Universidad de Málaga

Ingeniería de la Salud

Práctica 1

Patrones de diseño

Repositorio

https://github.com/GitHubAlejandroDR/P1_PdD_ADominguez

Autor

Alejandro Domínguez Recio

Curso

Ingeniería del Software Avanzada

Profesor

Antonio Jesus Vallecillo Moreno

Questiones

Q1. Consideremos los siguientes patrones de diseño: Adaptador, Decorador y Representante. Identifique las principales semejanzas y diferencias entre cada dos ellos.

Semejanzas:

- Pertenecen a la familia de los patrones de diseño estructurales. Estos modelan la relación entre clases y/u objetos.
- Modifican, mejoran o añaden responsabilidades a un objeto a través del encapsulamiento.
- Decorador y adaptador comparten el principio de responsabilidad única.
- Utilizan los principios de composición y delegación. Adaptador utiliza composición para enlazar las interfaces originales y adaptadas.

Diferencias:

- El uso de las interfaces. Adaptador utiliza distintas interfaces de objectos existentes encapsulandolas en uno nuevo, decorator les mejora las existentes y representante utiliza la misma.
- La suma o modificación de responsabilidades. Adaptador y decorador altera o añade comportamientos de las instancias existentes por el contrario que representante.
- Encapsulamiento de objetos. Adaptador y decorador encapsulan un objeto existente mientras que Representante no.
- Funcionalidad. Adaptador tiene el fin de *compatibilizar* dos interfaces, decorador de *añadir* nuevas funcionalidades a las existentes y representante de *delegar* su uso.

Q2. Consideremos los patrones de diseño de comportamiento Estrategia y Estado. Identifique las principales semejanzas y diferencias entre ellos.

Semejanzas:

- Principio Open Close de SOLID. Se permite añadir nuevos estados o estrategias sin modificar contexto.
- Basados en la composición. Delegan el comportamiento en otros objetos.
- Interfaz única. Tanto estado como estrategia poseen una única interfaz, la cual implementan los diferentes estados o estrategias.
- Objeto contexto. Tanto estado como estrategia poseen objetos contexto a partir de los cuales el cliente interactua con los diferentes estado o estrategias.
- Referencias estado y estrategia. Tanto estado como estrategia poseen una referencia en contexto sobre el estado o estrategia utilizada.

Diferencias:

- Referencias hacia el contexto. Las distintas estrategias no contienen ninguna referencia sobre el contexto por el contrario de los estados.
- Funcionalidad. Estado encapsula los diferentes estados de un objeto. Estrategia encapsula algoritmos o estrategias.
- Creación de los objetos. El objeto estado se crea junto con el contexto. En estrategia el objeto contexto añade la estrategia como parámetro posteriormente a su creación.
- Comportamiento. Le secuencias de estados tiene un orden fijo. El orden de ejecución de las estrategias lo tiene el cliente.
- Responsabilidad del comportamiento. Los cambios de estado pueden ser realizados a través del contexto o por los objetos estado. Los cambios de estrategia son realizados unicamente a través de contexto.
- Principio de responsabilidad única. Las distintas estrategias son independientes unas de otras, contrastando esto con la fuerte dependencia entre estados.

Q3. Consideremos los patrones de diseño de comportamiento Mediador y Observador. Identifique las principales semejanzas y diferencias entre ellos.

Semejanzas:

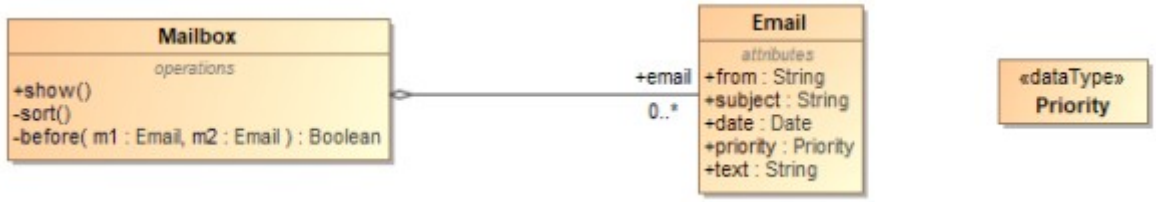
- Bajo acoplamiento. Tanto observador como mediador permite a sus objetos registrados estar desacoplados.
- Encapsulamiento. Ambos patrones hace uso del encapsulamiento de los distintos comportamientos que representan.

Diferencias:

- Forma de comunicación. Observador tiene comunicación unidireccional con sus observadores. En el caso de mediador la comunicación es bidireccional entre sus componentes.
- Referencias. En el caso de mediador cada objeto mediado mantiene una referencia sobre su mediador. Por el contrario, en observador, los observadores no mantienen referencia sobre su observable mientras que estos si las mantienen de cada uno de los observadores.
- Transmisión de la información. En observador el objeto observable es el encargado de transmitir la acción a los observadores. En el caso de mediador la acción puede ocurrir en cualquier objeto registrado y el encargado de la transmisión es el mediador. A su vez este último no tiene porqué enviar información sobre la acción a todos los objetos registrados.

Práctica 1. Cliente de correo e-look

e-look es un programa cliente de correo electrónico con el que el usuario puede gestionar sus mensajes de e-mail, almacenados en un mailbox.



Una de las funciones de e-look es la de permitir visualizar (método show()) los e-mails de un mailbox ordenados por diferentes criterios (from, subject, date, priority, …). Para ello, hemos implementado en la clase Mailbox una operación sort(), utilizando el llamado método de la burbuja:

```
private void sort() {
   for ( int i = 2; i <= email.size(), i++ )
       for ( int j = email.size(); j >= i; j-- )
           if ( before(email.at(j),email.at(j-1)) )
               // intercambiar los mensajes j y j-1
               ...
   }
```

que, como vemos, se basa a su vez en una operación before(). Como es lógico, el resultado que devuelva before dependerá del criterio de ordenación elegido para visualizar el mailbox (from, subject, date, priority, …), por lo que sería necesario parametrizar de alguna manera sort() de acuerdo a dicho criterio

Identifique un patrón de diseño que nos permita resolver la situación presentada, permitiendo incluso cambiar de un criterio de ordenación a otro mientras el usuario utiliza e-look, y de forma que sea fácilmente extensible (por ejemplo, para ordenar por otros criterios aparte de los indicados). Mostrar de forma esquemática la implementación en Java del patrón propuesto al problema descrito.

Identificación del patrón de diseño

Factores a tener en cuenta:

- Tenemos **diferentes algoritmos** de ordenación (from, subject, date, priority, text, …)
- La idea es **parametrizar** el método **sort()**.
- Fácilmente **extensible o escalable.**

Teniendo esto último en cuenta…

Elegimos el **patrón estrategia** para resolver la situación presentada.

Aplicando el patrón estrategia a nuestra situación tenemos…

- Nuestro objeto mailbox actúa como contexto. Este mantedrá una instancia sobre la estrategia actual (forma de ordenación). La parametrización del método sort proporcionará el cambio de estrategia en tiempo de ejecución.
- La interfaz SortStrategy permitirá instanciar tantas estrategias como sea necesario y por lo tanto permitiendo la extensión de estas.
- La propiedad de responsabilidad única de estrategia se ajusta perfectamente a la independencia de las distintas formas de ordenación.

Diagrama de clases de la implementación