TRABAJO FINAL DE INGENIERIA EN SOFTWARE
2016

Grupo: Los Masacre

INTEGRANTES:

#Callejo Alejandro

#Esperanza Manuel

#Pavón Diego

- **<u>Breve introducción acerca del proyecto</u>:**

  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.

NOTA DE ENTREGA

- **Estado del entregable:**

  Nuestro proyecto se entrega a través de un archivo comprimido de extensión .ZIP el cual contiene los archivos ejecutables .JAR, cada uno representa los requisitos de desarrollo que se nos pidió elaborar. El informe de nuestro proyecto en formato .PDF, y el código fuente del mismo con los tests unitarios desarrollados.

- **Listado de funcionalidad:**

  Afinador musical el cual ofrece la posibilidad de reproducir la cuarta octava de la escala musical con el fin de permitir afinar diversos instrumentos. Permitiendo poder elegir la nota que se quiere reproducir de dicha escala y recorrer la escala en orden ascendente y descendente.

- **Pass/Fail**:

Test	Pass/Fail
UT1: testSetBPM	Pass
UT2: testGetBPM	Pass
UT3:: testRegisterObserver_BeatObserver	Pass
UT4: testRemoveObserver_BeatObserver	Pass

Pass/Fail Ratio	100%

- **Bugs conocidos**:
  La vista no se adapta de acuerdo a la longitud del texto que muestra.

- **Link del entregable:**

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

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

  https://github.com/ManuEsperanza/IngSoft-2016--LosMasacre-/tree/master/ejecutables_jar

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

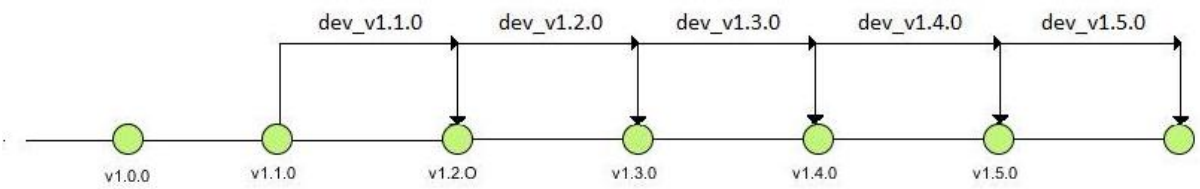
- **Normas de etiquetado y de nombramiento de los archivos:**

  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/

- **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.

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

Integrantes	Forma de contacto	Roles en la CCB
Callejo Alejandro	aleca77@gmail.com	**Máster**
	Tel 3541-555936	**Developer**
Esperanza Manuel	sojaku@gmail.com	**Developer**
	Tel. 351-2004427	**Tester**
		Colaborador General
Pavón Diego	diegoapvn@gmail.com	**Developer**
	Tel. 351-6329687	**Tester**
		Colaborador General

Colaborador General: Tiene injerencia en todos los aspectos del proyecto
Developer: Es el encargado de la programación en java
Máster: Vigila el cumplimiento de la metodología. Define prioridades, roles, proyectos.
Tester: Es el encargado de crear los test unitarios y de sistema.

- **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.

- **Proceso de gestión de cambios de la CCB**

El cliente deberá solicitar formalmente el cambio indicando la siguiente información en un documento:

- Solicitante / Cargo
- Fecha de solicitud
- Nivel de urgencia del cambio
- Importancia del cambio
- Descripción del cambio

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

La herramienta de gestión de defectos que se usa es la que viene integrada en Github ("Issues"). Se puede seguir los bugs y su evolución en el link:

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

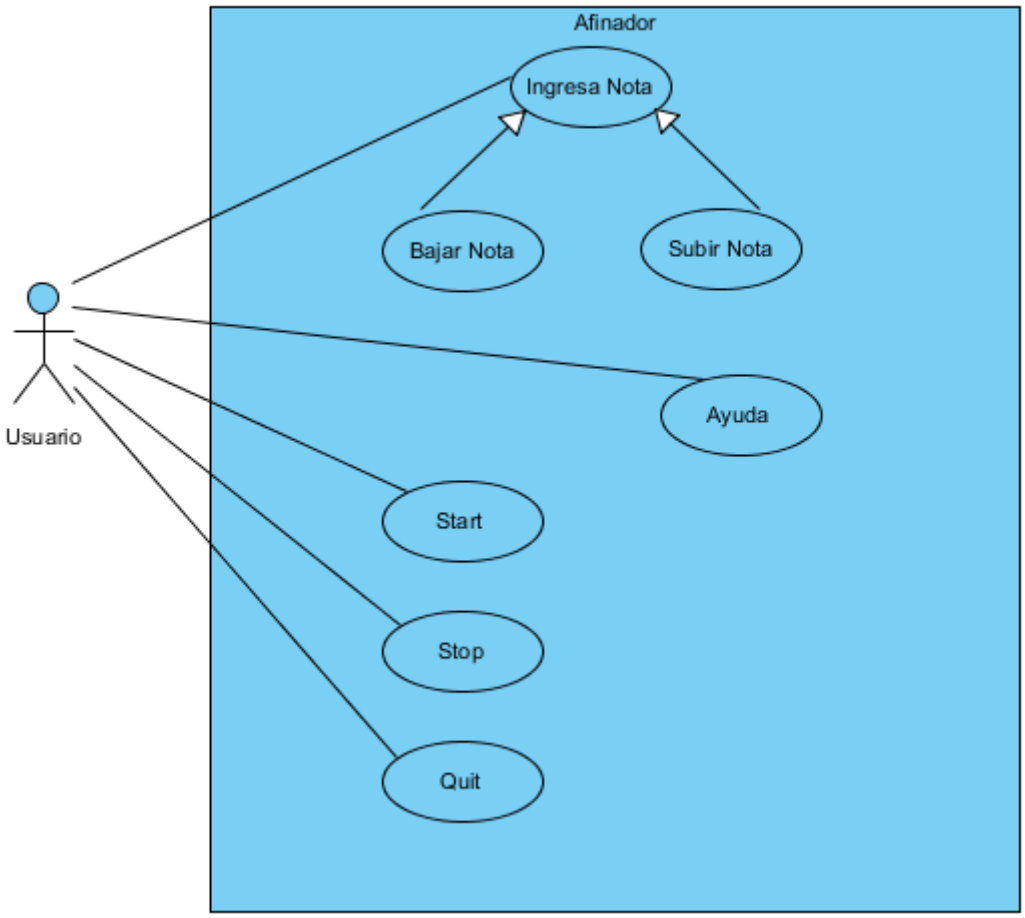
- **Dirección y forma de acceso a la herramienta de integración continúa:**

Decidimos utilizar "Travis" como herramienta de integración continua:
Link: https://travis-ci.org/ ManuEsperanza /IngSoft-2016-LosMasacre-

3. Diagramas

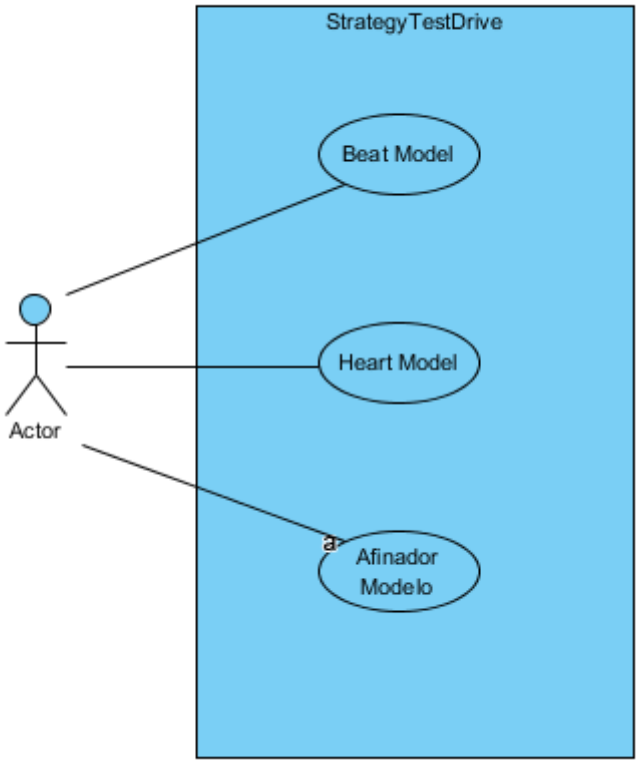
- **<u>Diagrama de Casos de Uso del Modelo Afinador</u>:**

En este diagrama se distinguen 2 actores: el usuario y el sistema “Afinador”. El usuario puede encontrar instrucciones de manejo del afinador pulsando “ayuda”; para comenzar el mismo 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”. El usuario puede encontrar instrucciones de manejo del afinador pulsando “ayuda”



- **Diagrama de Casos de Uso StrategyTestDrive:**

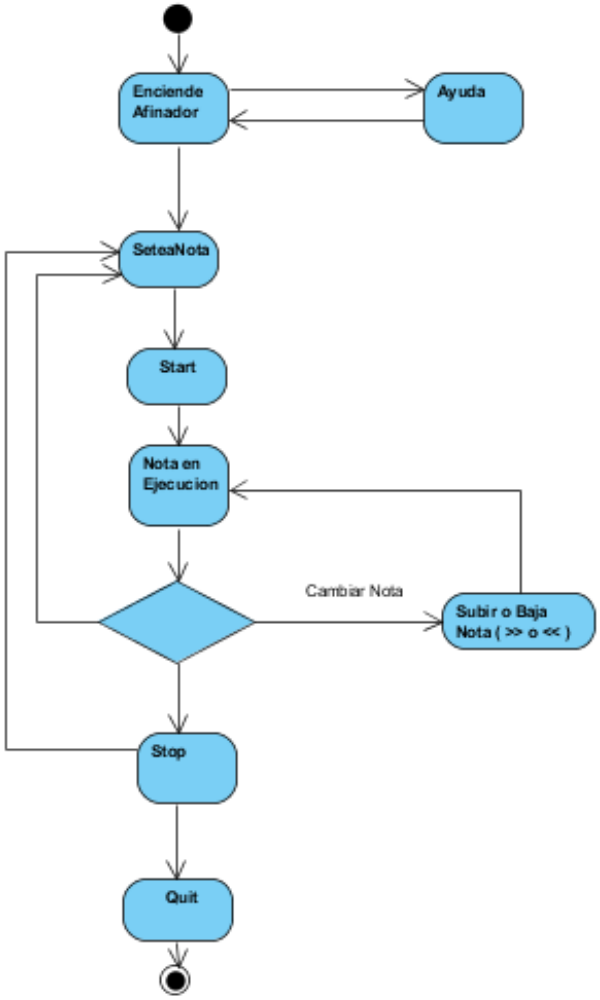
En este diagrama se distinguen 2 actores: el usuario y el sistema "StrategyTestDrive", en donde el usuario puede seleccionar entre los 3 modelos ("Beat Model", "Heart Model", "Afinador Modelo").



- **Diagrama de Actividades:**

El siguiente diagrama muestra las acciones que hay que llevar a cabo sobre el software para iniciar el afinador.

- ✓ El usuario deberá Ejecutar el Afinador.jar y Setear la Nota deseada; luego la misma será confirmada con "start", si el usuario aprieta ">>" o "<<" el afinador ejecutará la nota siguiente o la anterior de la que está en ejecución.
- ✓ el usuario en cualquier momento podrá poner "Quit" para salir del afinador o stop para detener la ejecución de la nota.
- ✓ El usuario luego de poner "Quit" para escuchar una nota debe Setear la misma y poner "Start".
- ✓ El usuario puede utilizar la pestaña ayuda para comprender el funcionamiento del afinador



4. Requerimientos Funcionales:

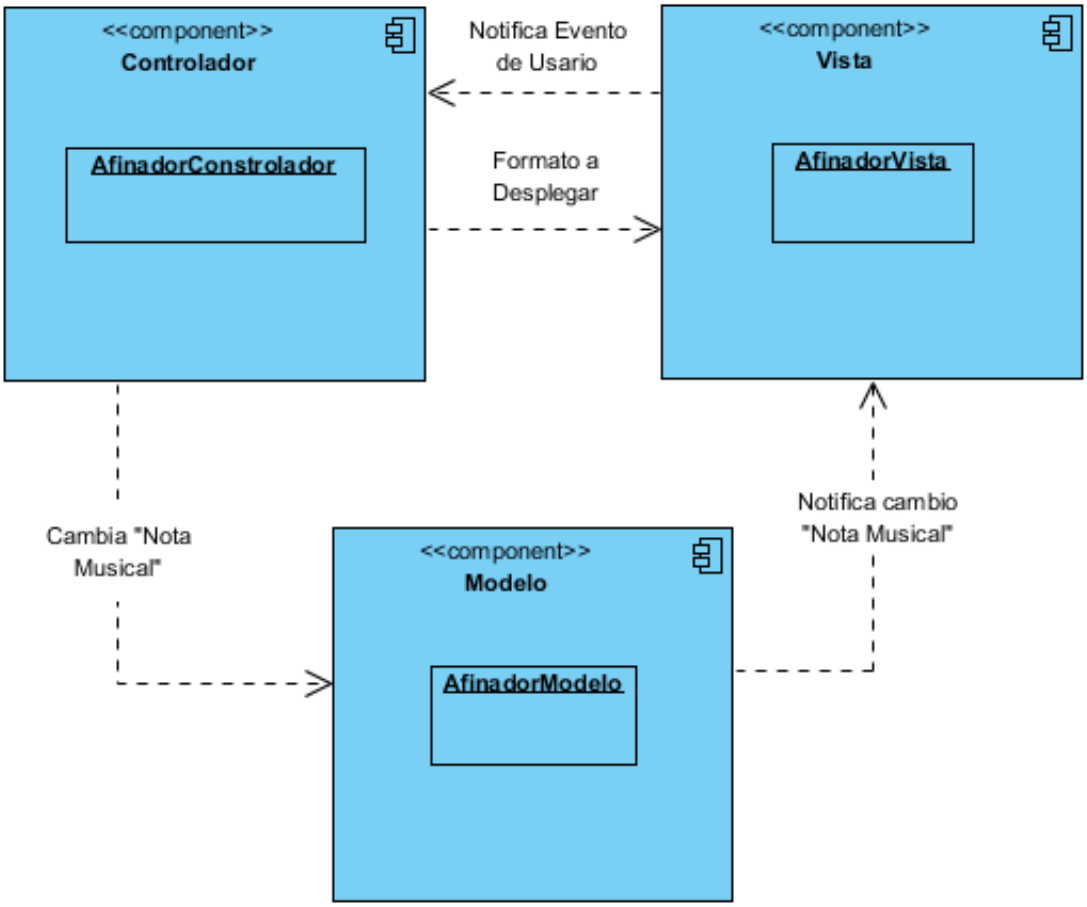
- F1) El modelo Afinador debe obtener un valor entre 0 y 6, que representa la octava y la nota que se quiere tocar.
- F2) Al iniciar deberá figurar la leyenda “Apagado”.
- F3) La nota se debe poder escuchar después que el usuario presione el botón “Start”.
- F4) Se debe poder detener el sonido después que el usuario presione el botón “Stop”.
- F5) El modelo debe ser capaz de tocar distintas notas, que vayan del DO al SI de la cuarta octava, y luego de la nota SI deberá volver a empezar.
- F6) Debe poder ser capaz de ir a la siguiente o a la anterior nota, luego de presionar “>>” o “<<”.

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 1 segundo como mínimo.
- NF3) Las acciones realizadas por el usuario deben verse reflejadas en la vista en un tiempo igual o menor a 2 segundos.
- NF4) El software debe ser desarrollado usando JAVA.

5. Diagrama de arquitectura preliminar:

En el siguiente diagrama vemos cómo se relacionan los requerimientos con los sistemas del Modelo Afinador.



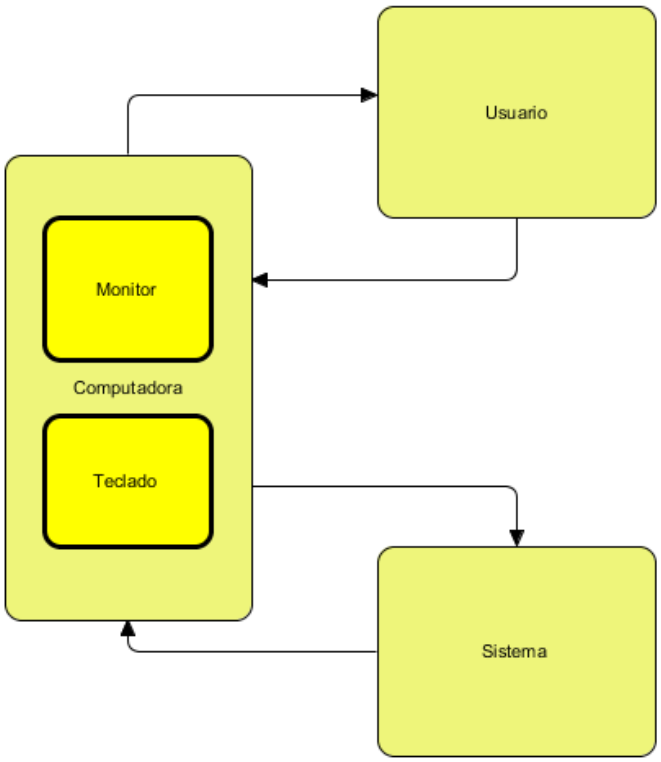
6. Matriz de Trazabilidad:

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

ARQUITECTURA

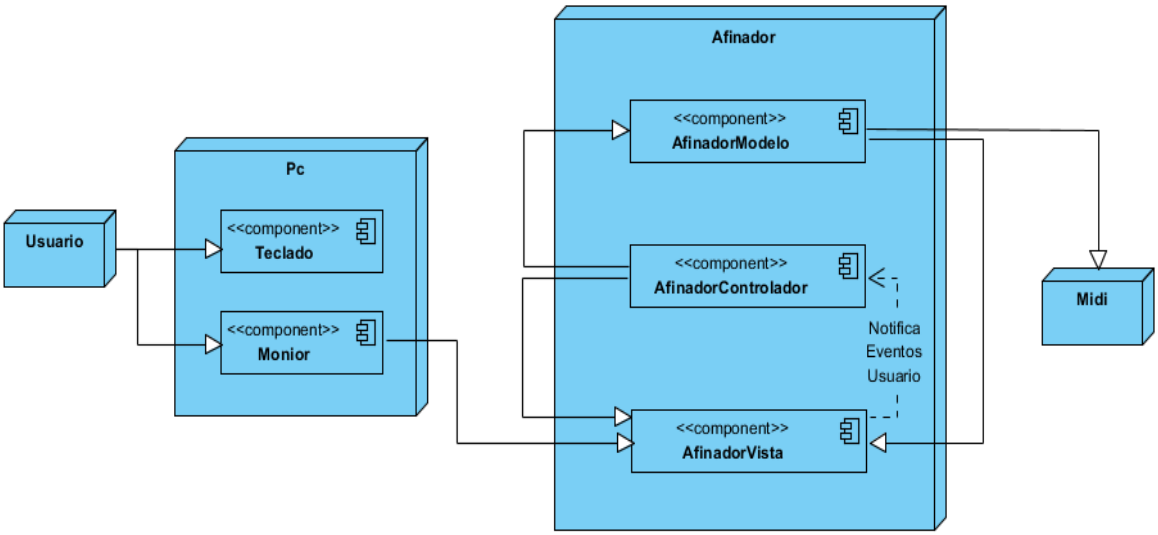
7. Para el desarrollo del proyecto fue utilizado el patrón de arquitectura MVC (Model – View - Controller) donde se separa la presentación de la información (View), la interacción (Controller) y el sistema (Model). Gracias a las ventajas de la separación e independencia entre las 3 capas nos permitió 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.

- **Diagrama de arquitectura:**



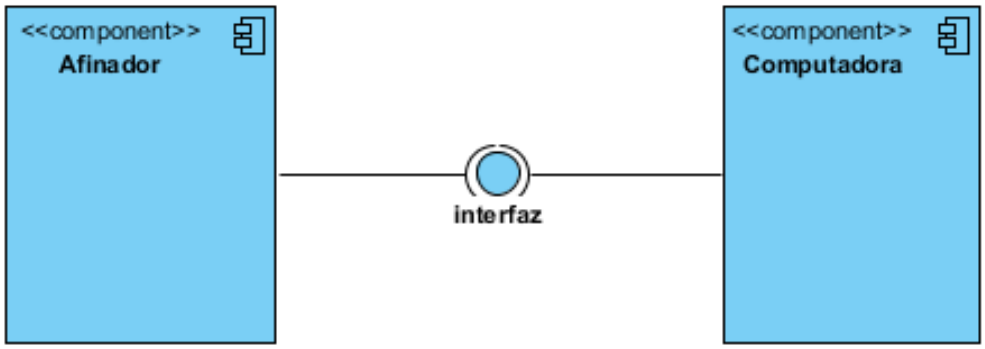
- **Diagrama de Despliegue:**

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



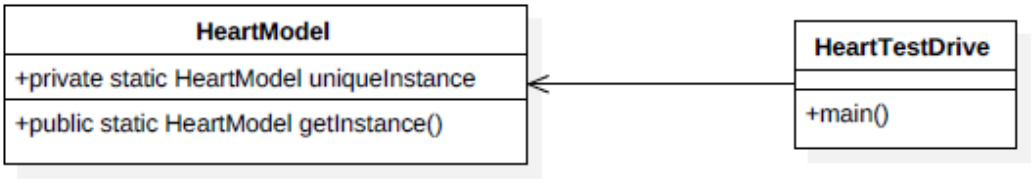
- **Diagrama de Componentes:**

  El siguiente diagrama permite visualizar la estructura de alto nivel del sistema y el comportamiento del Afinador a través de interfaces.

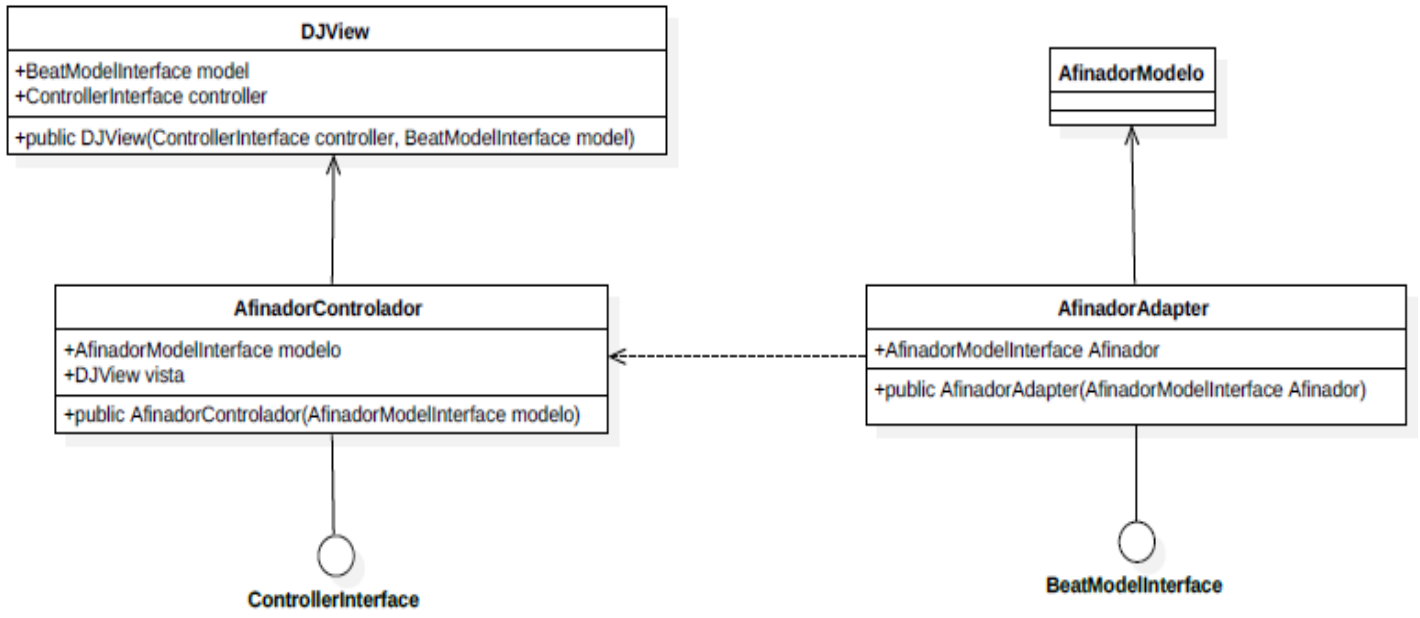


8. Diagrama de Clases

Patrón Singleton



Patrón Adapter DJView



Patrón Observer

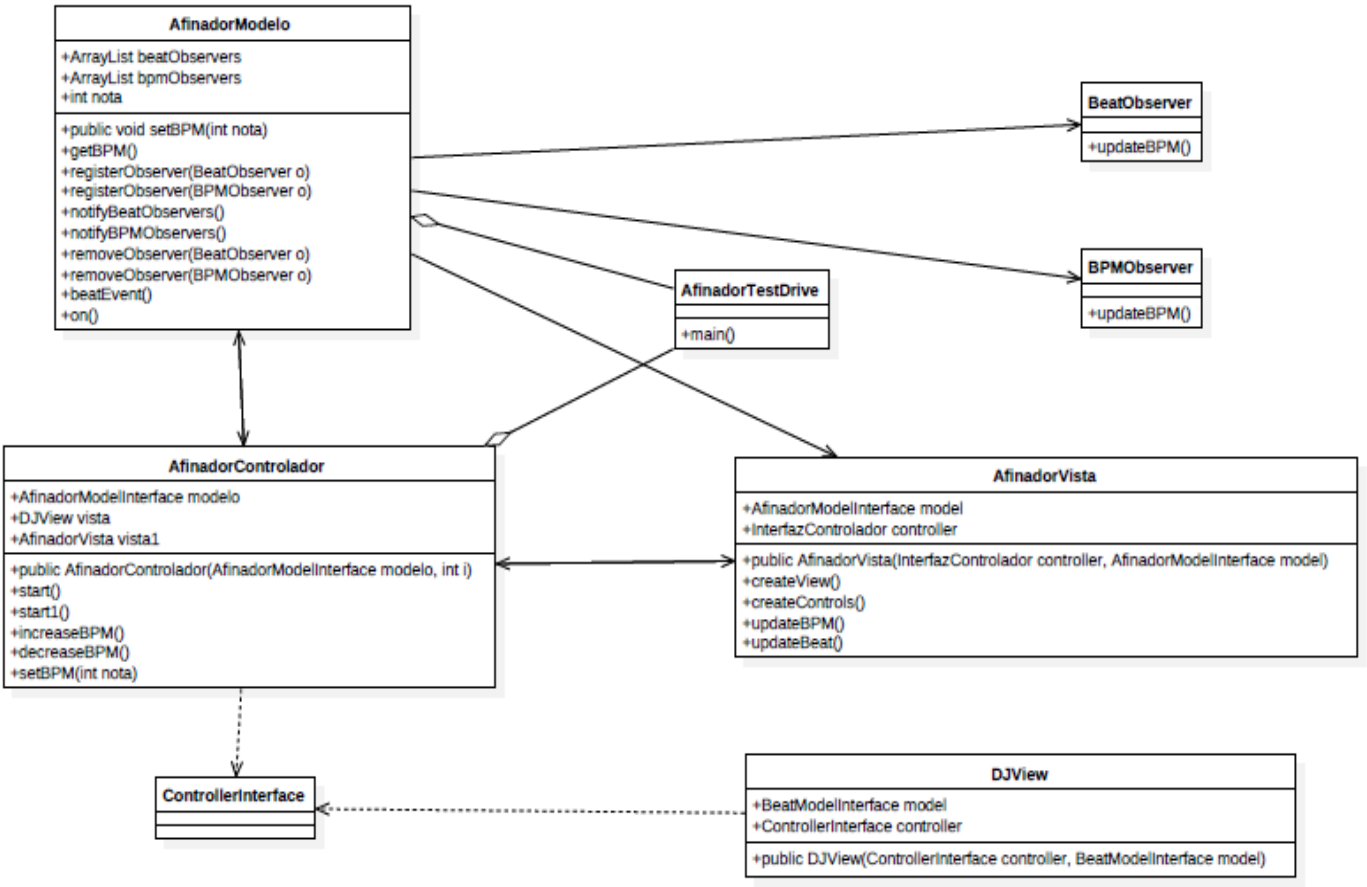


Diagrama de Clases

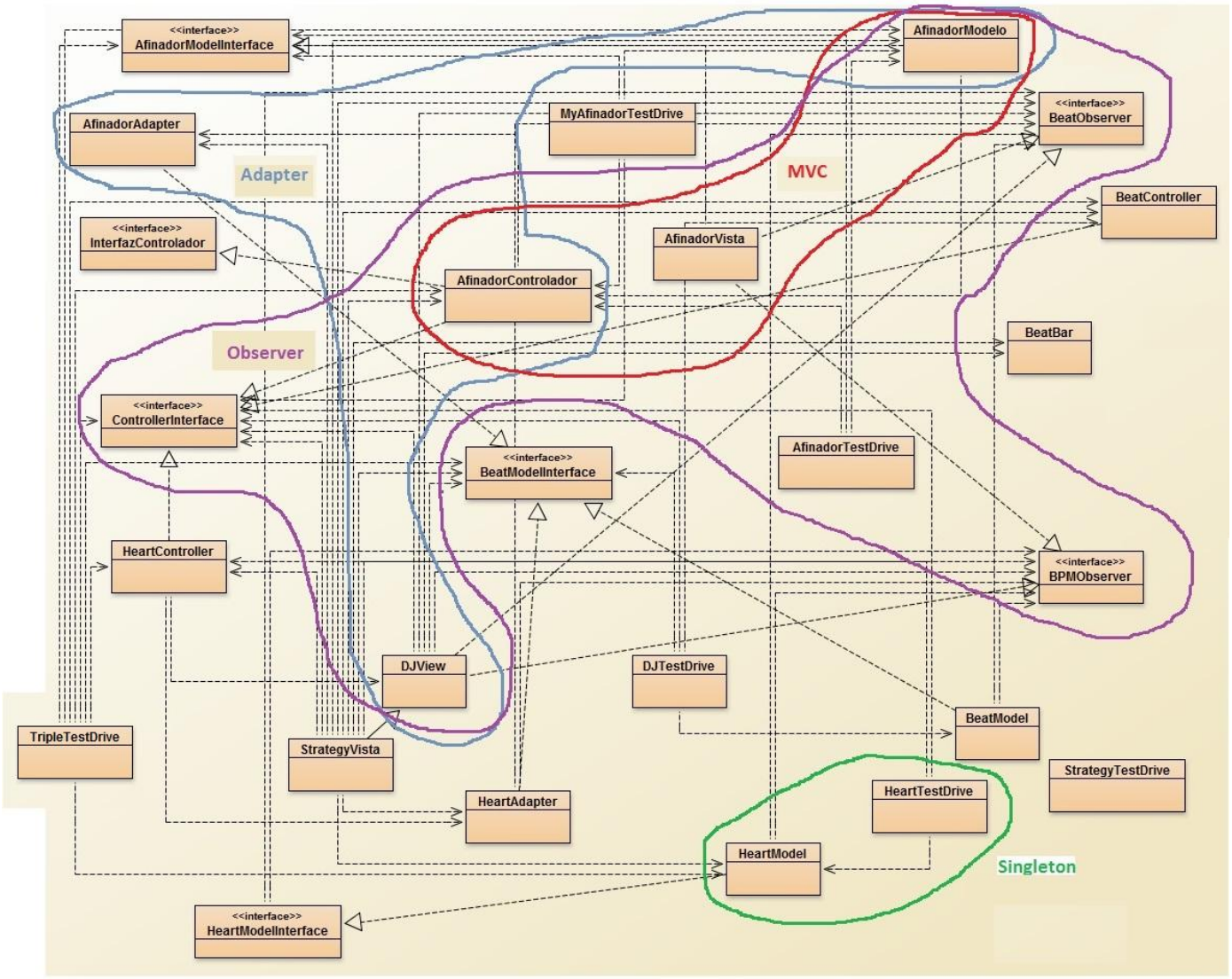
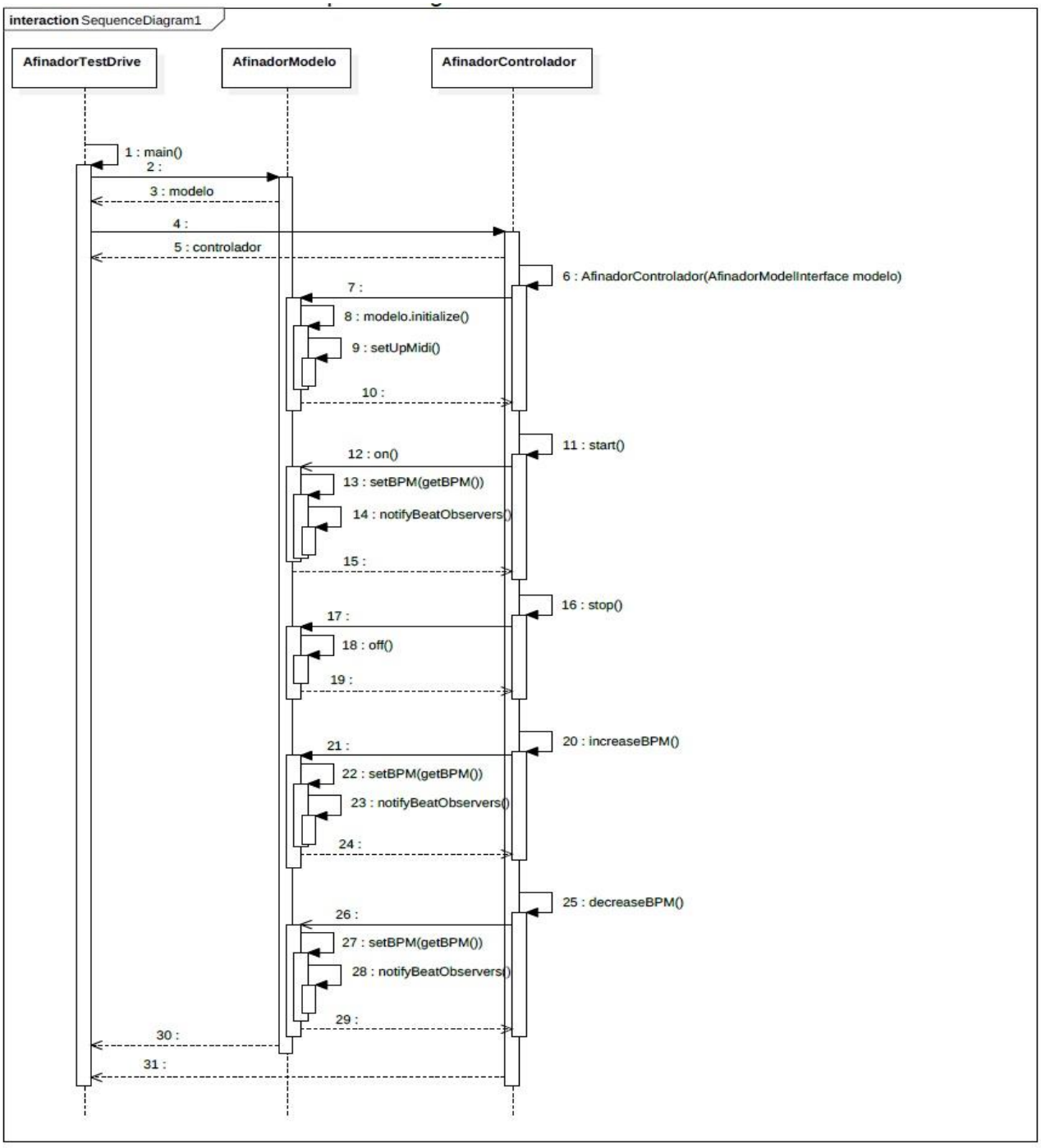
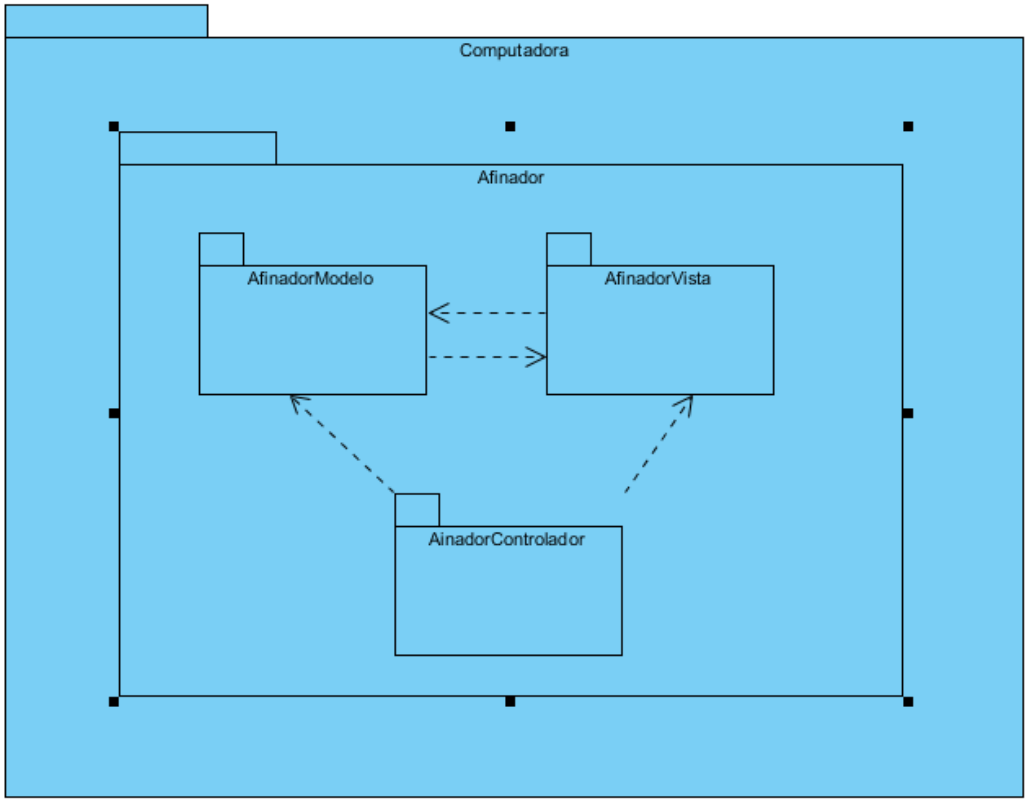


Diagrama de Secuencias:



9. Diagrama de Paquetes:

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



11. Pruebas unitarias automáticas:

Para dichas pruebas automáticas se utilizó la herramienta JUnit, a continuación se exponen los test realizados a nuestro proyecto y sus correspondientes estados

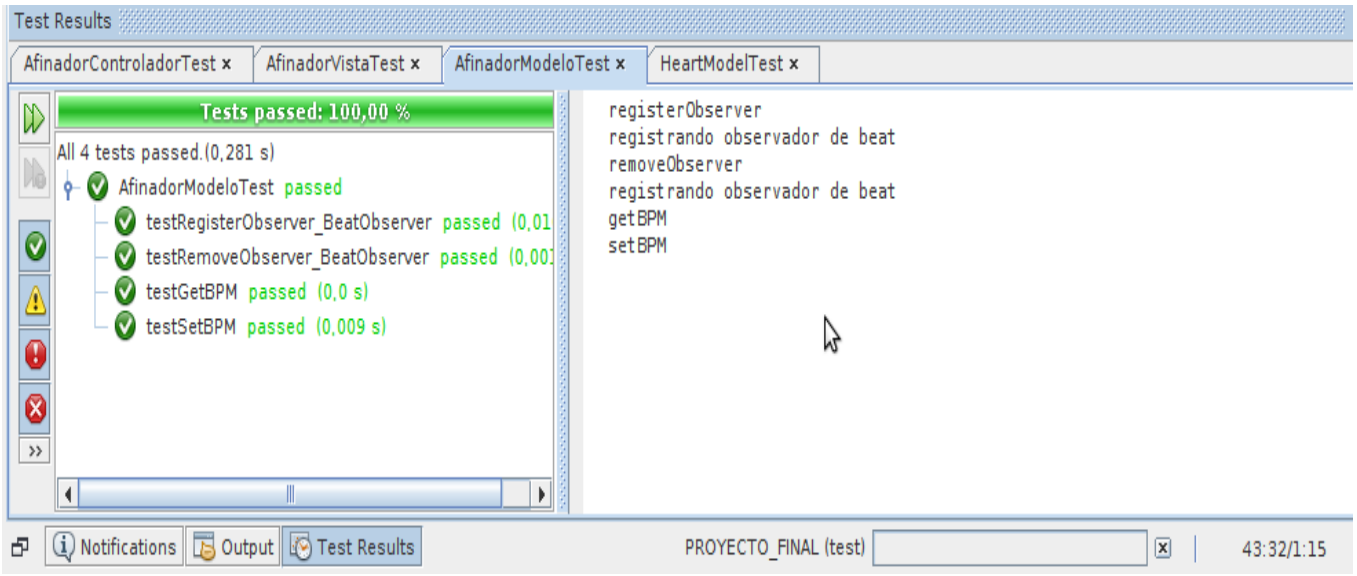
CLASE AfinadorModelo:

- UT1: testSetBPM
  En este test verificamos que el método setBPM este asignándole el valor especificado a la variable “nota”.

- UT2: testGetBPM
  En este test verificamos que el método getBPM este devolviendo el valor que le hemos asignado a la variable “nota” por medio del método setBPM.

- UT3: testRegisterObserver_BeatObserver
  En este test verificamos que el método registerObserver este añadiendo los observadores al arraylist que almacena los mismos.

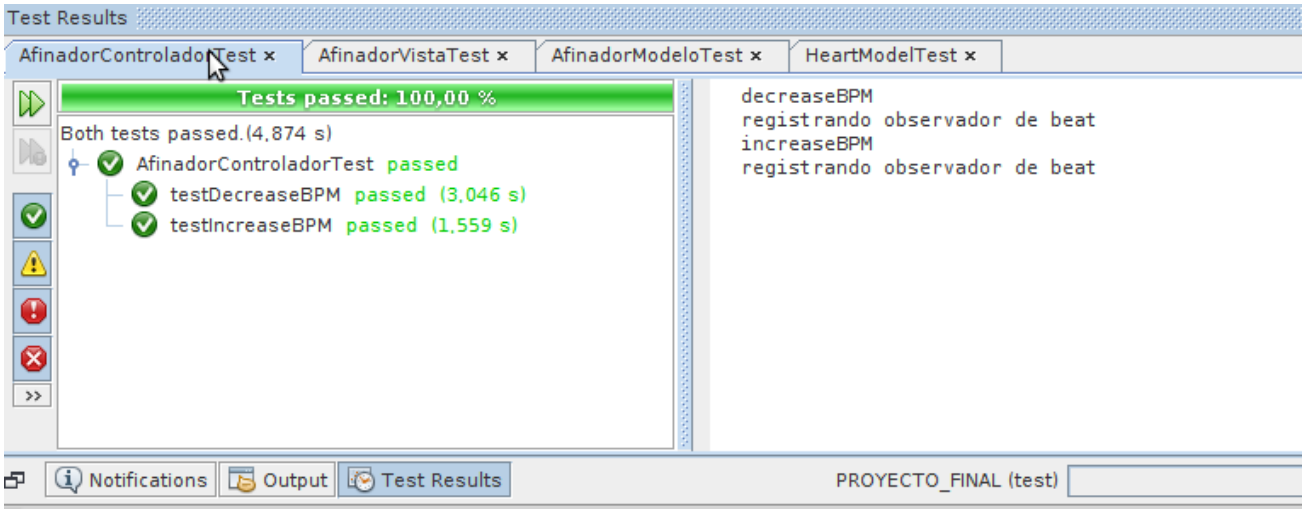
- UT4: testRemoveObserver_BeatObserver
  En este test verificamos que el método removeObserver este quitando los observadores al arraylist que almacena los mismos.



CLASE AfinadorControlador:

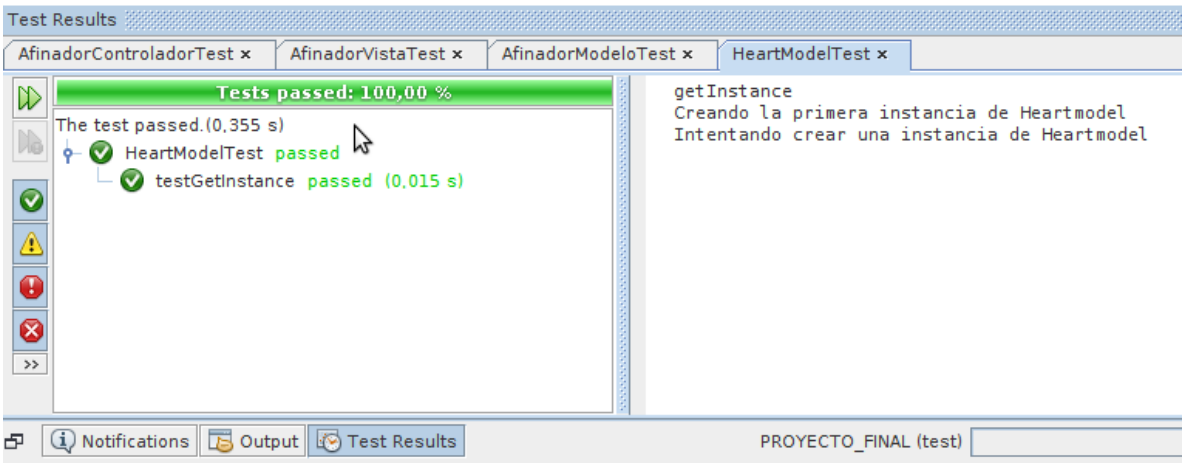
- UT5: testIncreaseBPM
  En este test verificamos que el método increaseBPM cada vez que se ejecuta este aumentando en 1 el valor de la variable “nota”

- UT6: testDecreaseBPM
  En este test verificamos que el método decreaseBPM cada vez que se ejecuta este disminuyendo en 1 el valor de la variable “nota”, además se verifica que estando en valor 0, pase a valer 6 cuando dicho método se ejecuta.



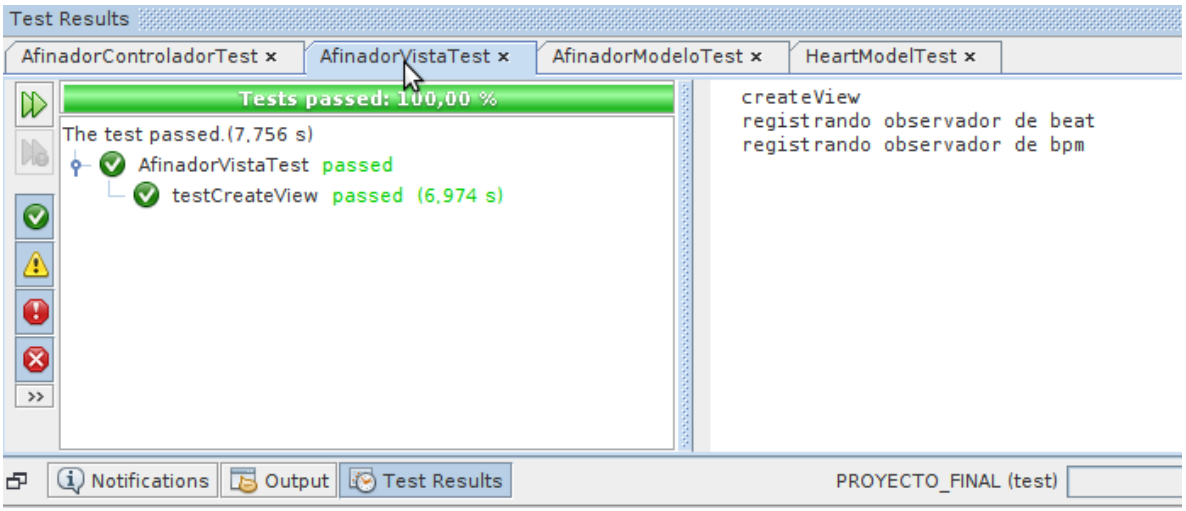
CLASE HeartModel:

- UT7: testGetInstance
  En este test verificamos el correcto funcionamiento del patrón “Singleton” utilizado, corroborando que no se cree más de una instancia de tipo HeartModel.



CLASE AfinadorVista:

- UT8: testCreateView
  En este test verificamos que se cree de manera correcta la vista de nuestro modelo



12. Casos de Prueba del Sistema

CPS1-Prueba de ajuste y corroboración de que luego de terminar la secuencia de las 6 notas volverá a comenzar la misma. (Requerimiento Funcional F1 y F5)

- Ejecutar AfinadorTestDrive.jar
- Presionar el botón “start”
- Presionar el botón “>>” y en la View saldrá la secuencia de notas DO-RE-MI-FA-SOL-LA-SI, a continuación se reinicia la secuencia.
- Comprobar que cuando se Presiona el botón “Quit” se cierra el afinadorTestDrive.

Pass/fail: **Pass**
Realizada por Pavón, Diego

CPS2-Prueba de que la nota se escuchara luego de presionar “Start” (Requerimiento Funcional F2 y F3)

- ✓ Ejecutar AfinadorTestDrive.jar
- ✓ Presionar “Set”, >>, <<, y comprobar que en la “view” continua la leyenda apagado.
- ✓ Presionar el botón “start” y comprobar que suena la nota y en la “view” muestra la nota y la frecuencia.

Pass/fail: **Pass**
Realizada por Esperanza, Manuel

CPS3- Prueba de que el sonido deja de sonar luego de presionar "stop" (F4)

- ✓ Ejecutar AfinadorTestDrive.jar
- ✓ Presionar el botón "start" y comprobar que suena la nota
- ✓ Presionar el botón "Stop" y comprobar que suena la nota

Pass/fail: **Pass**
Realizada por Callejo, Alejandro

CPS4- Prueba de que se puede cambiar a la siguiente o a la anterior nota (F6)

- ✓ Ejecutar AfinadorTestDrive.jar
- ✓ Setear nota deseada y presionar "Start".
- ✓ Comprobar de que cuando se presiona "<<" se escucha la nota anterior a la que está ejecutando.
- ✓ Comprobar de que cuando se presiona ">>" se escucha la nota siguiente a la que está ejecutando.

Pass/fail: **Pass**
Realizada por Callejo, Alejandro

14. Matriz de Trazabilidad de Casos de sistema VS Requerimientos Funcionales

Casos de Prueba de sistema	Requerimientos Funcionales					
	F1	**F2**	**F3**	F4	F5	F6
CPS1	X				X	
CPS2		X	X			
CPS3				X		
GPS4						X

16. Administración de bugs

Utilizamos la herramienta que nos proporciona GitHub llamada “Issues” como gestión de defectos, por lo que los bugs se fueron corrigiendo a medida que avanzamos con el proyecto. El link de dicha herramienta es:
https://github.com/ManuEsperanza/IngSoft-2016--LosMasacre-/issues

DATOS HISTÓRICOS

18. Detalle de dedicación de esfuerzo. En la tabla a continuación se muestra el esfuerzo de cada miembro del grupo medido en horas/persona para la realización de cada tarea.

	Junio													
	02	03	06	08	09	10	11	12	13	15	16	17	18	19	Total
Callejo	2	3	3	5	4	5	4	4	4	3	5	6	6	6	60
Esperanza	2	4	3	4	5	4	5	4	5	3	5	6	5	5	60
Pavón	2	3	3	4	5	4	5	4	3	4	5	6	5	6	59

INFORMACIÓN ADICIONAL

19. Lecciones aprendidas. Profundizamos las buenas prácticas con las herramientas apropiadas para el control de versiones, gestión y seguimiento de los bugs y para la realización de diagramas en las distintas etapas del proyecto.

Además, pudimos poner en práctica los conceptos aprendidos en clase lo cual nos generó preguntas sobre la implementación de los mismos a medida que avanzamos con el proyecto. Cabe destacar que nos sentimos realizados al poder aplicar lo visto durante el transcurso del cuatrimestre, ya que creemos que esta materia es realmente útil porque transmite conceptos y herramientas que son las que se utilizan actualmente en el mercado laboral, al cual esperamos ingresar en un futuro cercano.