Iván Valero Rodríguez, Ahmed El Mouktari Koubaa

# Práctica 1, Sesión 3. Filtros de intercepción.

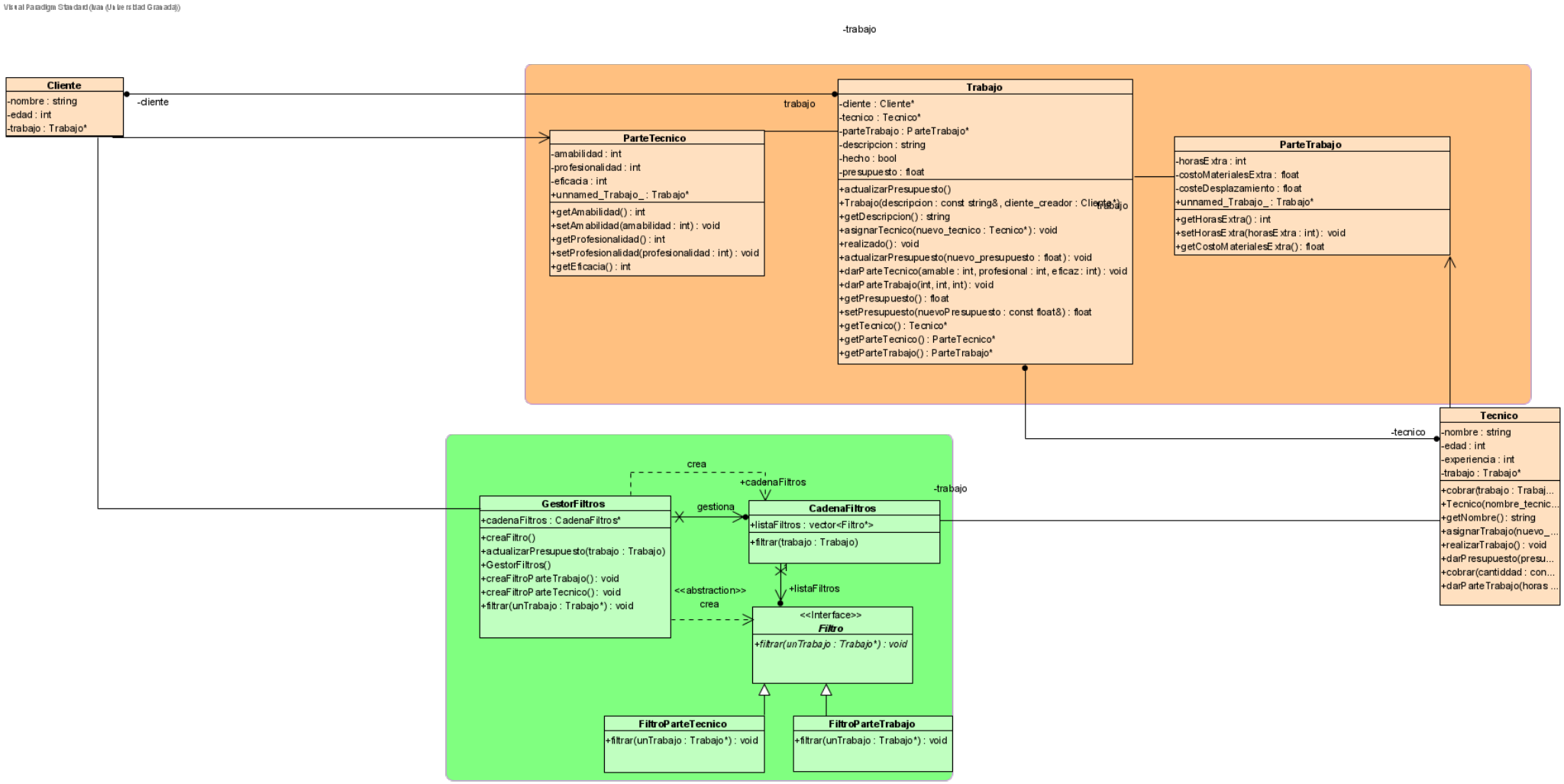
## Diseño del sistema

Para realizar los filtros de intercepción, necesitábamos:
Un *client* : Para tal fin, se ha cogido la clase Cliente, cuya comunicación con el target sea a través de la clase **`pagar()`.**
Un *target* : En este caso será el Técnico, que recibirá la comunicación del Cliente para terminar la transacción con el método **`cobrar()`**
Dos Filtros: Se han creado a tal fin las clases FiltroParteTecnico y FiltroParteTrabajo, que heredan de Filtro, una interfaz.
Una cadena de filtros : Se ha creado la clase CadenaFiltros a tal fin, que será quien contenga una lista de Filtros
Un Gestor de Filtros : Se ha creado a tal efecto la clase GestorFiltros, que crea la cadena de filtros y la gestiona.

Se adjunta el diagrama de diseño. Nótese que las clases correspondientes al modelo (adaptado) están en color naranja y que las clases correspondientes a la creación y gestión de filtros están en color verde.

Para hacer el manejo de los filtros y la interceptación de la información, se ha establecido el siguiente procedimiento:
1. Previamente se habría rellenado un Parte de Trabajo por parte del Tecnico y un Parte de Tecnico por parte del Cliente.
2. Cliente da la orden de pagar. Enseguida se envía al Gestor de Filtros.
3. El Gestor de Filtros crea la Cadena de Filtros que a su vez incluye los dos Filtros a usar, cada uno mira el Parte de cada implicado.
4. En el filtro asociado a cada Parte se puede dar un porcentaje extra que el Sistema daría al Tecnico si hace un buen trabajo y/o le ha costado más de la cuenta. Para ello, hace una media de los atributos y, con el presupuesto del Trabajo interceptado, se suma más o menos cantidad en función de la media del Parte y del porcentaje máximo de pago extra
5. El Tecnico podrá, una vez hecho todo este filtrado, cobrar por el Trabajo.

Iván Valero Rodríguez, Ahmed El Mouktari Koubaa



## Implementación:

### En C++

En C++ ha habido algunas observaciones importantes, ya que una de las características de este lenguaje es el manejo de la memoria dinámica y de los punteros.
- En los constructores se usan punteros que apuntan a los nuevos elementos.
- Se ha usado el STL **`vector<Filtro>`** para los arrays.

### En Dart

En Dart la programación se ha podido realizar sin grandes objeciones. Algunas observaciones:
- El array han pasado a ser **`List<Filtro>`**

## Reversión del código:

Tras realizar la implementación en C++ (en Dart no se puede revertir porque no es soportado por *Visual Paradigm* para hacer el *Instant Reverse*), se ha hecho una reversión del código, extrayendo como resultado de los *headers* el siguiente diagrama de clases: