TRABAJO FINAL DE INGENIERIA EN SOFTWARE
2016

Grupo: Los Masacre

INTEGRANTES:
#Callejo Alejandro
#Esperanza Manuel
#Pavón Diego

1. NOTA DE ENTREGA

- **Breve introducción acerca del proyecto:**

  En este proyecto final de la materia Ingeniería en Software, desarrollamos una aplicación llamada Afinador, en la cual su finalidad es la de proveer al usuario una interfaz grafica donde pueda ejecutar la cuarta octava de la escala musical que se utiliza para afinar diversos instrumentos, en base al estándar establecido de afinación con respecto a la nota musical "LA" de 440Hz incluida en dicha octava. Dicho proyecto se realizo en base al modelo llamado "Combined" provisto por el libro Head First - Design patterns.

- **Estado del entregable:**

  NOTA AL MARGEN:
  Para esta PRE – entrega, hemos podido desarrollar hasta la fecha, los 2 primeros puntos requeridos para el proyecto final, todos los diagramas provistos son en base a esta parte del proyecto.

- **Listado de funcionalidad:**

  AfinadorTestDrive: para esta PRE - entrega generamos un nuevo modelo con su respectivo controlador, utilizando la vista provista por el modelo base Djview del libro Head First - Design patterns, el cual su funcionalidad es la de generar y reproducir la cuarta octava de la escala musical que se utiliza para afinar diversos instrumentos, en base al estándar establecido de afinación con respecto a la nota musical "LA" de 440Hz incluida en dicha octava.

- **Link del entregable:**

  Dentro del repositorio se encuentra el código, el informe y el archivo .JAR ejecutable

  https://github.com/ManuEsperanza/IngSoft-2016--LosMasacre-/

2. MANEJO DE LAS CONFIGURACIONES

- **Dirección y forma de acceso a la herramienta de control de versiones:**

  La herramienta de control de versiones que utilizamos para nuestro proyecto es Github.
  Link del repositorio: https://github.com/ManuEsperanza/IngSoft-2016--LosMasacre-/

- **Esquema de directorios y propósito de cada uno:**

  - /: Raíz del proyecto
    - /bin: Archivos compilados
    - /src: código fuente y UnitTests
      - default package: aloja los archivos de código fuente
      - UnitTests: Tests Unitarios
    - /documentación: Informe del Proyecto

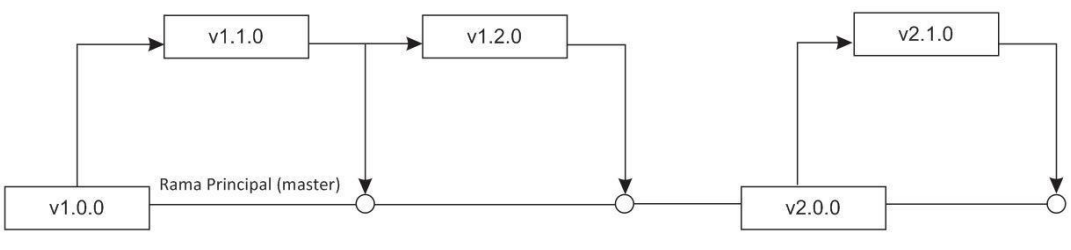
- **Normas de etiquetado y de nombramiento de los archivos:**

  Para el nombramiento de versiones, utilizaremos un patrón de 3 dígitos separados por puntos '.' Con un prefijo 'v' haciendo referencia a la palabra "versión" (Ejemplo: v1.0.0). Los dígitos representan los siguientes estados:

  - Primer dígito: Representa los reléase de mayor versión.
  - Segundo dígito: representa cambios o adición de funcionalidades.
  - Tercer dígito: representa correcciones de bugs o errores encontrados.

- **Plan del esquema de ramas a usar:**

  Implementaremos el esquema de branching "Lean & Lazy", donde existe una rama principal denominada máster, en la cual se desarrolla los releases de mayor peso o mayor versión, y otra rama denominada "dev", donde se desarrollan las adiciones de nuevas funcionalidades o donde se corrigen bugs encontrados en dichos releases.



- **Políticas de fusión de archivos y de etiquetado de acuerdo al progreso de calidad en los entregables:**

  La política de fusión de archivos que usaremos es la siguiente: El desarrollador primero deberá verificar que tenga la última versión de la rama principal (máster). Luego para realizar el merge de la rama en cuestión, El código deberá cumplir con los requisitos de ser, código funcional y que haya cumplido con los casos de prueba requeridos. Además se le debe aplicar el etiquetado de acuerdo a las normas de etiquetado establecidas.

- **Forma de entrega de los "releases", instrucciones mínimas de instalación y formato de entrega:**

  Los releases serán entregados a través de un archivo comprimido de extensión '*.zip', el cual se compone de un informe del proyecto realizado, un link de acceso al repositorio donde se encuentra alojado el código, y un archivo ejecutable de formato JAR autoejecutable.
  Para su ejecución el cliente necesitará tener instalada la herramienta JAVA JRE que lo puede descargar del siguiente enlace: http://java.com/es/download/

- **Listado y forma de contacto de los integrantes del equipo, así como sus roles en la CCB:**

  Callejo Alejandro. aleca77@gmail.com. Rol: Admin/Developer/Tester
  Esperanza Manuel. sojaku@gmail.com. Rol: Developer/Tester
  Pavón Diego. diegoapvn@gmail.com. Rol: Developer/Tester

- **Periodicidad de las reuniones:**

  Las reuniones serán periódicas y de lapso de tiempo corto en donde cada uno expondrá sus avances y problemas que se le presentaron, luego se hará una breve planificación de los pasos a seguir.

- **Frecuencia de reuniones de CCB**:

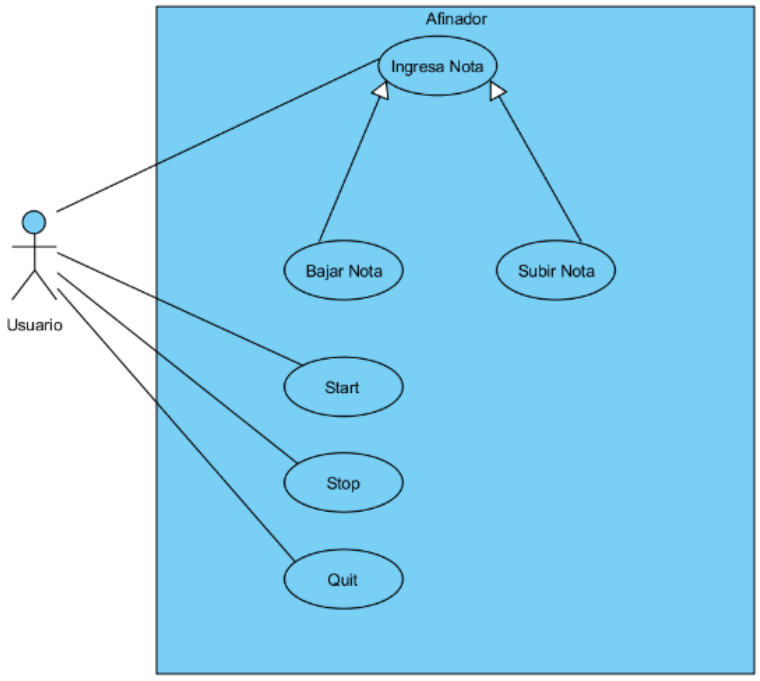
  Las reuniones se realizarán cuando haya solicitudes de cambios pendientes, donde se definirán los lineamientos a seguir con respecto a solicitudes de cambio del proyecto. Discutiendo su aprobación de la misma e implementación.

- **Herramienta de seguimiento de bugs usado para reportar los defectos descubiertos y su estado:**

3. REQUERIMIENTOS

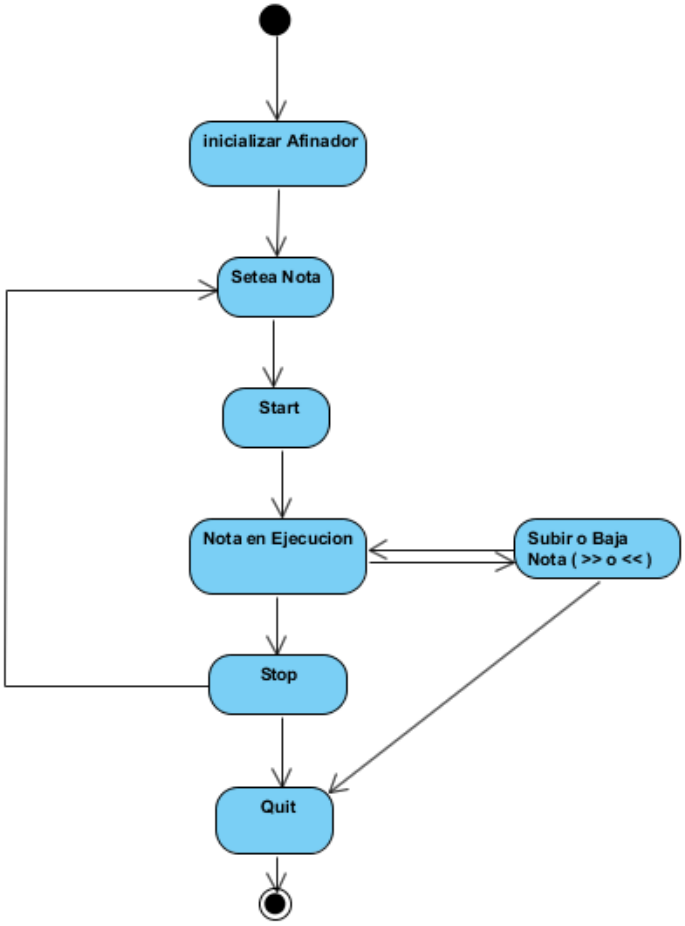
- **Diagrama de Casos de Usos:**

En este diagrama se distinguen 2 actores: el usuario y el sistema “Afinador”. El usuario debe setear un número entero entre 0 y 6: cada número corresponde a una nota entre DO y SI respectivamente. A continuación el usuario debe apretar “Start” para que comience a sonar la nota deseada. El usuario puede cambiar la nota con los botones de “Subir Nota” y “Bajar Nota”. Si el usuario quiere que deje de sonar la nota lo puede hacer mediante el botón de “Stop” y si quiere cerrar el afinador lo puede hacer mediante el botón de “Quit”.

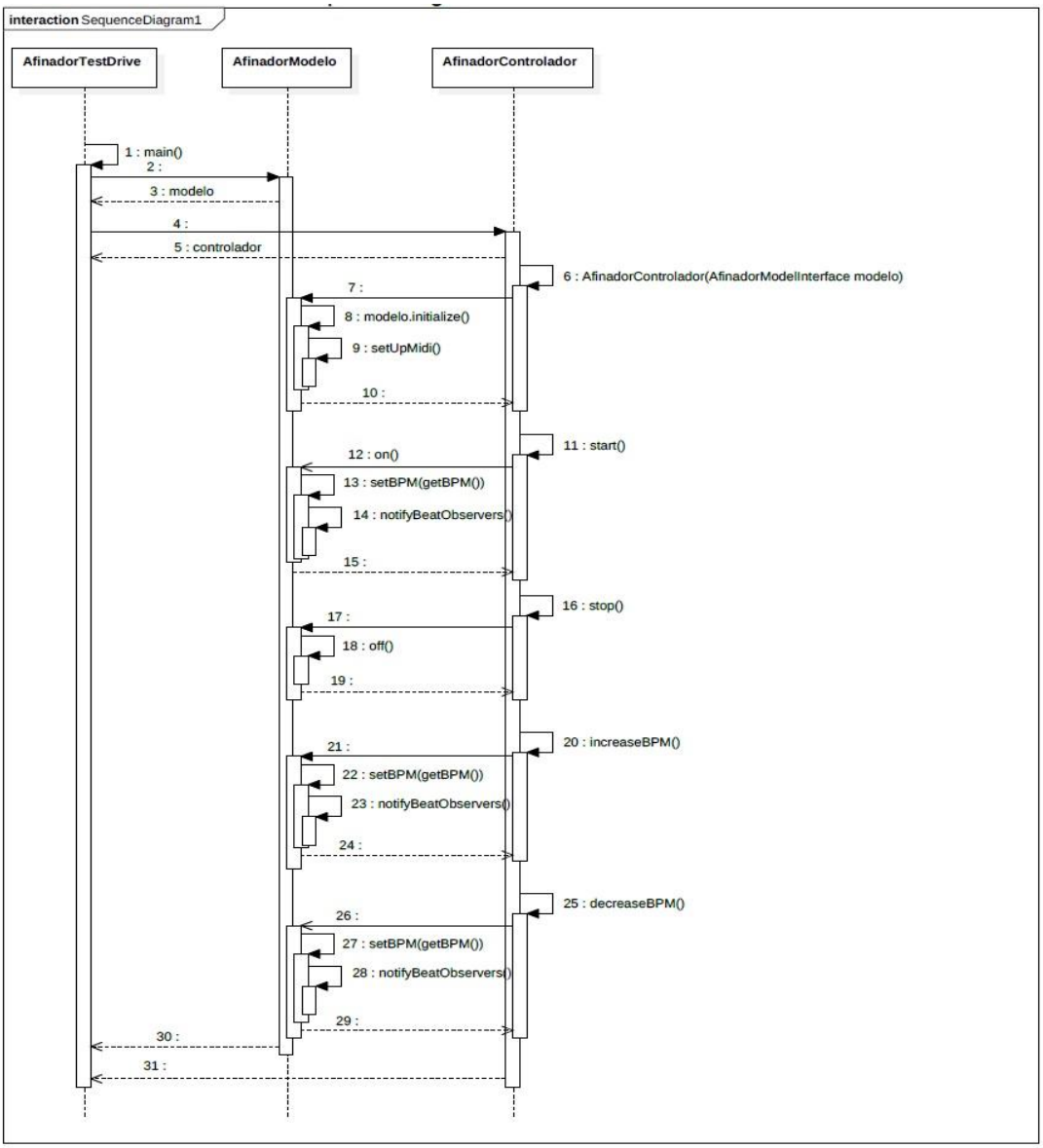


- **Diagrama de Actividades:**

Muestra las acciones que hay que llevar a cabo para que se logre escuchar una nota musical.

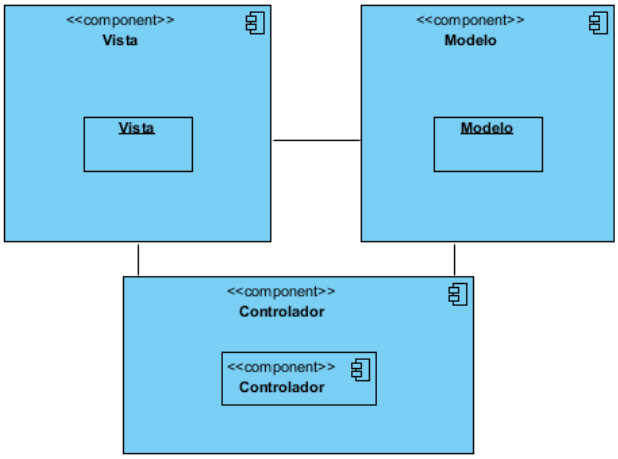


- **Diagrama de Secuencias:**



- **Diagrama de arquitectura preliminar:**

En el siguiente diagrama vemos cómo se relacionan los requerimientos con los sistemas y subsistemas.



- **Requerimientos Funcionales:**
  - F1) El modelo Afinador debe obtener un valor entre 0 y 6, que representa la octava y la nota que se quiere tocar.
  - F2) La nota se debe poder escuchar después que el usuario presione el botón “Start”.
  - F3) Se debe poder detener el sonido después que el usuario presione el botón “Stop”.
  - F4) El modelo debe ser capaz de tocar distintas notas, que vayan del DO al SI de la cuarta octava.
  - F5) Debe ser poder ser capaz de ir a la siguiente o a la anterior nota, luego de presionar “>>” o “<<”.
  - F6) El no puede tocar notas que no pertenezcan a la cuarta octava.
  - F7) Debe ser compatible con la vista del DJView.
  - F8) El modelo debe poder mostrar los datos con una nueva vista.

- **Requerimientos No Funcionales:**

  - NF1) La duración del sonido de la nota no puede ser superior a 2 segundos.
  - NF2) Entre un sonido y el siguiente debe haber una pausa de 2 segundos como mínimo.
  - NF3) Las acciones realizadas por el usuario deben verse reflejadas en la vista en un tiempo igual o menor a 1 segundo.
  - NF4) El software debe ser desarrollado usando JAVA

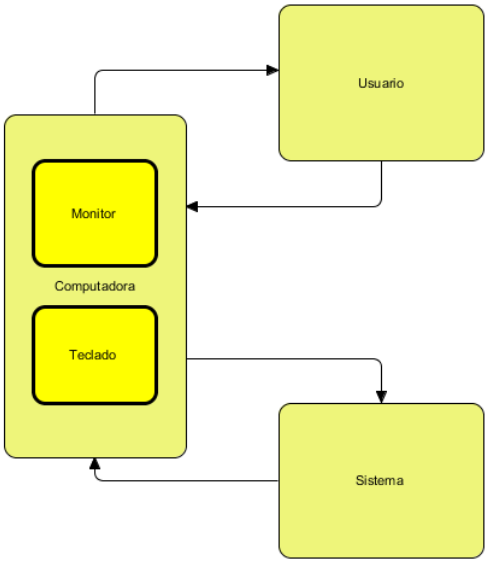
- **Matriz de Trazabilidad**:

	Establecer nota	Cambiar nota	Start	Stop	Quit
F1	1				
F2			1		
F3				1	
F4		1			
F5		1			
F6	1				
F7					
NF1					
NF2		1			
NF3					
NF4					

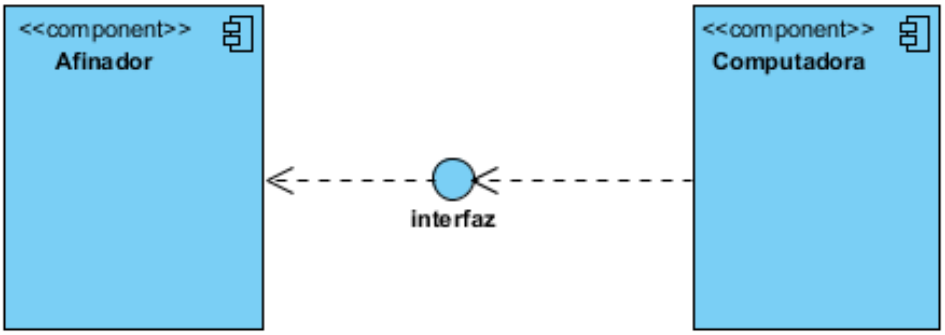
4. ARQUITECTURA

Para el desarrollo del proyecto fue utilizado el patrón de arquitectura MVC (Model – View - Controller) donde se separa la presentación de la información (View), la interacción (Controller) y el sistema (Model). Gracias a las ventajas de la separación e independencia entre las 3 capas nos permitió sin mayores dificultades cambiar entre la vista o el modelo sin tener mayores inconvenientes. Dicha arquitectura se implementó a través de los patrones Observer y Strategy.
Por los requerimientos de tiempo de respuesta, mantenibilidad y desempeño, se eligió el patrón MVC ya que en caso de fallo del sistema se puede reemplazar una interface por otra o un modelo por otro, sin tener que escribir mucho código, facilitando la reutilización del mismo.

- **<u>Diagrama de arquitectura</u>:**

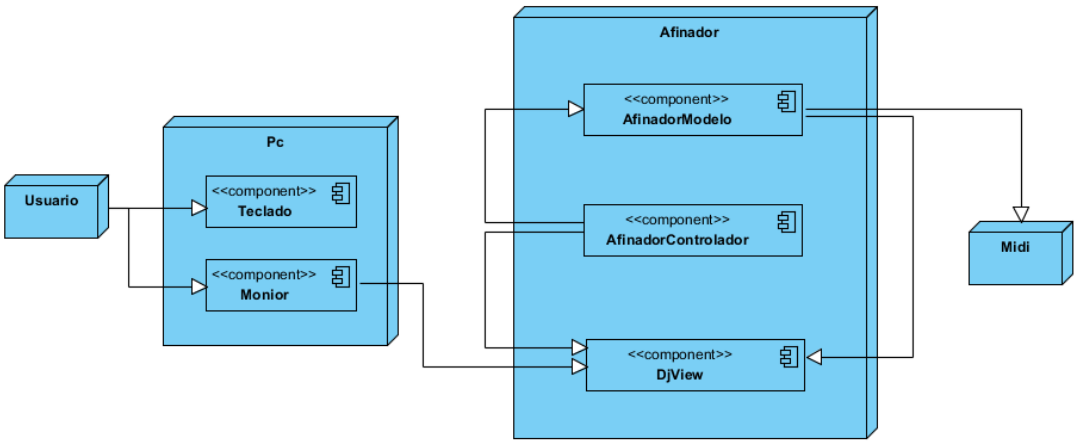


- **<u>Diagrama de Componentes</u>:**



- **Diagrama de Despliegue**:

  En el diagrama de despliegue se muestra la arquitectura del sistema con su interconexionado y las relaciones entre los distintos componentes a nivel físico.



5. DISEÑO E IMPLEMENTACIÓN

- **Diagrama de Paquetes**:

  Muestra cómo está dividido el sistema en agrupaciones lógicas mostrando las dependencias entre esas agrupaciones.