Trabajo Práctico N° 2 – Paradigmas de Programación

Universidad Nacional de La Matanza – Ingeniería Informática

Autores:

Rafael Ruiz, 44960383
Ramiro Fariello, 46124109
Santiago Ferreyra, 42956230

2DO CUATRIMESTRE 2025

1. INTRODUCCION

El presente informe describe el desarrollo del Trabajo Práctico cuyo objetivo principal fue integrar diferentes paradigmas a través de un caso práctico: la organización de un recital donde deben asignarse artistas a roles musicales considerando restricciones de disponibilidad, costo, habilidades y compatibilidad.

El trabajo incluyó el diseño de clases en Java basadas en el paradigma orientado a objetos, y la aplicación de pruebas unitarias con JUnit para validar la lógica del programa.

2. Desarrollo

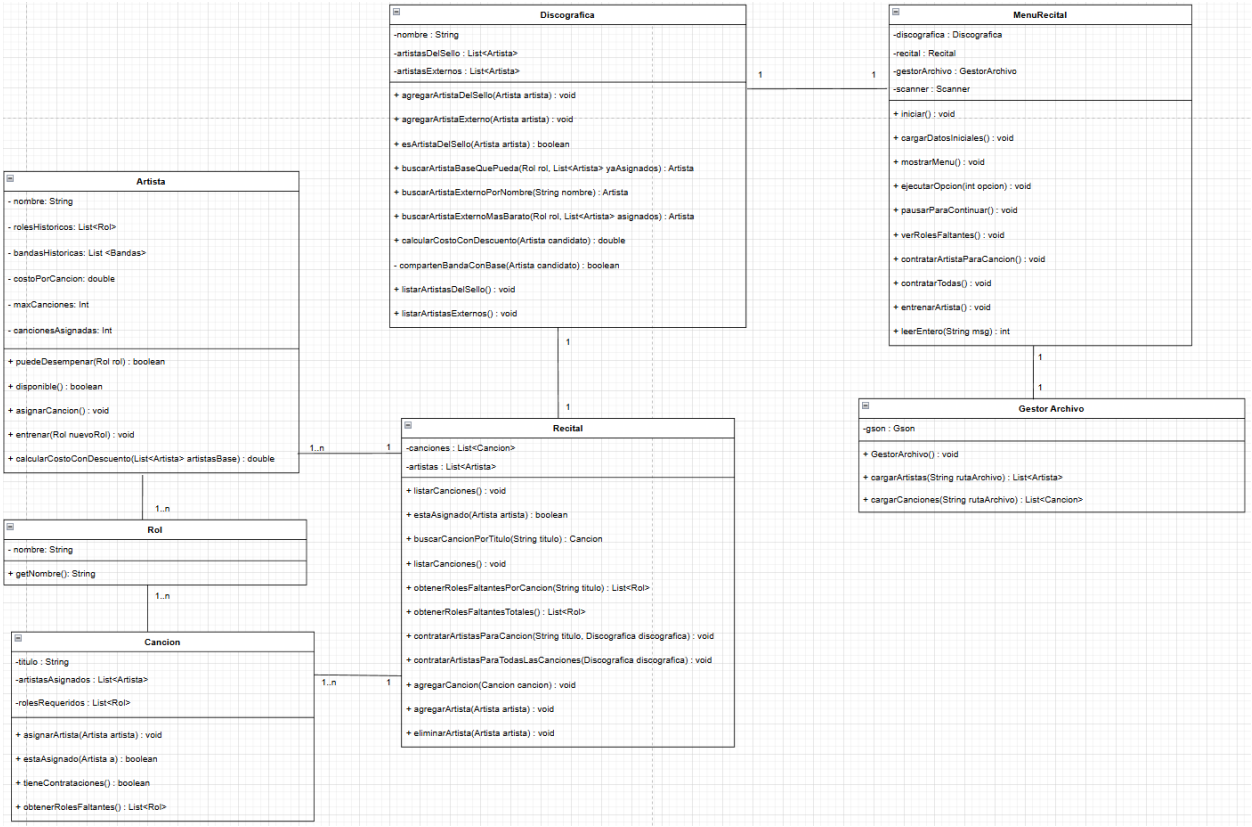
2.1. Planteamiento del problema

Se debía modelar un sistema para gestionar la contratación y utilización de artistas en un recital. Cada artista cuenta con:

- Roles musicales que puede desempeñar.
- Bandas históricas donde participó.
- Un costo por canción.
- Un máximo de canciones que puede interpretar.
- La posibilidad de ser entrenado para aprender nuevos roles *(excepto los artistas base)*.

Además, cada canción requiere un conjunto de roles, y un artista no puede desempeñar más de un rol por canción.

2.2. Diseño de clases en Java



El trabajo fue diseñado con orientación a objetos. Las clases principales fueron:

- **Artista**: contiene nombre, roles, bandas, costo, límite de canciones y métodos de disponibilidad, entrenamiento y cálculo de costo.
- **Rol**: representa un rol musical.
- **Canción**: contiene título y roles requeridos.
- **Discográfica:** administra listas de artistas del sello, artistas contratados, y las operaciones principales.

Se contemplaron decisiones de diseño como:

- Diferenciar artistas *base* (costo 0, no entrenables) de artistas contratados.

- Separar responsabilidades en métodos claros: `puedeDesempenar`, `asignarCancion`, `disponible`, `entrenar`, `calcularCostoConDescuento`.
- Mantener la cohesión y evitar clases innecesarias (como la clase *Contratación*, que se descartó por no aportar funcionalidad significativa).

2.3. Gestión de artistas, roles y canciones

Se garantizó que:

1. Un artista no pueda ser asignado más veces que su disponibilidad.
2. Los artistas se puedan entrenar y su entrenamiento aumenta el costo un 50%.
3. Se aplique descuento del 50% si el artista comparte banda histórica con alguno de los artistas base.
4. Un artista no pueda ocupar más de un rol en una misma canción.

Estas reglas fueron esenciales para el modelado del problema real y luego para las pruebas unitarias.

2.4. Lógica de asignación y restricciones

El núcleo del trabajo consistió en:

1. Calcular el costo total de contratación.
2. Detectar cuando un artista necesita ser entrenado para cubrir un rol faltante.
3. Asegurar que se cumplan todas las restricciones planteadas por la consigna.
4. Generar estrategias para cubrir roles con el mínimo costo y entrenamiento.

2.5. Pruebas unitarias con JUnit

Pruebas unitarias
Para validar el correcto funcionamiento del sistema, se desarrollaron pruebas unitarias utilizando JUnit 5.
En lugar de agrupar todas las pruebas en un único archivo, se implementó un archivo de test por cada clase principal del proyecto, siguiendo buenas prácticas de organización y mantenimiento del código.

Los archivos de prueba creados fueron:

- ArtistaTest.java
- RecitalTest.java
- DiscograficaTest.java
- CancionTest.java

Cada uno de estos archivos contiene casos de prueba diseñados para verificar el comportamiento de los métodos públicos, incluyendo creación de objetos, validaciones internas, operaciones de agregación y consultas sobre las estructuras del dominio.

Esta separación mejora la legibilidad, facilita el aislamiento de errores y permite ejecutar pruebas de forma modular.

2.6. Clase *GestorArchivo*

Se creó una clase `GestorArchivo` encargada de:

1. Leer archivos JSON desde `resources`.
2. Convertirlos a listas de objetos (Artista, Canción, etc.).

Esto permitió desacoplar completamente la lógica del programa del manejo de persistencia.

2.7. Uso de la librería Gson

Se utilizó **Google Gson** para:

1. Deserializar listas de artistas y canciones desde JSON.
2. Mapear automáticamente estructuras complejas (listas de roles dentro de artistas).

3. Simplificar el código evitando parseos manuales.

3. Conclusiones

El trabajo permitió aplicar de manera integrada conceptos fundamentales de programación orientada a objetos en Java, utilizando clases, encapsulamiento, composición y especialización para modelar entidades del dominio como artistas, canciones, discográficas y recitales.

Asimismo, se empleó programación imperativa y estructurada para implementar la lógica interna del sistema, incluyendo validaciones, asignaciones, recorridos de listas y control de restricciones.

Se cumplieron los objetivos planteados en la consigna:

- Se modelaron correctamente los artistas, los roles, las canciones, la discográfica y el recital.
- El sistema fue validado mediante **pruebas unitarias independientes** para cada clase (ArtistaTest.java, RecitalTest.java, DiscograficaTest.java y CancionTest.java), lo cual garantizó el correcto funcionamiento de los métodos y las interacciones.

El proyecto permitió comprender en profundidad cómo combinar diseño orientado a objetos con un enfoque imperativo para construir soluciones modulares, verificables y extensibles. Cada parte del sistema aporta claridad, organización y robustez al modelo implementado.

4. Referencias

Oracle. (2024). Java SE Documentation. https://docs.oracle.com/en/java/

Google. (2024). Gson user guide. https://github.com/google/gson

JUnit Team. (2024). JUnit 5 documentation. https://junit.org