TRABAJO PRÁCTICO

NOMBRE: ¿Qué Comemos? <Entrega 2>

MATERIA: Diseño De Sistemas

PROFESOR: NICOLAS PASSERINI

AYUDANTE ASIGNADO: PABLO TESONE

**GRUPO 6**

**INTEGRANTES:** RODRIGO MOCCA, LUCAS MUJICA, LUCIANO IASI, BRIAN LOBO, ALEXIS TABERNA

**CURSO:** K3004

-2015-

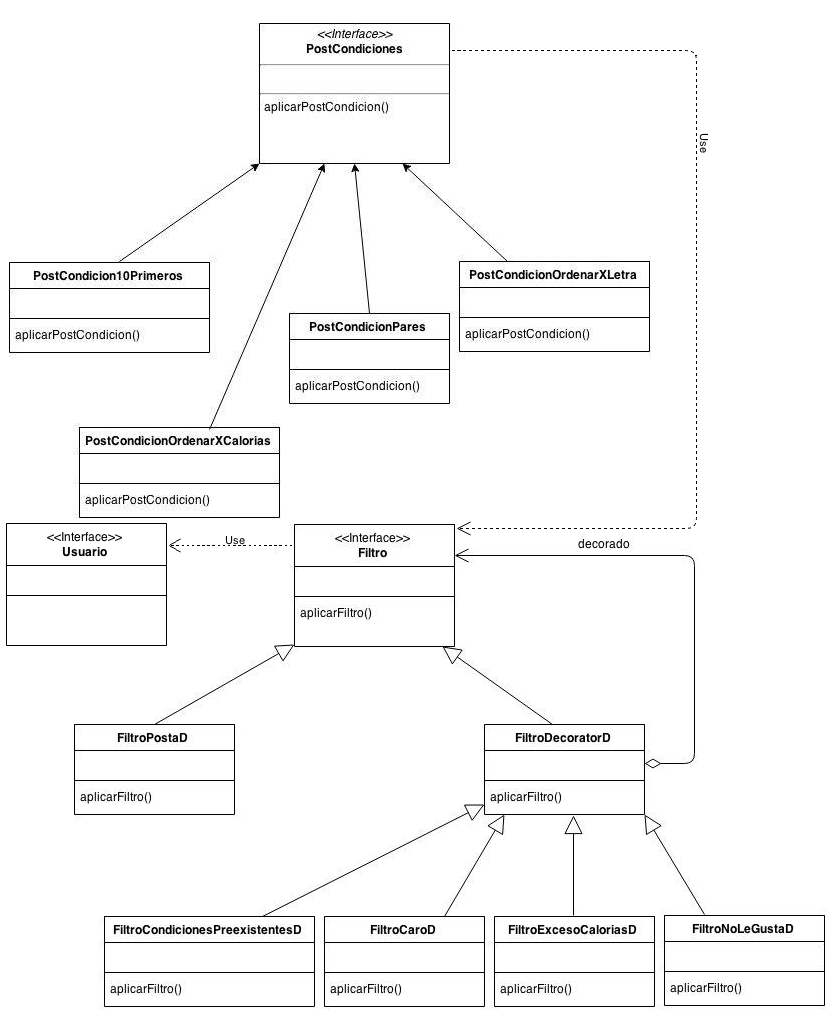
Entrega Nº 2

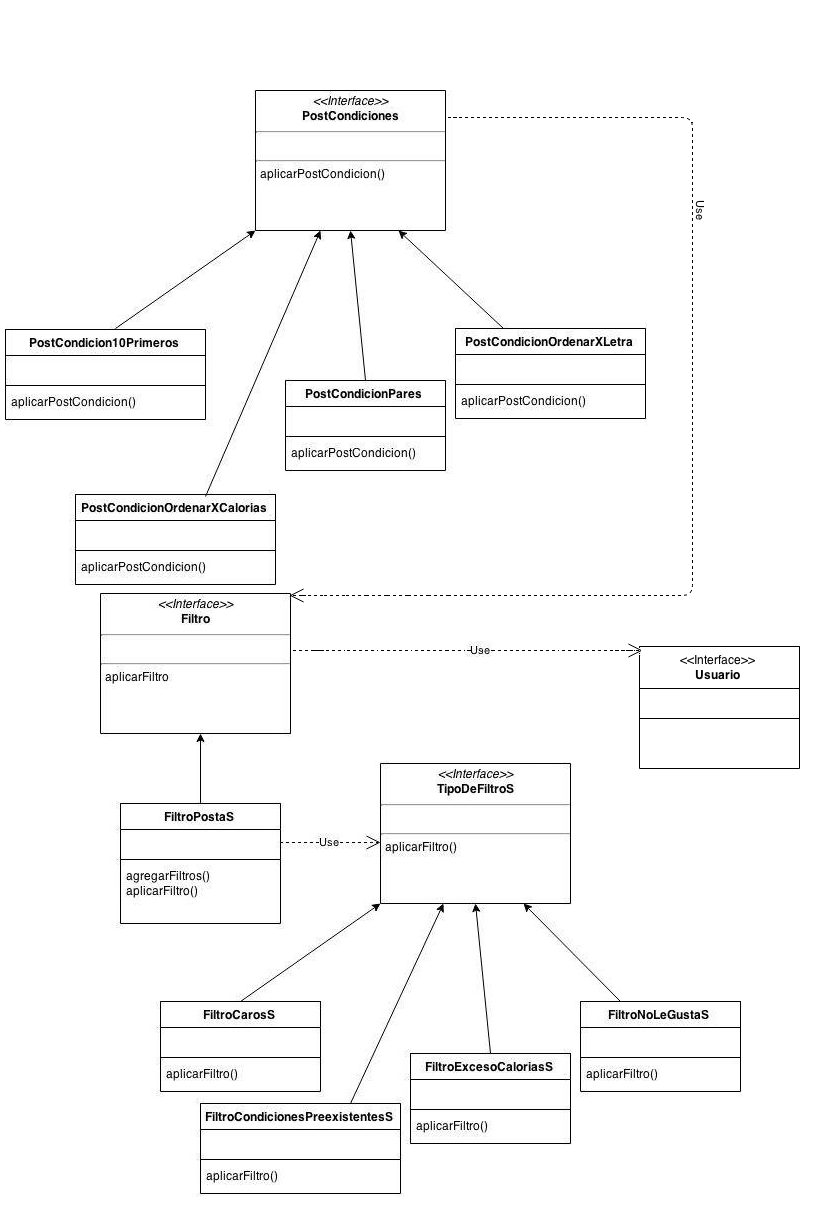


Para los nuevos requerimientos se crearon nuevas entidades. Una clase Grupo, que contiene usuarios y una interfaz Consumidor, que corresponde al comportamiento polimórfico entre un grupo y un usuario.

Se decidió que un grupo contenga usuarios y a la vez cada usuario conoce los grupos a los que pertenece, esto ayuda a resolver la pregunta si un usuario comparte un grupo con otro usuario.

Para lo corresponde a Filtros, se detalla abajo como quedaría implementando el patrón Decorator.



Se pide también desarrollar la funcionalidad con un Strategy.

Se realiza una comparación entre las dos soluciones:

|  |  |  |
| --- | --- | --- |
| **Solución** | **Filtros con Strategy** | **Filtros con Decorator** |
| **Facilidad para agregar nuevos criterios de filtro** | **Medio**, agregar nuevos filtros implica crear una clase nueva para ese filtro pero además agregar al FiltroBase una función para agregar el filtro a la lista de filtros a aplicar. | **Alta**, para agregar nuevos criterios en el peor de los casos será necesaria una nueva clase pero es posible dividir las responsabilidades de ese filtro en más de un “envoltorio” o sobre el objeto decorado |
| **Posibilidad de eliminar filtros dinámicamente** | **Alta**, los filtros se encuentran en una lista de objetos, su agregado, eliminación y movimiento es muy simple. | **Media**, debido que para “desdecorar” necesitaríamos conocer el decorador que queremos quitar, salvo que defina un orden de decoración |
| **Posibilidad de intercambiar el orden en que se aplican los filtros o el posterior manejo de resultados** | **Alta**, los filtros se encuentran en una lista, su movimiento es muy simple al igual que el manejo de los resultados que solo puede aceptar una pos condición | **Baja**, el decorado de un objeto por medio del Decorator se hace por encadenamiento de responsabilidades, esto complica la eliminación de estas sin romper dicho encadenamiento. |
| **Simplicidad** | **Media**, las estrategias generan muchos objetos que aplican diferentes estrategias o incluso una misma con diferente implementación, si bien su organización es simple su cantidad puede ser confusa. | **Baja**, Un diseño que usa el patrón Decorator suele dar como resultado sistemas formados por muchos objetos pequeños muy parecidos esto los hace difícil de entender |
| **Cohesión o cuántos objetivos resuelve el componente desarrollado** | **Media**, si bien la cohesión de las diferentes estrategias es alta la clase base asume la responsabilidad del manejo y aplicación de estas | **Alta**, al estar divididas las responsabilidades del objeto en los “envoltorios” el nivel de cohesión es alta. |
| **Mantenibilidad** | **Alta**, al estar las estrategias implementadas independientemente entre sí no son comunes los errores entre sus interacciones y por lo tanto es más fácil de mantener | **Baja**, Al estar presentes muchos objetivos pequeños muy parecidos que difieren en la forma en que están conectados su depurado puede ser complicado |