TRABAJO PRACTICO FINAL P.O.O. 2

“CIENCIA PARTICIPATIVA Y JUEGOS”

INTEGRANTES:

GARCIA, ALAN

alanguillermogarcia2001@gmail.com

LICCIARDELLO, MARTIN

martinlicciardello7@gmail.com

NARIZZANO, IGNACIO

narizzanoignacio@hotmail.com

DESARROLLO

AppProyecto es la clase central de nuestro trabajo, ya que esta contiene todos los proyectos. La app se encarga de filtrar los proyectos según las categorías dadas por el usuario administrador.

La clase **Proyecto** es la que representa cada proyecto de ciencia participativa de la app. Un proyecto tiene su nombre, una descripción, una lista de categorías a las que pertenece, una lista de desafíos correspondientes al proyecto y una lista de muestras que van agregando los usuarios participantes.

En el caso de la clase **Muestra**, es la que representa las muestras que los usuarios recolectan y agregan a los proyectos. Esta clase tiene un usuario que es el que recolectó la muestra, la fecha y la hora en la que se realizó, y la ubicación.

La clase **Usuario** es la que representa a los usuarios, los cuales realizan la recolección de las muestras. Estos tiene un nombre, una lista de muestras las cuales van recolectando, una lista de proyectos en los que participan, una lista de desafíos de usuario, los cuales son los desafíos que están aceptados o completados por el usuario en cuestión y un estado de recomendación para buscar match entre los desafíos mediante la relación con la interface *EntretegiaRecomendacion* en la cual se usó un patrón de diseño Strategy para manejar las 2 distintas recomendaciones, las cuales son, EstrategiaFavoritos y EstrategiaMisPreferencias. Tomamos la decisión de usar el patrón Strategy para modelar las estrategias de recomendación ya que notamos que el Usuario tenía la opción de elegir entre 2 estrategias de estas estrategias, entonces creamos una familia de algoritmos para buscar match de desafíos que son intercambiables entre sí, según como lo precise el Usuario.

La clase **Desafio** describen a actividades o una serie de actividades a completarse por un usuario. De esta se conoce el área, la restricción temporal, que puede ser entre 2 fechas distintas y combinaciones de estas mismas, cantidad de muestras necesarias para completar el desafío, dificultad del desafío, el cual es un enum que va del 1 al 5, y por último tiene una recompensa.

La clase **Restriccion Temporal** es la que limita el tiempo en el que se puede completar un desafío, esta tiene una fecha de inicio, una de fin y una estrategiaSemanal, para la cual se utilizó un patrón Strategy mediante la relación con la superclase *EstrategiaSemanal* para manejar las 3 posibles estrategias de fechas, estas son: EstrategiaSemanalDia, EstrategiaSemanalFinDeSem y EstrategiaSemanalSemCompleta, estas se utilizan para realizar las combinaciones entre fechas. Decidimos usar este patrón porque como en las recomendaciones del Usuario, notamos que Restricción Temporal tenía distintos tipos de estrategia según la fecha que se elija para el desafío, volviéndolas intercambiables entre sí por el Usuario creador del Desafío.

La clase **DesafioDeUsuario** modela el desafío del usuario, este tiene el desafío en cuestión, la cantidadMuestras recolectadas por el usuario para este desafío, un voto del usuario, el cual es un enum que tiene 6 posibles valores, del

peor (v) al mejor (v5), también tiene un estadoUsuario el cual es un patrón State ya que esta clase se relaciona con la superclase *EstadoDesafioUsuario* para manejar los 3 posibles estados del usuario:

EstadoDesafioUsuarioInactivo (el Usuario no acepto el desafío)**, EstadoDesafioUsuarioActivo** (el Usuario acepto el desafío)**, EstadoDesafioUsuarioCompletado** (el Usuario completo el desafío)**.**

Al notar los 3 posibles estados que tenía un Desafío, nos dimos cuenta que se trataba de un State, ya que por cada estado que tenía el desafío, este tomaría diferentes comportamientos. De este estado no se entera el Usuario, ya que este solo puede aceptar el desafió y automáticamente pasa a Activo, sino quedaría Inactivo, y al completar el desafió, este pasaría al estado Completo

Para realizar los filtros creamos la interface *Filtro*, que gracias a esta se realizan las búsquedas por las diferentes categorías. Esta interfaz está estructurada con el patrón Composite, mediante el cual se puede anidar las diferentes categorías. En donde las proposiciones compuestas con sus subclases **not** y la clase Abstracta ***FiltroComposite*** cumplen el papel de composite y las clases de ***FiltroCategoria*** y **Titulo** son las hojas.

Creamos la clase abstracta ***FiltroComposite*** para poder aprovechar la similitud entre las clases **and** y **or**, por lo que utilizamos el patrón Template Method, para solo definir una superclase abstracta y que sus hijos hereden de ella.

Usando otro Template Method hicimos la clase abstracta ***FiltroCategoria,*** la cual tiene como subclases a **IncluirCategorias** y **ExcluirCategorias**, estas dos heredan el método y atributos de su superclase ya que son similares, simplemente que se usa un not en **ExcluirCategorias.**

PATRONES

Patrón State en DesafioUsuario

La superclase EstadoDesafioUsuario, agrega los estados:

- EstadoDesafioUsuarioInactivo
- EstadoDesafioUsuarioActivo
- EstadoDesafioUsuarioCompletado

DesafioUsuario funciona como Contexto

EstadoDesafioUsuario es State

EstadoDesafioUsuarioInactivo, EstadoDesafioUsuarioActivo y EstadoDesafioUsuarioCompletado son estados concretos.

Patrón Strategy en RestriccionTemporal

La superclase EstrategiaSemanal, agrupa las estrategias:

- EstrategiaSemanalDia
- EstrategiaSemanalFinDeSem
- EstrategiaSemanalSemCompleta

RestriccionTemporal funciona como Contexto

EstrategiaSemanal es Strategy

EstrategiaSemanalDia, EstrategiaSemanalFinDeSem y EstrategiaSemanalSemCompleta son estrategias concretas.

Patrón Strategy en Usuario

La superclase EstrategiaRecomendacion, agrupa las estrategias:

- EstrategiaFavoritos
- EstrategiaMisPreferenciasEnJuego

Usuario funciona como Contexto

EstrategiaRecomendacion es Strategy

EstrategiaFavoritos, EstrategiaMisPreferenciasEnJuego son estrategias concretas.

Patrón Composite en Filtro

Filtro es el Component

Los Composite son:

- Not
- FiltroComposite

Las Hojas son:

- FiltroCategoria
- Titulo

AppPrueba es el Cliente.

Patrón Template Method en FiltroComposite

La clase abstracta es:

- FiltroComposite

La clase concreta son:

- And
- Or

Patrón Template Method en *FiltroCategoria*

La clase abstracta es:

- FiltroCategoria

La clase concreta son:

- IncluirCategoria
- ExcluirCategoria