# Informe Práctica de Diseño – Ejercicio 1

## Principios de diseños usados:

- ### Principio de sustitución de Liskov:
  Este principio indica que las clases derivadas deben poder subtituirse por la clase base. Es decir, que allí donde usemos una superclase, también podamos usar sus subclases sin alterar el comportamiento y la funcionalidad del programa.
  Este principio se puede ver en la clase Termostato, cuyo atributo “mode” es de tipo Mode, pero en realidad almacenamos instancias de subclases de Mode. También usamos este principio en el método “newMode” de la clase Termostato, donde el parámetro de entrada es de tipo Mode, pero nosotros le pasamos un parámetro que tiene como tipo una subclase de Mode.

- ### Principio de inversión de la dependencia:
  Este principio indica que siempre que podamos debemos depender de interfaces o clases abstractas en lugar de clases concretas.
  Lo utlizamos por un lado en el atributo historial de la clase Termostato, donde lo declaramos como “List” y no como “ArrayList”. De esta forma no dependemos de “ArrayList” que es una clase concreta, sino que de una interfaz: “List”.
  Por otro lado, también lo usamos al hacer que Manual, Off, Program y Timer implementen una interfaz, consiguiendo así que dependan de ella en lugar de depender de una clase concreta

- ### Principio de abierto cerrado:
  Este principio indica que se debería de poder extender el comportamiento de una clase sin modificarla. Es decir, que las clases tienen que estar abiertas para la extensión y cerradas para la modificación. Esto lo conseguimos a través de la herencia.
  Hemos empleado este principio en la Interfaz Mode, que es implementada por las subclases Manual, Off, Program y Timer y, por lo tanto, se puede extender pero no modificar al ser una interfaz.

## Patrón de diseño usado:

Para realizar este ejercicio hemos seguido el Patrón Estado. Este patrón permite que un componente modifique su conducta cuando se cambia de estado.

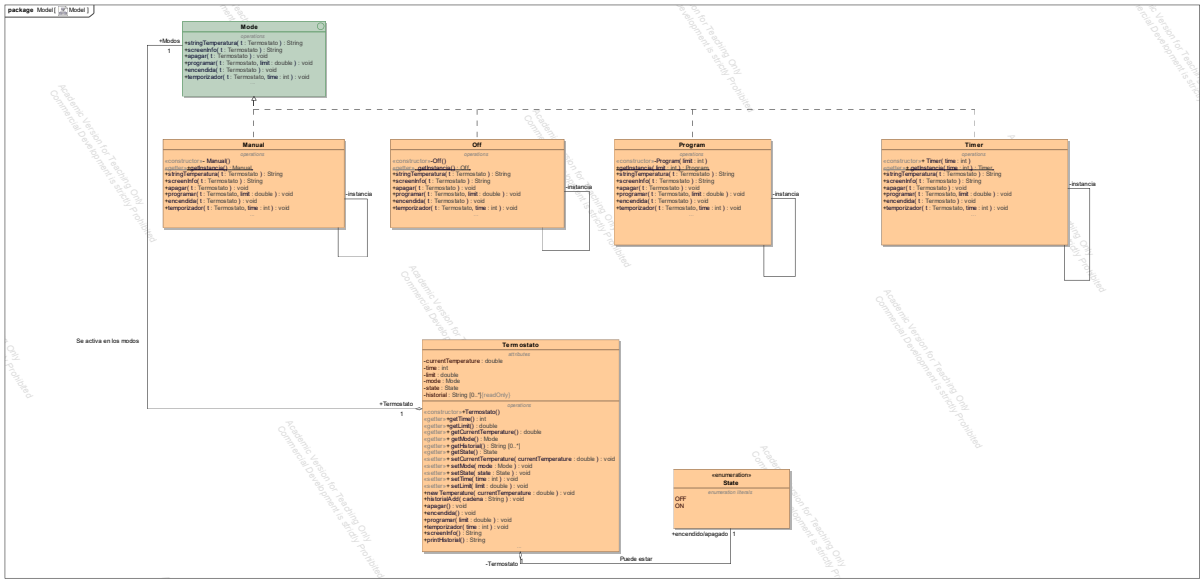
Hemos considerado que este patrón resulta de gran utilidad ya que necesitamos que el comportamiento del Termostato varíe en función del estado en el que se encuentre.

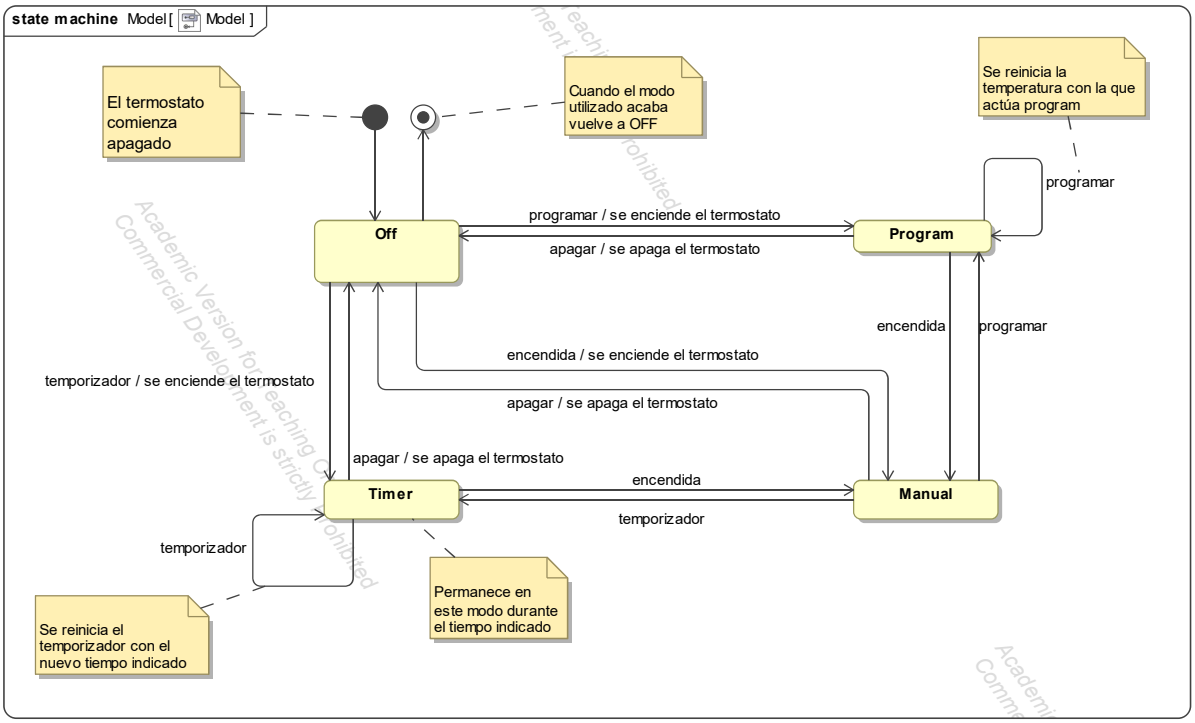
En este caso podríamos considerar lo siguiente:

- Contexto: La clase Termostato
- Estado: La clase Mode
- Estados concretos: Las clases que implementan a Mode (Off, Manual, Timer y Program).

Además, implícitamente, este patrón da lugar a la utilización de otro patrón: el Patrón Instancia Única. Este patrón lo que nos permite es la creación de una única instancia de la clase a la que se le aplica, proporcionando un punto global a esta instancia.

Este patrón dentro del Patrón Estado es útil ya que nos permite crear una única instancia de cada Estado Concreto. De este modo siempre usamos las mismas instancias y solo existe un estado por cada Estado Concreto.





# Informe Práctica de Diseño - Ejercicio 2

## Principios de diseños usados:

- ### Principio de sustitución de Liskov:
  Este principio indica que las clases derivadas deben poder sustituirse por la clase base. Es decir, que allí donde usemos una superclase también podamos usar sus subclases sin alterar el comportamiento y la funcionalidad del programa.

  Este principio se puede ver en la clase Proyecto, donde hay un atributo de tipo GrupoTrabajo, pero almacenamos instancias de sus subclases (Trabajador o EquipoTrabajo). Además, en varios de los métodos que usamos tienen como parámetro un objeto de tipo GrupoTrabajo, pero les pasamos un objeto de tipo Trabajador o de tipo EquipoTrabajo.

- ### Principio de inversión de la dependencia:
  Este principio indica que debemos depender de interfaces o clases abstractas en lugar de clases concretas.

  Este principio los usamos en la clase EquipoTrabajo para el atributo grupoTrabajo. Donde lo declaramos como "List" y no como "ArrayList". De esta forma no dependemos de "ArrayList" que es una clase concreta, si no de una interfaz que es "List".

  También lo usamos en las clases Trabajador y EquipoTrabajo, donde hacemos que se extienda de la clase abstracta GrupoTrabajo en lugar de una clase concreta

- ### Principio de abierto cerrado:
  Este principio indica que se debería de poder extender el comportamiento de una clase sin modificarla. Es decir, que las clases tienen que estar abiertas para la extensión y cerradas para la modificación. Esto lo conseguimos a través de la herencia.

  Esto lo podemos ver en la clase GrupoTrabajo que está abierta para ser extendida, ya que de ella heredan las clases Trabajador y EquipoTrabajo pero está cerrada a la modificación.

Mario López Cea
Marta Martín de Argenta Hernández
Grupo 4.2

## Patrón de diseño usado:

Para poder resolver este ejercicio hemos usado el Patrón Composición. Este patrón nos permite componer objetos en forma de árbol. De este modo se establecen jerarquías y hay objetos conteniendo a otros objetos.

Consideramos que este patrón se ceñía muy bien a lo que se nos pedía, resultando de gran utilidad, ya que en el ejercicio se plantea una estructura jerárquica en modo de árbol. En esta estructura existe un proyecto formado por un equipo de trabajo que a su vez puede estar formado por mas equipos de trabajo o trabajadores. De forma que las "hojas" son los trabajadores y los "nodos" los equipos de trabajo.

En este caso el cliente es la clase Proyecto, el Componente la clase GrupoTrabajo, el ComponenteConcreto la clase Trabajador y la Composición la clase EquipoTrabajo