APAW. ECP1. Patrones de Diseño

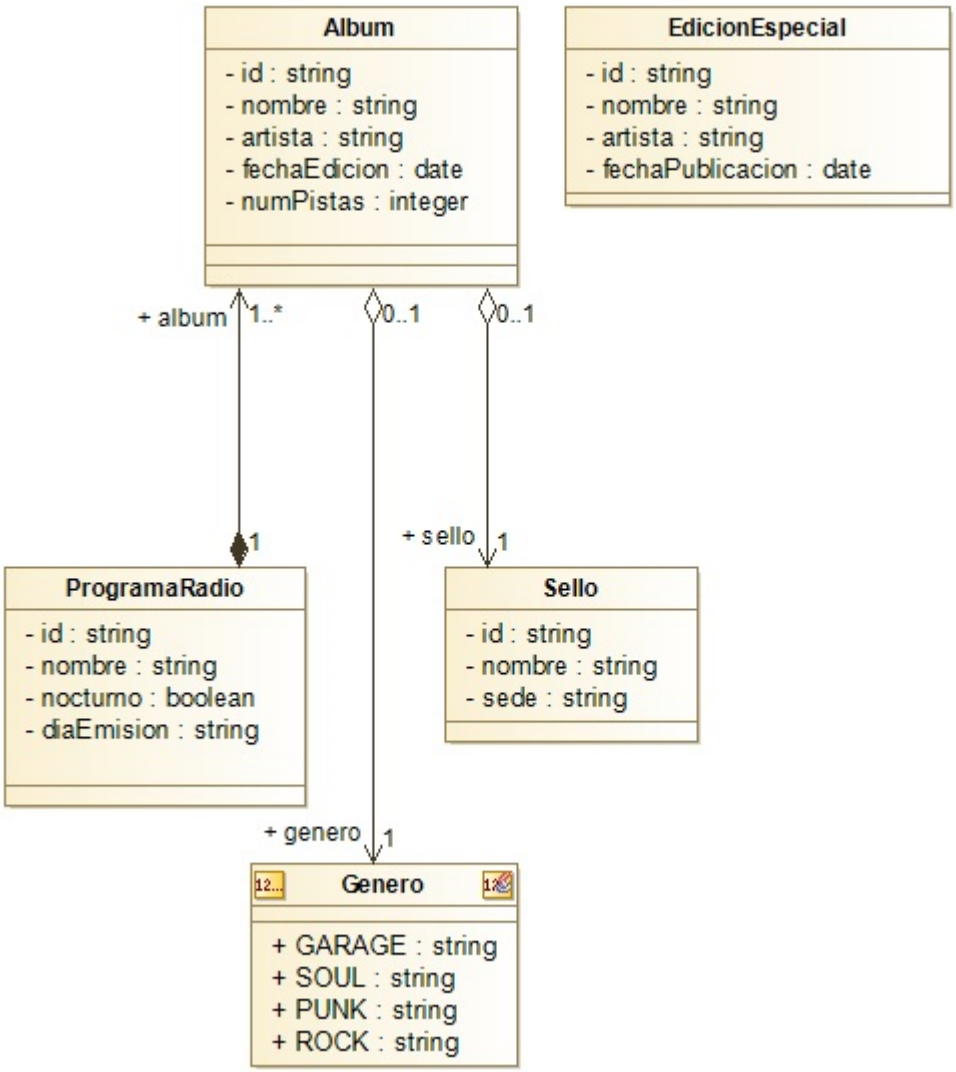
Diseño UML Entidades Album. Patrón Estrategia

Vanesa Paniego Seco

Contenido

Diagrama de clases entidades Álbum y relacionadas.

Gráfico de las entidades seleccionadas para la práctica de patrones de diseño y sus relaciones en UML:



Strategy Pattern (Patrón Estrategia)

Definición formal

El patrón Estrategia (*Strategy*) es un patrón de diseño para el desarrollo de software. Se clasifica como patrón de comportamiento porque determina cómo se debe realizar el intercambio de mensajes entre diferentes objetos para resolver una tarea. El patrón estrategia permite mantener un conjunto de algoritmos de entre los cuales el objeto cliente puede elegir aquel que le conviene e intercambiarlo dinámicamente según sus necesidades.

Objetivo

Definir una familia de algoritmos, encapsular cada uno de ellos y hacerlos intercambiables. El patrón estrategia permite cambiar el algoritmo independientemente de los clientes que lo utilicen.

Aplicación

Cualquier programa que ofrezca un servicio o función determinada, que pueda ser realizada de varias maneras, es candidato a utilizar el patrón estrategia. Puede haber cualquier número de estrategias y cualquiera de ellas podrá ser intercambiada por otra en cualquier momento, incluso en tiempo de ejecución. Si muchas clases relacionadas se diferencian únicamente por su comportamiento, se crea una superclase que almacene el comportamiento común y que hará de interfaz hacia las clases concretas.

Si un algoritmo utiliza información que no deberían conocer los clientes, la utilización del patrón estrategia evita la exposición de dichas estructuras. Aplicando el patrón a una clase que defina múltiples comportamientos mediante instrucciones condicionales, se evita emplear estas instrucciones, moviendo el código a clases independientes donde se almacenará cada estrategia.

Por lo tanto, el patrón *Strategy* es uno de los patrones más utilizados a la hora de diseñar software. Siempre que exista una posibilidad de realizar una tarea de distintas formas posibles, el patrón *Strategy* tendrá algo que decir al respecto.

La filosofía del patrón, por lo tanto, radica en el enlace entre la llamada clase de contexto y la propia interfaz. Esta clase de contexto será el intermediario entre el cliente y las clases que implementan la estrategia, y por lo tanto, sus funciones serán simples: cambiar la estrategia actual y ejecutarla.

Participantes

Contexto (*Context*). Es el elemento que usa los algoritmos, por tanto, delega en la jerarquía de estrategias. Configura una estrategia concreta mediante una referencia a la estrategia necesaria. Puede definir una interfaz que permita a la estrategia el acceso a sus datos en caso de que fuese necesario el intercambio de información entre el contexto y la estrategia. En caso de no definir dicha interfaz, el contexto podría pasarse a sí mismo a la estrategia como parámetro.

Estrategia (*Strategy*). Declara una interfaz común para todos los algoritmos soportados. Esta interfaz será usada por el contexto para invocar a la estrategia concreta.

EstrategiaConcreta (*ConcreteStrategy*). Implementa el algoritmo utilizando la interfaz definida por la estrategia.

Ejemplo práctico

Para realizar la implementación del patrón he seleccionado un ejemplo visto en la url http://codigolinea.com/2015/03/19/strategy-pattern-patron-estrategia/

Lo he seleccionado por parecerme un ejemplo interesante de cómo se puede aplicar el patrón en cualquier diseño.

Uso del patrón Strategy en un diseño

Eres el director técnico de un equipo de fútbol y tienes un partido muy importante el próximo domingo ¿Cómo plantearías el juego?

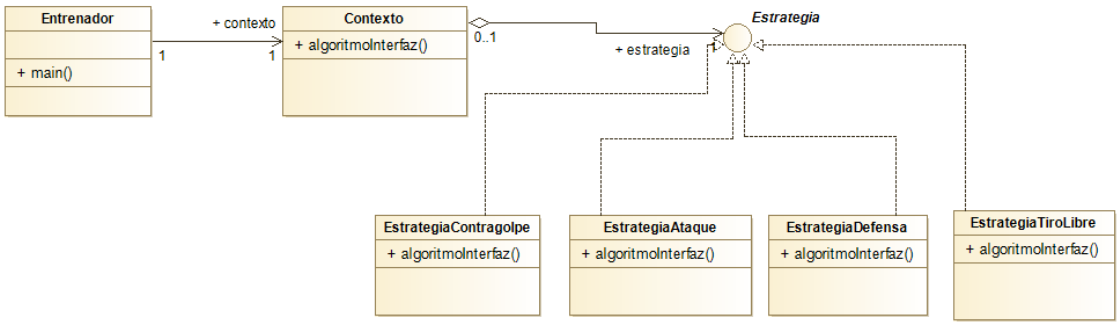
Eres un buen director técnico y se espera que preveas diferentes escenarios, por tanto debes tener preparada una estrategia adecuada para cada momento.

De esa manera la formación y el comportamiento de tus jugadores deberían cambiar dependiendo de los diferentes factores que se presenten en el desarrollo del partido:

- Cuándo atacar
- Cuándo defender
- Expulsión de uno de tus jugadores
- Ganas
- Pierdes
- Eres visitante
- Eres local, etc.

Nuestros objetos deben estar preparados para afrontar diversos contextos sin cambiar las interacciones de con el cliente.

Ante esto, el diseño del escenario y sus participantes, debería ser como esto:



Utilizado este diseño UML y el escenario planteado se ha realizado la implementación del patrón estrategia.

Fuentes disponibles en:

https://github.com/vpaniego/APAW.ECP1.PD.Vanesa.Paniego/tree/develop/src/main/java/es/upm/miw/strategy