Objetivos

1. Aprender a aplicar un patrón estructural a un contexto de diseño específico.
2. Comprender los detalles del patrón "Puente".

Contexto del problema

Para el diseño del framework EVAH se debe considerar que habrán aplicaciones en las que obtener la información de una adyacencia se repetirá miles de veces y podría impactar el desempeño, mientras en otras esa operación no será tan relevante. En el primer caso, una simulación de hormigas que recorre un laberinto para encontrar un circuito completo mínimo (cuya suma de pesos de arista es mínimo) requiere accesos continuos a la información en cada arista. En el segundo caso, una simulación de dispersión de virus de computadora no requiere del todo ninguna información en las aristas. Por tanto, es conveniente incluir al menos dos implementaciones diferentes para la clase Grafo de tal manera que el desarrollador de una aplicación pueda escoger la más adecuada: Puente al rescate!

Planteamiento del problema

Con base en las clases "Grafo", "Nodo" y "*Arista*" enriquecidas con algunos atributos y métodos en el laboratorio sobre Decorador, rediseñe la clase Grafo tomando en cuenta que puede implementarse utilizando "listas de adyacencia" o por medio de una "matriz de adyacencia". En el primer caso es necesario un vector de nodos (vector< Nodo >) y cada nodo requiere su propia lista de adyacencia con índices a los nodos adyacentes (list< int >). En el segundo caso igualmente se requiere un vector de nodos, pero para guardar las adyacencias se puede usar una matriz o "tabla de dispersión" (lo que permite ahorrar espacio sin sacrificar mucho el desempeño). Es claro que para cada implementación los métodos del Grafo van a cambiar. Siempre es posible que en el futuro se diseñe otra implementación porque las variantes de las dos comentadas son muchas. Por ejemplo, para las listas de adyacencia se pueden usar diferentes tipos de lista (simple o doble) o un vector. Para la matriz de adyacencia se pueden usar diferentes tipos de mapeo aparte de las tablas de dispersión: árboles balanceados.

Descripción de la solución

Qué clases hace falta agregar al modelo ya enriquecido para aplicar Puente en este contexto? Tome en cuenta que las aristas podrían o no tener información asociada dependiendo de la aplicación, además de que no es posible conocer, como diseñadores del framework, cuál va a ser la información requerida por cada aplicación futura del mismo.

Parte #1: agregue las nuevas clases requeridas, sus atributos y métodos. Cómo indicará el usuario desarrollador cuál implementación va a usar? Con base en el nuevo código, genere un modelo de clases usando Umbrello.

Parte #2: transforme todas las clases en plantillas y haga los cambios correspondientes aprovechando parámetros de tipo. Cómo indicará el usuario desarrollador cuál implementación va a usar? Con base en el nuevo código, genere un modelo de clases usando Umbrello.

Evaluación:

1. Código enriquecido para las clases "Grafo", "Nodo" y "*Arista*" y modelo generado (20%).
2. Código resultante de la parte #1 y modelo generado (40%).
3. Código resultante de la parte #2 y modelo generado (40%).

Fecha de entrega: domingo 14 de abril a las 23:59 vía mediacionvirtual.