Trabajo Integrador de Programación con Objetos 2

Integrantes: **Agustín Di Santo** **agusdisanto99@gmail.com**
Santiago Abregú santiago_abregu_@hotmail.com,
Fernando Tomás Centurión ferc721@gmail.com

Profesores: Diego Cano, Diego Torres y Matias Butti.

Fecha de entrega: 6/11/2022

Dominio

En el dominio nos empieza hablar acerca de cada clase cómo se implementa y cómo se relacionan, de ahí agarramos la idea de la aplicación móvil para el usuario.
Detalles importantes hablan en el segundo párrafo donde mencionan las posibles nuevas actividades, aunque más a posterior dicen que en este trabajo hay una sola actividad que es el desafío, estaría bien implementar las nuevas actividades que mencionan en el segundo párrafo
En el texto Proyectos, nos menciona el comportamiento de cómo sería un proyecto, con descripción, nombre etc.
Algo a destacar es que con el equipo pensamos que la palabra suscribir mencionaba un observer pero después nos dimos cuenta que no haría falta en este caso usar el patrón de diseño bajo el esquema que hemos planteado
Haciendo un resumen de los más importantes cada párrafo menciona especificaciones acerca de cada clase, los comportamientos y relaciones que tienen.

Deuda técnica

- Tenemos una deuda técnica en la restricción temporal debido a que en el enunciado se podría implementar un patrón de diseño composite para poder filtrar mediante varias condiciones que tenía la restricción, una de ellas era que esté entre semana o fin de semana o inclusive mixtas tanto semana como fin de semana. Se podría mejorar agregando un patrón de diseño en este caso el composite para poder mejorar las condiciones de las restricciones de los desafíos.

- Decimos que es una posible deuda técnica debido a que no la tenemos muy clara, decimos que es una deuda técnica si sería un error o mini parche a resolver o mejorar un futuro, en este caso no sería un error pero si se tendría que mejorar en un futuro si queremos implementar más actividades, en sí en este trabajo no nos mencionan otras acciones que no sean desafíos pero en un caso futuro estaría bueno que ya quede implementado para posibles casos de nuevas actividades. Se implementó como una clase abstracta debido a que tiene el mensaje que todas entienden si no es un desafío, lo cual esta bueno para poder mejorar el polimorfismo y poder filtrar por las actividades que uno quiera

Decisiones de modelo:

En el dominio nos empieza hablar acerca de cada clase cómo se implementa y cómo se relacionan, de ahí agarramos la idea de la aplicación móvil para el usuario.
Detalles importantes se hablan en el segundo párrafo, donde se mencionan las posibles nuevas actividades, aunque más a posterior dicen que en este trabajo hay una sola actividad que es el desafío.

Decidimos implementar esta solución debido a que queríamos delegarle las responsabilidades al usuario siempre pensando es así que ¿Porqué decidimos implementar esta solución? Decidimos implementar esta solución debido a que queríamos delegarle las responsabilidades al usuario siempre pensando en los principios Solid, ¿Había otras soluciones?
Claro que había otras soluciones, una de ellas era que el usuario quede con tenga todas las implementaciones que tendría la aplicación que nosotros implementamos.
Creemos que es la mejor solución para poder cumplir siempre con los principios Solid

En el usuario decidimos que el tenga la responsabilidad de tener su propio estado, este es el progreso del usuario que se vincula al desafío, cada usuario tiene un progreso independiente debido a que cada participante, juega el desafío de manera independiente, entonces los progresos que cada uno debe tener son individuales.

Decidimos tener diferentes recomendaciones para un usuario debido a que el usuario decide el algoritmo o la recomendación que más desea y el sistema lo vincula con un conjunto de desafíos acorde a su algoritmo / preferencia que el usuario/participante decidió. Por eso implementamos el patrón de diseño Strategy debido a que necesitamos un mediador en este caso el sistema, para que pueda encontrar los mejores desafíos en base a las recomendaciones del usuario/participante

Además, los desafíos pueden ser recomendados para otros Usuarios. Hay dos tipos de recomendación:
-Recomendación por preferencia: Se seleccionan los 5 desafíos con mejor coincidencia. Nuestro equipo tomó la decisión de implementarlo de la siguiente forma: Sumando los valores absolutos de las diferencias, para cada característica, entre el valor asignado al desafío y la preferencia del usuario en esa característica. Cuanto más baja sea esa suma mayor es el nivel de coincidencia.
-Recomendación por favoritos: Se seleccionan los 20 desafíos con mejor coincidencia, y se los ordena por similitud con el desafío favorito del usuario. Una vez ordenado por similitud, se recomiendan los primeros 5. Para esta implementación de las recomendaciones usamos el patrón Strategy.

Las búsquedas de proyectos se realizan a través de varios filtros individuales:
-Fitro AND
-Filtro OR
-Filtro NOT
-Filtro que incluye una categoría
-Filtro que no incluye una categoría

Elegimos el patrón de diseño composite, debido a que teníamos que cumplir varias condiciones acerca de un proyecto, para que no se vuelva tan tedioso. Nos pareció que con el Patrón Composite se soluciona todos los conflictos posibles. El filtrado se basa con ciertas condiciones que se nombran arriba con los tipos de filtros.
Estos filtros pueden ser combinables mediante los filtros binarios And, Or, Not, los cuales se extienden de la clase abstracta Filter.

¿Por qué decidimos implementar esta solución?
En el usuario decidimos que el tenga la responsabilidad de tener su propio estado, este es el progreso del usuario que se vincula al desafío, cada usuario tiene un progreso independiente debido a que cada participante, juega el desafío de manera independiente, entonces los progresos que cada uno debe tener son individuales

Decidimos tener diferentes recomendaciones para un usuario debido a que el usuario decide el algoritmo o la recomendación que más desea y el sistema lo vincula con un conjunto de desafíos acorde a su algoritmo / preferencia que el usuario/participante decidió. Por eso implementamos el patrón de diseño Strategy debido a que necesitamos un mediador en este caso el sistema, para que pueda encontrar los mejores desafíos en base a las recomendaciones del usuario/participante

Organización:

En nuestro equipo desglosamos las tareas y utilizamos un kanban para dividirnos las tareas. Por otro lado, los tests doubles, con la herramienta mockito, nos costó implementarlo, así que antes que usar los tests double usamos los objetos de reales como primera solución. Además siempre nos juntábamos los tres mediante a meet para poder codear juntos

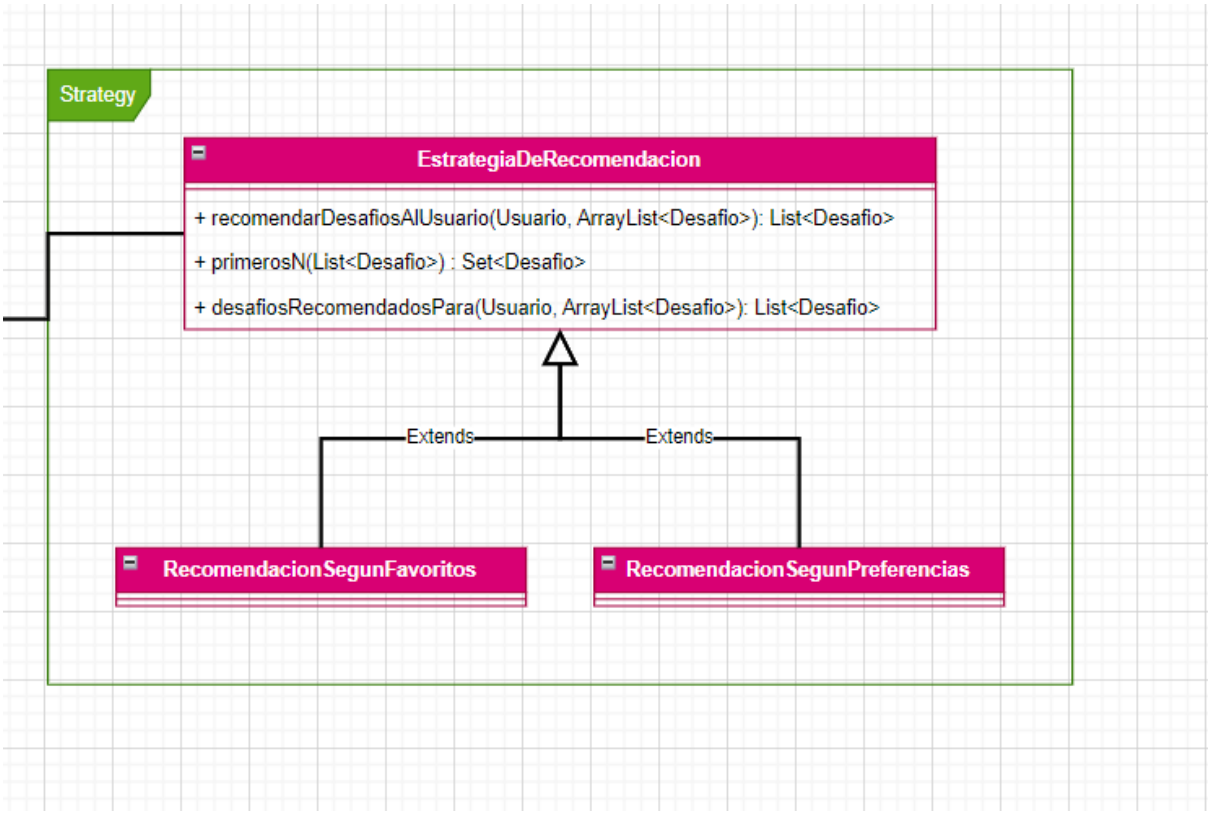
Patrones implementados

Patrón Strategy:

Roles

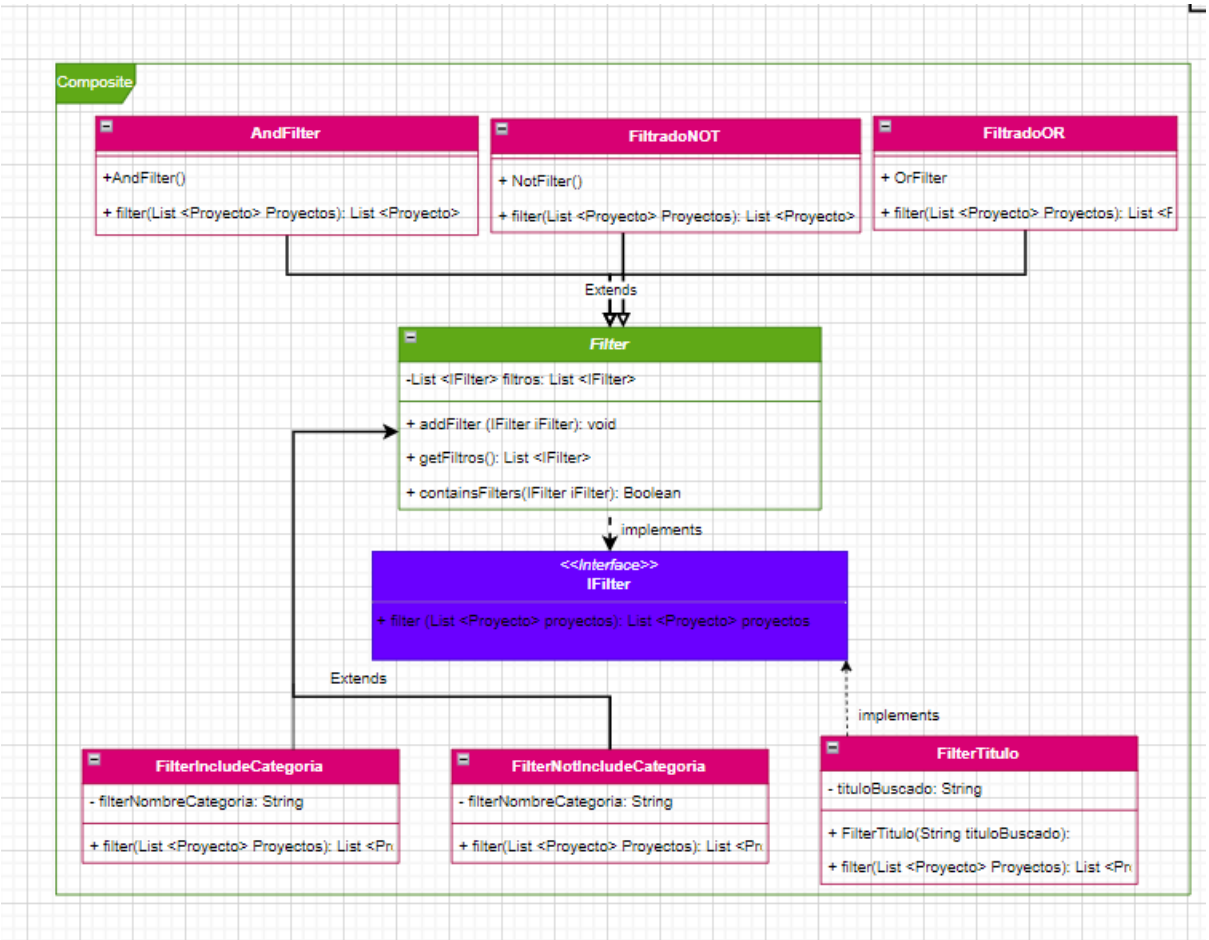
Contexto:Estrategia de recomendación

Estrategia Concreta: Recomendación Según Favoritos, Recomendación Según Preferencias



Patron Composite:

Roles: -Component -> IFilter
-Composite -> Filter
-CompositeSubClass -> AndFilter, OrFIlter, NotFilter, FilterIncludeCategoria, FilterNotIncludeCategoria
Leaf -> FilterTitulo



Patrón State:

Roles:

Context -> Usuario
State -> EstadoDelProgreso
State -> ProgresoDeDesafio
Concrete State Subclass -> ProgresoDeDesafioEnCurso, ProgresoDeDesafioTerminado,
ProgresoDeDesafioExpirado