Universidad de Málaga

Ingeniería de la Salud

Práctica 2

Patrones de diseño

Repositorio

https://github.com/GitHubAlejandroDR/P2_PdD_ADominguez

Autor

Alejandro Domínguez Recio

Curso

Ingeniería del Software Avanzada

Profesor

Antonio Jesus Vallecillo Moreno

Práctica 2. Triestables

a) Descríbase un patrón de diseño que permita implementar de manera satisfactoria dispositivos que, como el mencionado, reaccionan de forma distinta ante el mismo mensaje, dependiendo de su estado interno. Implementar en Java una particularización de dicho patrón de diseño para implementar el dispositivo Biestable descrito. [10 %]

Problema

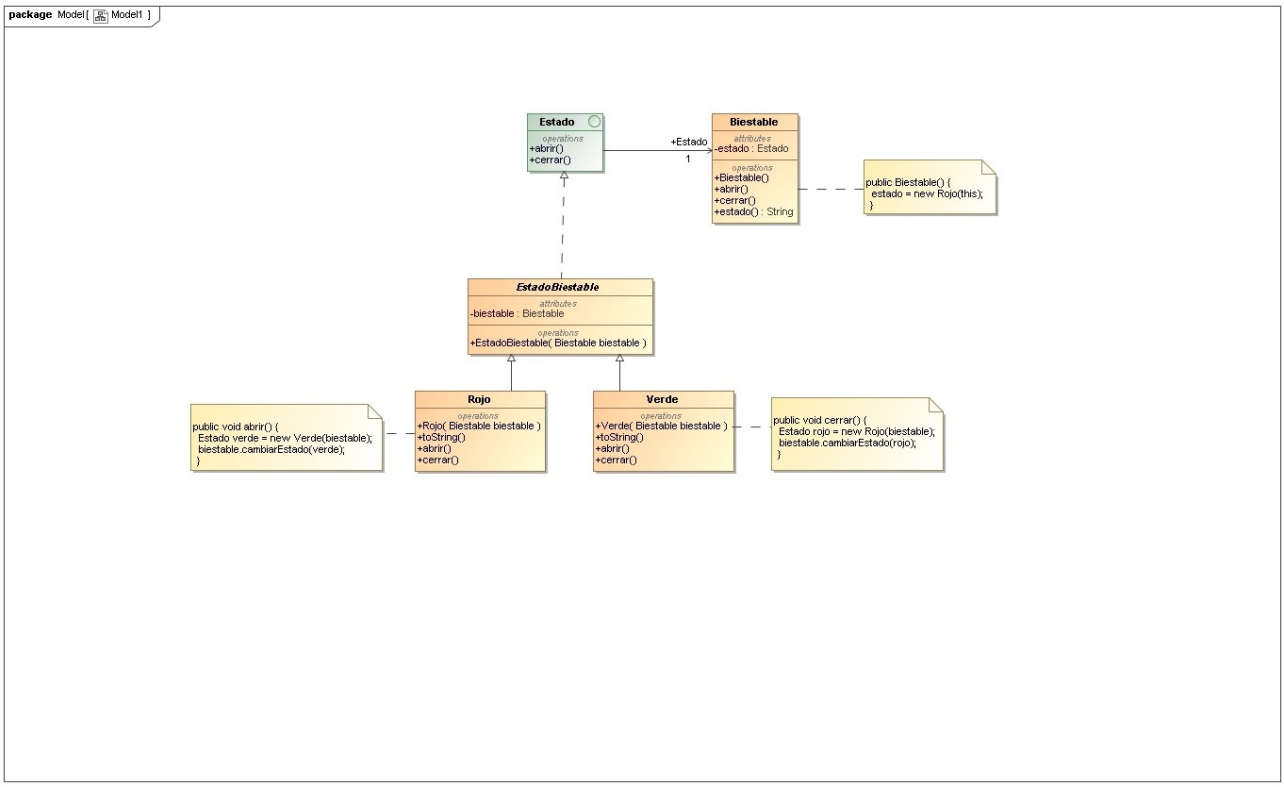
- Necesidad de modelar los distintos estados por los que pasa un biestable.
- Dependencia entre estados. El estado donde se encuentre el sistema marcará el comportamiento de este como las posibilidades de cambio.
- Ante el mismo mensaje o acción la reacción será diferente en cada estado.
- Modificar los distintos métodos del sistema para que actuén dependiendo del estado del sistema aumenta la complejidad de estos.

Solución: Patrón Estado

Aplicación:

- Creamos tantas clases como estados tenga el biestable (abierto a la extensión de estados).
- Nuestra clase biestable almacenará una instancia del estado actual. Las responsabilidades correspondientes al estado actual serán realizados por la instancia almacenada.
- La interfaz estado permite tratar como un estado tantos estados tenga nuestro modelo (rojo y verde). A su vez proporciona los métodos que estos deben implementar con sus características propias (responsabilidad única).
- La clase biestable proporciona un método para el cambio de instancia. Este último es utilizado por el método que corresponda en los distintos estados, actualizando así el estado según la situación.
- La clase abstracta EstadoBiestable contendrá una instancia del biestable. La instancia del biestable permitirá el cambio de estado a través del método propio cambiarEstado. Adémas permitirá mantener la asociación entre estado – biestable.

Diagrama de clases de la implementación



b) Supongamos ahora que deseamos implementar un dispositivo Triestable. Tal como muestra la Figura (b), un Triestable incorpora un estado intermedio Amarillo en el que la respuesta al método estado() será la cadena "precaución". Amplíese la solución propuesta en el apartado anterior para reutilizar todo lo posible el código ya desarrollado, teniendo en cuenta que en nuestro sistema deberemos disponer tanto de dispositivos Biestable como Triestable. Discuta las ventajas e inconvenientes de la solución propuesta, en particular comentando si la reutilización de código corresponde a lo que cabría esperar dadas las semejanzas de comportamiento de ambos dispositivos. Si conoce algún patrón de diseño que sea de utilidad para implementar la ampliación requerida, justifique su uso e implemente la solución al problema en Java utilizando dicho patrón. [30 %]

Problema

- Necesidad de modelar los distintos estados por los que pasa un triestable (rojo, amarillo, verde).
- Dependencia entre estados. El estado donde se encuentre el sistema marcará el comportamiento de este como las posibilidades de cambio.
- Ante el mismo mensaje o acción la reacción será diferente en cada estado.
- Modificar los distintos métodos del sistema para que actuén dependiendo del estado del sistema aumenta la complejidad de estos.

Prácticamente tenemos el mismo problema que en el apartado anterior con la diferencia del estado añadido amarillo y que desde los estados rojo y verde la reacción ante abrir o cerrar cambia a amarillo respectivamente. Por esto último reutilizamos el resto de la estructura modificando los puntos mencionados.

Solución: Patrón Estado

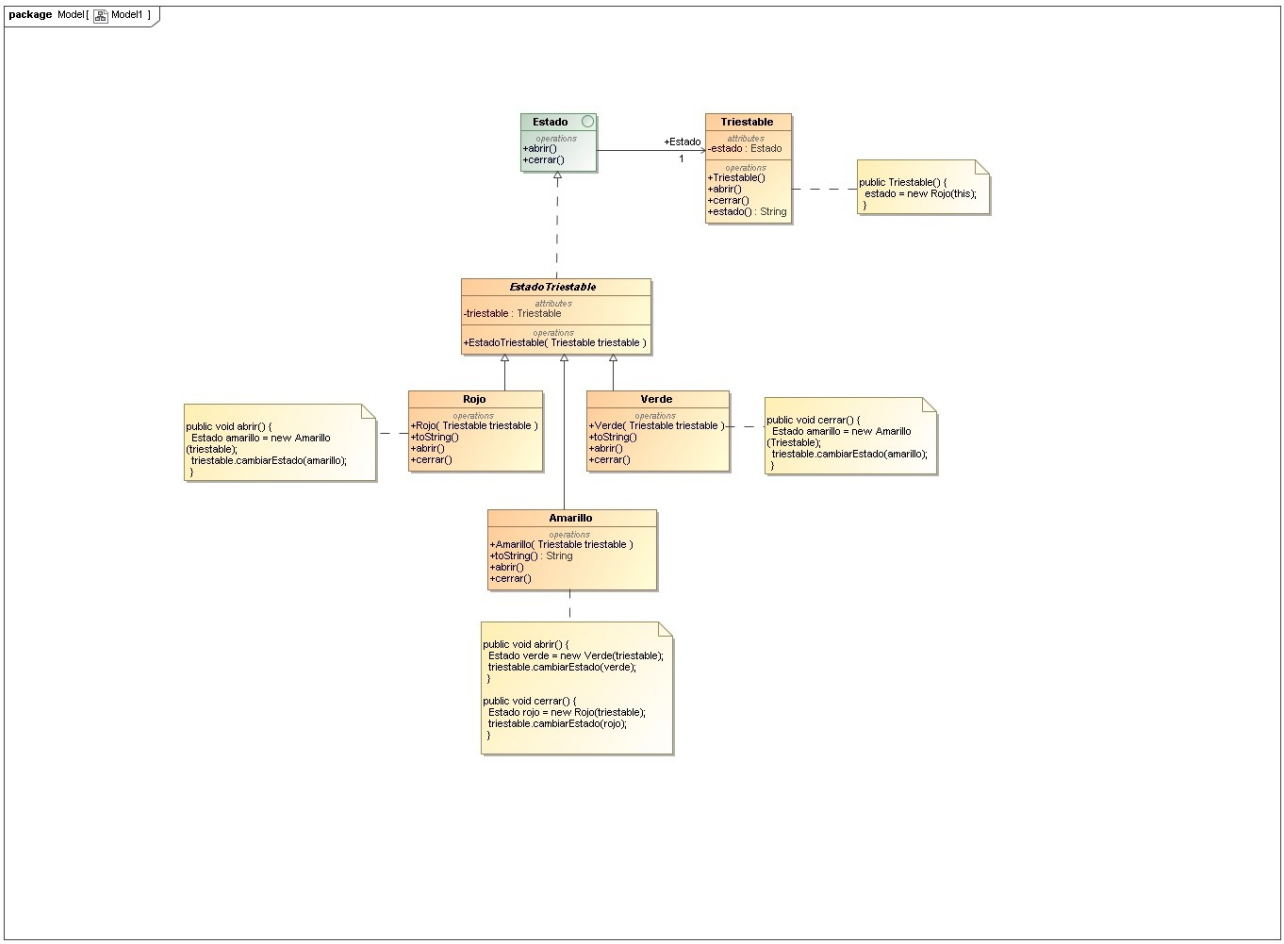
Aplicación:

- Creamos tantas clases como estados tenga el triestable (abierto a la extensión de estados).
- Nuestra clase triestable almacenará una instancia del estado actual. Las responsabilidades correspondientes al estado actual serán realizados por la instancia almacenada.
- La interfaz estado permite tratar como un estado tantos estados tenga nuestro modelo (rojo, verde, amarillo, . . .). A su vez proporciona los métodos que estas deben implementar con sus características propias (responsabilidad única).
- La clase triestable proporciona un método para el cambio de estado. Este último es utilizado por el método que corresponda en los distintos estados, actualizando así el estado según la situación.
- La clase abstracta EstadoTriestable contendrá una instancia del triestable. La instancia del triestable permitirá el cambio de estado a través del método propio cambiarEstado. Adémas permitirá mantener la asociación entre estado – triestable.

Diferencias con biestable

- Se añade un nuevo estado amarillo.
- Se modifican los métodos abrir y cerrar de los estados rojo y verde respectivamente para que su siguiente estado sea amarillo.

Diagrama de clases de la implementación



c) Supongamos por último que necesitamos realizar una nueva ampliación de nuestro sistema, en el que a la recepción de un mensaje cambio(), un dispositivo Biestable pasará a partir de ese momento a comportarse como un Triestable, y viceversa. Para ello, deberemos efectuar una fase de transición, tal como muestran las flechas discontinuas de la Figura (c), en el que pasaremos del diagrama de estados inicial (a la izquierda, con dos estados) al final (a la derecha, con tres estados) al recibir por primera vez los mensajes abrir() o cerrar() tras el mensaje cambio(). Discútase la mejor forma de modificar la solución propuesta en los apartados anteriores para tener en cuenta el nuevo requisito, de nuevo intentando la mayor reutilización de código posible. Discútase así mismo cómo tratar la situación de un Triestable que recibe un mensaje cambio() cuando se encuentra en estado Amarillo. Si conoce algún patrón de diseño que sea de utilidad para implementar la ampliación requerida, justifique su uso e implemente el sistema en Java utilizando dicho patrón. [60 %]

Problema

- Necesidad de modelar los distintos estados por los que pasa el sistema TriBiestable.
- Dependencia entre estados. El estado donde se encuentre el sistema marcará el comportamiento de este como las posibilidades de cambio.
- Ante el mismo mensaje o acción la reacción será diferente en cada estado.
- Modificar los distintos métodos del sistema para que actuén dependiendo del estado del sistema aumenta la complejidad de estos.

De la misma forma que en el apartado anterior el problema es el mismo. Tenemos un conjunto de estados (suma de estados biestable, triestables y intermedios) que se comportan de forma dependiente. Se reutilizará el total de los componentes de las dos implementaciones anteriores exceptuando que se añade un método más a la interfaz y que se crean dos estados intermedios. A su vez se renombrará la clase contexto por TriBiestable.

Solución: Patrón Estado

Aplicación:

- Creamos tantas clases como estados tenga el TriBiestable (abierto a la extensión de estados).
- Nuestra clase tribiestable almacenará una instancia del estado actual. Las responsabilidades correspondientes al estado actual serán realizados por la instancia almacenada.
- La interfaz estado permite como un estado tantos estados tenga nuestro modelo (rojobiestable, rojotriestable, verdebiestable, verdetriestable amarillo, cambioBiestable, cambioTriestable, . . .). A su vez proporciona los métodos que estas deben implementar con sus características propias (responsabilidad única).
- La clase tribiestable proporciona un método para el cambio de estado. Este último es utilizado por el método que corresponda en los distintos estados, actualizando así el estado según la situación.
- La clase abstracta EstadoTriBiestable contendrá una instancia del tribiestable. La instancia del tribiestable permitirá el cambio de estado a través del método propio cambiarEstado. Adémas permitirá mantener la asociación entre estado – tribiestable.

Diferencias con biestable y triestable

- Se crean dos nuevos estados intermedios.
- Se añade un método cambio() a la interfaz.

Diagrama máquina de estados

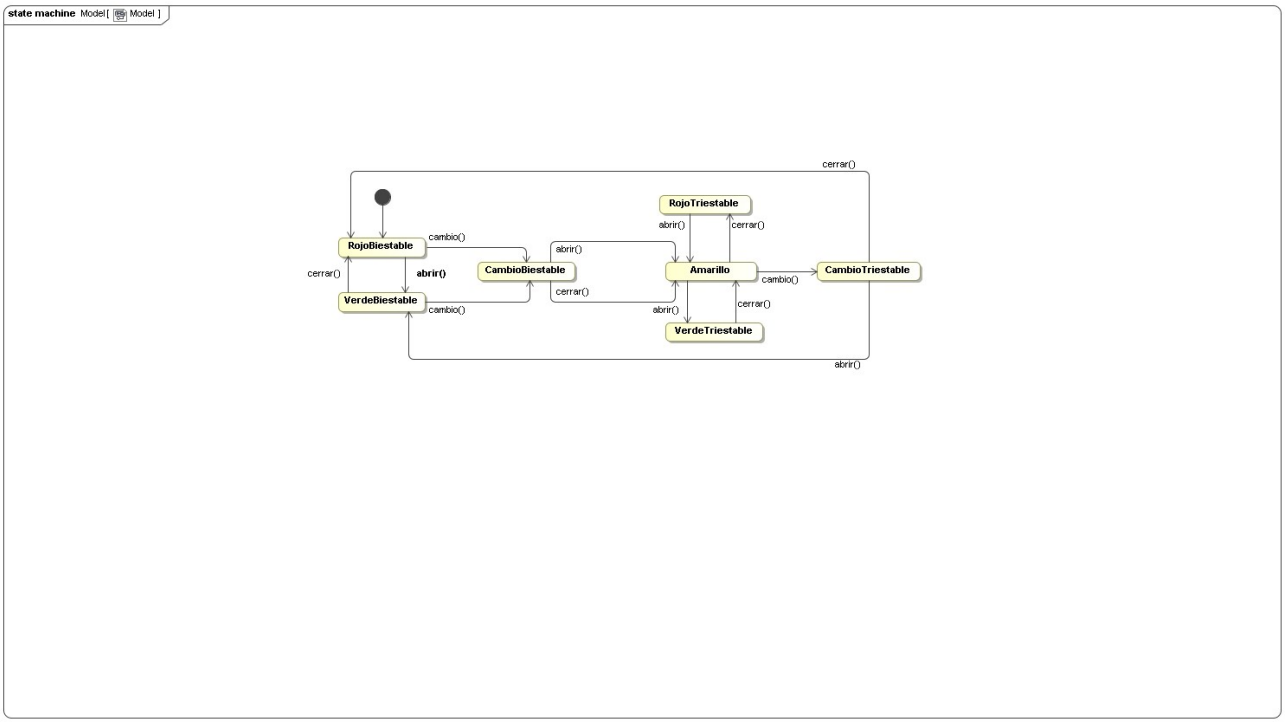


Diagrama de clases de la implementación