# MANUAL TECNICO

# HEAT-TECH


REALIZADO POR:

NATALIA ALEXANDRA SOLORZANO PAZ 00120310

CARMEN ELISA LOPEZ ALVARADO 00100619

# Capturas de pantalla:

# Diagramas UML:

# Patrones de Diseño:

## PATRÓN DE DISEÑO PROXY

Tipo implementado: Proxy de Protección.

### ACERCA DE:

El patrón proxy trata de proporcionar un objeto intermediario que represente o sustituya al objeto original con motivo de controlar el acceso y otras características del mismo.

El patrón Proxy se usa cuando se necesita una referencia a un objeto más flexible o sofisticada que un puntero.
Según la funcionalidad requerida podemos encontrar varias aplicaciones:
Proxy de protección: controla el acceso a un objeto.
Proxy remoto: representa un objeto en otro espacio de direcciones.
Proxy virtual: retrasa la creación de objetos costosos.
Referencia inteligente: tiene la misma finalidad que un puntero, pero además ejecuta acciones adicionales sobre el objeto, como por ejemplo el control de concurrencia.

### VENTAJAS O CONSECUENCIAS:

Según el tipo de Proxy, las consecuencias positivas pueden ser:
Proxy de protección y referencias inteligentes: permiten realizar diversas tareas de mantenimiento adicionales al acceder a un objeto.
Proxy remoto: oculta el hecho de que un objeto reside en otro espacio de direcciones.

Proxy virtual: puede realizar optimizaciones, como la creación de objetos bajo demanda.

### ¿POR QUÉ SE ELIGIÓ ESTE PATRÓN?

### IMPLEMENTACIÓN:

## PATRÓN DE DISEÑO DECORADOR

### ACERCA DE:

Nuestro sistema requiere que la funcionalidad de ciertos componentes pueda extenderse y modificarse en tiempo de ejecución. Además, pretendemos que esta característica no se implemente mediante herencia para poder aprovechar al máximo las clases existentes sin introducir jerarquías complejas y extensas.

Se aplica cuando:

- Necesitamos añadir responsabilidades a objetos individuales de forma dinámica y transparente
- Se pueden revocar responsabilidades antes asignadas a nuestros objetos.
- La extensión mediante herencia viola los principios SOLID.
- Necesitamos extender la funcionalidad de una clase, pero la herencia no es una solución viable.
- Necesitamos extender la funcionalidad de un objeto en tiempo de ejecución e incluso eliminarla si fuera necesario.

### VENTAJAS O CONSECUENCIAS:

- Es más flexible que la herencia.
- Permite añadir y eliminar responsabilidades en tiempo de ejecución.
- Evita la herencia con muchas clases y la herencia múltiple.

- Limita la responsabilidad de los componentes para evitar clases con excesivas responsabilidades en los niveles superiores de la jerarquía.

## ¿POR QUÉ SE ELIGIÓ ESTE PATRÓN?

## IMPLEMENTACIÓN:

# Script de la Base de Batos:

```
CREATE TABLE public."DEPARTAMENTO"
(
    "idDepartamento" integer NOT NULL DEFAULT,
    nombre character varying(100) COLLATE,
    ubicacion character varying(150) COLLATE,
    CONSTRAINT pk_iddepartamento PRIMARY KEY ("idDepartamento")
);

CREATE TABLE public."REGISTRO"
(
    "idRegistro" integer NOT NULL DEFAULT,
    "idUsuario" integer,
    entrada boolean,
    "fechaYhora" timestamp without time zone,
    temperatura double precision,
    CONSTRAINT "pk_idRegistro" PRIMARY KEY ("idRegistro"),
    CONSTRAINT "fk_idUsuario" FOREIGN KEY ("idRegistro")
        REFERENCES public."USUARIO" ("idUsuario") MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
);
```

```
CREATE TABLE public."USUARIO"
(
    "idUsuario" integer NOT NULL DEFAULT,
    "idDepartamento" integer,
    contrasena character varying(100) COLLATE,
    nombre character varying(100) COLLATE,
    apellido character varying(100) COLLATE,
    dui character varying(15) COLLATE,
    "fechaNacimiento" date,
    CONSTRAINT "pk_idUsuario" PRIMARY KEY ("idUsuario"),
    CONSTRAINT "fk_idDepartamento" FOREIGN KEY ("idUsuario")
        REFERENCES public."DEPARTAMENTO" ("idDepartamento")
MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
);
```