Grupo Hello World

Trabajo Final - Reproductor Mp3

Configuration Management Plan

Autores: Castagno Gustavo - 34582890
Gonzalez Somerstein Gustavo -7721064
Maero Facundo - 38479441

Historial de Revisiones

Versión	Fecha	Resumen de cambios
1.0.0	09/06/2016	Primera versión del Informe Final
1.1.0	17/06/2016	Se agregan requerimientos
1.1.1	19/06/2016	Actualizo diagramas de clase
1.2.0	20/06/2016	Correcciones generales

Tabla de Contenidos

1. Integrantes

El grupo HelloWorld está conformado por:

Castagno Gustavo Matrícula: 34582890	
Gonzalez Somerstein Gustavo Matrícula:7721064	
Maero Facundo Matrícula: 38479441	

2. Release Notes

El documento de notas de entrega se puede encontrar en la siguiente URL (Google Docs)

3. Manejo de las Configuraciones

El plan de Gestión de las Configuraciones utilizado en el proyecto se encuentra en la siguiente URL (GitHub).

4. Requerimientos

El documento de Especificación de Requerimientos de Software (SRS) utilizado en el proyecto se encuentra en la siguiente URL (GitHub).

5. Arquitectura

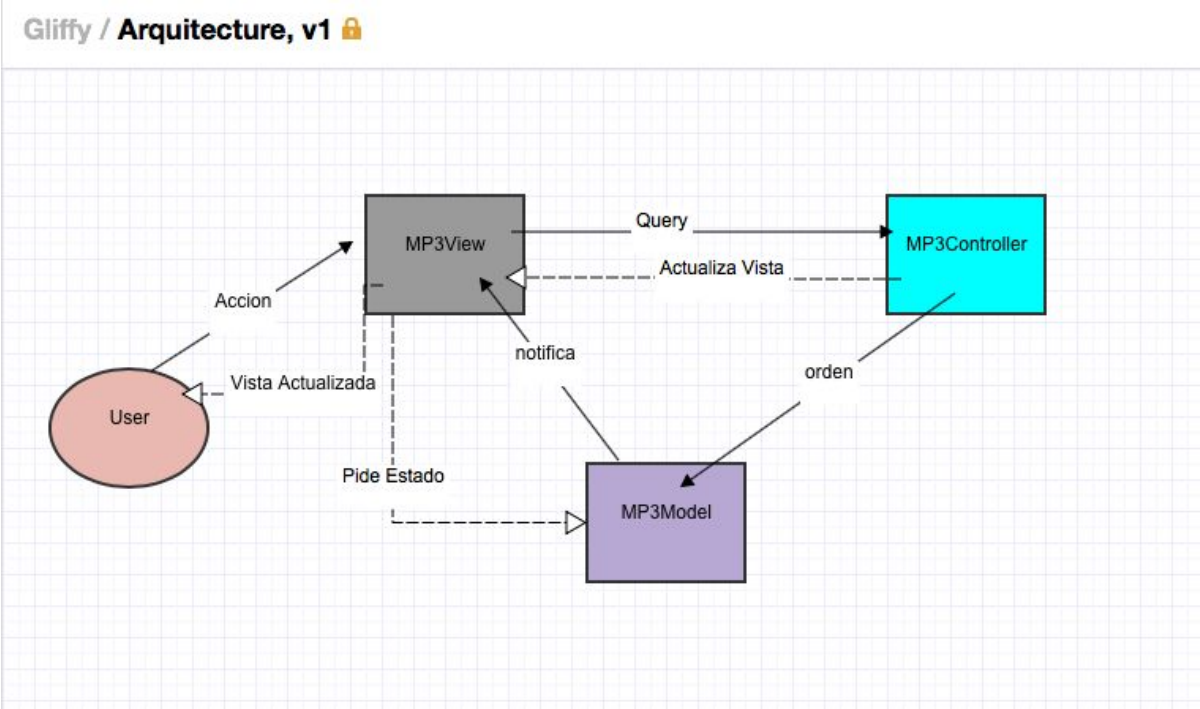
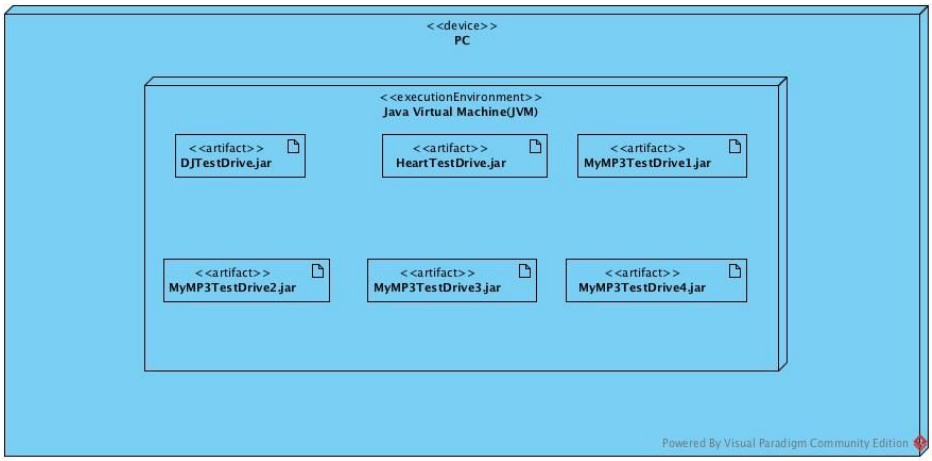


Diagrama de Arquitectura

En el diagrama anterior se puede ver la arquitectura básica para la aplicación MP3, donde las líneas llenas (->) indican el flujo principal del programa(desde que el usuario realiza una acción interactuando con la UI hasta que el modelo cambia su estado por orden del controlador) y las punteadas(-->) indican el flujo secundario(las órdenes de actualización a la vista).

El patrón utilizado es evidentemente el MVC (Model View Controller) ya que permite solucionar el requerimiento de mantenibilidad. Al estar separada la lógica (modelo) del programa de la interfaz de usuario (vista) y la interpretación de las acciones (controlador) el acoplamiento es bajo. Esto permite reutilizar el modelo con distintas vistas(como se realiza en este proyecto con DJView y MP3View, donde cada vista puede explotar funcionalidades distintas del modelo.

Otra gran ventaja del acoplamiento bajo es que implementar una nueva funcionalidad o cambiar la vista resultan operaciones sencillas gracias a la arquitectura. Diagrama de Deployment



En el diagrama de Deployment se puede observar los distintos ejecutables corriendo en la JVM, la cual a su vez se encuentra en la PC del usuario ejecutandolos. No se incluye el sistema operativo, ya que este depende de la persona utilizando el programa. Además, el software posee la ventaja de ser portable (probado en Windows 10 y OSX El Capitan).

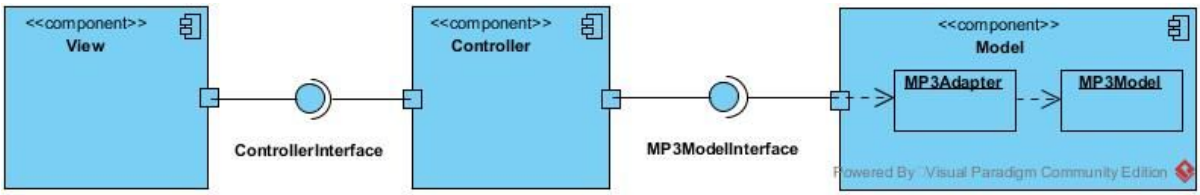


Diagrama de Componentes

6. Diseño e Implementación

En el presente proyecto las clases se separaron en diferentes paquetes, según su funcionalidad (referido al Model View Controller). Estos son:

- Modelos
- Controladores
- Vistas
- TestDrives
- Estados (referido al Patrón de Diseño State)

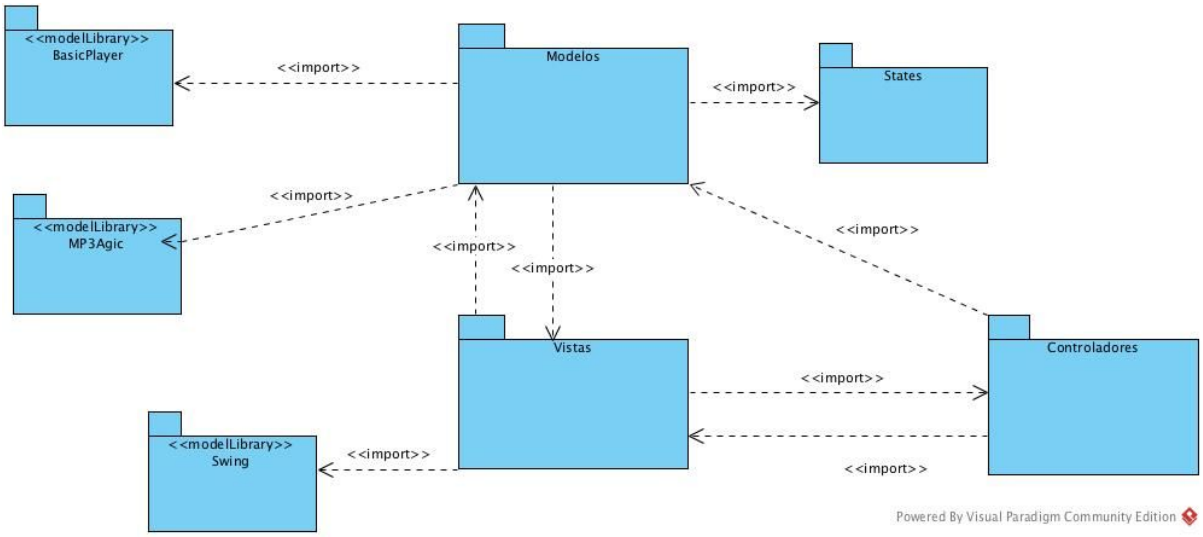
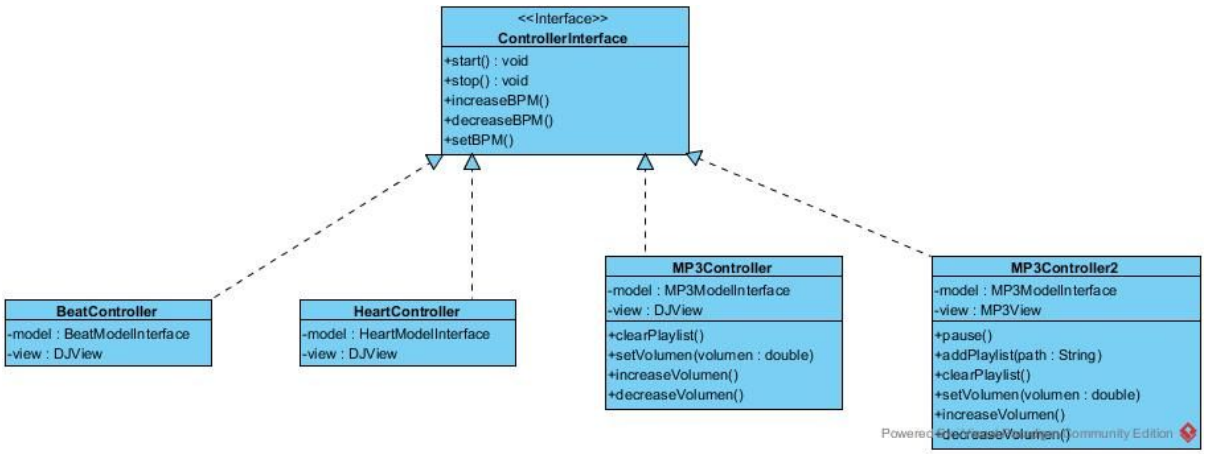


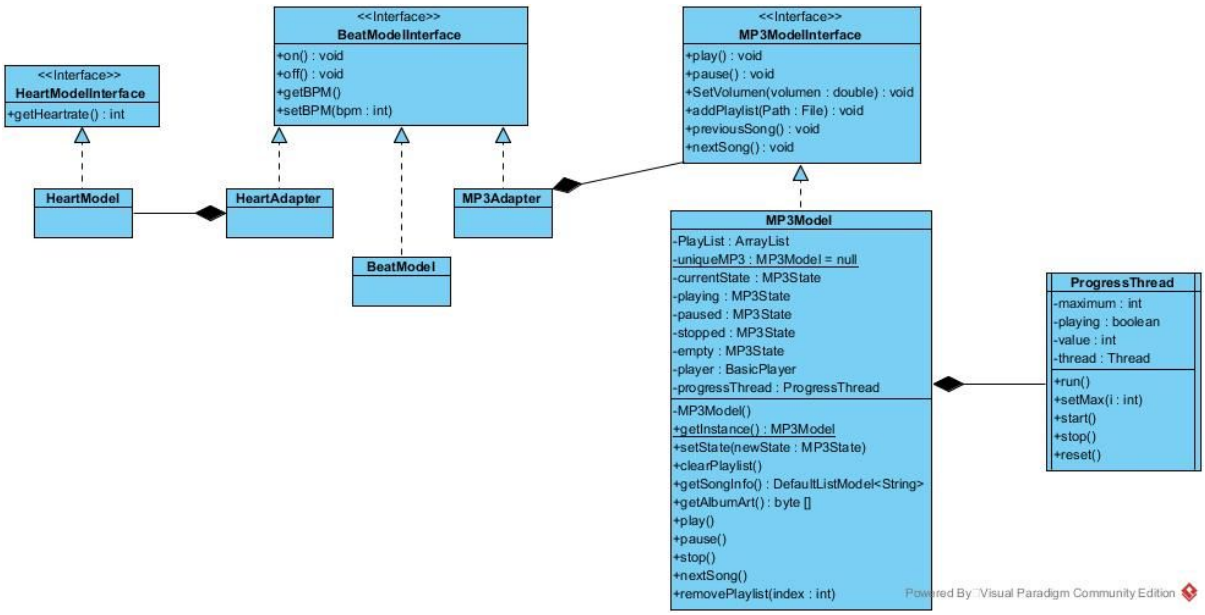
Diagrama de Paquetes

A continuación se muestran diagramas de clase que dejan clara la relación entre clases de cada paquete:

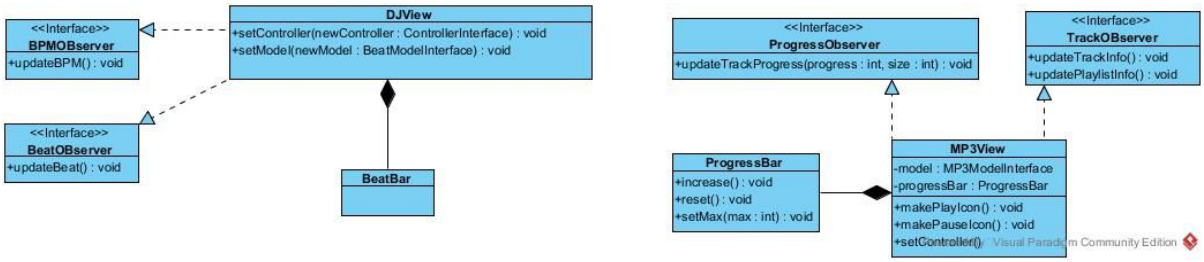
- Controladores: Se tiene una interfaz común ControllerInterface, y dos implementaciones propias, MP3Controller y MP3Controller2, que comunican respectivamente la vista DJView y nuestra vista propia MP3View con el modelo. Diagrama de Clases: Controladores



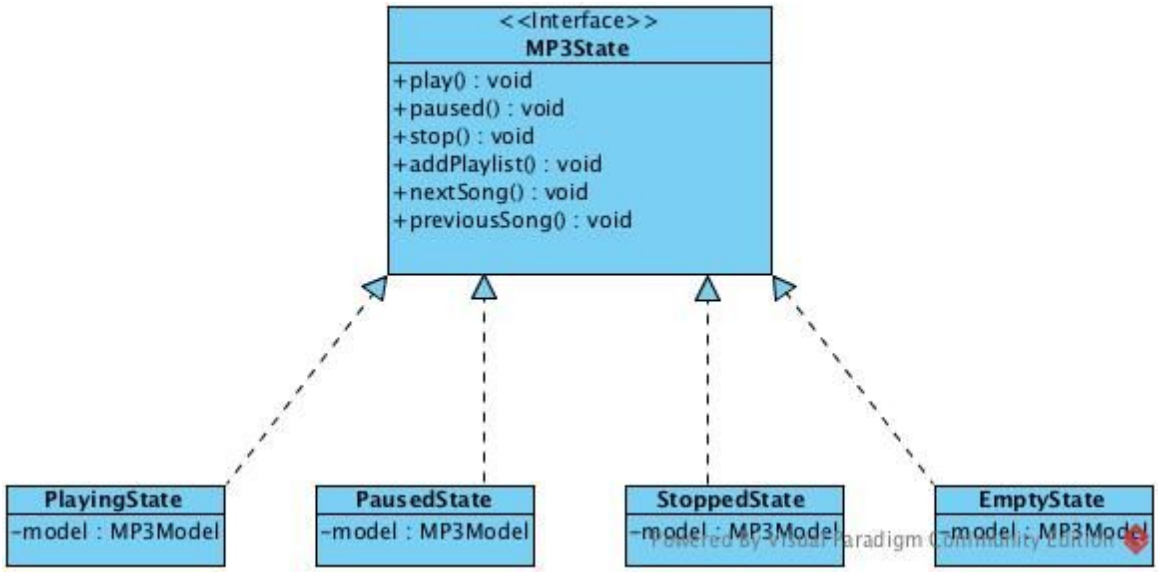
- Modelos: Se creó la interfaz MP3ModelInterface, que provee los métodos principales que debería tener nuestro reproductor. A partir de esta clase se implementa MP3Model, que cuenta con más métodos para gestionar la lista de canciones, como la posibilidad de limpiar la lista de reproducción, obtener los metadatos y la portada de la canción actual, entre otros.
  Para poder utilizar nuestro modelo con la vista DJView proporcionada en el ejemplo se creó la clase MP3Adapter, que proporciona la interfaz correcta para que la vista DJView se comunique con nuestro modelo, al actualizarse (por medio del patrón Observer).
  Además, para implementar la funcionalidad de reproducción continua de música, se incluyó la clase ProgressThread, que cuenta el tiempo transcurrido y avisa al modelo que comience a sonar la pista siguiente. Diagrama de Clases: Modelos



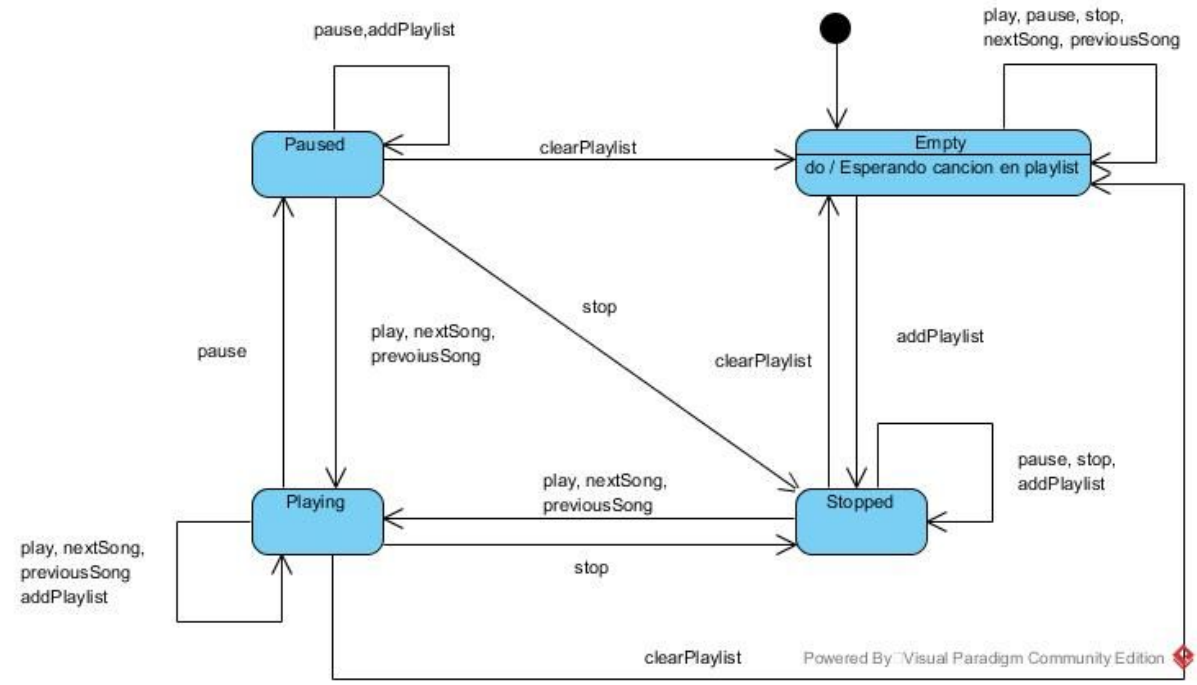
- Vistas: se tiene la vista DJView por defecto, que implementa las interfases BPMObserver y BeatObserver. Luego, como creación original se escribió la clase MP3View, que proporciona una mayor capacidad de relacionarse con el modelo, mediante la adición de botones y un visualizador de canciones a reproducir. Para actualizar ciertos componentes de la misma se creó la interfaz TrackObserver, que complementada con la información del modelo, amplía el patrón Observer ya implementado en el ejemplo.
  Se añadió también a la vista MP3View una barra de progreso, a través de la clase ProgressBar, que dinámicamente va llenándose a medida que avanza la canción en curso. Esta funcionalidad se implementa gracias a que la vista implementa la interfaz ProgressObserver, para ser notificada a cada segundo. Diagrama de Clases: Vistas



- Estados: se muestra la interfaz MP3State, que tiene todos los métodos que pueden causar un cambio de estado en el modelo. Se crearon 4 clases concretas, correspondientes a los 4 estados de nuestro reproductor. Estas clases tienen una referencia al mismo(modelo), para poder realizar las acciones que les son delegadas, y cambiar el estado actual según corresponda. Diagrama de Clases: Estados



Como patrón de diseño adicional, se utilizó el patrón State. Su definición dice: permite que un objeto altere su comportamiento, cuando su estado interno cambia. El objeto aparenta cambiar su clase.
Como se desarrolló un reproductor de música, se decidió tratarlo como una máquina de estados.
Las distintas etapas en las que puede encontrarse son: reproduciendo, pausado, detenido, esperando playlist(empty).
Las acciones de usuario que pueden alterar esos estados son: play, pausa, stop, siguiente, anterior, agregar canciones a la lista. Estas son delegadas por el modelo, para que los estados las interpreten y respondan acorde a la situación.
Si bien el modelo puede realizar otras acciones, como modificar el volumen, realizar operaciones relacionadas a sus observadores, entre otras, estas no modifican el estado global del sistema, por lo que el modelo se encarga directamente de procesarlas.
El comportamiento del patrón mencionado puede verse en el siguiente diagrama de Estados:
Diagrama de Estados: Patrón State



Este patrón puede resultar muy útil para simplificar la lógica de una clase, y reemplazar secuencias de if-else por simplemente delegar el método al estado actual, el cual realiza la acción correcta sin necesidad de complicar el código.
Como ventaja adicional, si a futuro se considera necesario expandir la lógica del sistema, se puede crear un nuevo estado, agregarlo al proyecto y establecer sus acciones frente a cada situación.

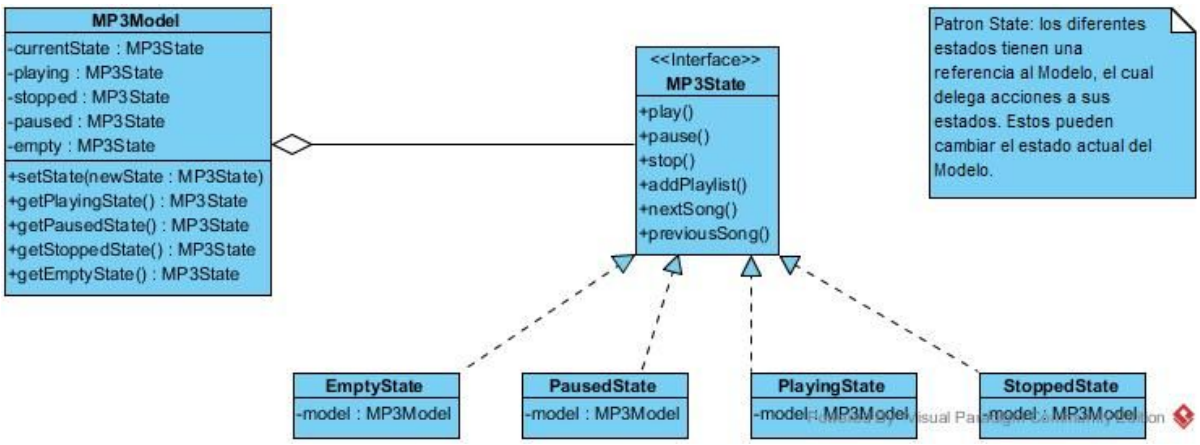
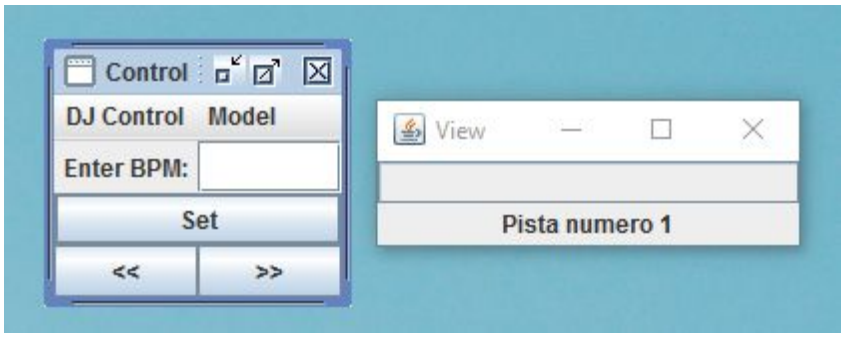


Diagrama de Clases: Patrón State

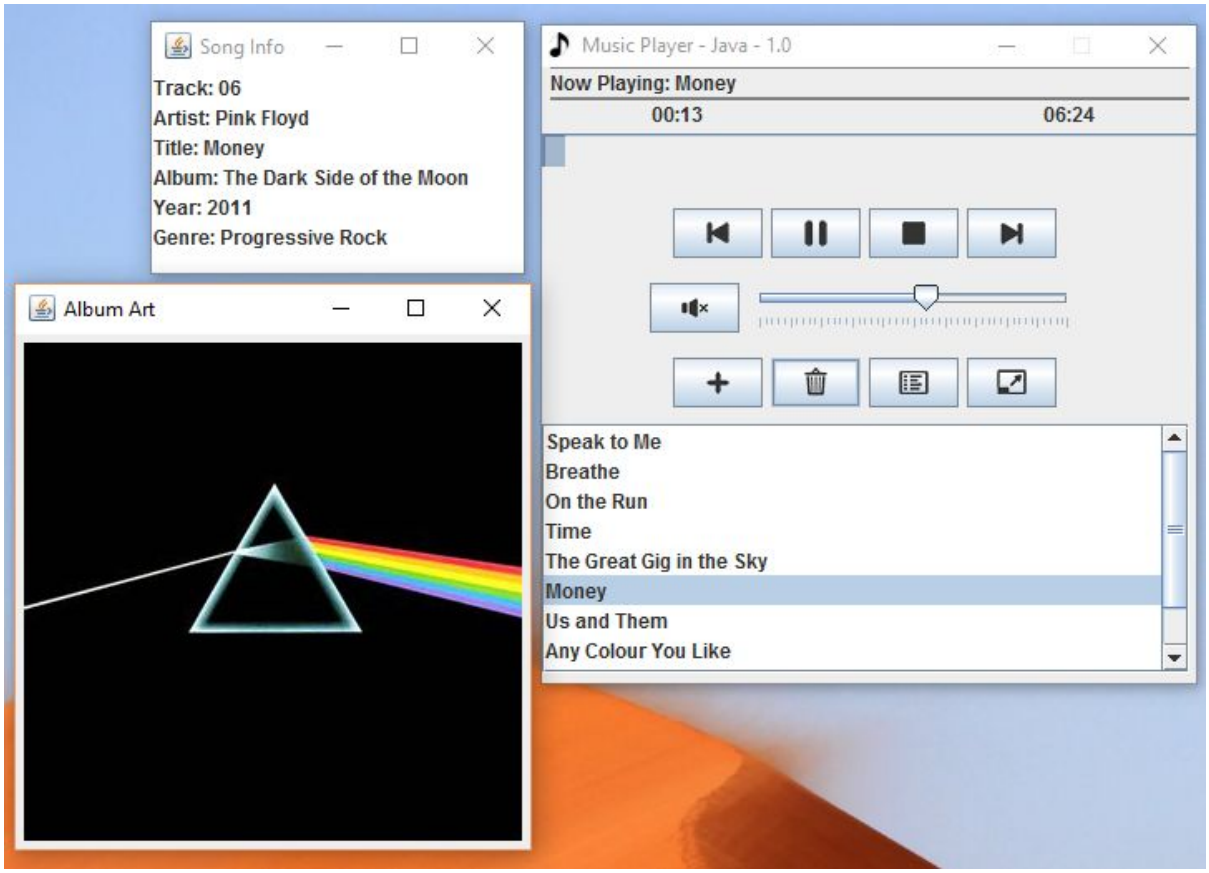
Finalmente las librerías externas utilizadas para realizar el proyecto son:

- BasicPlayer API: utilizada para leer y reproducir archivos .mp3. URL
- Mp3agic: librería utilizada para leer los metadatos del archivo en reproducción, y poder así mostrar datos como artista, álbum, año, género y arte de portada. URL
- Swing : librería de Java utilizada para el manejo de la UI

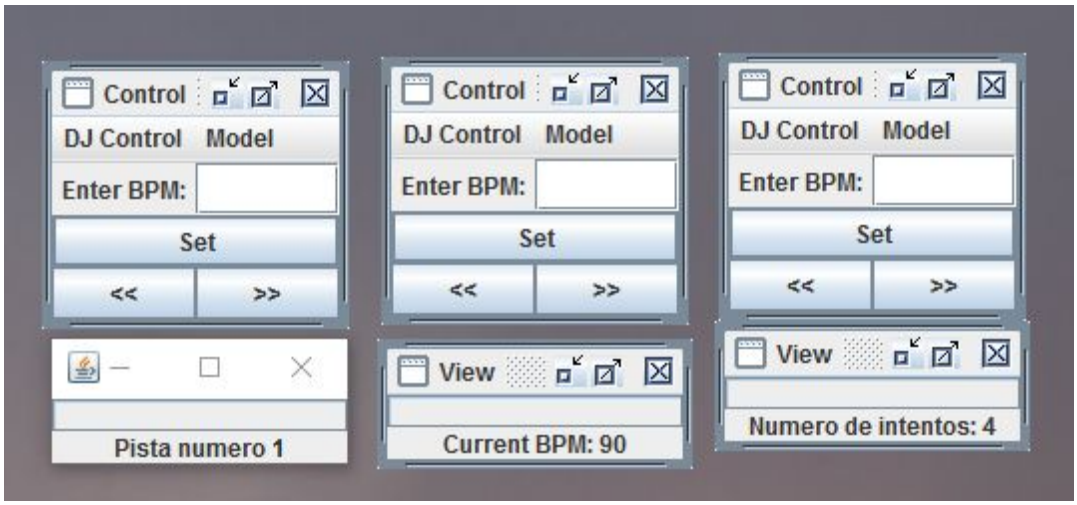
Se muestran capturas de pantalla del proyecto en funcionamiento.



MP3Model con la vista DJView



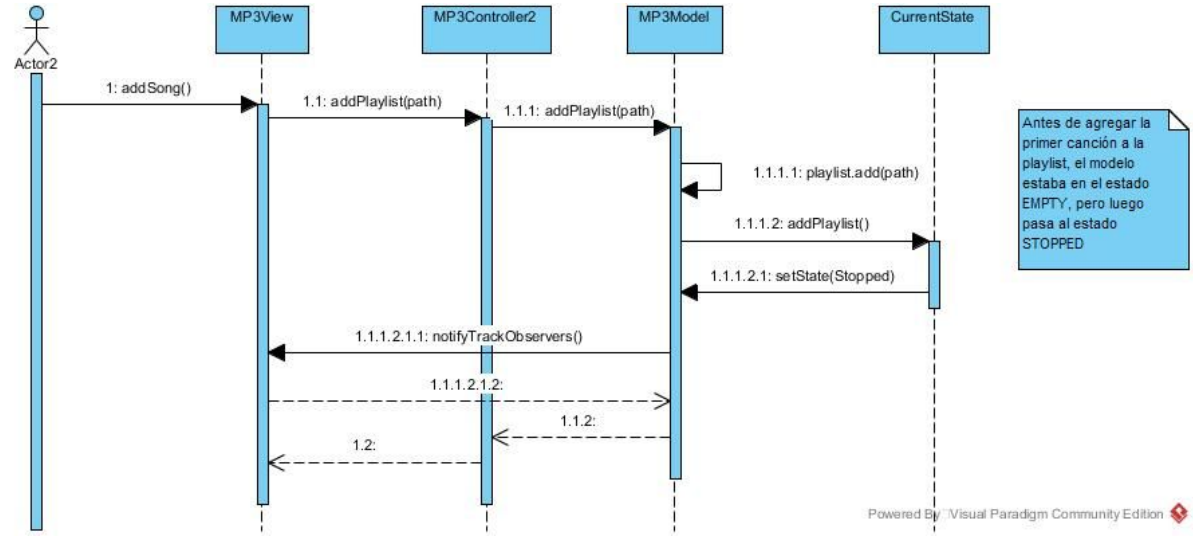
MP3Model con la vista propia MP3View, y además mostrando información de la canción actual.



DJView funcionando con los tres modelos al mismo tiempo.

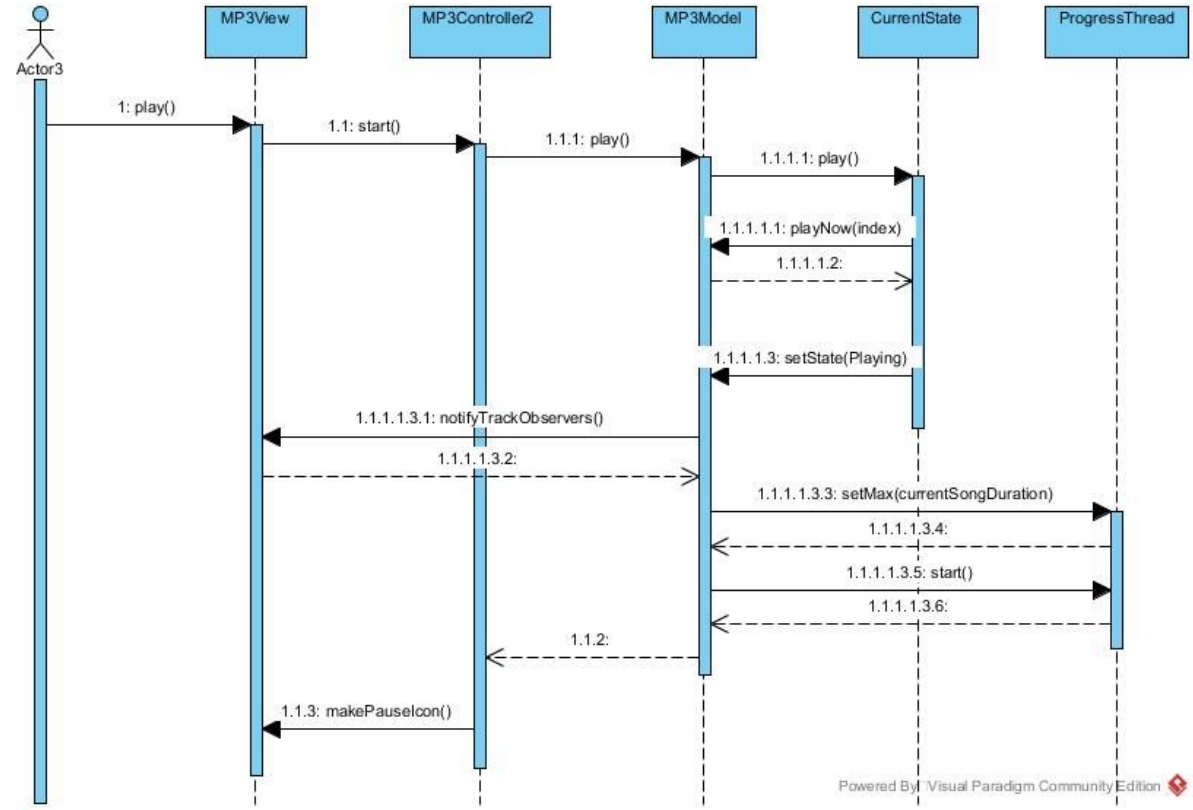
Se muestran también dos situaciones típicas en el sistema:

1. Agregar una canción a la playlist cuando la misma está vacía, lo que causa un cambio de estado.



Link

2. Iniciar la reproducción de una canción cuando la música estaba detenida.

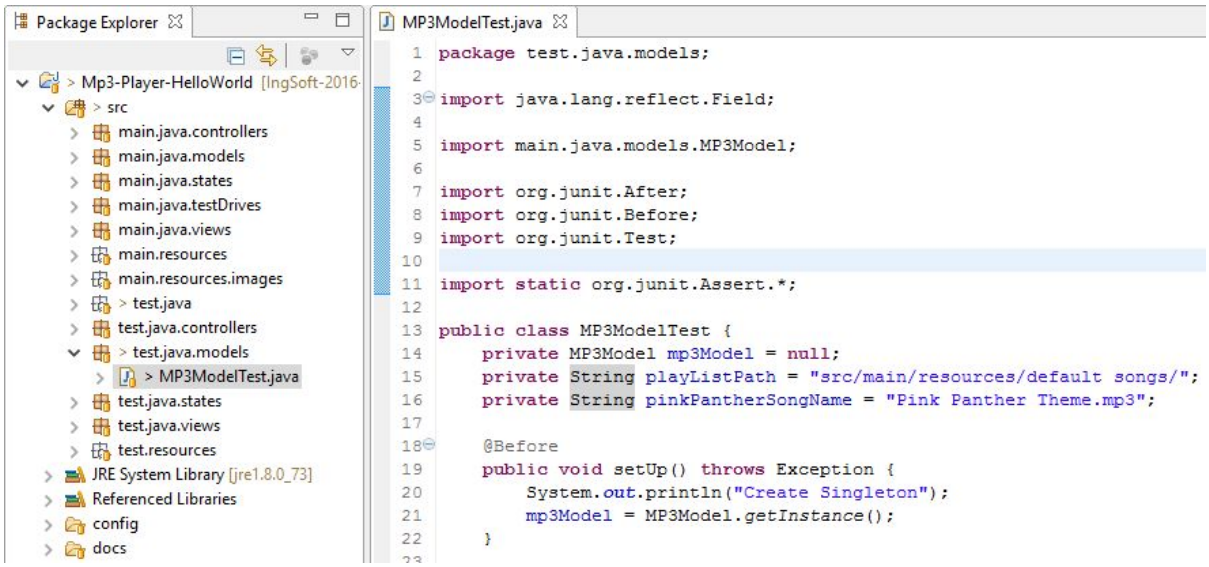


Link

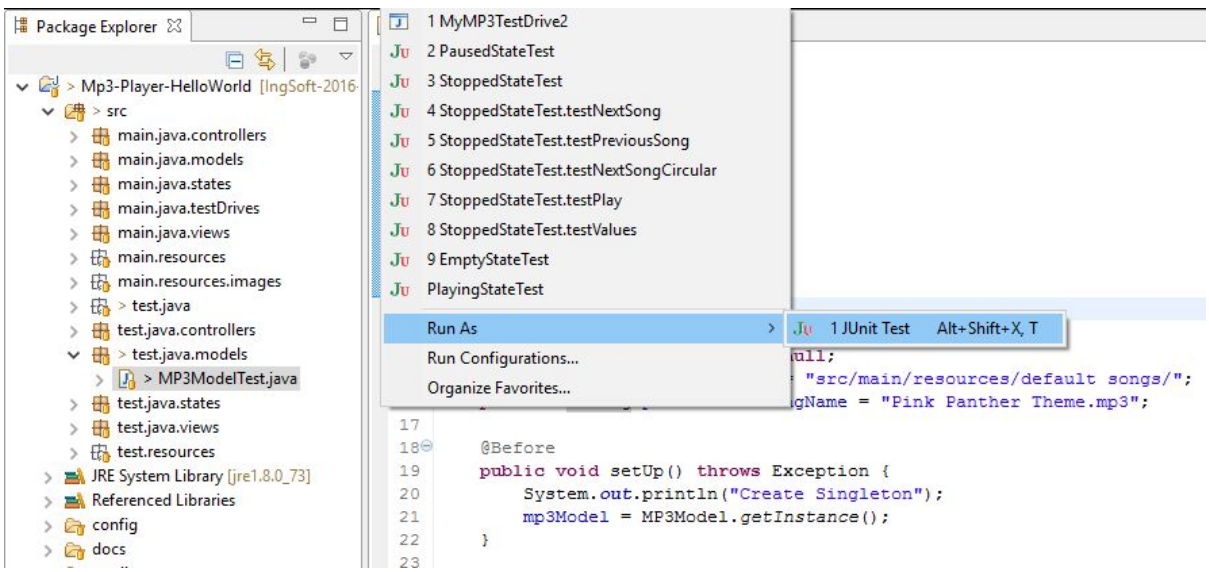
7. Pruebas Unitarias y del Sistema

Se generaron pruebas unitarias automáticas basadas en JUnit. Las mismas contribuyen a verificar el correcto funcionamiento de las clases que manejan la lógica de nuestro sistema, en particular las clases correspondientes a los distintos estados (States), y el modelo MP3Model.
Para correr dichos tests, se deben seguir los siguientes pasos:

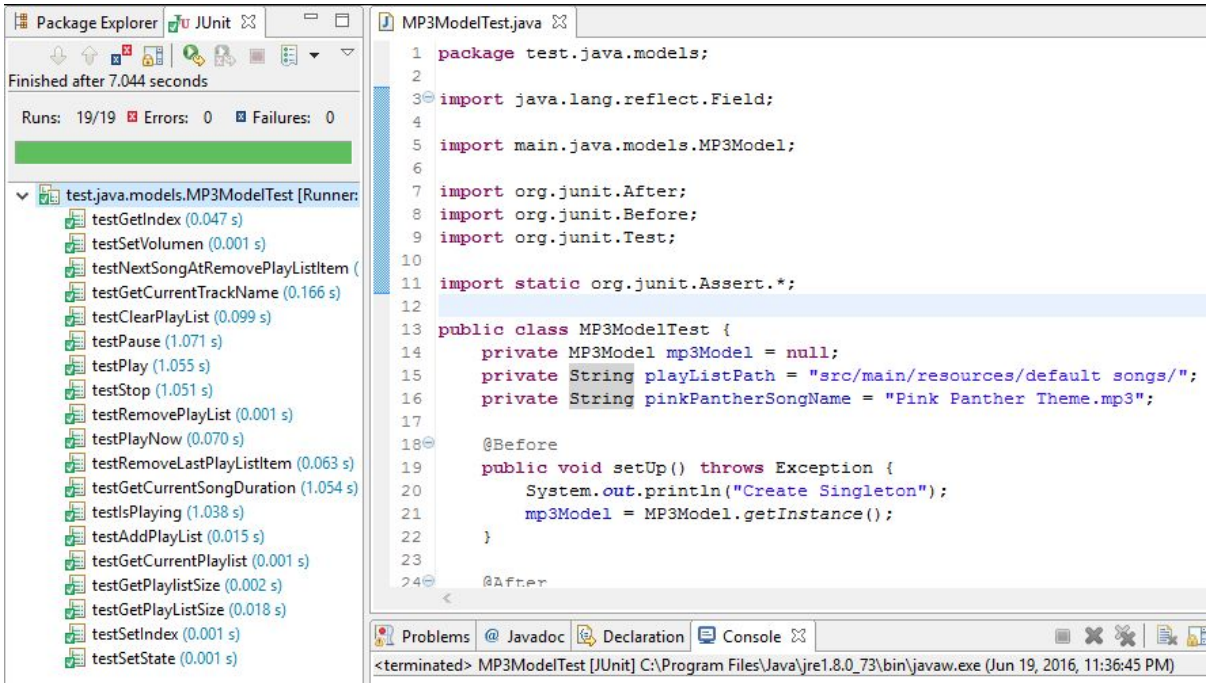
1. Importar el proyecto en un IDE de Java. Ej: Eclipse, IntelliJ.
2. Buscar en el explorador de paquetes las clases Test.
3. Abrir la clase test que se quiera probar. (Link imagen)



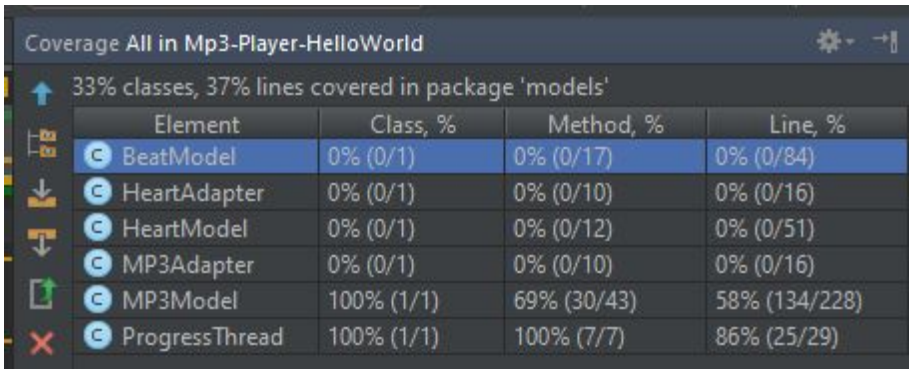
4. Correr la clase como JUnit TestCase. (Link imagen)

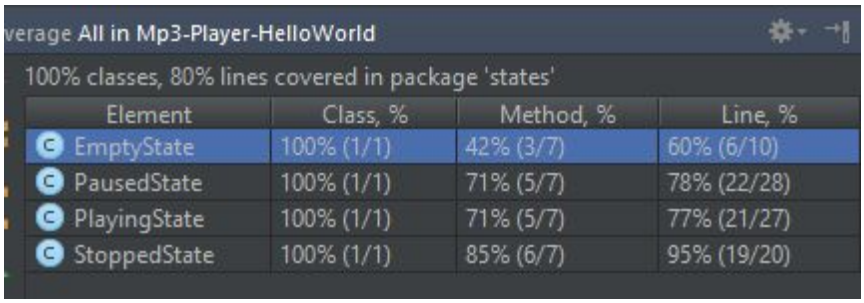


5. Luego de unos segundos, aparecerán en pantalla los resultados. (Link imagen)



A continuación se muestran las estadísticas de coverage de código, obtenidas con el IDE IntelliJ. Se procedió a testear las clases MP3Model, y los cuatro estados del mismo (Stopped, Paused, Playing, Empty).





A continuación se presentan los Casos de Prueba de Sistema contra los requerimientos descritos en el documento correspondiente:

Link Casos de Prueba de Sistema

De los casos de prueba anteriores, se seleccionaron los siguientes como Sanity Tests, los que permiten comprobar que la aplicación cumple sus funcionalidades básicas y puede ser entregada al cliente. La descripción completa de los tests se puede encontrar en el documento sobre tests de sistema.

Test
Reproducir canción
Pausar
Siguiente Canción
Barra de volumen
Reanudar

Se actualizaron las matrices de trazabilidad:

Relaciones entre Tests Unitarios MP3Model y Requerimientos de Usuario:

Tests unitarios MP3Model \Requerimentos de usuario	test Get Current Playlist	test Add PlayList	test Clear PlayList	test Get Play List Size	test GetI ndex	test SetIndex	test Play Now	test Play	test Is Playing	test Stop	test Pause	test Get Playlist Size	test Get Current Track Name	test Get Current Song Duration	test Remove PlayList	test Set State	test Set Volumen	test Remove Last PlayList Item	test Next Song At Remove PlayList Item
4. Agregar canciones	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
5. Eliminar canciones			X												X	X		X	X
6. Reproducir música						X	X	X	X							X			
7. Pausar música											X					X			
8. Detener música										X						X			
9. Reanudar reproducción																X			
10. Saltear canciones						X										X			
11. Mostrar información													X	X					
12. Mostrar portada																			
13. Modificar volumen																	X		

Relaciones entre Tests Unitarios de las clases State y Requerimientos de Usuario:

	EmptyState	PausedState						StoppedState					PlayingState				
Tests unitarios Clases State \Requerimentos de usuario	test Add PlayList	test Values	test Play	test Next Song	test Next Song Circular	test Previous Song	test Stop	test Values	test Play	test Next Song	test Next Song Circular	test Previous Song	test Values	test Paused	test Next Song	test Previous Song	test Stop
4. Agregar canciones	X	X						X					X				
5. Eliminar canciones																	
6. Reproducir música		X	X	X	X	X		X	X	X	X	X	X	X	X	X	
7. Pausar música					X	X					X	X					
8. Detener música							X										X
9. Reanudar reproducción					X						X						
10. Saltear canciones				X	X	X				X	X	X			X	X	
11. Mostrar información																	
12. Mostrar portada																	
13. Modificar volumen		X						X					X				

Relaciones entre Tests de Sistema y Requerimientos de Usuario:

Tests de Sistema \Requerimentos de usuario	Reproducir canción	Pausar	Reanudar	Detener	Play luego de un stop	Siguiente Cancion	Borrar canción	Ver información de la canción	Ver portada del album	Barra de volumen	Botón Mute	Añadir un archivo con extension distinta a mp3	Tratar de reproducir cuando no hay canciones	Borrar la canción que se esta reproduciendo
4. Agregar canciones	X	X	X	X	X	X	X	X	X	X	X	X		X
5. Eliminar canciones							X							X
6. Reproducir música	X	X	X	X	X	X				X	X		X	X
7. Pausar música		X	X											
8. Detener música				X	X									
9. Reanudar reproducción			X											
10. Saltear canciones						X								
11. Mostrar información								X						
12. Mostrar portada									X					
13. Modificar volumen										X	X			

Relaciones entre Clases principales del proyecto y Requerimientos de Usuario:

Clases \\\\\\\\\\\\\\\\\\\\\\\\\\\\ Requerimentos de usuario	MP3Model	Progress Thread	MP3 Controller 2	Empty State	Paused State	Playing State	Stopped State	MP3View	Progress Bar
4. Agregar canciones	X		X	X	X	X	X	X	
5. Eliminar canciones	X		X		X	X	X	X	
6. Reproducir música	X	X	X		X	X	X	X	
7. Pausar música	X	X	X		X	X	X	X	
8. Detener música	X	X	X		X	X	X	X	
9. Reanudar reproducción	X	X	X		X	X	X	X	
10. Saltear canciones	X	X	X		X	X	X	X	
11. Mostrar información	X							X	X
12. Mostrar portada	X							X	
13. Modificar volumen	X		X					X	

Los bugs identificados se encuentran en la sección de Issues del repositorio en GitHub: Link
A continuación se listan los bugs corregidos:

Bugs severos:

- Al borrar una canción anterior a la que se estaba reproduciendo, se producía un error y el sistema entraba en conflicto, dejando de actualizar la vista correctamente y comportándose de manera errónea.
- La playlist no se actualizaba correctamente al eliminar una canción.
- La música no se detenía al cambiar de modelo usando patrón Strategy en la vista DJView, dado que el hilo encargado de la tarea seguía corriendo en el background.
- No se podía importar un archivo mp3 al crear el .jar ejecutable. Se decidió incluirlo en una carpeta junto con el .jar.

Bugs leves:

- La canción siguiente no se comenzaba a reproducir al finalizar la actual, sino que el reproductor se detenía a la espera de interacción del usuario.
- Si se agregaban canciones sin tags, no se podían ver en la playlist, ya que se muestra su tag: Nombre de Pista.
- Al cambiar de canción, el volumen no se actualizaba correctamente, sino que siempre comenzaba en el valor máximo

Es importante destacar que en el repositorio los tests están comentados porque Travis no los puede ejecutar correctamente, aunque, como se puede ver en la imagen arriba, en nuestra PC corren sin problema.

8. Datos Históricos

Para realizar este trabajo hubo que coordinar actividades, discutir decisiones, codificar, diseñar, testear y documentar un proyecto de software en Java de complejidad media.
El esfuerzo y contribución personal de los integrantes fue repartido en cada actividad:

Diseño: 14 horas
Codificación: 20 horas
Testing: 6 horas
Documentación: 14 horas

El esfuerzo personal de cada integrante del grupo fue de:
Gonzalez Somerstein Gustavo: 24 horas
Maero Facundo: 24 horas
Castagno Gustavo: 6 horas

9. Información Adicional

Durante la elaboración del presente trabajo práctico se aprendió a trabajar de manera conjunta en un proyecto concreto, aplicando las prácticas de ingeniería de software vistas en clase.
Se aprendió a utilizar en mayor profundidad herramientas como Eclipse IDE, Travis CI, GitHub. Se tuvo que investigar online para encontrar las librerías necesarias para reproducir archivos mp3, importarlas en nuestro proyecto, y trabajar con ellas. Se tuvo que leer documentación al respecto, foros y páginas de ayuda.
Aprendimos a separar la lógica de un programa de su interfaz de usuario, para así tener un proyecto fácilmente expandible. También leímos sobre patrones de diseño e integramos uno en nuestro código. Esto resultó un poco complicado, pero luego de terminada la tarea se hicieron evidentes las mejoras.
La lección más importante aprendida durante la realización del práctico es a trabajar en equipo, dividir las tareas, compartir ideas e implementarlas.
Nos equivocamos seguido al programar, tuvimos que borrar y comenzar de nuevo más de una vez. Al intentar corregir bugs en el programa las cosas no resultaban como se esperaba. Además, nos dimos cuenta la dificultad y el trabajo necesario para llevar a cabo un proyecto de software, tarea que suele ser subestimada.