Problema: Validación de Strings

- ✓Supongamos que estamos diseñando un formulario para ingresar datos.
- ✓Los input fields deben ser validados, pero la validación depende del dominio del texto ingresado
  - ✓Teléfono.
  - ✓Número entero.
  - ✓E-mail.
  - ✓Dirección.
  - ✓...
- ✓No podemos preveer de antemano todas las posibles formas de validación.

Solucion

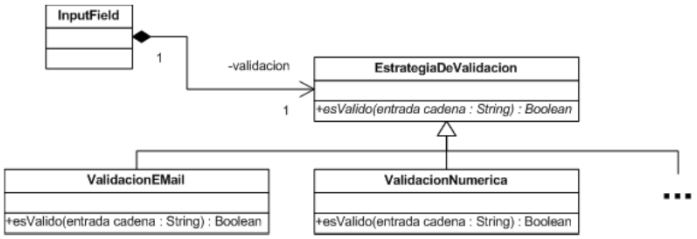
- ✓En la clase donde esta el String, por ejemplo Field, chequear que se esta ingresando….

Field
aString
esValido?

Problemas?

Validación de Strings

- ✓ Solución: Encapsular el algoritmo de validación en un objeto.



Patrón Strategy

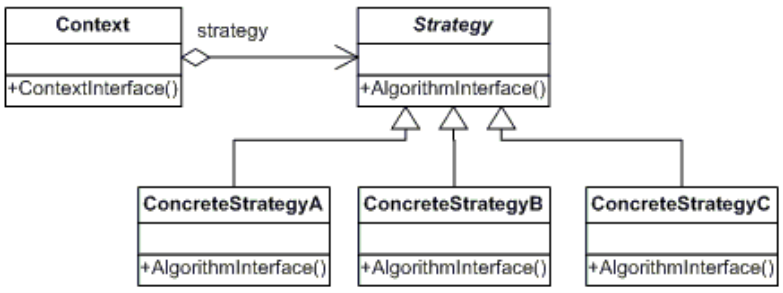
- ✓ **Intent:**
  - ✓ Desacoplar un algoritmo del objeto que lo utiliza.
  - ✓ Permitir cambiar el algoritmo que un objeto utiliza en forma dinámica.
  - ✓ Brindar flexibilidad para agregar nuevos algoritmos que lleven a cabo una función determinada.

Patrón Strategy

- **Applicability:**
  - Existen muchos algoritmos para llevar a cabo una tarea.
  - No es deseable codificarlos todos en una clase y seleccionar cual utilizar por medio de sentencias condicionales.
  - Cada algoritmo utiliza información propia. Colocar esto en los clientes lleva a tener clases complejas y difíciles de mantener.
  - Es necesario cambiar el algoritmo en forma dinámica.

Patrón Strategy

- **Solución:**
  - Definir una familia de algoritmos, encapsular cada uno en un objeto y hacerlos intercambiables.

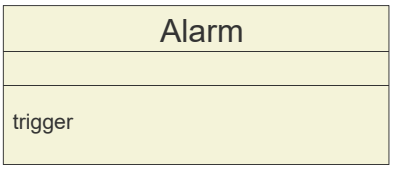


Patrón Strategy

- ✓ **Consecuencias:**
  - \+ Alternativa a subclasificar el contexto, para permitir que se pueda cambiar dinámicamente.
  - \+ Desacopla al contexto de los detalles de implementación de las estrategias.
  - \+ Se eliminan los condicionales.
  - – Overhead en la comunicación entre contexto y estrategias.
  - – Los clientes deben conocer las diferentes estrategias para poder elegir.
- ✓ **Implementación**:
  - ✓ El contexto debe tener en su protocolo métodos que permitan cambiar la estrategia
  - ✓ Parámetros entre el contexto y la estrategia

Sistema de Alarmas

- ✓ Supongamos una clase Alarma con un comportamiento “trigger” que reacciona a mensajes enviados por sensores

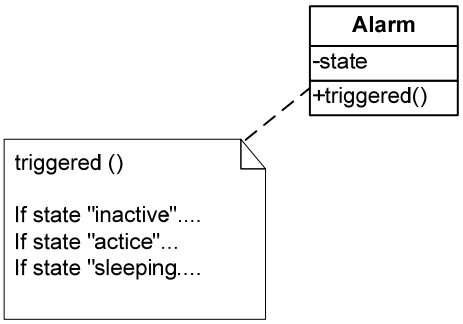


Pero….

- ✓La alarma puede estar en diferentes estados y en funcion de eso reacciona:
  - ✓Si esta inactive no toma en cuenta ningun aviso de los sensores
  - ✓Si esta active tiene que reaccionar de acuerdo a su comportamiento como Alarma
  - ✓Si esta “sleeping” se activa…..
  - ✓Otras combinaciones….

Como resolvemos el problema?

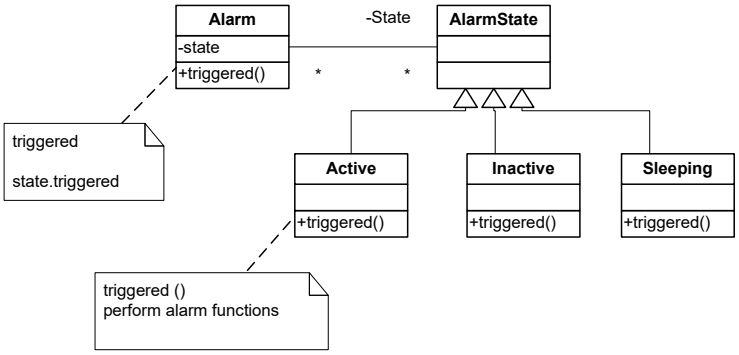
- ✓ Solucion “ingenua”:



Problemas con esta solucion?

Una solucion mejor

- ✓ “Objetificar” el estado



Patrón State

- ✓ **Intent:**
  - ✓ Modificar el comportamiento de un objeto cuando su estado interno se modifica.
  - ✓ Externamente parecería que la clase del objeto ha cambiado.

Patron State

- **Aplicabilidad:**

Usamos el patron State cuando:

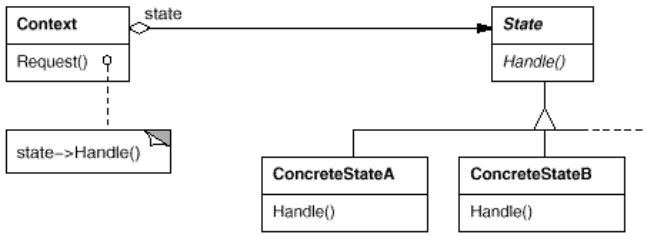
- Elcomportamiento de un objeto depende del estado en el que se encuentre.
- Los metodos tienen sentencias condicionales complejas que dependen del estado.Este estado se representa usualmente por constantes enumerativas y en muchas operaciones aparece el mismo condicional. El patron State reemplaza el condicional por clases (es un uso inteligente del polimorfismo)

Patrón State

- **Detalles:**
  - Desacoplar el estado interno del objeto en una jerarquía de clases.
  - Cada clase de la jerarquía representa un estado concreto en el que puede estar el objeto.
  - Todos los mensajes del objeto que dependan de su estado interno son delegados a las clases concretas de la jerarquía (polimorfismo).

Patron State

- Estructura



Temas interesantes

- Ejecucion de los comportamientos de la alarma. Donde estan ubicados?
- Como cambiamos de estado?
- Muchos objetos Alarma, comparten la jerarquia de estados?

Patron State

- **Participantes**
  - **Context (Alarm)**
    - **Define la interfaz que conocen los clientes.**
    - **Mantiene una instancia de alguna clase de ConcreteState que define el estado corriente**
  - **State (AlarmState)**
    - **Define la interfaz para encapsular el comportamiento de los estados de Context**
  - **ConcreteState subclases (Active, Inactive,Sleeping)**
    - **Cada subclase implementa el comportamiento respecto al estado especifico.**

Patrón State

- **Consecuencias:**
  - Localiza el comportamiento relacionado con cada estado.
  - Las transiciones entre estados son explícitas.
  - En el caso que los estados no tengan variables de instancia pueden ser compartidos.