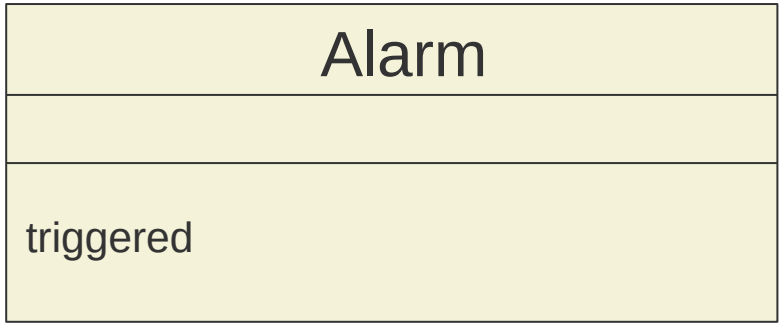
State

Laboratorio de Investigación y Formación en Informática Avanzada
Facultad de Informática, Universidad Nacional de La Plata
Calle 50 esq. 115 - Primer piso (1900) La Plata, Argentina
TE: (0221) 422 8252 http://www-lifia.info.unlp.edu.ar

Sistema de Alarmas

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



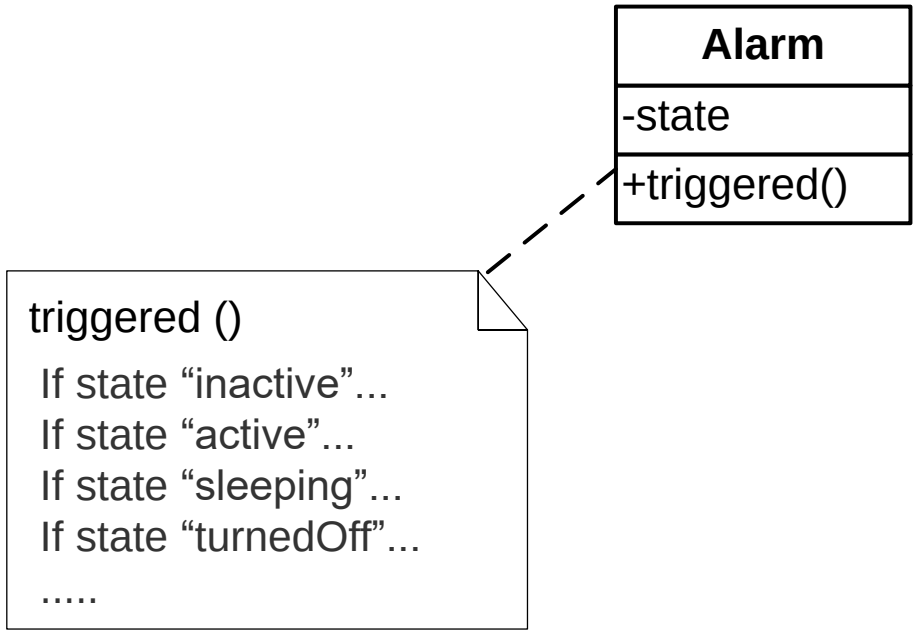
Pero….

- 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?

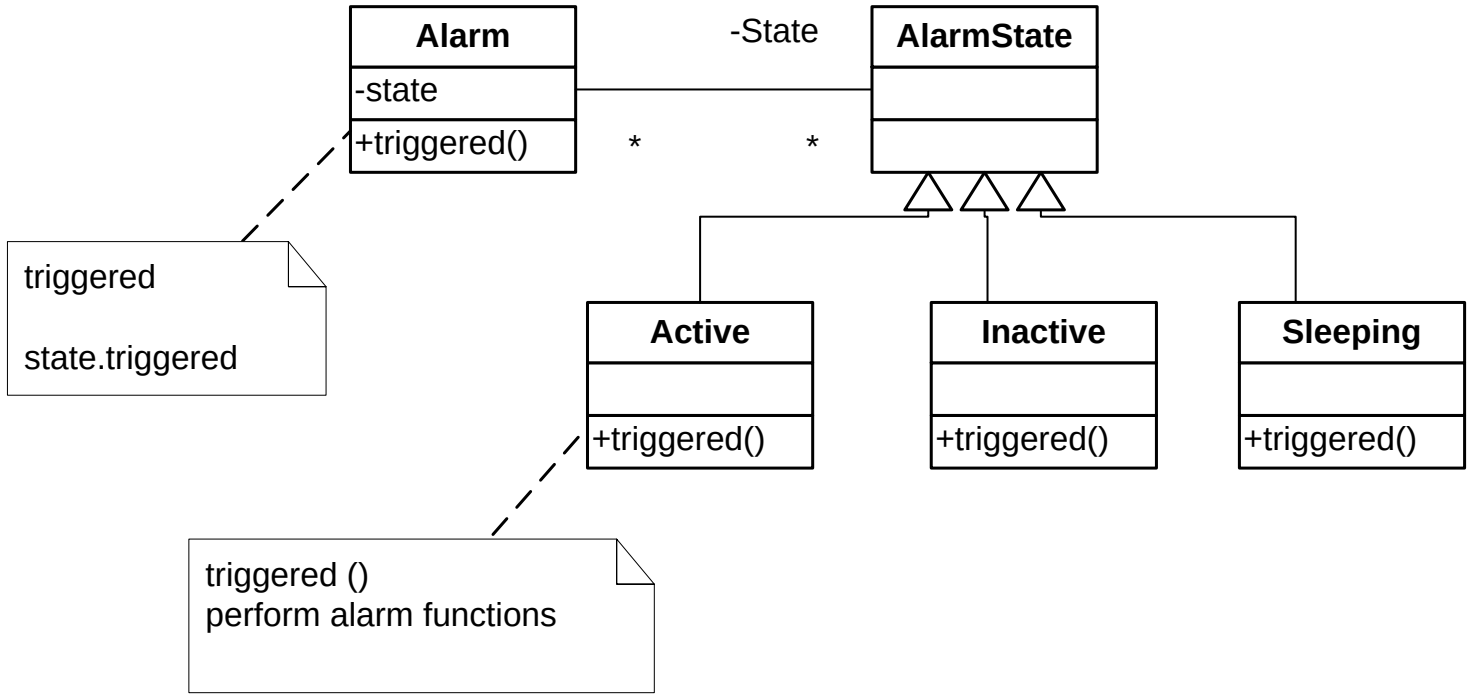
- ✓ Solucion “ingenua”:

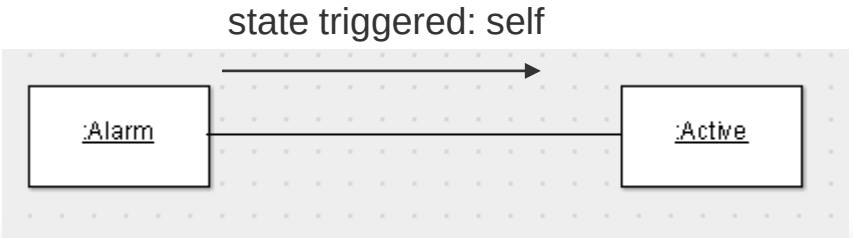


Problemas con esta solucion?

Una solucion mejor

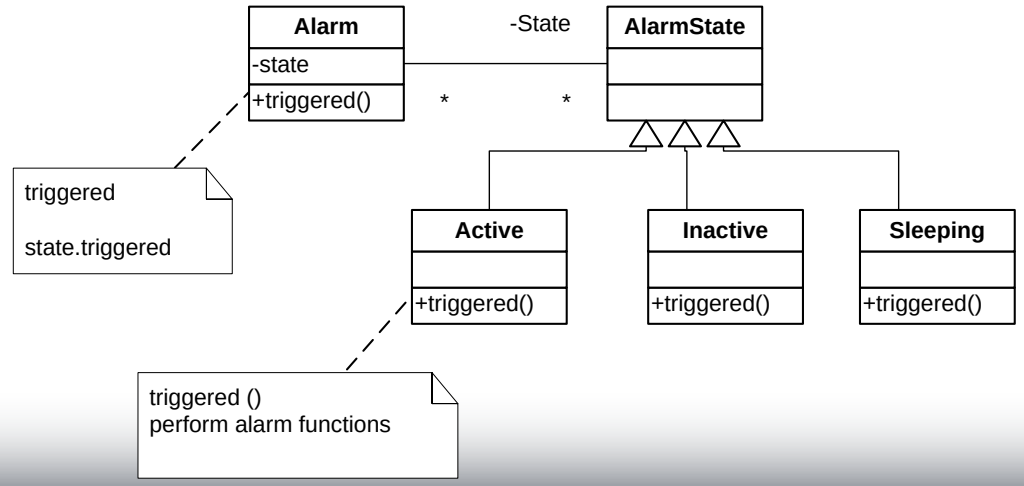
- ✓ “Objetificar” el estado





Aspectos a considerar

- ✓ La alarma cambia de estado con cierta frecuencia (tenemos que ver como la manejamos en el código)
- ✓ Hay diversas razones para cambiar de estado
- ✓ En cada estado tendremos otros métodos que son los que provocan cambio de estado
- ✓ Necesitamos manipular el estado que es una variable en Alarma



Conclusiones

- ✓ La alarma delega la operación en su estado actual
- ✓ El estado podría devolver (como respuesta) cual es el nuevo estado
- ✓ El estado necesita conocer a la Alarma? Por que?
- ✓ Como conseguimos esto?

En el código

- ✓ En la Clase Alarma, todos los mensajes cuya respuesta depende del estado se delegan en el estado

```
triggered
 state triggered: self
```

- ✓ En cada estado se manipulan las acciones y el cambio de estado comunícandose con “su” alarma

```
Class Active
   triggered: miAlarma
      miAlarma sonarBocina
      ^(Sonando new)
```

```
Class Active
   triggered: miAlarma
      miAlarma sonarBocina
      miAlarma setState: (Sonando new)
```


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:

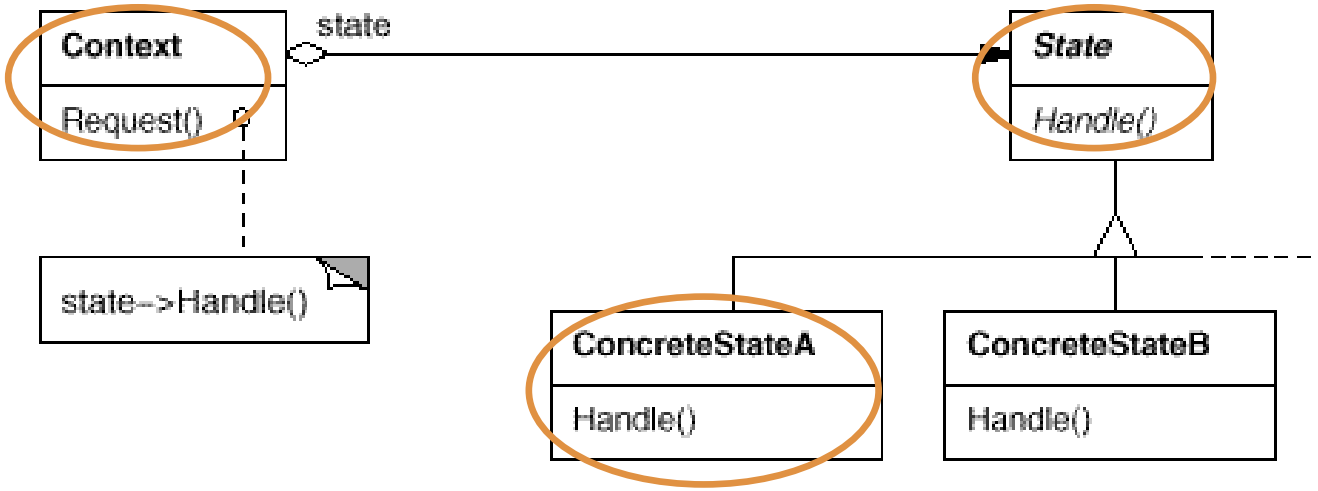
- ✓ Elcomportamiento de un objeto depende del estado en el que se encuentre.
- ✓ Los metodos tienen sentencias condicionales complejas que dependen del estado.Este estado se representa usualmente por constantes enumerativas y en muchas operaciones aparece el mismo condicional. El patron State reemplaza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



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.