CONTROL DE LA TEMPERATURA

1. PRINCIPIOS DE DISEÑO UTILIZADOS EN LA PRÁCTICA:

Durante la elaboración de la practica fueron utilizados los siguientes principios de diseño, entre ellos podemos destacar los SOLID:

- Principio de responsabilidad única:

Consiste en que cada clase presente una propia responsabilidad (cada objeto representa una responsabilidad encapsulada en su clase). Esto se hace para evitar caer en clases dios que lo hacen prácticamente todo.

En el código podemos observar el principio en el uso de los interfaces donde cada uno de los estados depende única y exclusivamente de si mismo. De esta forma podremos ampliar en un futuro el número de estados del Termostato sin grandes dependencias con el resto del código.

- Principio de inversión de dependencias:

Este principio declara la posibilidad de usar módulos de un nivel de abstracción alto sin tener dependencia de los módulos de bajo nivel de abstracción.

En este código en concreto se ve perfectamente en el uso del interfaz general de los distintos estados, ahí podemos observar cómo el estado del termostato no tiene dependencia alguna con los módulos de bajo nivel, sino que tiene una única procedente del interfaz.

- Principio de abierto-cerrado:

Este principio indica que las entidades software deben ser abiertas para permitir su extensión, pero cerradas a posibles modificaciones. Es decir, queremos cambiar o añadir funcionalidades sin tocar el código ya existente.

En nuestro código podemos observar que gracias al uso de los interfaces de estados podemos añadir funcionalidades en cualquier momento sin modificar el código ya existente.

2. PATRON DE DISEÑO UTILIZADO:

El patrón de diseño que se ha utilizado en esta práctica es el patrón estado donde tenemos una clase principal que puede pasar por varios estados dinámicamente en tiempo de ejecución. Esto se consigue usando un interfaz general como estado que luego será implementado por los distintos estados en sus clases específicas.
Esto permite modificar el "estado" de la clase principal de forma dinámica y a través de ligadura dinámica su comportamiento, ya que, un estado sobrescribe los métodos instaurados en el interfaz.

La ventaja principal del uso de este patrón es la simplicidad de comprensión del código con respecto a su antigua versión basada en condicionales anidados y la poca capacidad de modificación que estos implican. Al mismo tiempo, nos permite delegar las acciones de nuestra clase principal favoreciendo al encapsulamiento.

3. DIAGRAMAS.

Diagrama de clases:

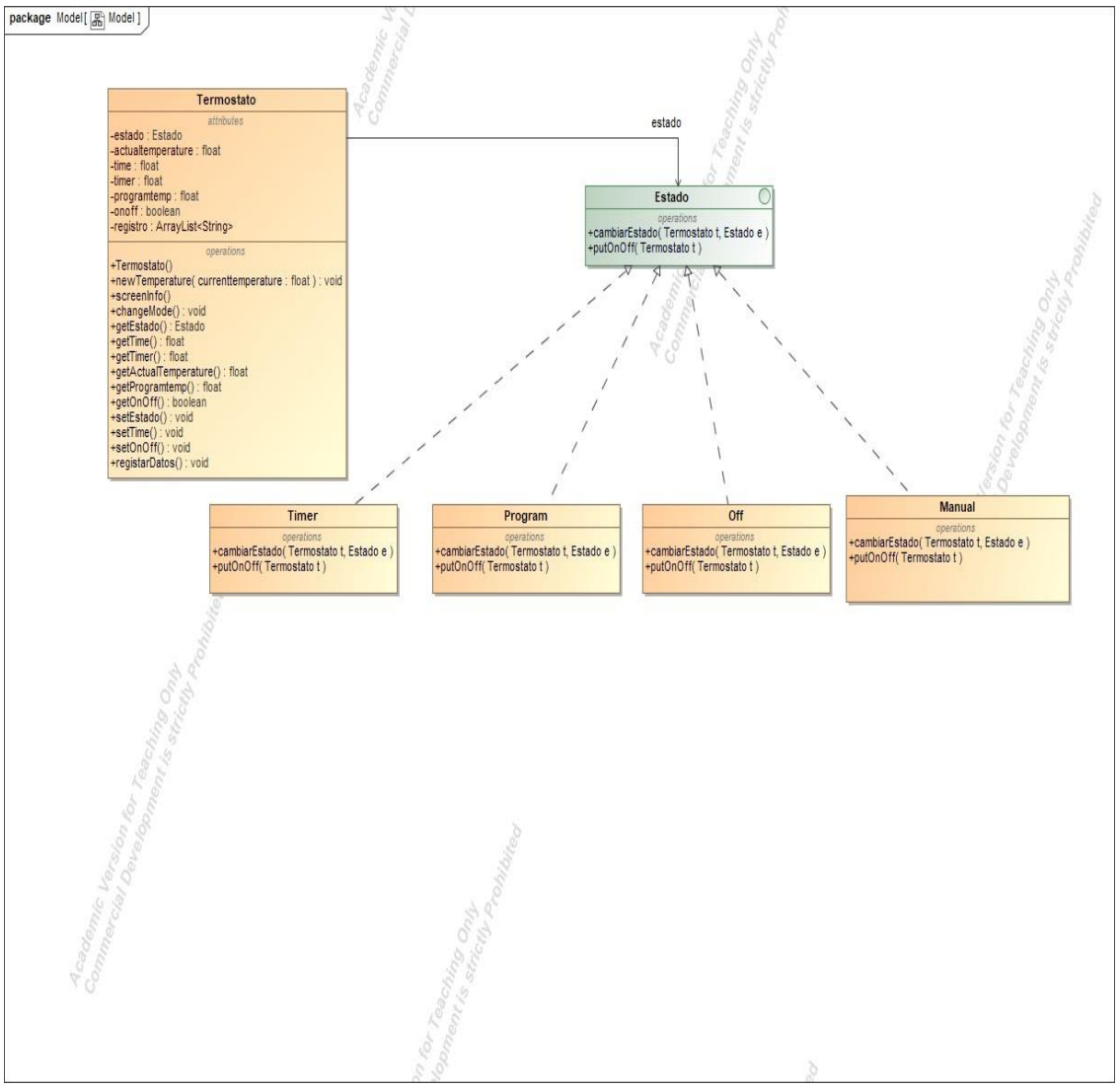

Diagrama Dinámico: para este ejercicio se optó por usar un diagrama dinámico de estados debido a que permite ver con mayor claridad el funcionamiento del programa.