

Practica 1 DS - Grupo DS2_4

- Luis Guerra Batista
- Jose Jiménez Cazorla
- Alberto Llamas González
- Oscar López Maldonado

Para facilitar la corrección hemos proporcionado todos los diagramas de clase de cada módulo, así como quién ha realizado cada módulo y una imagen de una ejecución que muestra el resultado final.

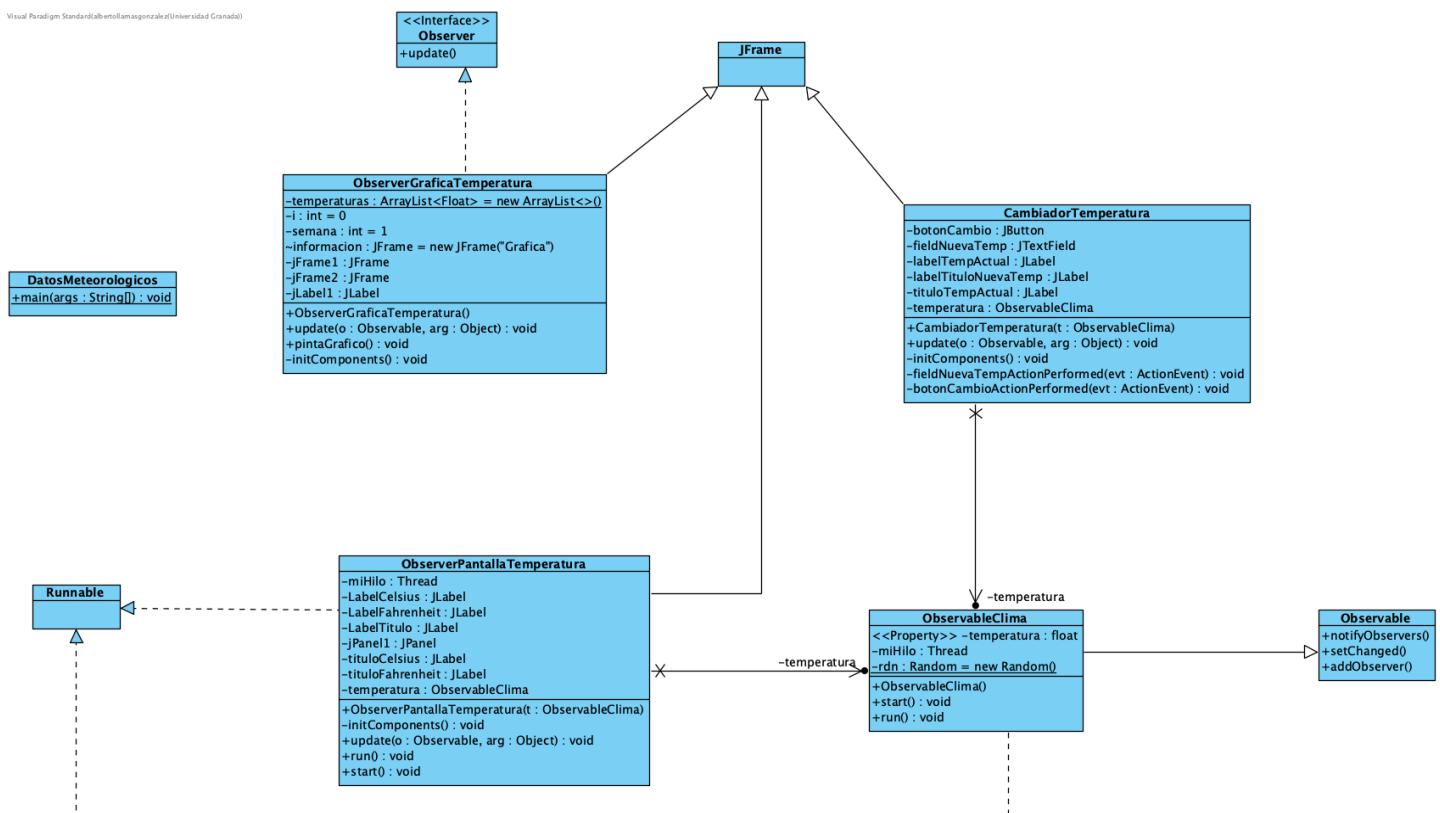
Sesión 1

Módulo 1 : Patrón Observador para la monitorización de datos

meteorológicos (Jose Jiménez y Alberto Llamas)

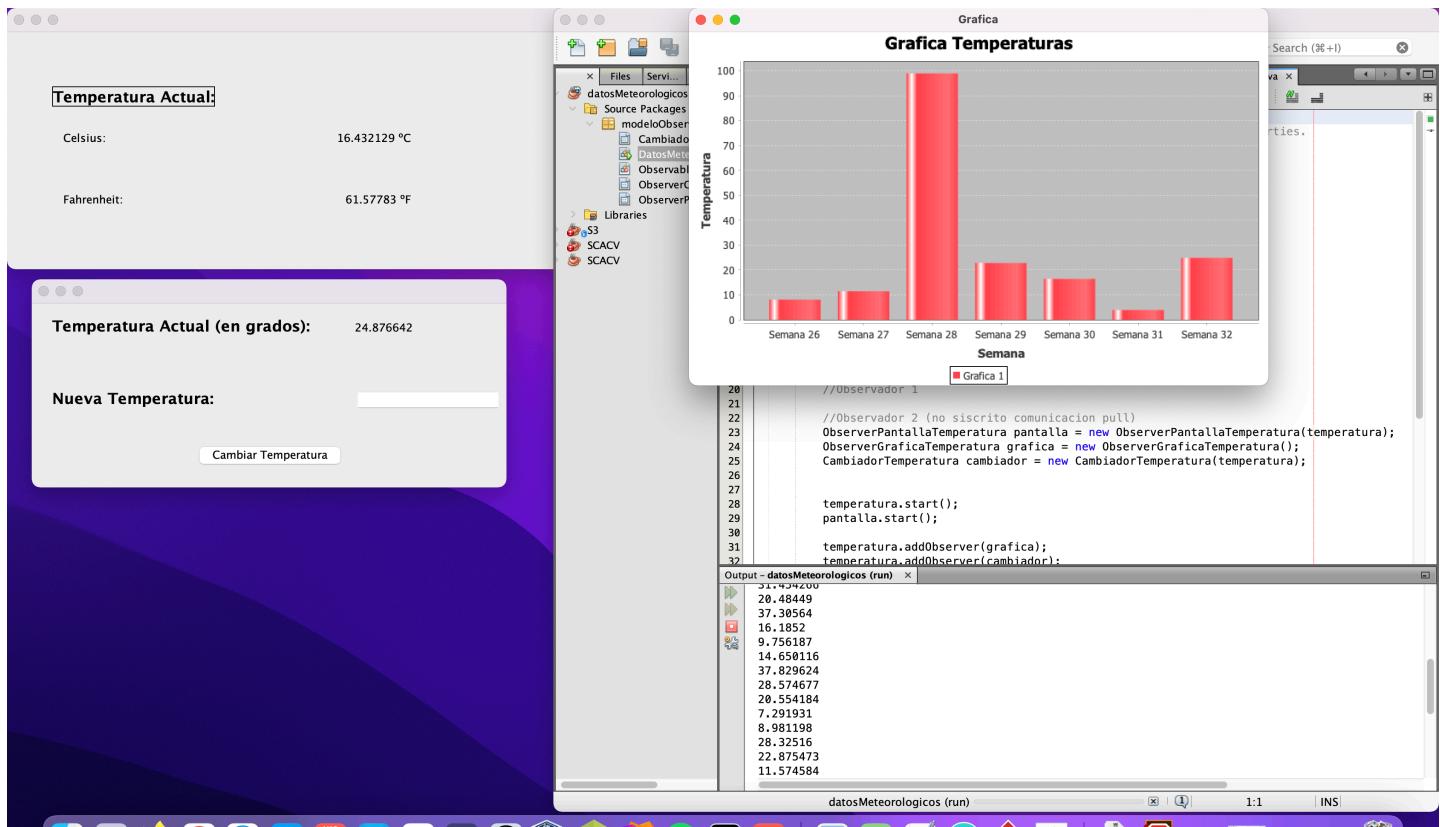
Hemos implementado la clase Clima (ObservableClima) como **sujeto Observable**. Tenemos además tres **Observers**; gráficaTemperatura, pantallaTemperatura y botonCambio (en nuestro caso le hemos llamado CambiadorTemperatura). De estos Observers, gráficaTemperatura y botonCambio están suscritos (aunque botonCambio rompe parte de la filosofía de este diseño) mientras que pantallaTemperatura no está suscrito. Mediante una hebra cada 3 segundos se comprueba si ha habido una modificación del observable.

Diagrama de clases



Captura de ejecución

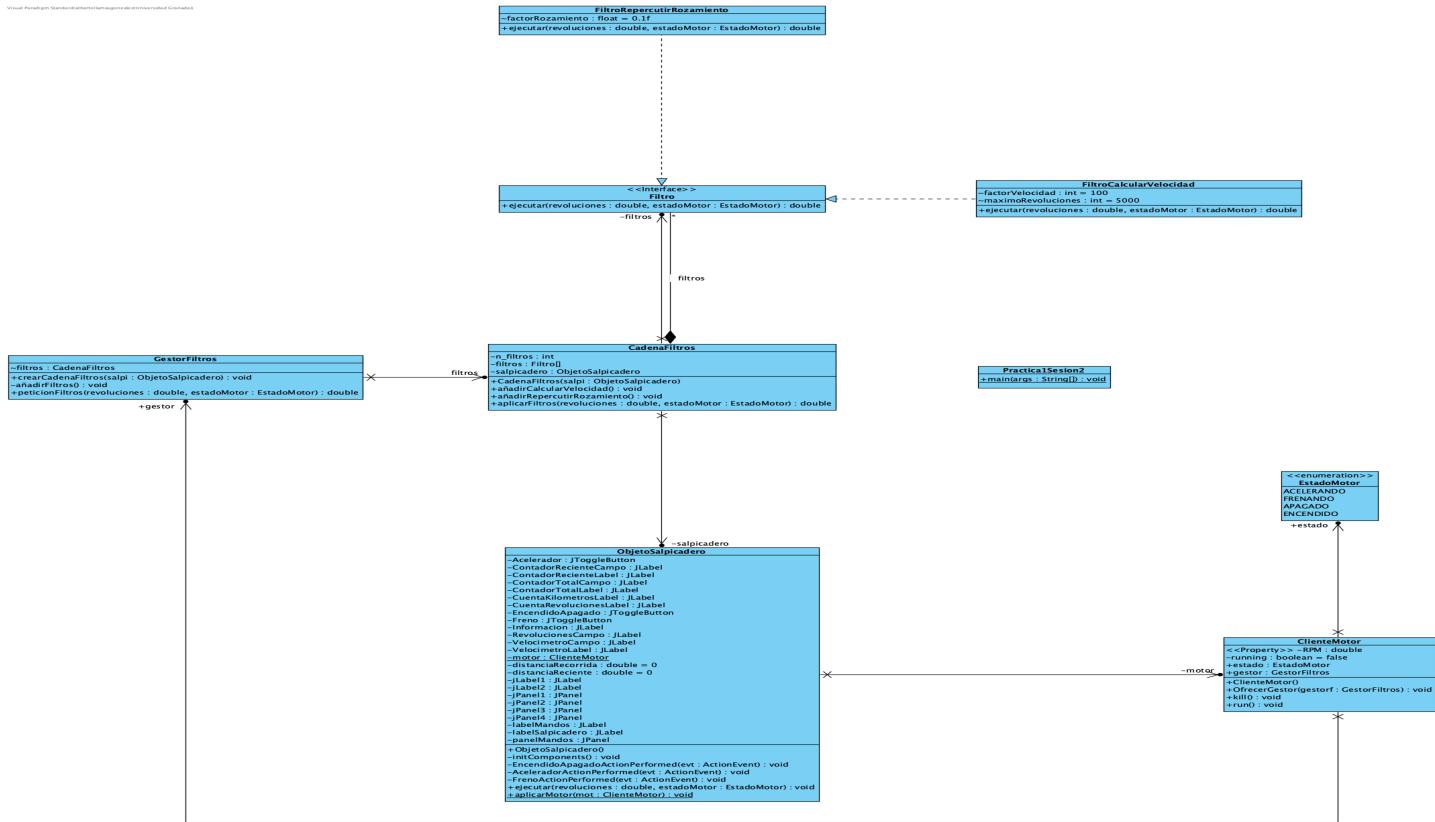
Para mostrar el funcionamiento, hemos introducido con en la ventana CambiadorTemperatura la temperatura 99 y como vemos en la gráfica, se actualiza correctamente.



Módulo 2 : Patrón arquitectónico Filtros de Intercepción para simular el movimiento de un vehículo con cambio automático (Luis Guerra y Oscar López)

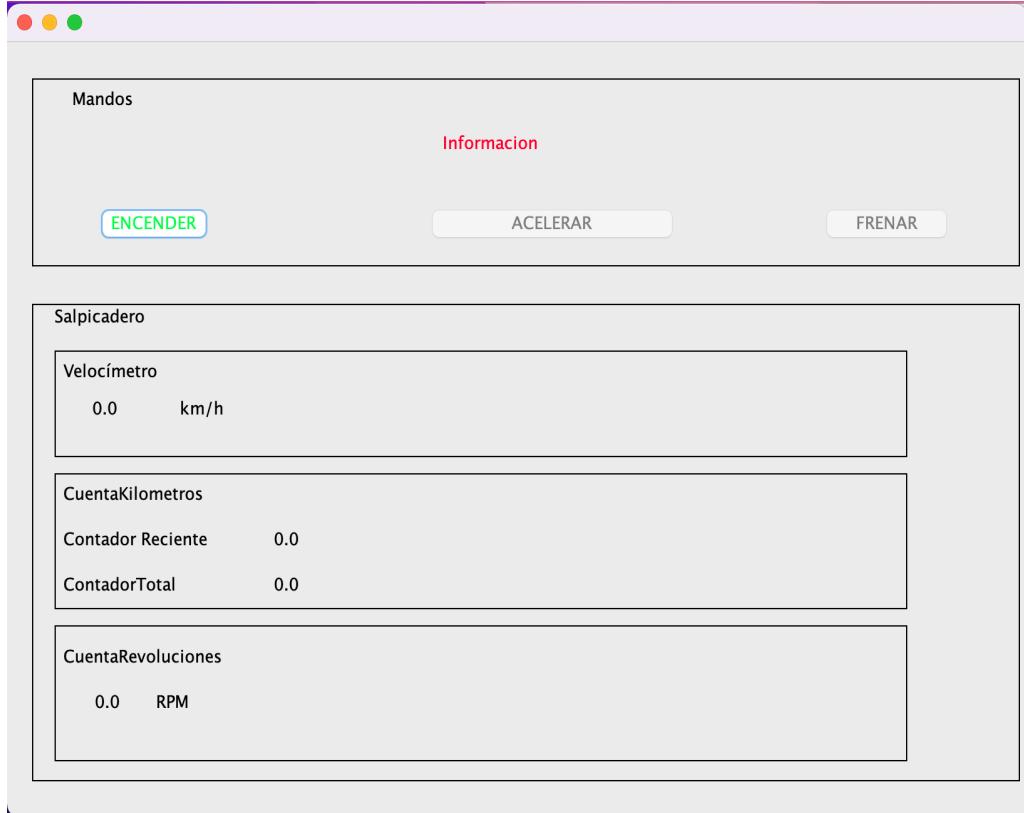
En este supuesto, y bajo el patron arquitectonico Filtros de Intercepcion, hemos ido dando a cada uno de los elementos de nuestro vehiculo su papel dentro del patron. Ya sea el papel del cliente para el motor ("ClienteMotor") y el papel de Objetivo/Target al Salpicadero ("ObjetoSalpicadero"). Por tanto, y siguiendo el patron designado, seria el motor el que estaria constantemente invocando al gestor de filtros para que este, seguidamente, pasara la informacion dada por el motor a su cadena de filtros. Esta poseeria los filtros pertinentes por los que debiese pasar la informacion para, finalmente, terminar dandole esta informacion al Target, que en nuestro caso es el salpicadero, y vista, de nuestro programa.

Diagrama de clases

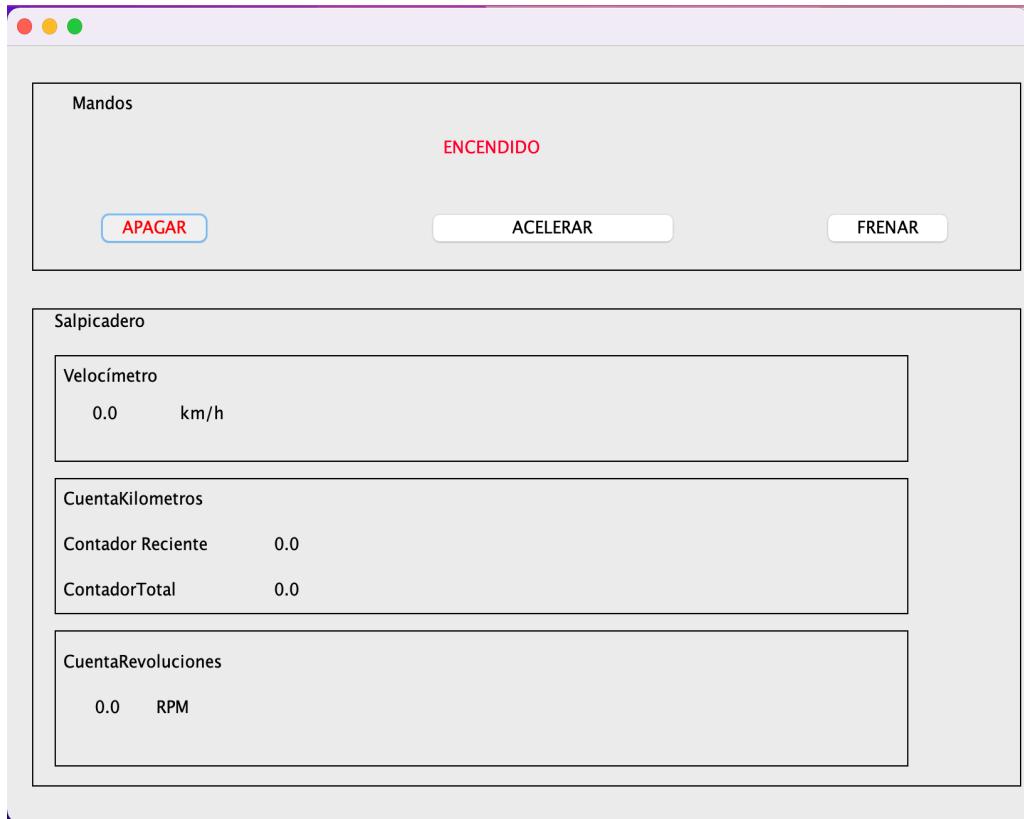


Captura de ejecución

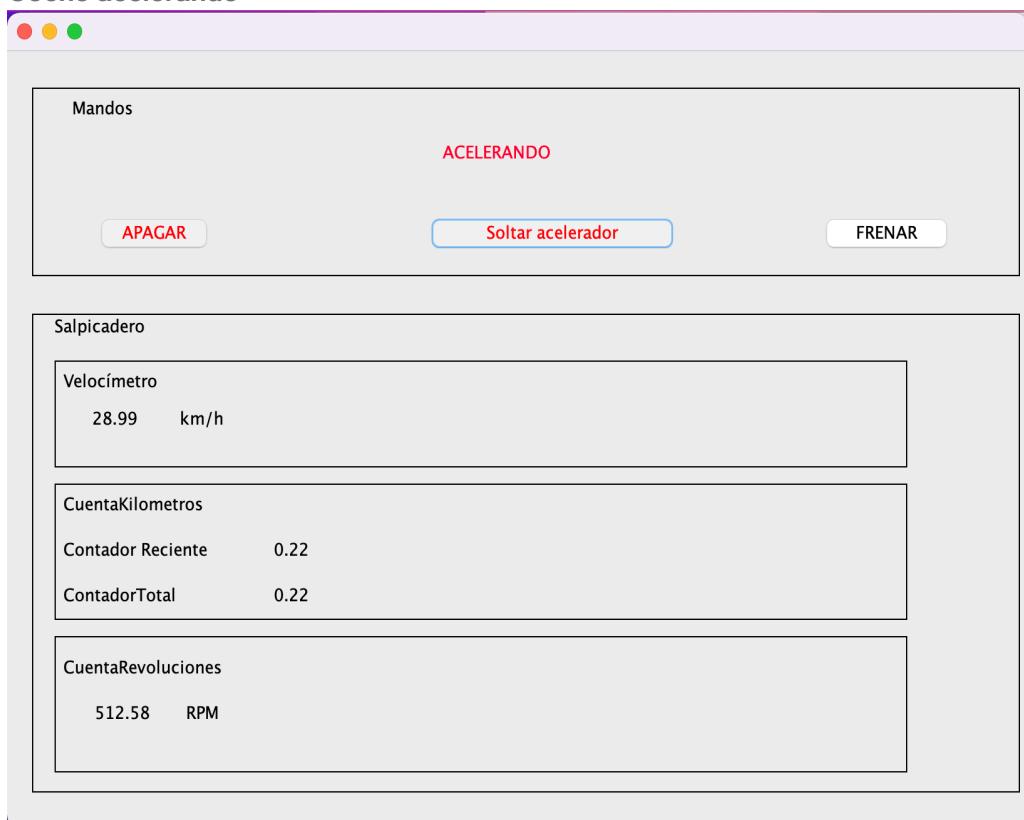
Coche apagado



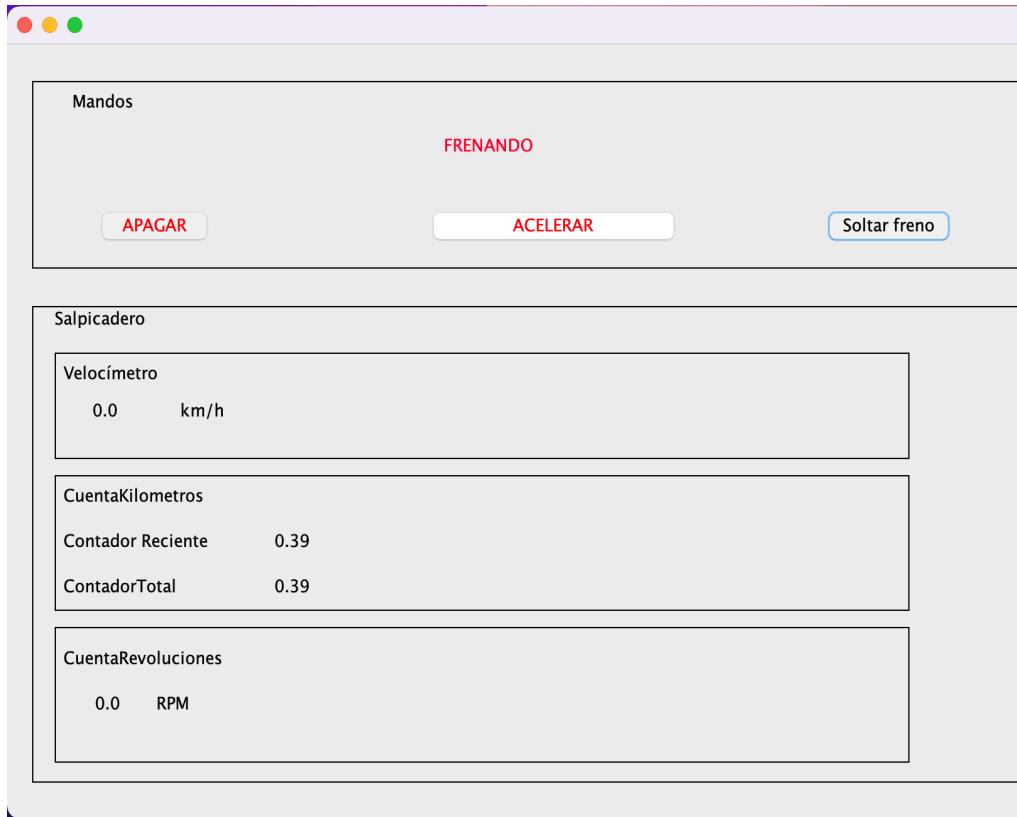
Coche encendido



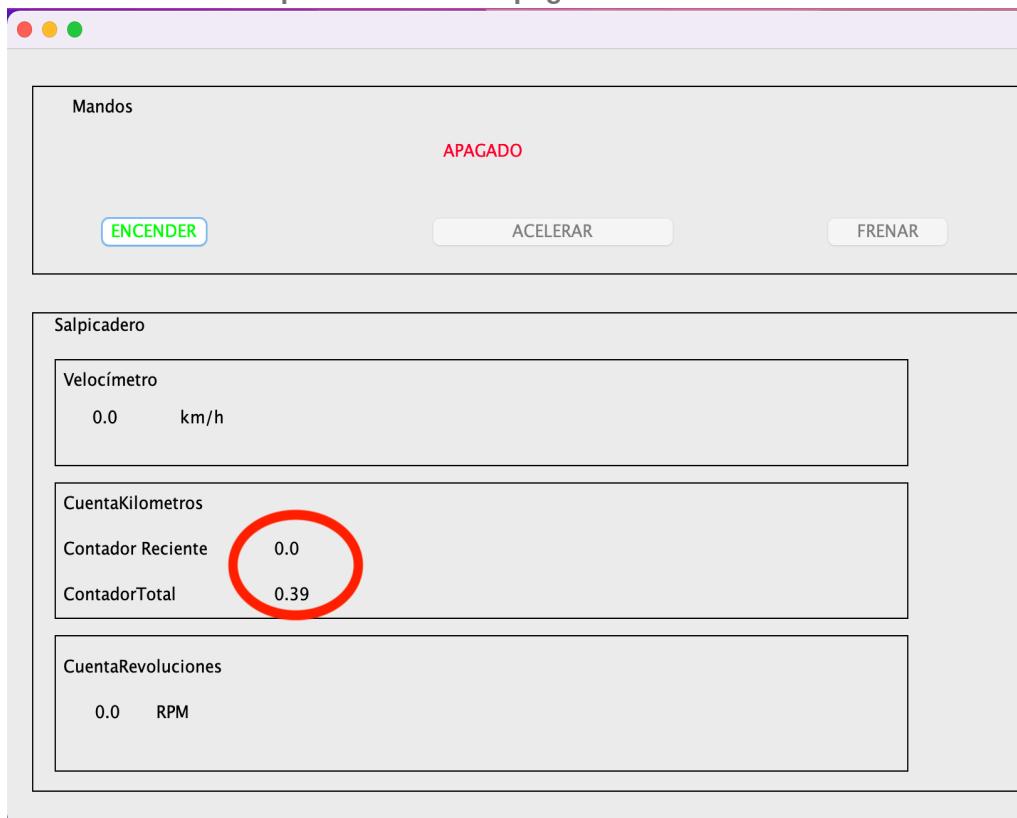
Coche acelerando



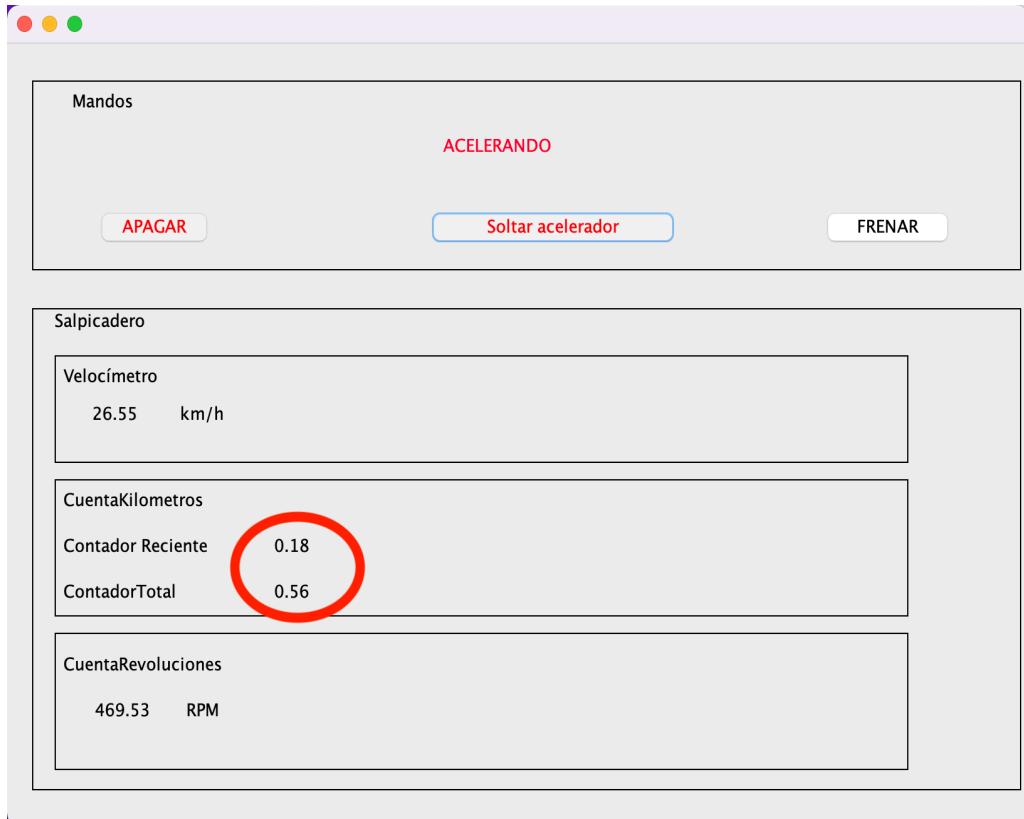
Coche frenando



Contador reciente se pone a 0 cuando apagamos el coche



Contador reciente aumenta distancia recorrida y contador total sigue aumentando



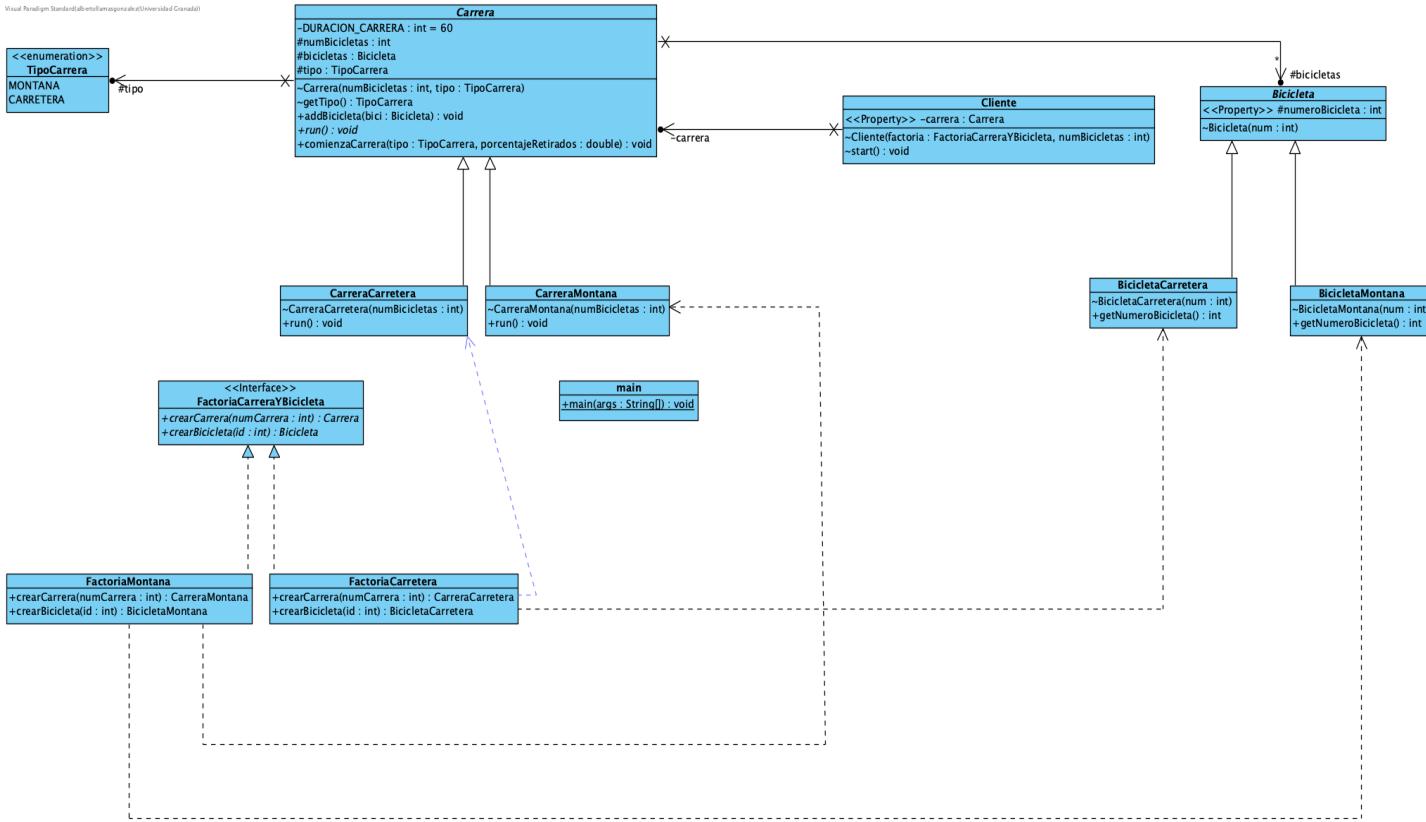
Sesión 2

Módulo 1 : Ej 1 (Oscar López y Alberto Llamas) Patrón Factoría

Abstracta y patrón Método Factoría

En este ejercicio, nos pedían utilizar el patrón Factoría Abstracta para simular dos carreras distintas de ciclistas que durasen 60s y se retiraran un % dependiendo del tipo de carrera. Para el retiro de ciclistas, hemos supuesto que se retira un ciclista aleatorio cada segundo a partir de un cuarto de la carrera por un motivo distinto. Hemos realizado un método synchronized en el que cada hebra "corre" una carrera (Carretera y Montaña) y después mediante el método join() de Thread, hemos fusionado las hebras para que acaben a la vez.

Diagrama de clases



Captura de ejecución

```

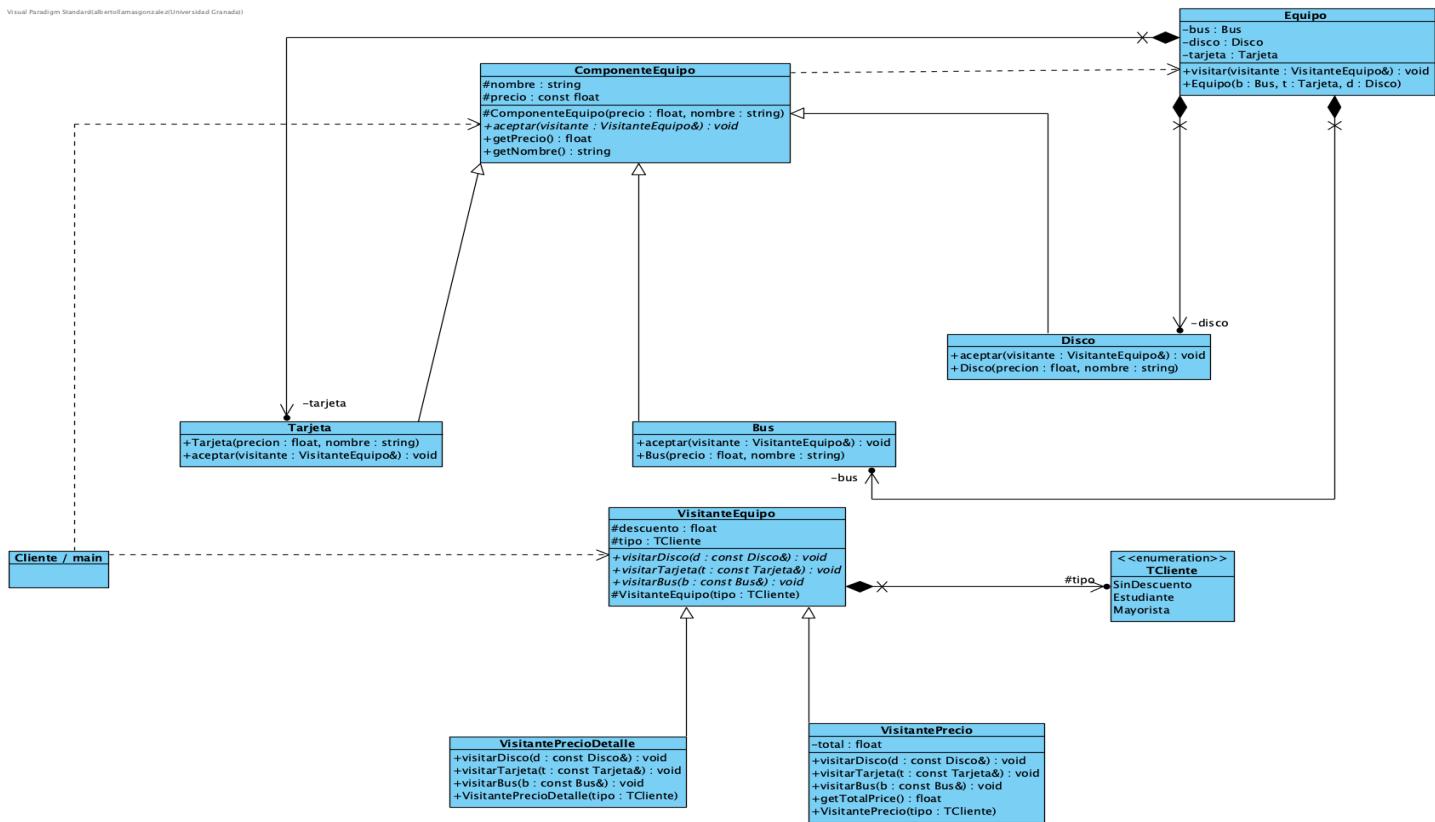
run:
Las carreras tienen un total de 17 participantes.
Comienza la carrera de MONTANA !
Comienza la carrera de CARRETERA !
Se retira el ciclista 9 porque le ha dado un tirón. Quedan 17 ciclistas.
Se retira el ciclista 2 porque le ha dado un tirón. Quedan 17 ciclistas.
Se retira el ciclista 5 porque le ha dado un tirón. Quedan 16 ciclistas.
Se retira el ciclista 7 porque le ha dado un tirón. Quedan 15 ciclistas.
El ciclista 0 ha acabado la carrera de MONTANA .
El ciclista 1 ha acabado la carrera de MONTANA .
El ciclista 2 ha acabado la carrera de MONTANA .
El ciclista 3 ha acabado la carrera de MONTANA .
El ciclista 4 ha acabado la carrera de MONTANA .
El ciclista 6 ha acabado la carrera de MONTANA .
El ciclista 7 ha acabado la carrera de MONTANA .
El ciclista 10 ha acabado la carrera de MONTANA .
El ciclista 11 ha acabado la carrera de MONTANA .
El ciclista 12 ha acabado la carrera de MONTANA .
El ciclista 13 ha acabado la carrera de MONTANA .
El ciclista 0 ha acabado la carrera de CARRETERA .
El ciclista 14 ha acabado la carrera de MONTANA .
El ciclista 1 ha acabado la carrera de CARRETERA .
El ciclista 15 ha acabado la carrera de MONTANA .
El ciclista 16 ha acabado la carrera de MONTANA .
El ciclista 3 ha acabado la carrera de CARRETERA .
Se ha acabado la carrera de MONTANA !!
El ciclista 4 ha acabado la carrera de CARRETERA .
El ciclista 5 ha acabado la carrera de CARRETERA .
El ciclista 6 ha acabado la carrera de CARRETERA .
El ciclista 7 ha acabado la carrera de CARRETERA .
El ciclista 8 ha acabado la carrera de CARRETERA .
El ciclista 9 ha acabado la carrera de CARRETERA .
El ciclista 10 ha acabado la carrera de CARRETERA .
El ciclista 11 ha acabado la carrera de CARRETERA .
El ciclista 12 ha acabado la carrera de CARRETERA .
El ciclista 13 ha acabado la carrera de CARRETERA .
El ciclista 14 ha acabado la carrera de CARRETERA .
El ciclista 15 ha acabado la carrera de CARRETERA .
El ciclista 16 ha acabado la carrera de CARRETERA .
Se ha acabado la carrera de CARRETERA !!
BUILD SUCCESSFUL (total time: 1 minute 0 seconds)
|

```

Módulo 2 : Patrón visitante básico Ejercicios 1 y 2 (Luis Guerra y Jose Jiménez)

En este ejercicio se nos pedia un modelo en el cual se pudiera visitar un equipo que poseyese tres elementos unicos (Tarjeta, Bus, Disco) de dos formas distintas: por precio y por detalle-precio. Ademas, hemos realizado la parte opcional, dando la posibilidad de que cada visitante tenga un tipo y un descuento en el precio en relacion a este. Es por ello que, siguiendo el patron visitante, hemos "encapsulado" los elementos del equipo en una clase ComponenteEquipo de la que heredan los atributos comunes como precio y nombre, junto con el metodo de aceptar() abstracto que cada uno implementaria. Mismamente hemos hecho con los visitantes en la clase VisitanteEquipo, en la que heredarian el atributo de Tipo y descuento, e implementarian los metodos visitarBus(), visitarTarjeta() y visitarDisco(), abstractos en su clase Padre("VisitanteEquipo").

Diagrama de clases



Captura de ejecución

=====

EQUIPO 1

=====

MAYORISTA : El precio total por el visitante 1 es : 25.925

SinDescuento :

El Bus "BusPro" tiene un valor de : 2

El disco "Frisbe" tiene un valor de : 5.5

La Tarjeta "Nvidia me das" tiene un valor de : 23

=====

EQUIPO 2

=====

ESTUDIANTE : El precio total por el visitante 1 es : 24.525

Mayorista :

El Bus "Buskk" tiene un valor de : 4.25

El disco "AritoPapa" tiene un valor de : 6.1625

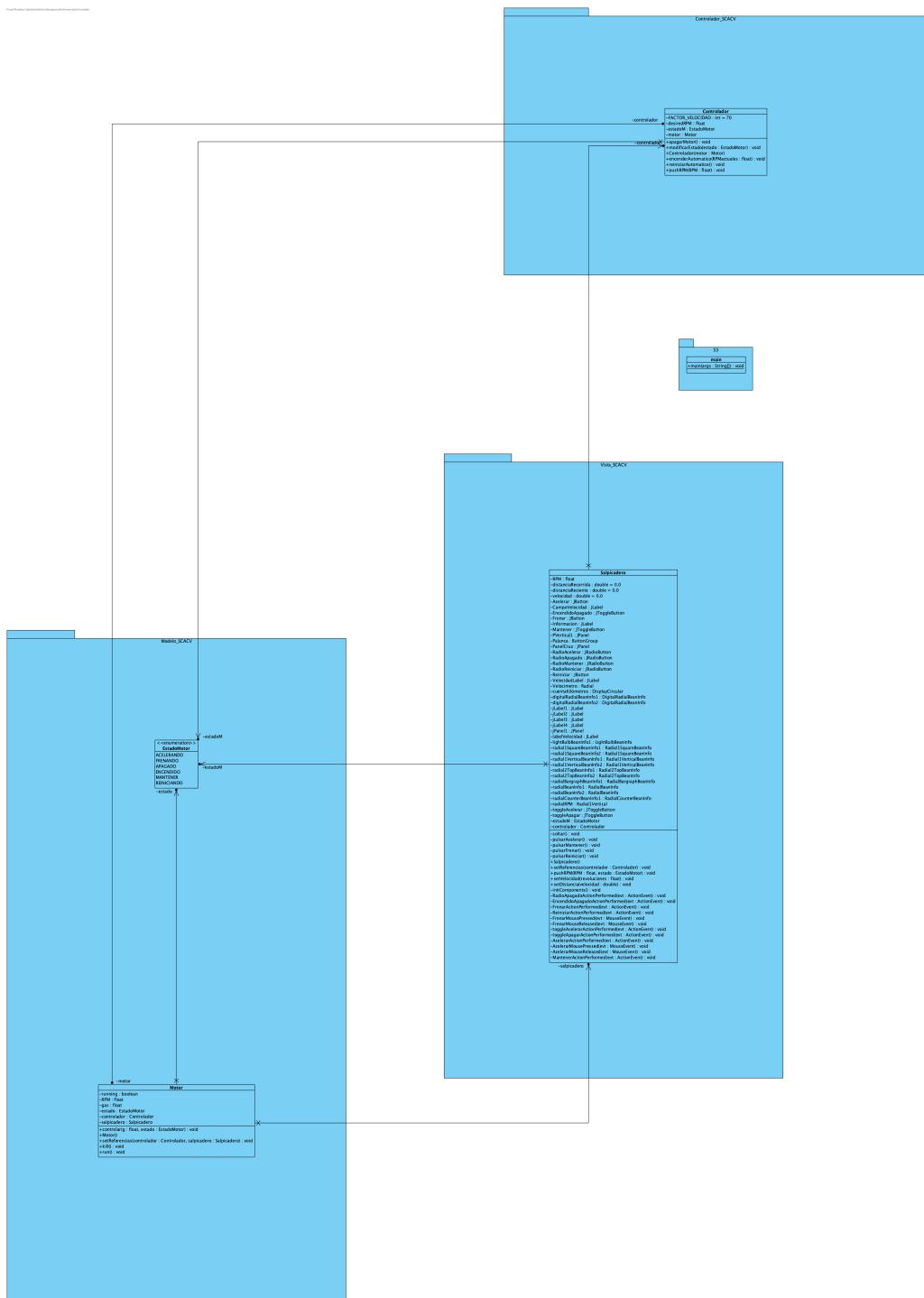
La Tarjeta "Amedias" tiene un valor de : 12.75

Sesión 3

Como patrón, hemos usado el estilo "Controlador de procesos" ya que hemos considerado que es el que mejor se adapta a nuestro caso debido a la idea principal de éste, separar la funcionalidad del sistema de la interfaz de usuario y un controlador que monitorice constantemente las acciones que debe realizar el modelo (motor). De esta forma, hemos podido repartirnos el trabajo más fácilmente:

- Motor, Controlador: Luis Guerra, Jose Jiménez, Oscar López.
- Salpicadero: Alberto Llamas.

Diagrama de clases



Captura de ejecución

