

Universidad Nacional de Córdoba

Facultad de Ciencias Exactas, Físicas y Naturales

Ingeniera de Software

Trabajo Práctico Final

Integrantes:

Román Gasparini
Martín Tarres
Franz Yépez Hinostroza

Resumen:

En el presente informe se abordará lo aprendido en la Cátedra Ingeniería de Software aplicándolo al trabajo práctico que se explicará a continuación.

Se parte para el siguiente de líneas de código pertenecientes al ejemplo del libro Head first Design Patterns. En base de ello, se procura realizar un proyecto propio, haciendo uso de las prácticas aprendidas.

Desarrollo:

1. Modifique el HeartModel para que sólo se pueda crear una instancia (usando el patrón Singleton)

y extienda la ventana de control del BeatController para “tratar” de generar nuevas instancias cada vez que se clickea en el botón “>>”. La ventana de la BeatBar debería mostrar en texto el número de intentos de creación de un nuevo HeartBeat model en el texto donde se mostraba la frecuencia cardíaca.

2. Crear un nuevo modelo con su controlador específico que pueda usarse para verse desde la vista

DJView en la ventana BeatBar. Generar un java main class para poder ejecutar tal modificación llamándolo “My<modelName>TestDrive.java” (similar to “HeartTestDrive”). Se proveerán puntos adicionales por la originalidad del modelo creado.

3. Generar una vista propia que permita usar su modelo sin modificar el código existente del ejemplo

pero que permita mostrar los cambios gráficamente y por medio de texto.

4. Generar un TestDrive que permita mostrar a los tres modelos trabajando al mismo tiempo (se esperarán ver al menos 3 ventanas BeatBar con los 3 modelos andando simultáneamente).

5. Modificar la vista BeatBar para que permita cambiar gráficamente en tiempo de ejecución (ej. Mediante un dropdown box) el modelo usado (el Beat model, el Nuevo modelo y el Heartbeat model). Por favor use para tal implementación el patrón strategy. Generar un TestDrive que permita ejecutar tal acción.

6. Utilizar algún sistema como el sistema de Issues de GitHub para gestionar las tareas y defectos encontrados.

7. Incluir todos los documentos generados en el repositorio dentro de una carpeta llamada docs. Los documentos generados deben estar en formato Markdown. Pueden utilizar algún programa para editar este tipo de archivos como MarkDown Pad. De esta manera se podrá observar el historial de cambios sobre los documentos y quien realizó cada cambio.

Nota de Entrega

Estado de Entrega:

Nuestro proyecto se encuentra disponible en el repositorio de Github, en la sección correspondientes a los ejecutables, según corresponde a los criterios establecidos en Manejo de las configuraciones.

Listado de Funcionalidad:

Simula el comportamiento de una heladera, en la cual se le permite establecer la temperatura deseada a alcanzar por medio de dos opciones: seteando por teclado el valor específico que quiere tomar, o subiendo y bajando por unidad con los botones “<<” y “>>”.

Bugs Conocidos:

Se le puede introducir caracteres al momento del seteo, siendo una opción no válida.

Link:

https://github.com/martintarres/Final_InqSoft

Ejecutables:

https://github.com/martintarres/Final_InqSoft/tree/master/Ejecutables

Manejo de las Configuraciones

Dirección y forma de accesos a la herramienta de control de versiones

Para el desarrollo del software usaremos la herramienta de control de versiones GitHub. Creamos un repositorio, en el cual los colaboradores podrán no sólo acceder a los items en desarrollo sino también podrán aportar sus propios progresos de forma compartida.

La dirección del repositorio es: https://github.com/martintarres/Final_IngSoft

Esquema de directorios y propósito de cada uno

La sección correspondiente al código desarrollado, para el caso de la rama principal, se encontrará en la carpeta src/main/java/headfirst/combined/djview/.

En el caso de la rama principal será:

https://github.com/martintarres/Final_IngSoft/tree/master/src/main/java/headfirst/combined/djview

Los archivos ejecutables se encuentran almacenados en la carpeta Ejecutables del repositorio, correspondiente a la última versión de la rama master.

https://github.com/martintarres/Final_IngSoft/tree/master/Ejecutables

Los documentos, como imágenes agregadas, serán guardados en la carpeta docs.

Normas de etiquetado y de nombramiento de los archivos

En concordancia con los miembros del equipo, las normas para nombrar un nuevo archivo son:

Los nombres de las clases e interfaces creadas comienzan con mayúscula, y si el nombre es compuesto de dos o más palabras, lleva mayúscula la primera letra correspondiente a dicha palabra.

Ejemplo: BeerFridgeModel.java

La parte final del nombre hace referencia a su función.

Ejemplo: BeerFridgeController especifica que es el controlador correspondiente a BeerFridge.

Con respecto del etiquetado, se optó por la forma convencional.

Ej Versión 4.2.3

El primer número hace referencia a cambios muy profundos realizados en esta versión en comparación con la anterior.

El segundo, a mejoras de importancia relativa.

El tercer número denota que ha habido modificación que no hace demasiada variación.

Plan del esquema de ramas a usar

Se inicializa el repositorio subiendo el código del Head First Design Pattern a la rama master, la única existente en el inicio. Luego se bifurcó el proyecto en ramas para el desarrollo de cada uno de los miembros. Las versiones más estables se subirán al master, haciendo uso de las ramas sólo para el desarrollo temporal, previniendo el futuro mergeo a la rama principal (master).

La rama Test es en la que se desarrolló todo lo relativo a testeo. No consideramos necesario mergearlo a la rama principal, para que quedara lo más simplificada posible.

Políticas de Mergeo

Sólo se podrá mergear a la rama master una vez que el ítem desarrollado en la rama se encuentre estable. Las condiciones de estabilidad en esa etapa son: que compile necesariamente, y que cumple la función por la cual que creada la rama a la que pertenece.

Especificaciones de Release

El proyecto será entregado en formato JAR, por lo que no es necesaria ningún instalador específico del proyecto para ejecutarlo.

Cabe destacar que sí es necesario que el cliente tenga JRE (JAVA Runtime Environment). El anterior se encuentra disponible:

<https://www.java.com/es/download/>

Integrantes:

Nombre	Función	Forma de contacto
Román Gasparini	Desarrollador/Gestor de proyectos	roman.gasparin@gmail.com
Martín Tarres	Desarrollador/Diseñador	martintarres@gmail.com
Franz Yépez Hinostroza	Desarrollador/Arquitecto	alexander.fyh@gmail.com

La comunicación entre los integrantes del equipo se realiza generalmente de forma presencial, por otro medio se usa solamente para acordar reuniones semanales y dar detalles generales.

En la reunión, es necesaria la presencia de cada uno de los miembros. En ella se especifica las nuevas directrices a tomar, luego de ello se procede a trabajar de forma conjunta si la situación lo requiere.

Herramienta de seguimiento de bugs

Para ello, se utiliza una herramienta propia de github.

https://github.com/martintarres/Final_IngSoft/issues

En ella se debe especificar el error, a qué versión corresponde, y mostrar si se pudo solucionar en el progreso.

Varios:

Cabe destacar que se hizo uso de la herramienta de integración continua TRAVIS CI.

https://travis-ci.org/martintarres/Final_IngSoft

Para diagrama UML, se empleó WhiteStar UML.

Requerimientos

Para facilitar el entendimiento, se prosigue a presentar diagramas UML.

Diagrama de Casos de uso:

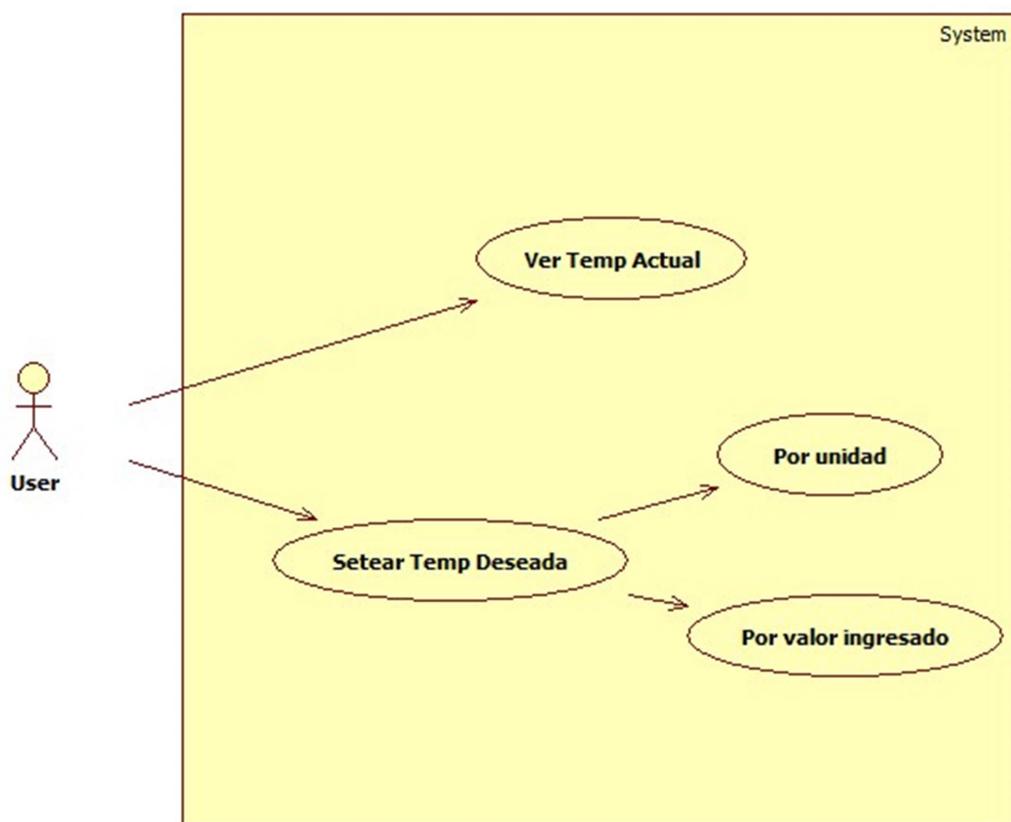
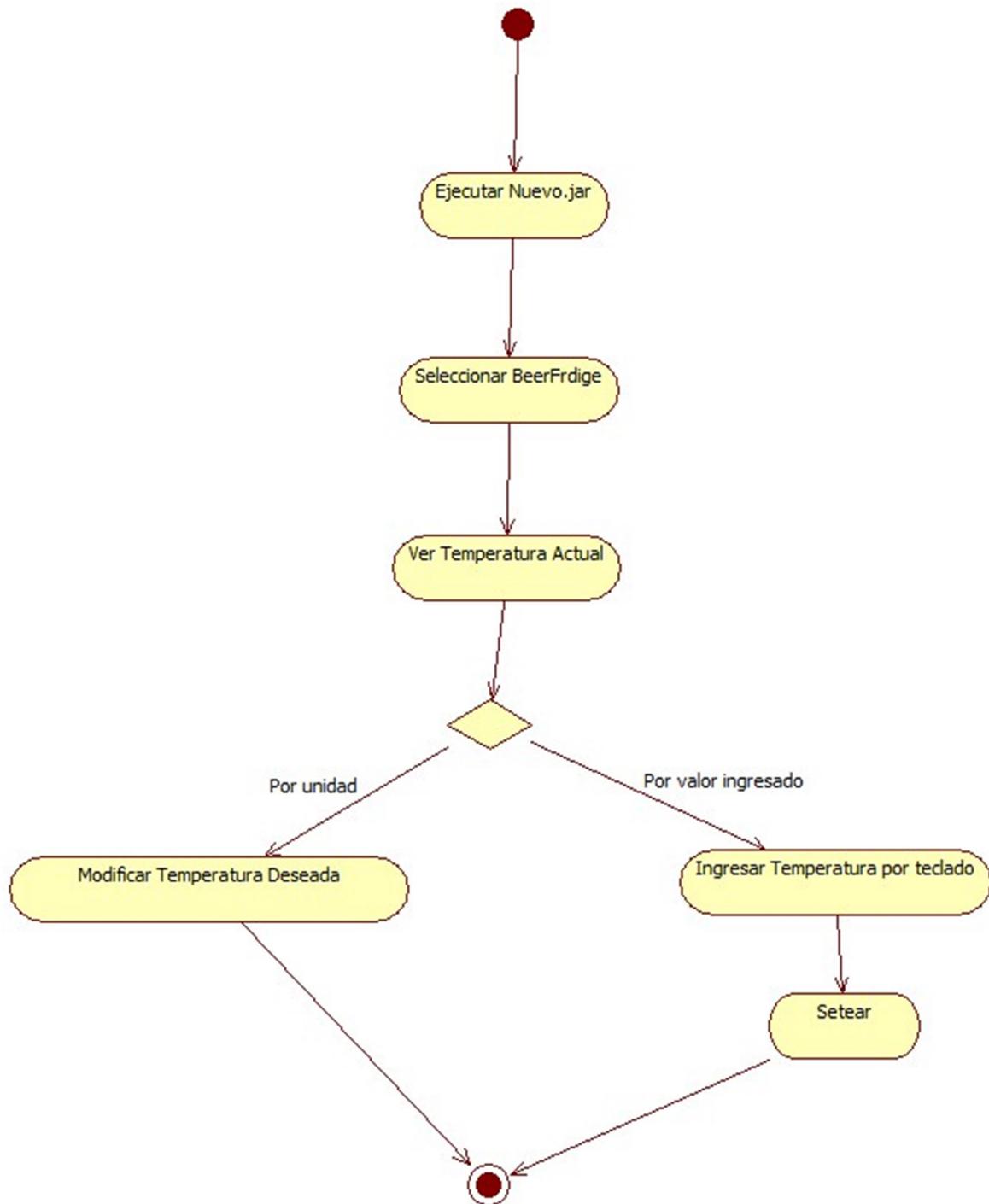


Diagrama de Actividad:



Requerimientos Funcionales:

1. El software debe asegurar una sola instancia de la clase HeartModel. Para el caso que se intente crear otro nuevo objeto tipo HeartModel, se debe mostrar la cantidad de veces fallidas que se trató de instanciar.
2. El código debe permitir la implementación de un nuevo modelo con su propio controlador utilizando la vista predefinida anteriormente.
3. Se debe ofrecer una nueva vista propia para el nuevo modelo a desarrollar.
4. Es necesario verificar el funcionamiento de los 3 modelos actuando en paralelo, sin que se comprometa el comportamiento independiente de ambos.
5. Modificar la vista para poder elegir qué modelo correr. Esto debe poder hacerse en tiempo de ejecución.
6. El comportamiento de los sistemas anteriores no debe verse afectado por la creación de un nuevo sistema. Esto quiere decir, que al crearse el sistema correspondiente a BeerFrdige, no se alteró las funcionalidades de los demás.
7. Debe ser posible setear el valor de la temperatura deseada al modelo de BeerFrdige, ya sea por medio del botón Set o por los botones >> y <<.

Requerimientos no funcionales:

1. La temperatura actual debe tender a la temperatura deseada. Para ello, luego de un lapso de tiempo definido proporcionalmente con la diferencia inicial entre la temperatura actual y la deseada, la diferencia en ese instante no debe ser mayor a 5.
2. El sistema requiere de una capacitación previa del usuario, así los valores a setear son lógicos (se debe ingresar nada más que valores enteros, y no caracteres, por ejemplo).

Matriz de Trazabilidad

	Ver temperatura Deseada	Setear temperatura actual
F1	1	1
F2		
F3		
F4		
F5	1	
F6	1	1
F7		1
NF1		1
NF2		

Arquitectura:

El patrón de arquitectura utilizado para nuestro proyecto fue el MVC (model-view-controller), lo que nos permitió desarrollar cada parte del trabajo en módulos relativamente independientes entre sí.

A su vez, también se implementó el patrón Strategy, lo que nos ofreció poder cambiar la vista en tiempo de ejecución, por ejemplo.

El patrón Observer fue utilizado para notificar de cambios a la vista, habiendo variado valores del modelo.

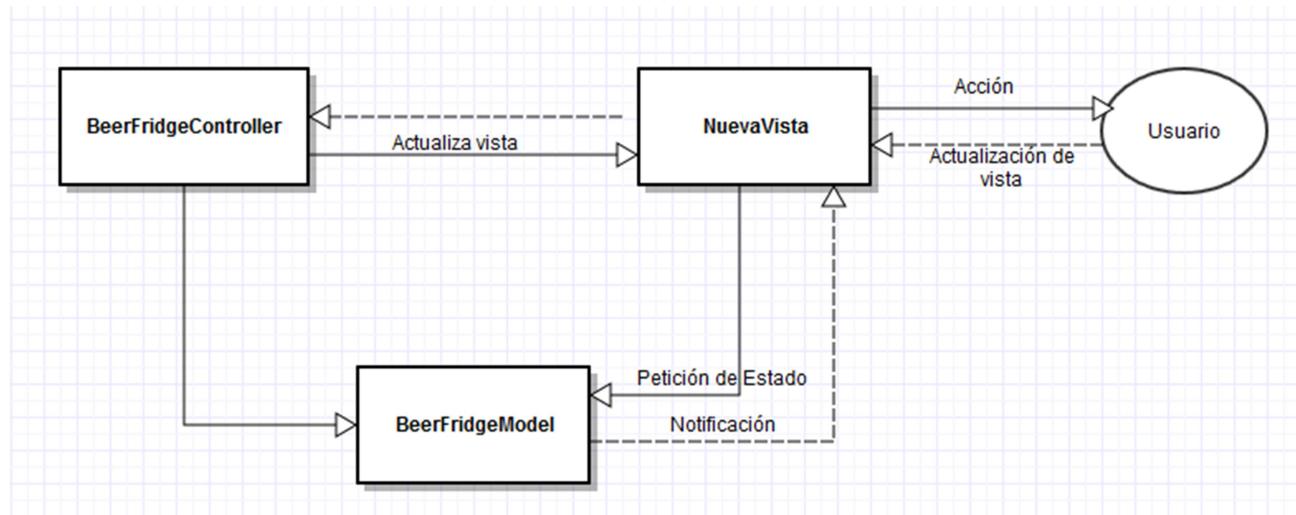
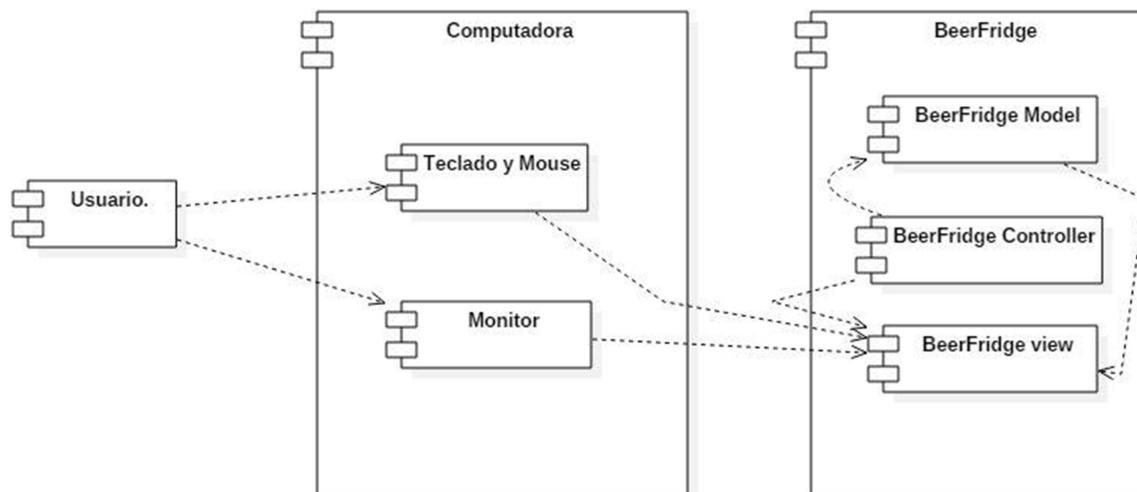


Diagrama de Despliegue



Diseño e Implementación:

Diagrama de clases

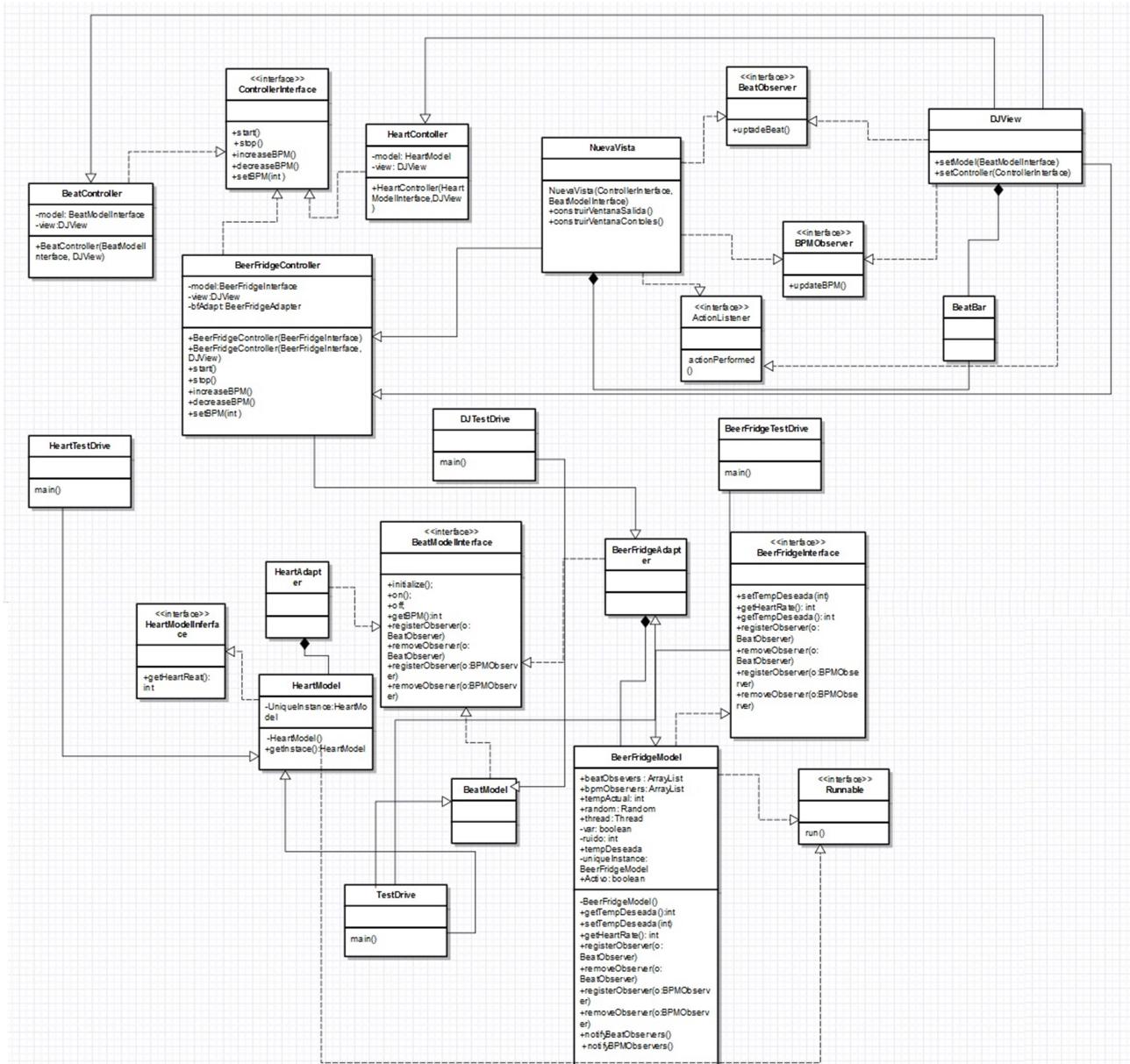


Diagrama de Secuencia:

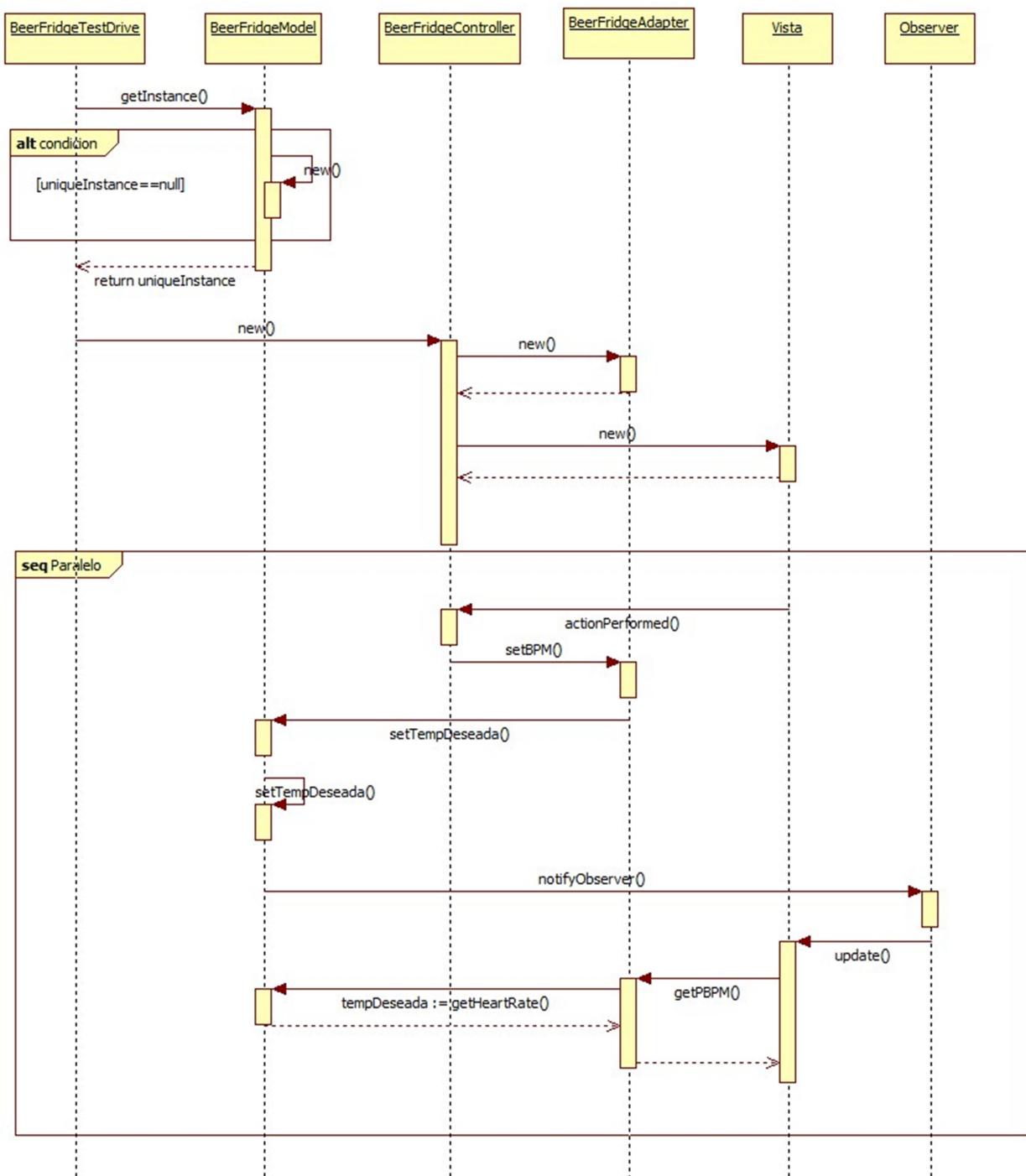
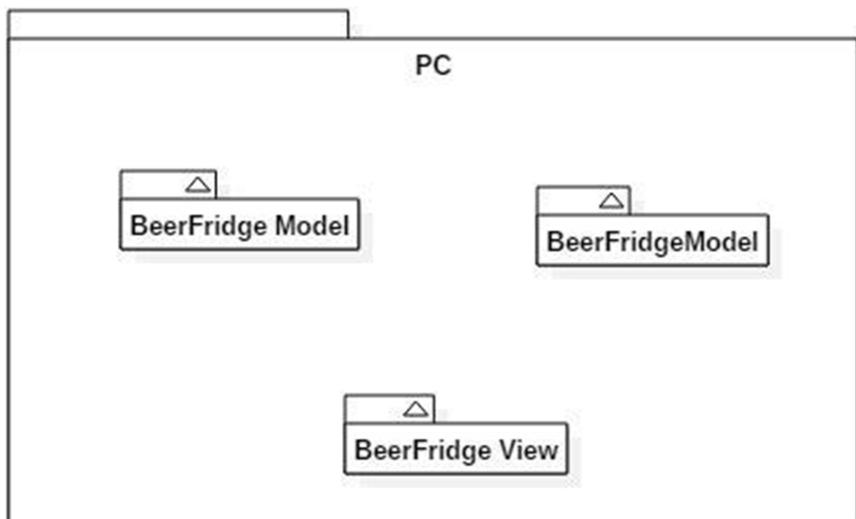


Diagrama de Paquete:



Pruebas.

Unitarias:

Para el uso de pruebas unitarias se utilizó las herramientas provistas por la IDE IntelliJ Idea.

Se uso la test BeerFridgeTestDriveTest para confirmar que se permitiera sólo una instanciación de la clase BeerFridgeModel.

Se configuró para que la prueba corriera diez veces, y saltara excepción si no cumplía su objetivo el patrón Singleton.

Todas las veces que se corrió, la prueba pasó exitosamente como se esperaba, no se crearon en los intentos más que una sola instancia, por lo que se puede concluir que se satisfizo el requerimiento funcional previsto.

The screenshot shows the IntelliJ IDEA interface with the following details:

- Code Editor:** The main window displays the `BeerFridgeTestDriveTest.java` file. The code is as follows:

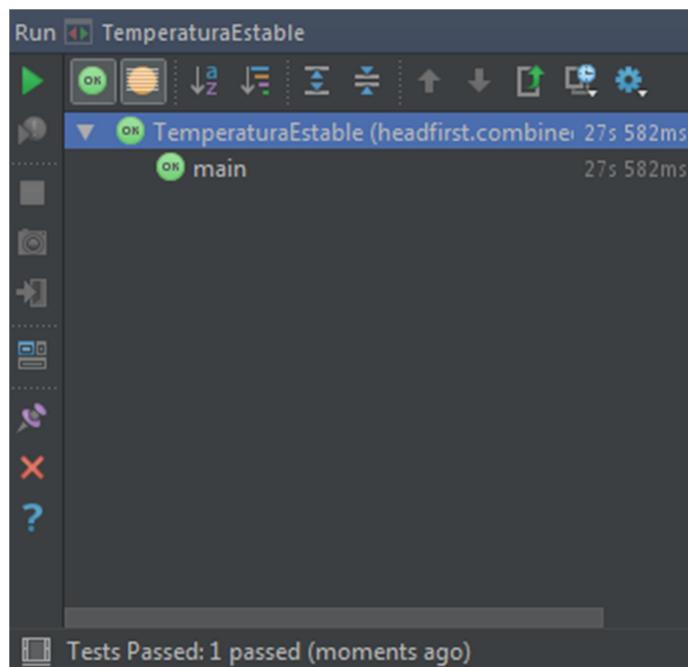
```
import ...  
  
/** * Created by Yepez Hinostroza on 19/06/2016.  
 */  
public class BeerFridgeTestDriveTest {  
    @Test  
    public void main() throws Exception {  
        BeerFridgeModel beer1= BeerFridgeModel.getInstance();  
        BeerFridgeModel beer2= BeerFridgeModel.getInstance();  
        if(beer1!=beer2)  
        {  
            throw new Exception("Se ha creado otra modelo");  
        }  
    }  
}
```

- Run Tool Window:** The bottom panel shows the results of the run. It indicates "All 10 tests passed - 170ms". The log output shows 10 entries of "main" taking 0ms each, followed by "Process finished with exit code 0".
- Status Bar:** The bottom right corner shows the status bar with the time (10:4), file encoding (CRLF), and Git status (master).

Test de Sistema:

Para comprobar la efectividad del sistema, se tomó como parámetro que llegara a la temperatura deseada luego de cierto tiempo definido proporcionalmente por su diferencia inicial . Si la diferencia entre la actual y la deseada luego de ese tiempo era muy grande, se consideraría que no cumple con el requerimiento no funcional: que enfrié un un tiempo estipulado por el cliente.

Para ello, se utiliza el Test TemperaturaEstable. Sólo se lo hizo correr 3 veces, ya que demora muchísimo más que el anterior cada prueba.



Cobertura:

73% classes, 70% lines covered in package 'headfirst.combined.djview'			
Element	Class, %	Method, %	Line, %
BeatBar	100% (1/1)	100% (2/2)	100% (12/12)
BeatController	100% (1/1)	100% (7/7)	100% (31/31)
BeatModel	100% (1/1)	82% (14/17)	77% (65/84)
BeerFridgeAdapter	100% (1/1)	50% (5/10)	61% (11/18)
BeerFridgeController	100% (1/1)	71% (5/7)	60% (15/25)
BeerFridgeModel	100% (1/1)	83% (10/12)	81% (50/61)
BeerFridgeTestDrive	0% (0/1)	0% (0/1)	0% (0/4)
Diferenciador	0% (0/1)	0% (0/2)	0% (0/7)
DJTestDrive	100% (1/1)	100% (1/1)	75% (3/4)
DJView	100% (4/4)	83% (15/18)	95% (148/155)
HeartAdapter	100% (1/1)	60% (6/10)	71% (10/14)
HeartController	100% (1/1)	85% (6/7)	52% (10/19)
HeartModel	100% (1/1)	100% (10/10)	100% (49/49)
HeartTestDrive	0% (0/1)	0% (0/1)	0% (0/4)
NuevaVista	0% (0/1)	0% (0/6)	0% (0/78)
TestDrive	0% (0/1)	0% (0/1)	0% (0/8)

Diagrama de Arquitectura

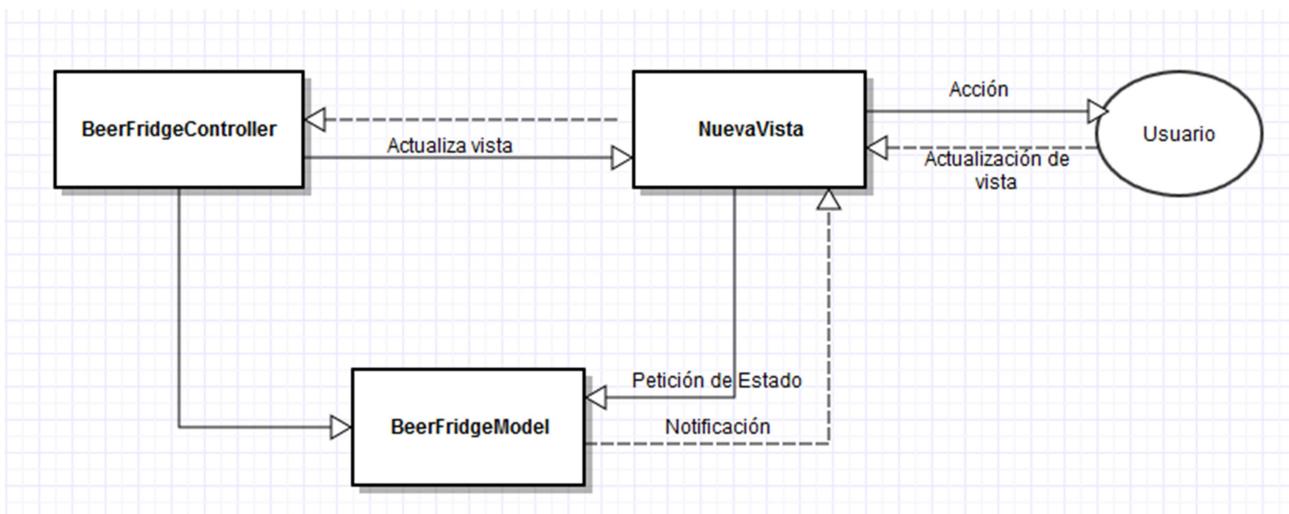
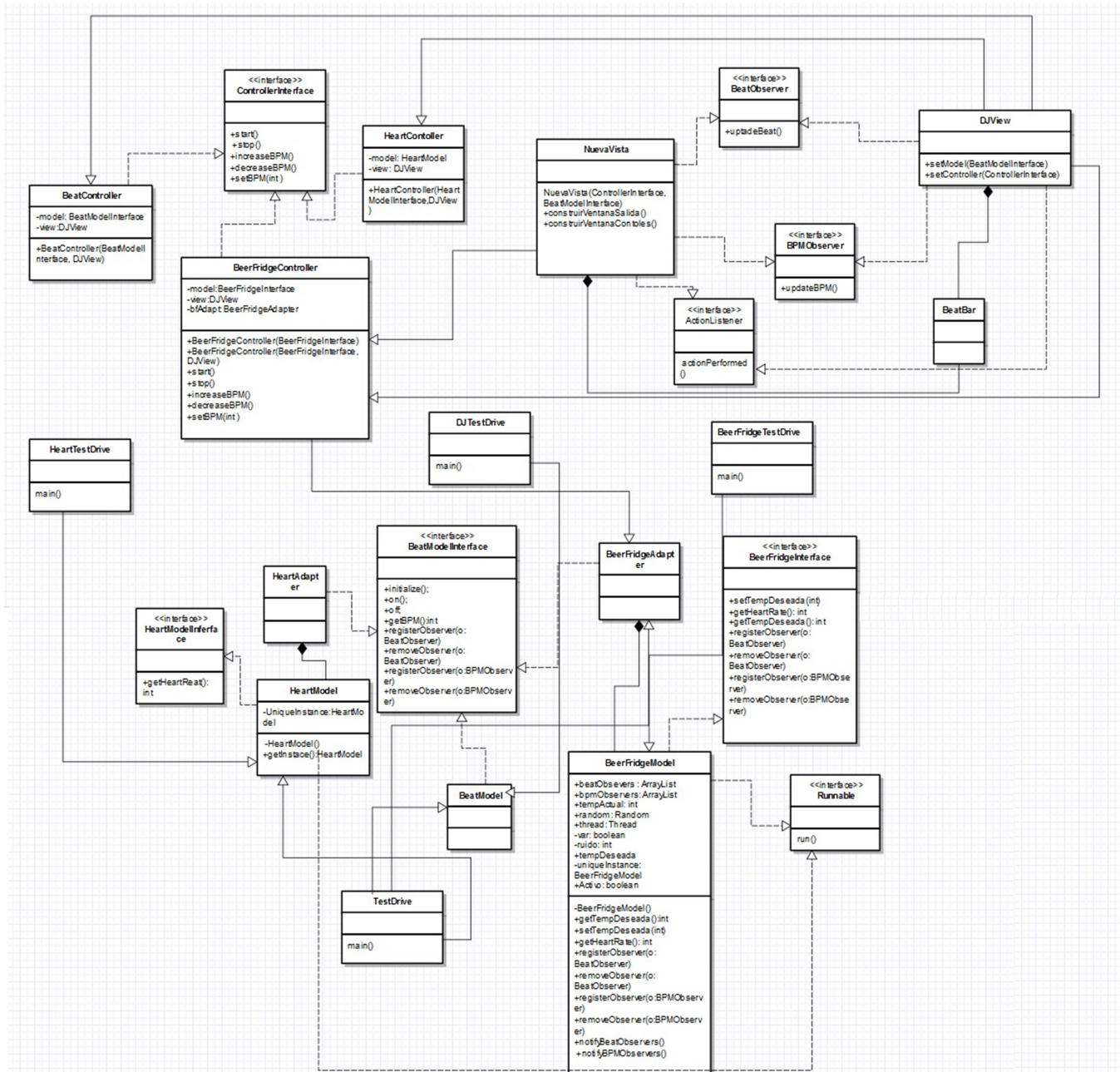
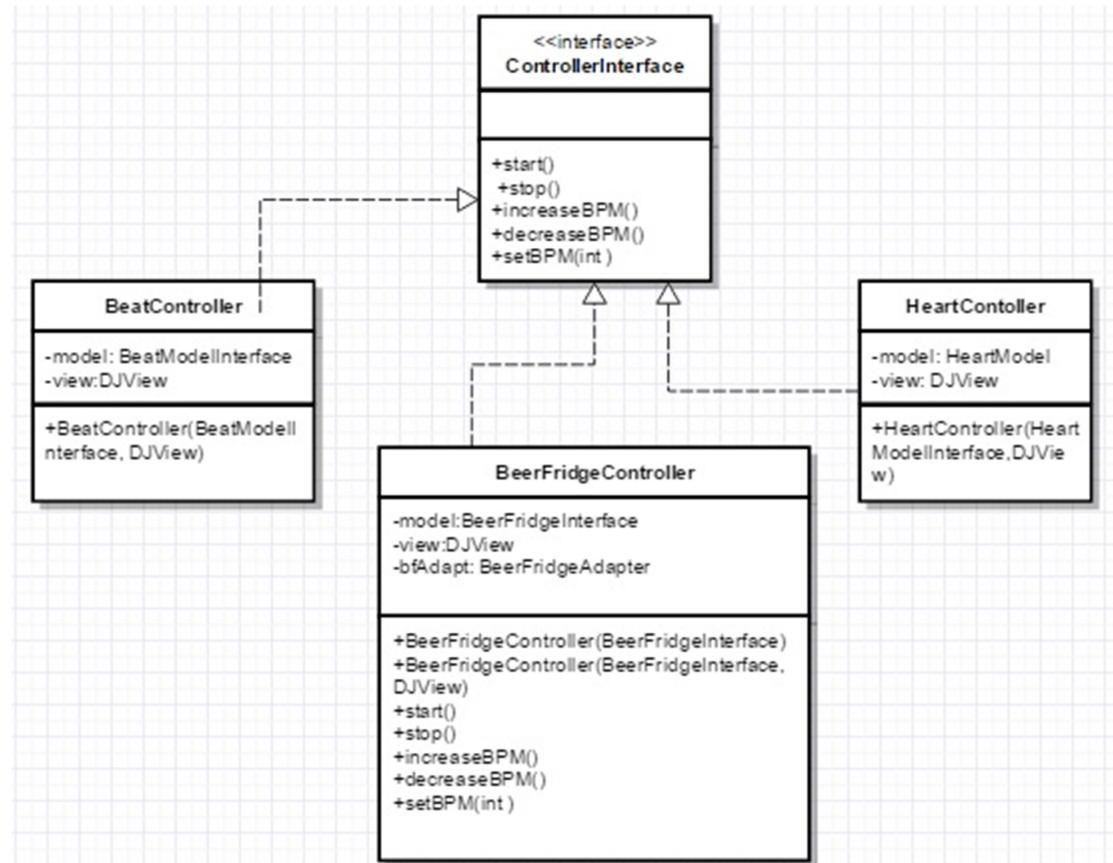


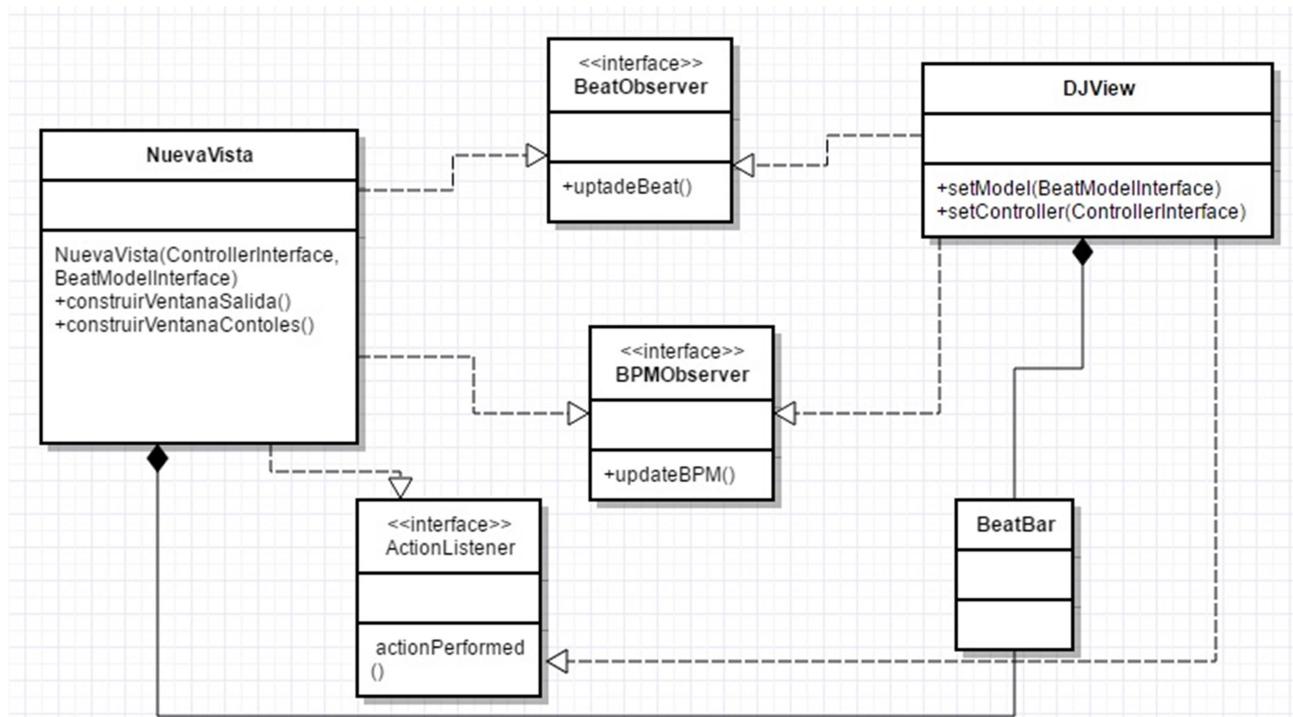
Diagrama de clases



controllers



vistas



Models

