4) 5)

Patrones utilizados:

**Singleton**

Problema: Se admite exactamente una instancia de una clase- es un “singleton”. Los objetos necesitan un único punto de acceso global.

Solución: Defina un método estático (nivel de clase) de la clase que devuelva el singleton.

Justificación: Se necesita solo una instancia de las fábricas por ejemplo y se evita el trabajo de creación si nunca se accede a la misma, por eso lo utilizamos.

Patrones Grasp relacionados: Indireccion, Fabricación pura.

**Estrategia**

Problema: ¿Cómo diseñar diversos algoritmos o políticas que están relacionadas?

¿Cómo diseñar que estos algoritmos o políticas puedan cambiar?

Solución: Defina cada algoritmo/política/estrategia en una clase independiente con 1 interfaz común.

Justificación: Podrían incluirse nuevas políticas/algoritmos/estrategias de asignación, las mismas podrían cambiar en el tiempo o incluso podrían ser válidas más de una al mismo tiempo, con lo cual se evita un impacto global, y sólo deben modificarse las estrategias particulares y asociarse a la interfaz común.

Patrones Grasp relacionados: Variaciones protegidas, Polimorfismo, Fabricación pura.

**Factoría**

Problema: ¿Quién debe ser el responsable de la creación de objetos cuando existen consideraciones especiales, como una lógica de creación compleja, el deseo de separar las responsabilidades de la creación para mejorar la cohesión, etc.?

Solución: Asignar la responsabilidad a un objeto de fabricación pura denominado factoría que maneje la creación.

Justificación: La Factoría permite separar la responsabilidad de la creación compleja en objetos de apoyo cohesivos y ocultar la lógica de creación potencialmente compleja, como por ejemplo para crear los Adaptadores para cada puerta.

Patrones Grasp relacionados: Fabricación pura, Indirección, Singleton.

**Adaptador**

Problema: ¿Cómo resolver interfaces incompatibles, o proporcionar una interfaz estable para componentes parecidos con diferentes interfaces?

Solución: Convierta la interfaz original de un componente en otra interfaz mediante un objeto adaptador intermedio.

Justificación: Los adaptadores utilizan interfaces y el polimorfismo para añadir un nivel de indirección. Un adaptador envuelve el acceso a un subsistema o sistema con un único objeto, por eso los utilizamos para las puertas.

Patrones Grasp relacionados: Variaciones protegidas, Polimorfismo, Indirección, Fabricación Pura.

**DTO**

Problema: ¿Cómo puedo intercambiar datos entre diferentes capas de una aplicación de manera de mantener un bajo acoplamiento entre ellas?

Solución: Crear pseudo-entidades a medida de las necesidades. Es decir, definir clases con los atributos necesarios para representar los datos que se desean intercambiar entre las diferentes capas.

Justificación: Permite enviar y recibir datos entre diferentes capas, contenidos en una única estructura, además un cambio en la capa generadora de estos objetos no implica un cambio en las demás capas, consiguiendo así el desacoplamiento deseado.

Patrones Grasp relacionados: Alta cohesión, Bajo acoplamiento, Fabricación pura.

**Ejercicio adicional**

public class ExpertoValidarAcceso {

FabricaAdaptadores fabrica;

AdaptadorPuerta adaptador;

Puerta p;

public boolean ValidarAcceso(){

boolean confirmacion;

fabrica = FabricaAdaptadores.getInstancia();

adaptador =fabrica.getAdaptadorPuerta(p);

confirmacion = adaptador.abrirPuerta(p);

if(confirmacion){

// hubo un error

}

}/\* otros métodos

\*/

}

public class FabricaAdaptadores {

private static FabricaAdaptadores instancia;

public static FabricaAdaptadores getInstancia(){

if(instancia==nul){

instancia= new FabricaAdaptadores();

}

return instancia;}

public AdaptadorPuerta getAdaptadorPuerta(Puerta puerta){

if(puerta.getTipo().compareTo(“IP”)==0){

return new AdaptadorPuertaIP();

}

else{

if(puerta.getTipo().compareTo(“serie”)==0){

return new AdaptadorPuertaSerie();

}

}

}

}

public interface AdaptadorPuerta {

public boolean abrirPuerta(Puerta puerta);

}

public class AdaptadorPuertaSerie implements AdaptadorPuerta {

pPublic boolean abrirPuerta(Puerta puerta){

boolean error = false;

int nSerie=((PuertaSerie)puerta).getNumeroSerie();

// faltan detalles técnicos de apertura

return error;

}

}

public class AdaptadorPuertaIP extends AdaptadorPuerta {

public boolean abrirPuerta(Puerta puerta){

boolean error = false;

int nIp=((PuertaIP)puerta).getNumeroIP();

// faltan detalles técnicos de apertura

return error;

}

}