

Trabajo Práctico Final

Objetos II

Ciencia Participativa y Juegos

Integrantes del grupo 10:

Ignacio Pablo - ignaciopablodev@gmail.com

Fabrizio Améndola - fabrizzioamendola@gmail.com

Gonzalo Zamorano - g.zamorano256@gmail.com

PATRÓN DE DISEÑO - STATE

Se decidió seguir la estructura del patrón de diseño state para manejar el estado de la clase **DesafioDelUsuario**. Tal aplicación en el diseño cuenta con la siguiente identificación de roles según la definición de Gamma et. al.

- **Context:** quien participa con el rol de contexto sería la clase **DesafioDelUsuario**, ya que posee la instancia del estado concreto que define el estado actual y los métodos que ocasionan cambios(request) del mismo, siendo **aceptarDesafio()**, **calificarDesafio()**, **incrementarCantidadDeMuestrasRecolectadas()** y **comprobarCompleitud()**.
- **State:** quien participa con el rol de Estado sería la clase **EstadoDesafio**, es abstracta y posee todos los métodos abstractos que deberán implementar los estados concretos que extiendan de ella, siendo **aceptarDesafio(DesafioDelUsuario)**, **calificarDesafio(DesafioDelUsuario)**, **superarDesafio(DesafioDelUsuario)**, **incrementarCantidadDeMuestrasRecolectadas(DesafioDelUsuario)**, así representando los métodos Handle(). Además, para conocer la instancia del **DesafioDelUsuario** sobre la que se efectuará la manipulación de estado se coloca un parámetro de tal clase en los métodos Handle().
- **ConcreteState:** finalmente el rol de los estados concretos lo ocupan las clases que extienden de la clase abstracta **EstadoDesafio**, siendo **DesafioCompletado**, **DesafioAceptado**, **EsperandoAceptacion**. Esos son los estados que puede tomar un **DesafioDelUsuario**, y aprovechando el polimorfismo de la clase abstracta mencionada cada clase con el rol de estado concreto podrá realizar, cuando sea conveniente, la transición a otro estado para el desafío dado por parámetro en cada método.

PATRÓN DE DISEÑO - STRATEGY

Se decidió seguir la estructura del patrón de diseño strategy para encapsular los algoritmos correspondientes a la recomendación de desafíos para los participantes.

-Composition: este rol lo ocupa la clase **Usuario**, es la que posee la variable de instancia **recomendacionDesafio**, justamente de tipo **RecomendaciónDesafio**. Implementa el método **buscarMatchDesafios()**, en donde se hace uso del valor del atributo mencionado y dependiendo de tal valor se usará un algoritmo particular para realizar la búsqueda de desafíos.

-Compositor: este rol lo asume la clase **RecomendadorDesafio**, la cual es abstracta y posee un método abstracto **recomendacionDesafiosPara(Usuario)** que dado un usuario devuelve una lista de proyectos, tal método representa al método **Compose()** en la estructura del patrón.

-SimpleCompositor, TeXCompositor, ...: este rol lo cumplen las clases **RecomendadorPorPreferencias** y **RecomendadorPorFavoritos**, las cuales extienden de **RecomendadorDesafio**, teniendo que implementar el método **recomendacionDesafiosPara(Usuario)** por heredar de la mencionada. Cada clase que cumpla este rol, implementa y encapsula un algoritmo para lograr recomendar desafíos al usuario dado por parámetro, y así, dependiendo de la estrategia actual que se configure (siendo instancias concretas de las clases del rol en cuestión) actuará un algoritmo u otro.

DISEÑO DE RESTRICCIONES DE FECHAS

Para diseñar las fechas se decidió por crear a la clase abstracta `RestriccionTemporal` que cuenta con 3 subclases que heredan su metodo `cumple(fecha)` que son

-**RestriccionPorRango:** que trabaja con 2 fechas y se encarga de corroborar que una fecha en especifico pertenece a este rango.

-**RestriccionDíaDeSemana:** Que trabaja con 1 fecha en especifico y se encarga de corroborar que la fecha pertenece a un día de semana.

-**RestriccionFinDeSemana:** Que trabaja con 1 fecha en especifico y se encarga de corroborar que la fecha pertenece a un día de fin de semana.

La intención de realizarlo de esta manera es para que su extensión sea mas sencilla a la hora de querer crear nuevas restricciones.

DISEÑO DE CIRCULO

Para poder calcular el área se opto por crear la clase **Circulo**. La intención de esta clase es que quedase lo mas generica posible para así luego poder ser utilizada en distintos contextos, la clase cuenta con los atributos centro y radio para calcular la circunferencia.

Esta clase cuenta con el método `estaEnLaCircunferencia(coordenada)` que dada una coordenada me permite saber si esta pertenece a la circunferencia.