Ejerció 1 Control de la temperatura

Explicación breve del principio.

-El principio de inversión de la dependencia:

El principio de diseño de inversión de la dependencia consiste en depender de abstracciones y no de concreciones.

Este principio lo usamos en la clase Termostato depender del interfaz EstadoTermostato para representar el estado en el que se encuentra actualmente el la clase Termostato. Utilizando el interfaz EstadoTermostato evitamos tener una dependencia concreta que tendría al utilizar cualquiera de las clases que representan los estados (Off,Manula,Timer,.Program), por el contraria conseguimos tener una dependencia abstracta que es menos propensa a cambiar que una clases concreta.

Ademas se utilizan en la clase abstracta que Subject y en el interfaz Notificar respecto a la clase Termostato que se utilizaran para representar los cambios que se producen en la clase Termostato, estés cambios serían informados a la clases que representan los estados atrevas de la abstracción de la clase abstracta Subject y del interfaz Notificar.

-Principio de segregación de interfaces

El principio de diseño de segregación de interfaces consiste en utilizar muchos interfaces específicos para cada cliente son mejores que un único interfaz de propósito general.

Este principio lo usamos al dividir los interfaces que tenemos en dos grupos que hacen diferentes funciones cada uno, en un grupo estaría el interfaz EstadoTermostato que la funcion que hacer las operaciones de cambio de estado del termostato y el modo de funcionamiento y mostrar la información que tiene ese modo de funcionamiento en un momento concreto con la funciona print. En el otro grupo se encontraría el interfaz Notificar y la clase abstracta Subject que tendría la función de representar los cambios que se producen en la clase Termostato y avisar de esos cambios a la clases interesadas .Como se puede ver se tenemos dos interfaces específicos uno relacionado con las operaciones de los estados que puede transitar la clase termostato y el otro con informar a las clases Timer y Program de los cambios producidos en la clase Termostato.

-Principio de responsabilidad única

El principio de responsabilidad única consiste en cada objeto debe tener una responsabilidad única que esté enteramente encapsulada en la clase.
Este principio lo usamos al tener varias clases que tienen pequeñas responsabilidades. Esto se puede ver en las clase Termostato que delega el funcionamiento de la implementación de los diferentes funcionamientos que puede tener a otras clases que serían la clase Off que representar el comportamiento del termostato haciendo que estea siempre apagado y el mismo razonamiento para las clases Manual, Timer yProgram. Ademas estas clases también cambia a encendido ó apagado el termostato si su funcionamiento se lo ve necesario .También se divide la responsabilidad que tendría el termostato de informar a las clases anteriormente mencionadas si se produce algún en alguno de sus atributos , esto se relega a la clase Subject .

-Principio de substitucion de Linskov

el principio de sustitucion de liskov consiste en que las clases derivadas deben ser sustituibles por sus clases base. Este principio lo utilizamos en la clase Subject y la clase Termostato que hereda los métodos de Subject pero la clase Termostato nunca cambia el funcionamiento de cualquiera de sus métodos.

-Principio abierto – cerrado

El principio de diseño solid abierto-cerrado consiste en que las entidades software deberían ser abiertas para permitir su extensión, pero cerradas frente a la modificación
Este principio lo usamos al poner las clases Off,Manual,Timer y Program que implementen el interfaz EstadoTermostato y poniendo un atributo en la clase Termostato de tipo EstadoTermostato y poniendo unas funciones que llaman a las clases del EstadoTermostato para poder hacer los cambios de funcionamiento de la clase Termostato. Con esto conseguimos si queremos añadir una nueva clase que representaría un nuevo funcionamiento del termostato una manera que no se tendría que tocar mucho el código para poder engadir un nuevo cambio. Solo tendría que hacer para crear una nueva clase que esta nueva clase implementaría el interfaz EstadoTermostato y poner en la clase termostato una función que le permita ser llamada para poder cambiar

de estado y si hiciese falta que esa nueva clase tuviese información de la clase Termostato también tendría que implementar el interfaz Notificar .Con estés dos interfaces EstadoTermostato y Notificar eliminamos las dependencias con las clases concretas haciendo más fácil el añadir y eliminar clases que representan los estados del termostato.

Principio Tell don´t ask

El principio Tell don´t ask consiste en decirle a los objetos lo que quieres que hagan mejora la abstracción y se reduce e, acoplamiento.

Este principio lo utilizamos cuando llamados desde la clase Termostato cualquiera de las funciones que hacen el cambio de estado por ejemplo cambiarOff estas funciones lo que hacen es llamar a la función que cambia de estado que es implementa por todas las clases que representan los estados del termostato , cada función implementaría de modo diferente dependiendo de la clase en la que se encuentre.

Patrones:

Patrón Estado:

El patrón estado consiste en permitir a un objeto modificar su conducta al cambiar su estado interno.

En el patrón Estado lo utilizamos en la clases Termostato,Off,Manual,Timer,Program y en el interfaz EstadoTermostato .

Utilizamos este patrón porque con este patrón nos permite representar los diferentes estados y cambios interno que se producen en la clase Termostato mediante la modifican del atributo encendido que representa si el termostato esta encendido ó no de la clase Termostato de una manera eficiente y sencilla.

Los roles que juegan las clases en este problema para el patrón estado serían

La clase termostato juega el rol de contexto porque es la clase que se va cambiar el atributo encendido dependiendo del estado que se encuentre,el interfaz EstadoTermostato juega el rol de Estado porque representar los diferentes estados por los que puede pasar el termostato. Las clases Off,Manual, Timer y Program juegan el rol de estado al ser las clases que cambian el atributo de encendido de la clases Termostato dependiendo de las

circunstancias de cada clase.

Patron Instancia Única

el patrón Instancia Única se utiliza para asegurarnos que una clase tiene sólo una instancia y proporcionar un punto global de acceso a ella.

Se utiliza este patrón en este problema porque queremos que no solo se pueda crear un único instancia de cada clase en el caso de que sea en modo Off ó Manual que representa el modo de funcionamiento del termostato y haciendo esto conseguimos que estos dos funcionamientos del termostato se puedan utilizar en otros termostatos.

En este problema la instancia única se encontrar en las clase Off y Manual al tener un atributo del mismo tipo de que su clases en modo estático y privado y una función stactic getEstado que devolvería la instancia de cada una de las clases mencionadas anteriormente. En nuestro caso utilizamos inicializan tardía al oficializarlo cuando no piden la instancia atrevas de la llamada getEstado.

Patrón Observador

El patrón observador permite definir una dependencia entre objetos de tal forma que , cuando el objeto cambie de estado , todos sus objetos dependientes sean notificados y actualizados automáticamente.

Este patrón lo utilizamos porque nos permite notificar los cambios de temperatura y del tiempo transcurrido en la clase termostato a las clases que lo necesiten para encender o apagar el termostato dependiendo de estés atributos .En nuestro caso serian las clases interesadas en la información del termostato la clase Timer y Program al depender de estos atributos para hacer el cambio de encendido a apagado ó viceversa.

Los roles que juega en este problema las clases son Subjec juega el rol de Subjeto, Termostato juega el rol de Subjeto concreto al ser la clase que en la que se producen los cambios ,Notificar juega el rol de Obervador al ser la clase que notifica a las clases interesadas si se producio algún cambio y Timer y Program juegan el rol de Observador concreto al ser las clases en interesadas en recibir información de la clase Termostato.

Diagrama de clases del termostato

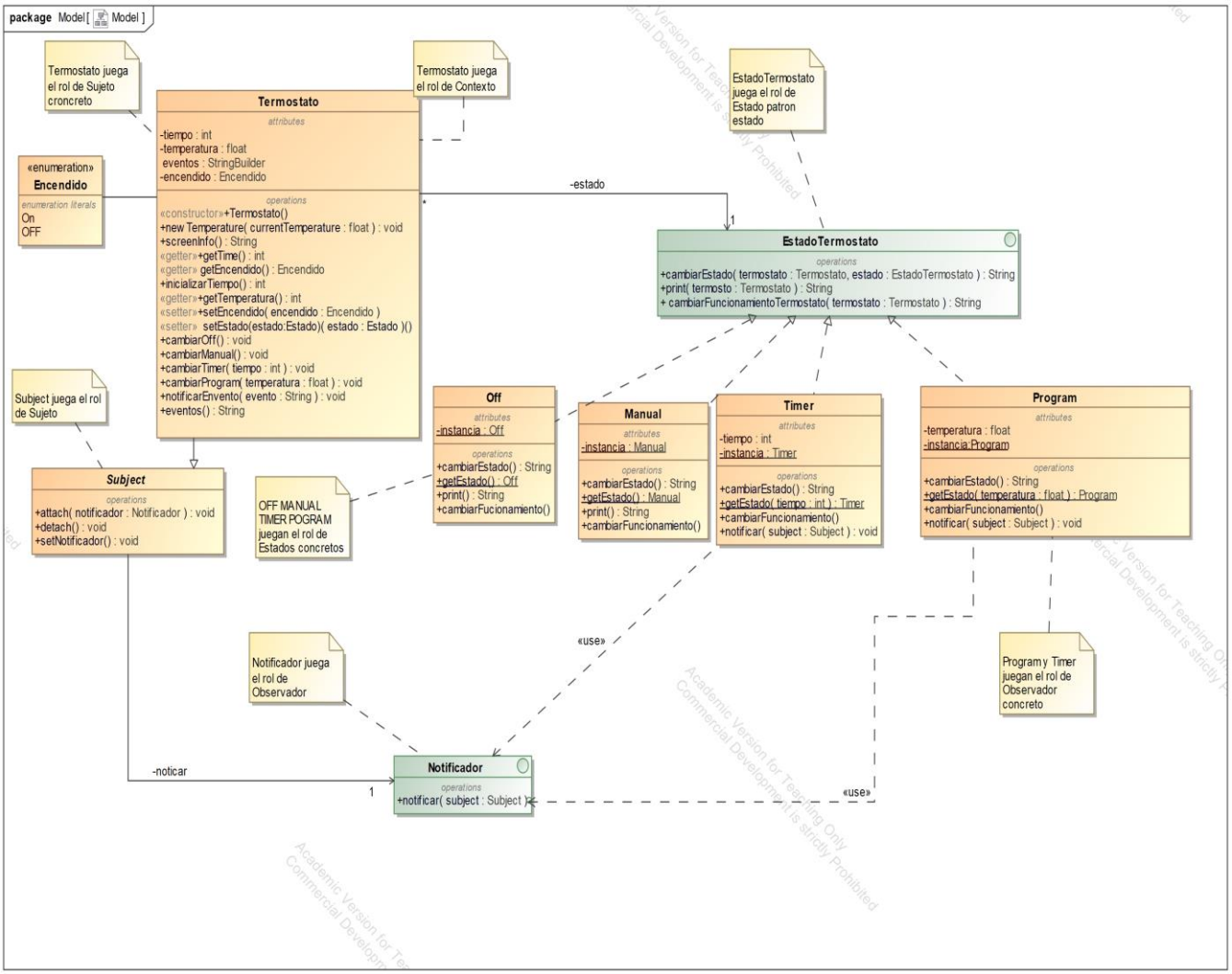

Diagrama de estados del Termostato