica.json”, los mismos cuentan con los siguientes datos:

    - Nombre: Identifica al artista
    - Roles: Lista de roles que puede desempeñar (ej: “Voz Principal”,”Batería”).
    - Bandas: Conjunto de bandas históricas en las que participó.
    - Costo: Costo de contratación del Artista
    - maxCanciones: Cantidad máxima de canciones que esta dispuesto a tocar en un recital.
    - indicador de si es **artista base** o **artista contratado**.

  Sobre la misma se puede:
    - Verificar si puede cubrir un rol determinado.
    - Disminuir su disponibilidad al asignarlo a una nueva canción.
    - Aplicar incrementos de costo cuando el artista es entrenado para tener nuevos roles.

- **Canción**: Representa cada tema musical del recital, la misma toma los datos del archivo “recital.json”, la misma presenta los datos:
    - Título : Nombre de la canción.
    - rolesRequeridos: lista de roles necesarios para interpretarla.

  Sobre esta se puede realizar
    - Consultar qué roles aún están libres.
    - Asociar asignaciones de artistas.
    - Determinar si una canción se considera cubierta o no.

- **Recital**: Modela el evento completo y agrupa:
    - La lista de canciones que forman parte del show.
    - Las asignaciones de artistas a cada canción y rol.
    - El costo total acumulado de las contracciones realizadas.

- **Asignación:** La misma se utiliza como entidad auxiliar para relacionar:
    - Una canción específica.
    - Un rol dentro de esa canción.
    - Y el artista que se encargará de interpretarlo.

  La función de este objeto permite registrar qué función cumple cada persona dentro de cada canción del recital permitiendo:
    - Calcular el costo total de la contratación.
    - Verificar límites de canciones por artista.
    - Listar, al final, las asignaciones efectuadas.

Principios de diseño aplicados:

En el desarrollo del sistema se aplicaron de manera explícita diversos principios de la Programación Orientada a Objetos. Estos principios fueron utilizados en el diseño de las clases, teniendo como misión el hecho de mejorar el código y volverlo mantenible.
Los principios utilizados fueron:

- **Principio de Responsabilidad Única (SRP)**: Esto quiere decir que cada clase del modelo cumple exactamente con un propósito:

    - **Artista**: representa el estado y comportamiento de un músico (roles, bandas, costo, disponibilidad).
    - **Canción**: representa únicamente una canción y sus roles requeridos.
    - **Asignación**: relación artista–canción–rol.
    - **Recital**: agrega las canciones y asignaciones del evento.
    - **Productora**: concentra la lógica del negocio sin mezclarla con datos o persistencia.
    - **LectorJSON**: única responsabilidad de leer/escribir archivos.
    - ServicioProlog: única responsabilidad de comunicarse con Prolog.

- **Encapsulamiento**: Las clases exponen solo lo necesario mediante metodos públicos y ocultan:
    - Atributos privados.
    - Listas Internas.
    - Detalles del cálculo de costo.
    - Como se determina si un artista tiene disponibilidad.
    - Cómo se actualiza el estado del recital.
- **Bajo acoplamiento:** El código está diseñado para que las clases dependan lo menos posible entre sí, un ejemplo de esto es:
    - **Productora**: utiliza interfaces de alto nivel (como listas de Artistas) pero no necesita conocer la implementación interna de cada clase.
    - **LectorJSON**: no depende de cómo se usa un artista posteriormente.

- **Composición sobre Herencia:** El modelo prioriza composición:
    - Un recital contiene muchas canciones.
    - Una canción contiene una lista de roles requeridos.
    - Un artista contiene una lista de roles y bandas.
    - Una asignación contiene referencias a Artista + Canción + Rol.
- **Principio Abierto/Cerrado:** El sistema está abierto a extensiones pero cerrado a modificaciones.
    - Se puede agregar un nuevo tipo de rol sin modificar la clase artista.
    - Se puede agregar una métrica de costo nueva sin cambiar la estructura del modelo.
    - Se pueden crear nuevos métodos de contratación sin modificar la lógica de lectura del JSON.

Estrategia de contratación de artistas:

La contratación de artistas externos es el núcleo del sistema y uno de los requisitos centrales de la consigna. La Productora debe seleccionar artistas capaces de cubrir los roles faltantes del recital, respetando restricciones y minimizando el costo total.

1. **Identificación de roles faltantes:**
    a. Se obtienen sus roles requeridos.
    b. Se analizan los artistas base y las asignaciones ya realizadas.
    c. Se eliminan de la lista de roles ya cubiertos.

    d. El resultado es la lista de roles faltantes para esa canción.

2. **Búsqueda de candidatos para cada rol: (**Para cada rol faltante**)**
    a. Pueden desempeñar ese rol.
    b. No superan su límite de maxCanciones.
    c. No se encuentran ya asignados en la misma canción.
3. **Cálculo del costo real del artista:**
    a. Calcula si tiene algún descuento a aplicar. (La verificación se realiza por la intersección entre “artistas.bandas” y “bandasDeArtistasBase”) si la intersección no es vacía se aplica el descuento.
4. **Selección del artista óptimo** (se usa el criterio del menor costo):
    a. Se obtienen todos los candidatos válidos
    b. Se calcula el costo real.
    c. Se selecciona el artista de menor costo total.
5. **Registro de la asignación** (Luego de elegir un artista para un rol):
    a. Se crea una Asignación(canción,rol,artista).
    b. Se actualiza la disponibilidad del artista:
        i. aumenta su contador de canciones asignadas.
        ii. queda bloqueado para otros roles de la misma canción.
6. **Flujo completo por canción**, (para cada canción del recital):
    a. Obtener roles faltantes.
    b. Para cada rol:
        i. Obtener candidatos.
        ii. Calcular costos reales.
        iii. Seleccionar el más económico.
        iv. Crear la asignación.
    c. Si algún rol no se puede cubrir:
        i. La canción queda “Imposible de cubrir”.
        ii. El sistema puede devolver un aviso o lanzar una excepción según la implementación.
7. **Flujo para el recital completo**, (El proceso de contratación para el recital completo sigue los mismos pasos que el caso por canción, pero ejecutado de manera secuencial para todas las canciones del evento)
    a. La disponibilidad de los artistas se va reduciendo a medida que son asignados a distintas canciones.
    b. Algunos artistas ya habrán recibido descuentos previamente, por lo que su costo actualizado se mantiene para las siguientes evaluaciones.
    c. Algunos artistas pueden llegar a su límite de participación, quedando imposibilitados de ser considerados como candidatos para canciones posteriores.
8. **Manejo de Casos límite:**
    a. Artista sin disponibilidad : Se descarta para esa canción.
    b. Ningún artista puede cubrir un rol : el sistema detiene la contratación de esa canción.
    c. Costo igual entre varios artistas: se selecciona el primero del listado.
    d. Roles duplicados: permiten asignar artistas repetidos siempre que su disponibilidad lo permita.
    e. Entrenamiento Previo : incrementa el costo de forma acumulativa

Estrategia de entrenamiento de artistas:

El sistema incluye un mecanismo de entrenamiento de artistas, cuyo objetivo es ampliar la cantidad de roles que puede desempeñar un músico. Esta función se utiliza cuando los artistas base y los artistas externos disponibles no son suficientes para cubrir todos los roles requeridos por una canción o por el recital completo.

Condiciones necesarias para entrenar a un artistas:

- No pertenece a los músicos base de la discográfica.
- No ha sido contratado previamente en el recital.
- No posee el rol que se desea agregar.

Efectos del entrenamiento:

- Incorporación del nuevo rol.
- Incremento del costo del artista.

Proceso de entrenamiento en el flujo general:

1. Se detecta un rol faltante en una canción para el cual no existe ningún artista disponible
2. La Productora analiza cuáles artistas externos pueden ser entrenados para ese rol.
3. Se selecciona un artista apto (sin restricciones violadas) y se ejecuta : entrenar(rol).
4. El artista actualiza su lista de roles y su costo con el incremento correspondiente.
5. Se vuelve a ejecutar el proceso de contratación, donde el artista entrenado ahora es considerado candidato.
6. Si su costo (incrementado) sigue siendo competitivo, se le asigna a la canción.

Requisitos del sistema y cómo fueron implementados:

A continuación se presenta un análisis comparativo entre cada requisito y la forma concreta en la que fue implementado en el proyecto.

Identificar roles faltantes para una canción:

- Se utiliza la clase Canción, que contiene los rolesRequeridos.
- La clase Productora implementa un método que compara los roles de la canción con los roles que pueden cubrir los artistas base.
- Los roles no cubiertos se devuelven como roles faltantes, usados luego en la contratación.

Identificar roles faltantes para todo el recital:

- “Productora” recorre todas las canciones del Recital.
- Para cada una ejecuta el análisis de roles faltantes.
- Se genera una estructura unificada que resume los roles faltantes a nivel global del recital.

Contratar artistas externos para cubrir roles faltantes:

- La clase Productora aplica una estrategia *greedy* de contratación.
- Filtrar candidatos por:
    - Rol disponible.
    - Límite de canciones (maxCanciones).
    - Estado del artista.
- Calcula el costo real (Incluyendo descuentos).
- Selecciona el artista de mínimo costo.

Aplicar descuentos por bandas compartidas:

- “Artista” almacena sus bandas históricas.
- “Productora” detecta coincidencias con “bandas” de artistas base.
- Si existen coincidencias el costo se reduce al 50%,
- Este cálculo se encapsula en un método del propio artista (Encapsulamiento + SRP).

Respetar el límite de canciones por artista:

- Cada “Artista” mantiene un contador interno de canciones asignadas.

- Antes de asignarlo, la Productora verifica si “cancionesAsignadas” < “maxCanciones”.
- Si supera el límite, se descarta automáticamente para el rol en cuestión.

Formación de asignaciones (artista–rol–canción):

- La clase “Asignación” modela exactamente esta relación..
- Cada vez que se contrata a un artista, se crea una instancia “Asignación”.
- El “Recital” mantiene la lista global de asignaciones.

Entrenar artistas:

- La lógica se ubica dentro de “Artista.entrenar(rolNuevo)”.
- El rol se agrega a la lista de roles del artista.
- Se actualiza el costo con incremento acumulativo.
- La Productora controla quién es elegible para el entrenamiento.

Integración con Prolog:

- Clase “ServicioProlog”: carga archivo “recital.pl”.
- Se traducen artistas y roles a hechos Prolog.
- Se consulta la regla que determina los entrenamientos mínimos.
- Se devuelve el resultado a Java para mostrarlo o utilizarlo en cálculos posteriores.

Lectura y escritura de archivos JSON:

- “LectorJSON” se encarga exclusivamente de:
    - leer artistas.json.
    - leer recital.json.
    - leer artistas-discografica.json.
    - generar recital-out.json.
- Este diseño reduce el acoplamiento y facilita futuros cambios de formato.

Resumen final del recital:

- La clase “Recita”l mantiene toda la información global.
- La Productora genera el archivo JSON final con todas las asignaciones y costos actualizados.
- Este archivo representa el estado final del recital y cumple con los requisitos de salida.

Pruebas y Validación

1. Pruebas de lectura de datos (JSON)

Se verificó que los archivos:

- artistas.json,
- recital.json,
- artistas-discografica.json

fueron correctamente interpretados por la clase LectorJSON.
Las pruebas incluyeron:

- carga exitosa de artistas con roles y bandas,
- identificación correcta de artistas base,
- carga de canciones con sus roles requeridos,
- manejo adecuado de listas y estructuras internas.

Estas pruebas confirmaron que los datos externos son traducidos de forma confiable al modelo interno.

2. Pruebas de identificación de roles faltantes

Se evaluó que el sistema determine correctamente qué roles no están siendo cubiertos por los artistas base.

Se contemplaron casos:

- canciones totalmente cubiertas,

- canciones parcialmente cubiertas,
- canciones sin ningún rol cubierto,
- canciones con roles repetidos.

El resultado fue exitoso en todos los escenarios, obteniéndose listas de roles faltantes consistentes con la lógica definida.

3. Pruebas del algoritmo de contratación

Se diseñaron pruebas para validar la selección de artistas externos bajo distintos escenarios de complejidad:

- varios artistas disponibles para un mismo rol,
- aplicación del descuento por bandas compartidas,
- comparación de costos reales,
- respeto del límite de canciones (maxCanciones),
- elección del artista más económico entre múltiples candidatos.

Se verificó además que, en caso de no existir candidatos posibles, el sistema indique correctamente que la canción es imposible de cubrir.

4. Pruebas de entrenamiento de artistas

Se evaluó que:

- los artistas elegibles cumplan con las restricciones,
- el costo se incremente correctamente en un 50% por cada rol incorporado,
- el artista adquiera el nuevo rol y pueda participar luego en la contratación,
- el entrenamiento se refleja correctamente en el estado interno del sistema.

Se realizaron pruebas con un artista:

- sin roles suficientes,
- ya entrenado,
- con múltiples entrenamientos acumulados.

5. Pruebas de integración (armado del recital completo)

Se probó el flujo completo:

1. Cargar artistas y canciones.
2. Identificar roles faltantes por canción.
3. Realizar contrataciones según costo.
4. Entrenar artistas cuando es necesario.
5. Registrar asignaciones y actualizar el recital.
6. Generar el archivo recital-out.json.

Se validó que:

- todas las canciones quedaran en estado consistente,
- cada asignación fuera válida respecto de roles y disponibilidad,
- los costos totales reflejaran correctamente contrataciones y entrenamientos.

6. Pruebas con datos de prueba de la cátedra

Además de los datos principales, se probaron:

- canciones personalizadas,
- artistas con roles ambiguos,
- casos límite intencionalmente preparados,
- Estructura alternativa de JSON.

Estas pruebas confirmaron que la solución es generalizable y robusta ante cambios en los datos.

Conclusiones Técnicas:

El desarrollo del trabajo permitió integrar conceptos principales de la Programación orientada a objetos POO, mencionadas previamente, a nivel técnico los principales resultados y conclusiones son las siguientes:

1. **Diseño modular y mantenible:**
   El uso de paquetes diferenciados (app, model, service, util) permitió establecer una arquitectura clara basada en separación de responsabilidades.
   Esto redujo el acoplamiento entre componentes y facilitó la escalabilidad del sistema.
2. **Modelo de dominio consistente:**
   Las clases “Artista”, “Canción”, “Asignación” y “Recital” representan fielmente el dominio planteado por la consigna.
   El diseño prioriza la composición sobre la herencia, logrando una estructura flexible, modificable y coherente.
3. **Integración con Prolog:**
   La comunicación con Prolog permitió complementar la solución orientada a objetos con consultas declarativas.
   La combinación con Prolog posibilitó:
   - Análisis de entrenamientos mínimos.
   - Verificación lógica independiente del modelo OO.
   - Flexibilidad ante cambios en las reglas del dominio.
4. **Uso de archivos JSON como fuente y destino:**
   La entrada de archivos JSON permitió:
   - Entrada flexible de artistas, canciones y staff base.
   - Generación del resultado final del recital.
5. **Cobertura de casos límite:**

    - roles imposibles de cubrir.
    - artistas sin disponibilidad.
    - conflictos por repetición de roles.
    - descuentos acumulados.
    - entrenamientos sucesivos.
6. **Aprendizajes del equipo:**
    - diseño modular.
    - abstracción y encapsulamiento.
    - manejo de JSON.
    - integración de paradigmas.
    - Definición de algoritmos.
    - trabajo colaborativo sobre un mismo modelo.

La solución final cumple completamente con los requisitos establecidos en la consigna y refleja un entendimiento sólido tanto del dominio del problema como de los principios fundamentales de la Programación Orientada a Objetos. El sistema obtenido es extensible, claro y capaz de adaptarse a nuevos escenarios con modificaciones mínimas.