Santiago Muñoz Castro
Francisco Javier Jimenez Legaza

DS-sesion1

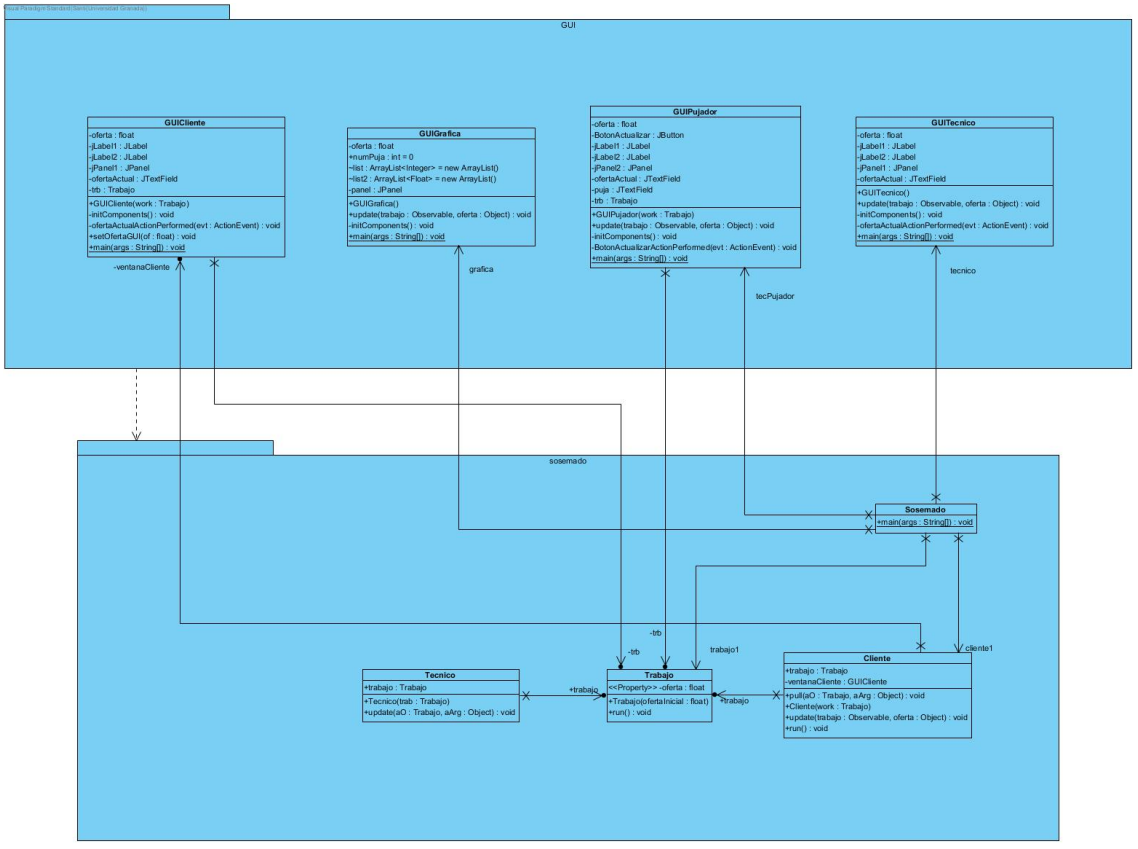
Para la realización de esta sesión inicialmente hemos realizado la parte obligatoria y de la opcional hemos hecho todas las opciones menos la del patrón compuesto(la realizaremos posteriormente).

Nuestro observable va a ser el atributo oferta de la clase Trabajo ya que nuestro software consiste en una puja para poder adquirir un mantenimiento.

Nuestra clase cliente será nuestro observador no suscrito el cual podrá ver la oferta actual de la subasta cada 12 segundos sin ninguna forma de poder consultarlo entre ese periodo.

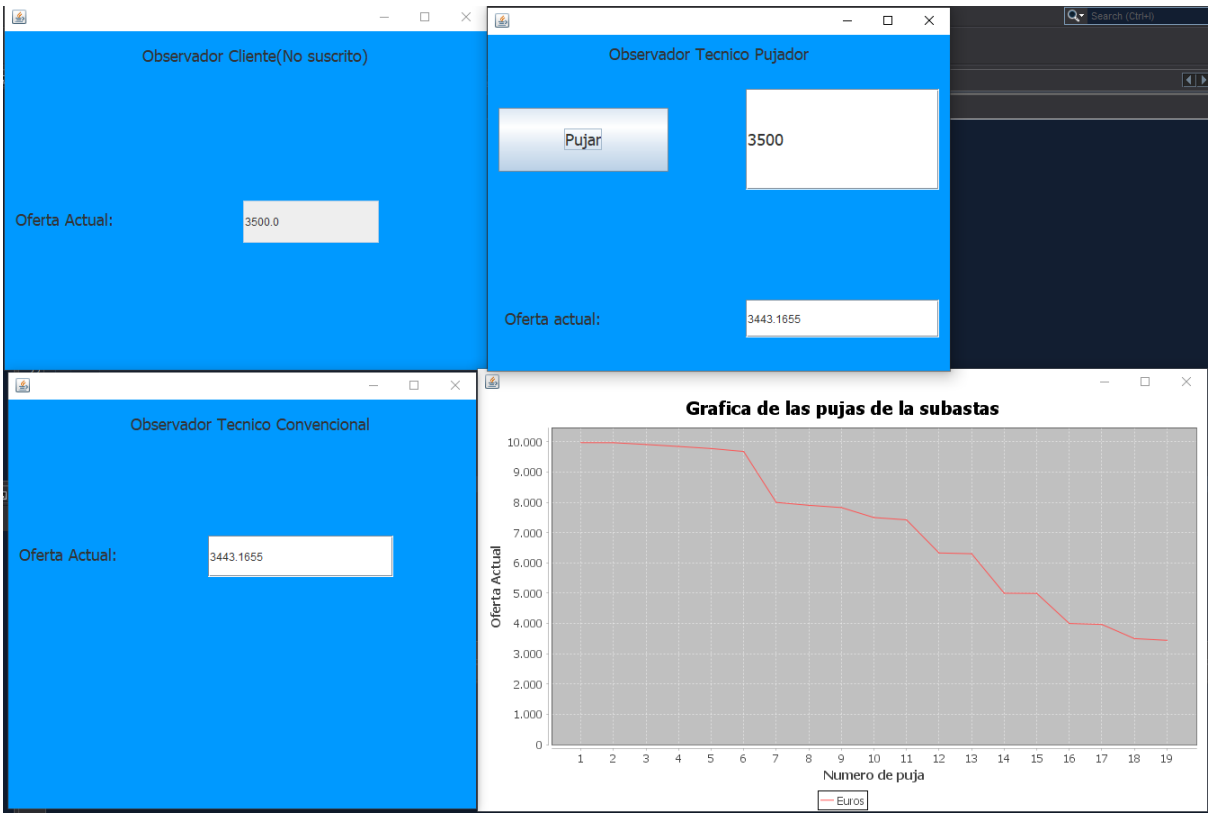
Luego tenemos a la clase Técnico que será el resto de observadores de la práctica, por un lado será nuestro suscrito convencional(sin interacción) de tal forma que siempre que se actualice la oferta actual de la puja este es notificado al instante, también es un observador suscrito que interacciona con un botón, este sirve para realizar una puja(valor menor o igual a la oferta actual) y podrá observar al mismo tiempo la oferta actual.

Por último también se utilizará igual que el suscrito convencional pero en vez de mostrarle el valor decimal de la oferta, se le representa una gráfica representando en euros las ofertas realizadas a lo largo de la puja.



Aqui una demostracion del código

Observador cliente -> observador no suscrito, se actualiza por medio de una hebra cada 12 segundos
observador Técnico convencional -> observador suscrito, se actualiza cada 5 segundos
observador Técnico pujador-> observador suscrito que puede cambiar el precio de la oferta
observador gráfica -> observador suscrito que va mostrando los cambios de oferta a lo largo de la ejecución.



DS-sesion2

Para la realización de esta práctica hemos optado por realizar tanto la parte obligatoria como la opcional.

Para la parte obligatoria:

Había que implementar el prototipo factoría abstracta junto con el patrón de método factoría.

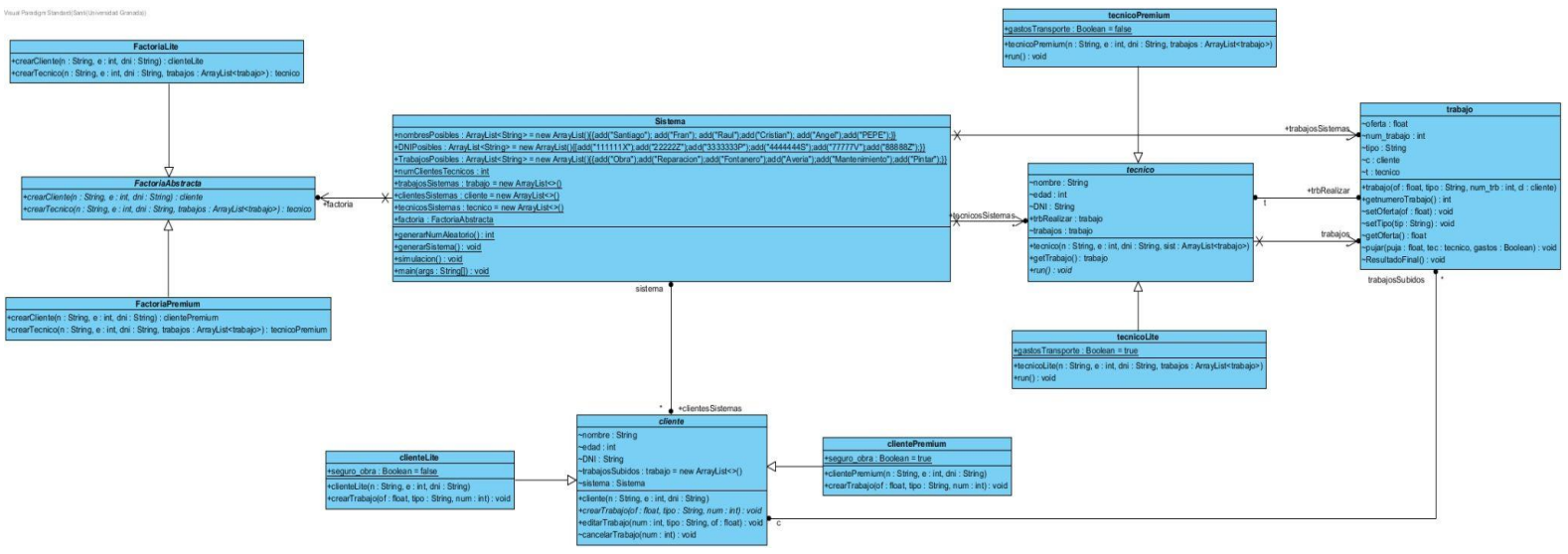
Nuestro programa sigue teniendo la misma temática que la sesión 1, tenemos clientes(del sistema), técnicos y trabajos. Los clientes pueden ser premium o lite al igual que los técnicos. Los clientes crearán un trabajo, los técnicos pujarán con una oferta y el trabajo será asignado al mejor técnico pujador.

Tenemos nuestra factoriaAbstracta, una interfaz de Java, que declara los métodos de fabricación públicos:

crearTecnico que devuelve un objeto de alguna subclase(premium o lite) de la clase abstracta técnico

crearCliente que devuelve un objeto de alguna subclase(premium o lite) de la clase abstracta cliente

Las clases factoría específicas heredan de nuestra factoriaAbstracta y cada una de ellas se especializa en un tipo de cliente y técnico: los clientes y técnicos premium y los cliente y técnicos lite. Por lo que asumimos que tenemos dos clases factoría específicas: FactoriaPremium y FactoriaLite, que implementarán cada una de ellas los métodos de fabricación crearCliente y crearTécnico



Una demostración de la ejecución

La ejecución consiste en llamar a la factoría y que esta factoría nos cree mitad de clientes lite y mitad de clientes premium, lo mismo con técnicos pero no lo mostramos ya que damos por hecho que están en el sistema.

Una vez creados los trabajos de cada cliente, ejecutamos las hebras de los técnicos para que realicen la puja por los trabajos que hay en el sistema de manera aleatoria.

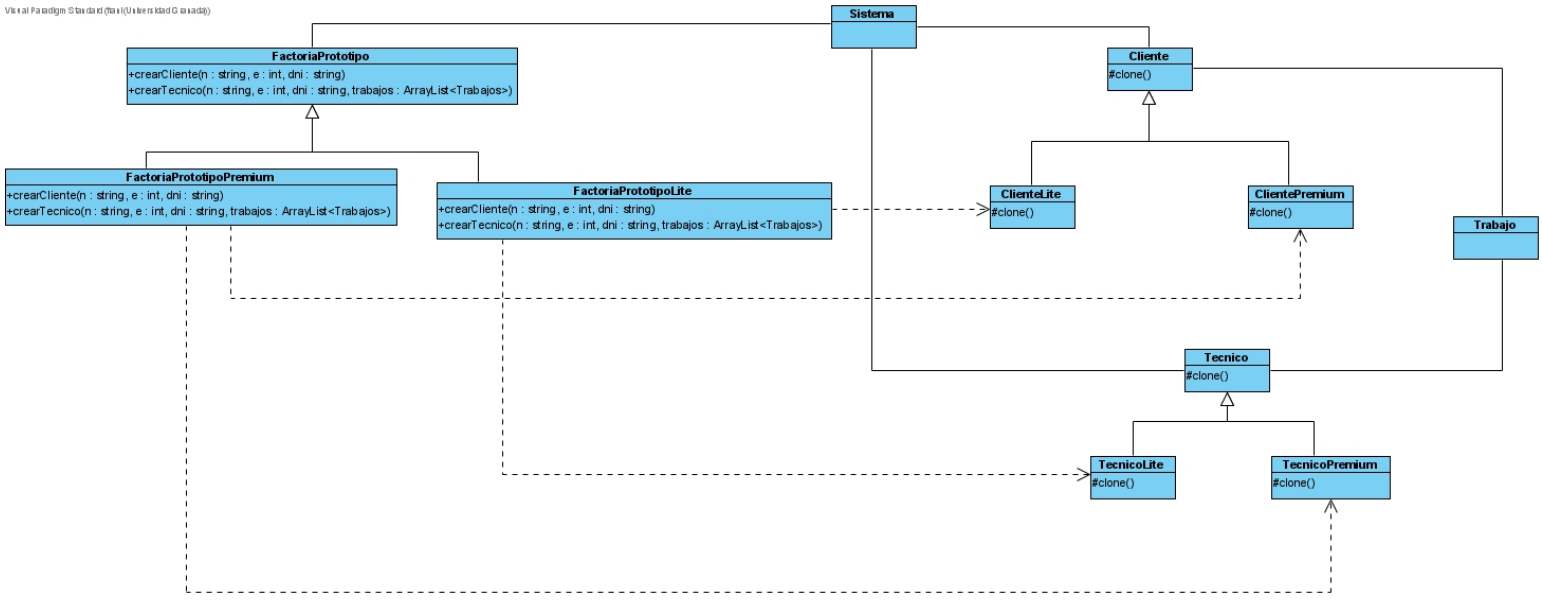
Una vez finalizada la subasta mostramos el resultado final de la subasta, los técnicos que han conseguido los trabajos.

```
run:
4
El cliente LITE Santiago ha creado el trabajo con id 0
El cliente LITE Fran ha creado el trabajo con id 1
El cliente PREMIUM Raul ha creado el trabajo con id 2
El cliente PREMIUM Cristian ha creado el trabajo con id 3



El tecnico LITE Santiago realiza una puja al trabajo con id 2 de 4764.039
El tecnico PREMIUM Cristian realiza una puja al trabajo con id 2 de 4741.2656
El tecnico LITE Fran realiza una puja al trabajo con id 2 de 4581.932
El tecnico PREMIUM Cristian se queda por ahora con la subasta con una oferta de 4741.2656 del trabajo 2
El tecnico LITE Fran se queda por ahora con la subasta con una oferta de 4581.932 del trabajo 2
El tecnico PREMIUM Raul realiza una puja al trabajo con id 1 de 4661.52
El tecnico PREMIUM Raul se queda por ahora con la subasta con una oferta de 4661.52 del trabajo 1
El tecnico LITE Fran se queda por ahora con la subasta con una oferta de 4581.932 del trabajo 2
El tecnico LITE Santiago realiza una puja al trabajo con id 3 de 4565.118
El tecnico PREMIUM Raul realiza una puja al trabajo con id 3 de 4925.942
El tecnico LITE Santiago se queda por ahora con la subasta con una oferta de 4565.118 del trabajo 3
El tecnico LITE Santiago realiza una puja al trabajo con id 1 de 4383.3164
El tecnico LITE Santiago se queda por ahora con la subasta con una oferta de 4383.3164 del trabajo 1
El trabajo con id 1 ha sido otorgado al tecnico Santiago
El trabajo con id 2 ha sido otorgado al tecnico Fran
El trabajo con id 3 ha sido otorgado al tecnico Santiago
```

Para la parte opcional:

Para la parte opcional había que implementar la factoría abstracta con el patrón prototipo en ruby. En general ha sido muy parecido a la implementación en java, seguimos teniendo nuestra clase cliente con sus dos subclases(premium ylite), nuestra clase técnico con sus dos subclases(premium y lite) y la clase trabajo, donde realizamos la misma dinámica de siempre.

Esta vez nuestras factorías van a ser las mismas pero van a implementar el prototipo, es decir, en los métodos de crearCliente y crearTécnico de cada factoría específica llamaremos al método clone declarado en cada subclase específica de cliente y técnico, el método clone devuelve un objeto de la propia subclase. Por lo dicho es muy similar a la parte obligatoria, lo único que hemos cambiado es el método de creación añadiendo el método clone, además de la implementación en ruby.



Demostración de ejecución de código

Básicamente es lo mismo que el anterior sin hebras

Creamos los clientes llamando a la factoría. Llamamos a los distintos técnicos para que realicen la puja por un trabajo aleatorio. Y se muestra el trabajo asignado al técnico ganador.

```
/usr/bin/ruby /home/santiago/Escritorio/DS/Prototipo/Sistema.rb
El cliente LITE Fran ha creado el trabajo con id 0
El cliente PREMIUM Santiago ha creado el trabajo con id 1
El cliente PREMIUM Raul ha creado el trabajo con id 2


El tecnico LITE Fran realiza una puja al trabajo con id 1 de 368
El tecnico LITE Fran se queda por ahora con la subasta con una oferta de 368 del trabajo 1
El tecnico PREMIUM Santiago realiza una puja al trabajo con id 1 de 97
El tecnico PREMIUM Santiago se queda por ahora con la subasta con una oferta de 97 del trabajo 1
El tecnico PREMIUM Raul realiza una puja al trabajo con id 2 de 4446
El tecnico PREMIUM Raul se queda por ahora con la subasta con una oferta de 4446 del trabajo 2

El trabajo con id 1 ha sido otorgado al tecnico Santiago
El trabajo con id 2 ha sido otorgado al tecnico Raul

Process finished with exit code 0
```

DS-sesion3

Para esta sesión hemos realizado tanto la parte obligatoria como la opcional, en esta última hemos elegido hacer la parte de Dart.

Para la parte obligatoria:

Para la parte obligatoria había que aplicar el estilo arquitectónico de filtros de intercepción en C++, el sistema es el de todas las sesiones, tenemos un cliente(usuario que usa el sistema) que crea un trabajo, también tenemos a los técnicos que escogen los trabajos y los realizan. Un trabajo es creado con un presupuesto inicial, después de realizarlo se rellenará dos partes, una parte de técnico que completará el cliente valorando la actitud del técnico y otra parte de trabajo que rellenará el técnico sobre gastos y dificultad del trabajo.

Nuestro presupuesto se verá modificado por estos partes que actúan como filtros. En resumen tendremos dos filtros(clases FiltroParteTécnico y FiltroParteTrabajo, que implementa la interfaz Filtro para modificar el presupuesto del trabajo en función de las clases de parteTrabajo y parteTecnico).

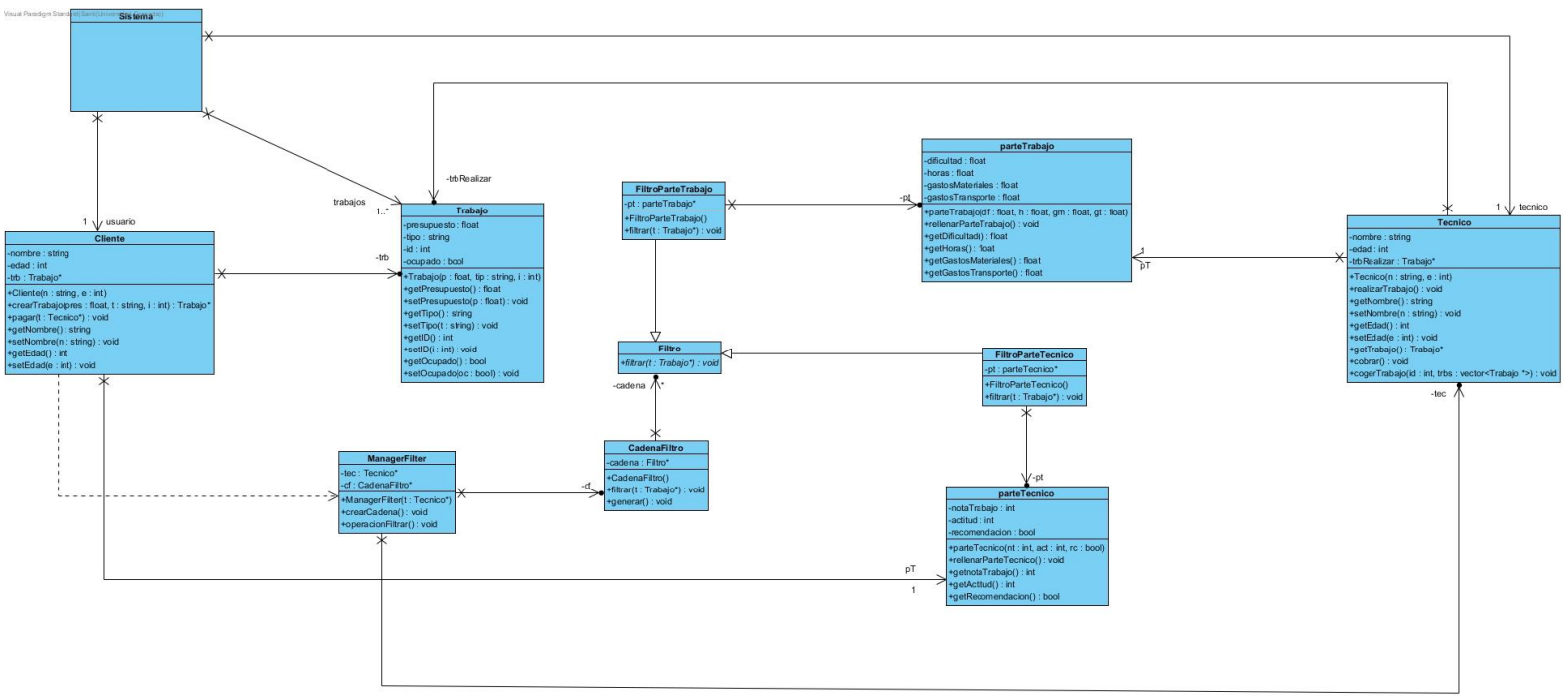
A continuación se explican las entidades de modelado necesarias para programar el estilo Filtros de intercepción para este ejercicio:

- *Objetivo(target)*: Representa el técnico que recibe el pago del trabajo realizado.

- *Filtro:* Esta interfaz es implementada por las clases parte FiltroParteTécnico y FiltroParteTrabajo arriba comentadas. Estos filtros se aplicarán antes de que el técnico (objeto de la clase *Objetivo*) ejecute sus tareas propias(método *cobrar)* para recibir el presupuesto del trabajo.

- *GestorFiltros:* Crea la cadena de filtros y se encarga de que se apliquen (método *filtrar)*

- *Sistema:* Representa las acción de cliente/usuario (pagar) que será interceptada por los filtros de la *CadenaFiltros* para añadir o quitar cantidad monetaria del trabajo.



Demostración de ejecución de código

Como en este caso no hay hebras, creamos un cliente que crea varios trabajos y los mete en el array de trabajos del sistema.

Creamos un técnico que coge un trabajo aleatorio del array de trabajos del sistema.

Una vez que el técnico realiza el trabajo, llamamos a la función pagar del cliente, que paga al técnico.

Antes de pagar al técnico se rellenan los partes, tanto del trabajo(por parte del técnico) como del técnico(por parte del cliente).

Que a partir de esos parámetros de los partes, el técnico recibe más o menos dinero por el trabajo.

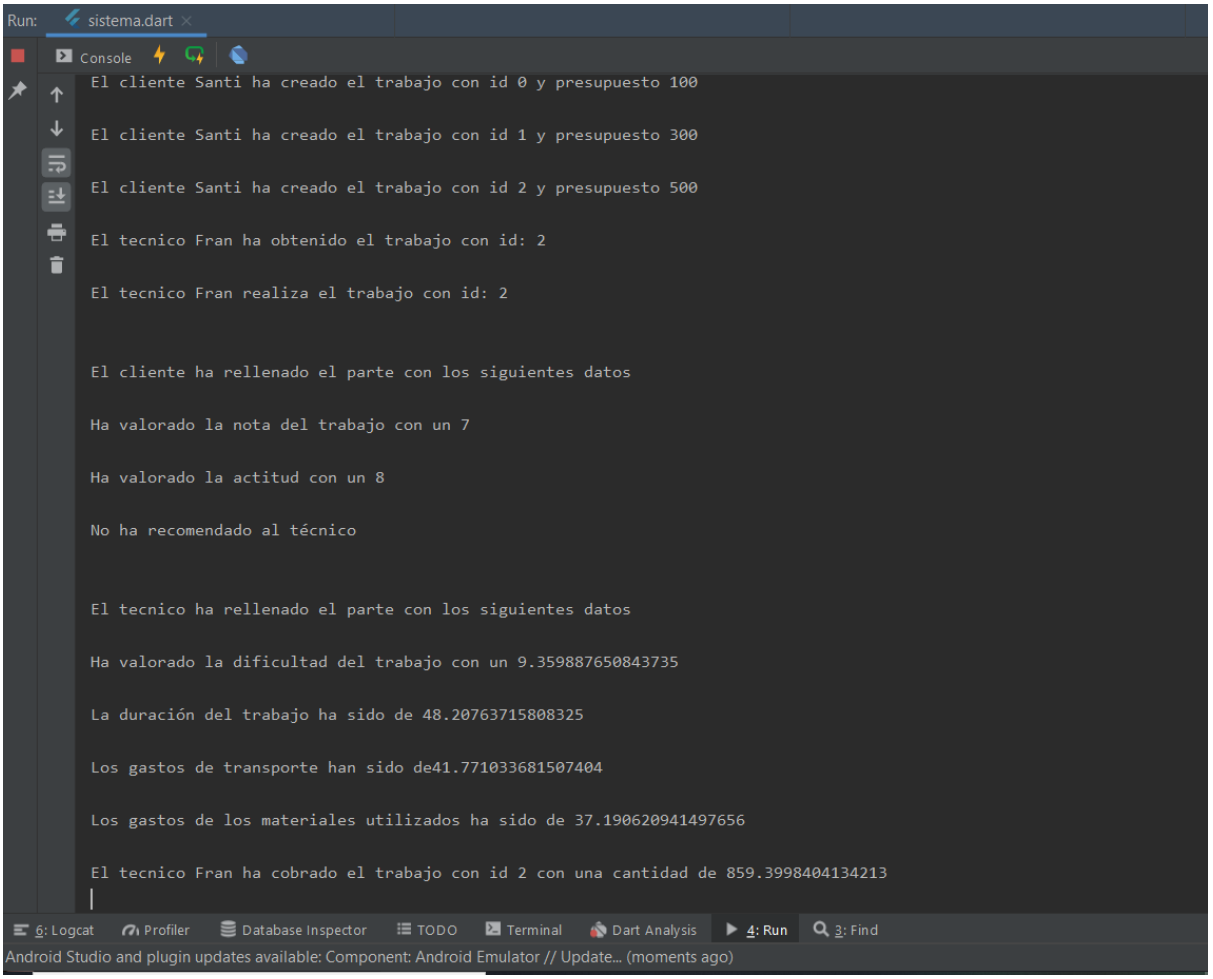
```
santiyagito22@LAPTOP-RT9H1L8V:/mnt/c/Users/yagoe/OneDrive/Escritorio/IS/DS/Sesion3/S3$ make
Enlazando para: Linux
./pracs_exe
El cliente Fran ha creado el trabajo con id 0 y presupuesto 100
El cliente Fran ha creado el trabajo con id 1 y presupuesto 300
El cliente Fran ha creado el trabajo con id 2 y presupuesto 500
El tecnico Santi ha obtenido el trabajo con id: 0
El tecnico Santi realiza el trabajo con id: 0

Vamos a rellenar el Parte del Tecnico

Introduzca una nota del trabajo efectuado por el tecnico(0-10):
8
Introduzca la actitud que ha tenido el tecnico(0-10):
7
¿Recomiendas al técnico?(Si o No):
si

Vamos a rellenar el Parte del Trabajo

Introduzca la dificultad del trabajo(0-10):
7
Introduzca cuantas horas ha durado la reparación(No acepta negativo):
6
Introduzca los gastos materiales(No acepta negativo):
100
Introduzca los gastos de transporte (No acepta negativo):
200
El tecnico Santi ha cobrado el trabajo con id 0 con una cantidad de 472.6
```

Para la parte opcional:

Para esta parte hemos escogido hacer el proyecto en dart.

Solamente ha sido ir traduciendo código de c++ a dart.

Lo único que hemos cambiado ha sido la entrada de datos por terminal que le hemos asignado directamente los parámetros de forma aleatoria de los partes, ya que por problemas técnicos no nos cogía los datos por terminal.


Run:
sistema.dart
Console
El cliente Santi ha creado el trabajo con id 0 y presupuesto 100
El cliente Santi ha creado el trabajo con id 1 y presupuesto 300
El cliente Santi ha creado el trabajo con id 2 y presupuesto 500
El tecnico Fran ha obtenido el trabajo con id: 2
El tecnico Fran realiza el trabajo con id: 2
El cliente ha rellenado el parte con los siguientes datos
Ha valorado la nota del trabajo con un 7
Ha valorado la actitud con un 8
No ha recomendado al técnico
El tecnico ha rellenado el parte con los siguientes datos
Ha valorado la dificultad del trabajo con un 9.359887650843735
La duración del trabajo ha sido de 48.20763715808325
Los gastos de transporte han sido de41.771033681507404
Los gastos de los materiales utilizados ha sido de 37.190620941497656
El tecnico Fran ha cobrado el trabajo con id 2 con una cantidad de 859.3998404134213
6: Logcat
Profiler
Database Inspector
TODO
Terminal
Dart Analysis
4: Run
3: Find
Android Studio and plugin updates available: Component: Android Emulator // Update... (moments ago)