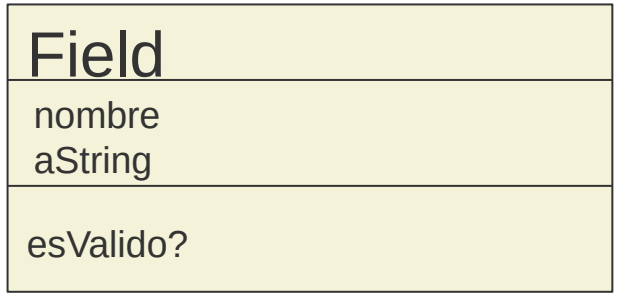
Problema: Validación de Strings

- 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….



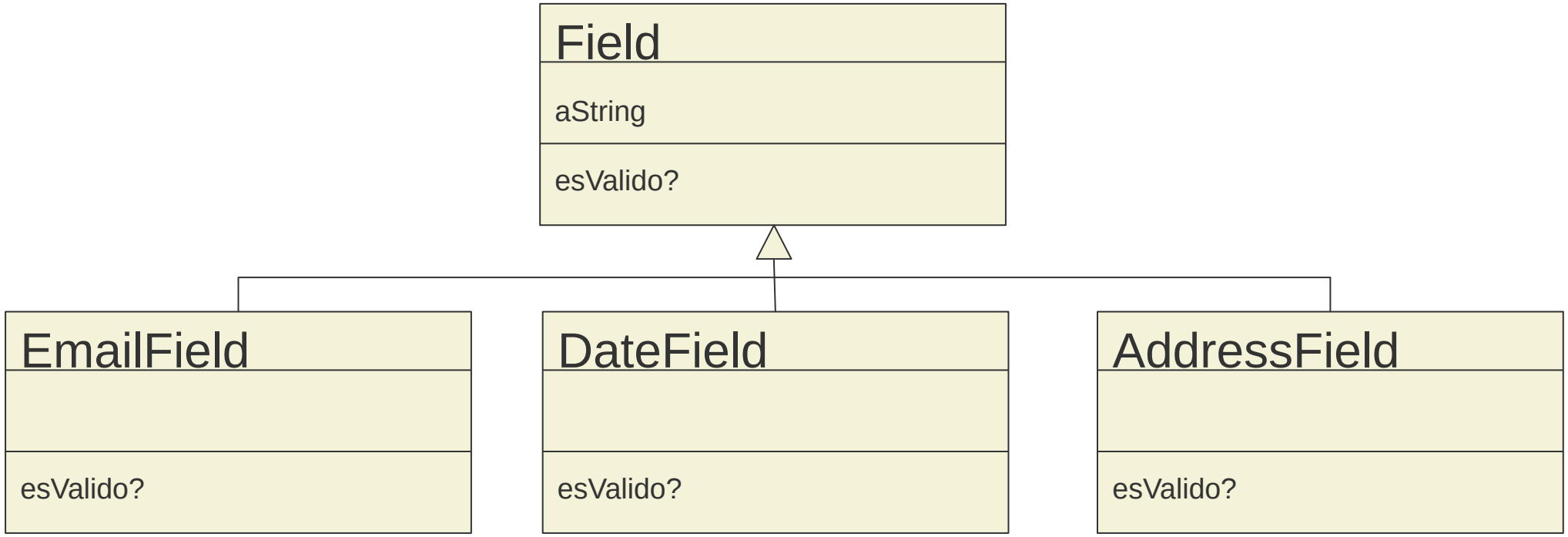
El código de validación depende de que cosa representa aString

```
esValido
 string isA email then self validateEmail
 string isA address then self validateAddress
 string isA  phone then self validatePhone
 .....
 .....
 .....
```

Problemas?

Otra Solución

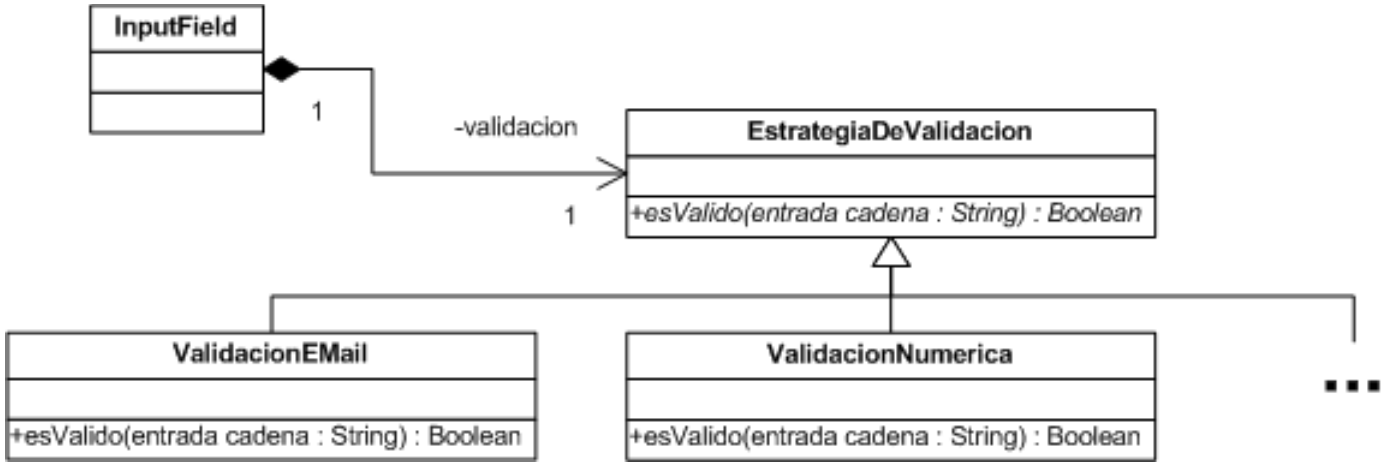
- ✓ Jerarquía de Clases de Input Fields



Problemas?

Validación de Strings

- ✓ Objetificar lo que varía. Sacarlo fuera del objeto (y de la clase, por supuesto)
- ✓ Solución: Encapsular el algoritmo de validación en un objeto.

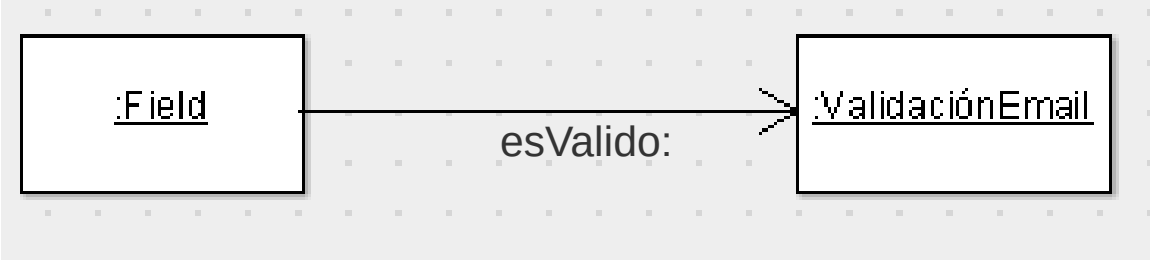


Como se ve una instancia de Field?

- ✓ Esta solución permite que cada instancia de la clase Field pueda tener un algoritmo de validación diferente

En Class Field

esValido
  validación esValido:string



En Cada EstrategiaDeValidación

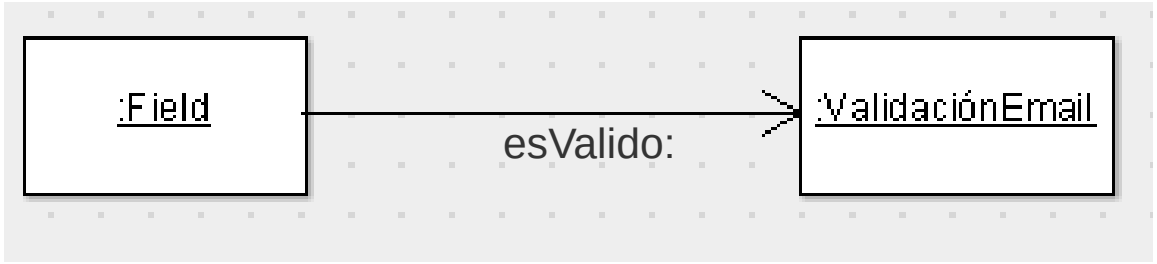
esValido: aString
  validar como corresponde

Entonces....

- A un objeto Field le tenemos que asignar un validador cuando lo creamos
- O sea, la inicialización de un Field consiste entre otras cosas en crear un objeto de una clase de EstrategiaDeValidación y “conectárselo” al Field
- De la misma manera, podemos cambiarle la Estrategia de Validación a un objeto Field en forma dinámica



Resumiendo



Cambiando el valor de la variable que conecta aField con el validador (apuntando a otro validador) cambiamos la estrategia de validación

Patrón Strategy

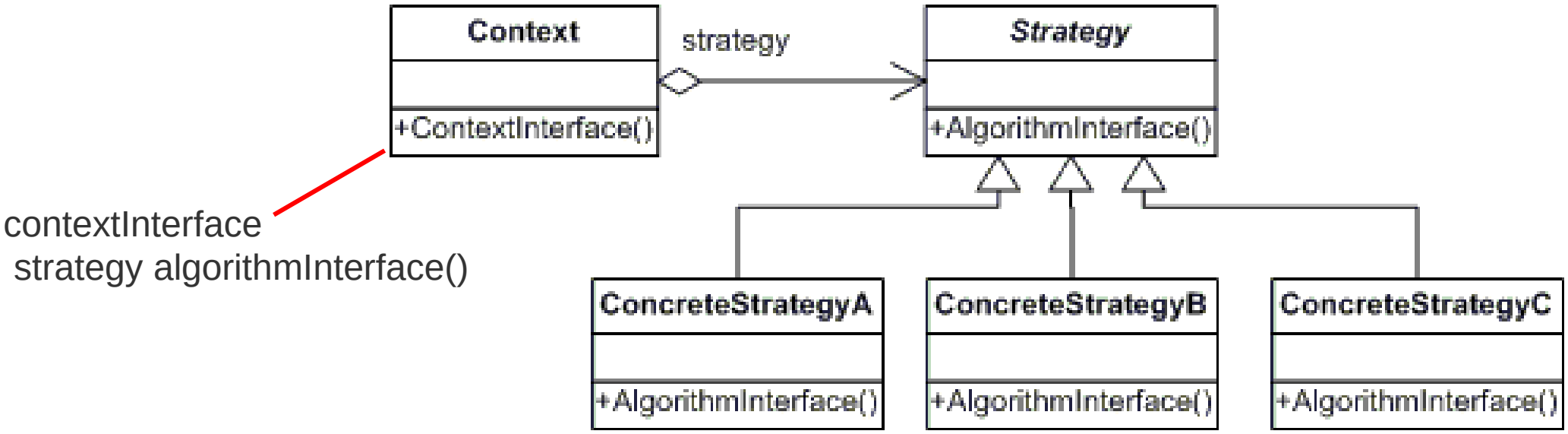
- **Intent:**
  - Desacoplar un algoritmo del objeto que lo utiliza.
  - Permitir cambiar el algoritmo que un objeto utiliza en forma dinámica.
  - Brindar flexibilidad para agregar nuevos algoritmos que lleven a cabo una función determinada.

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.
  - Es necesario cambiar el algoritmo en forma dinámica.

Patrón Strategy

✓ Structure



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.
- ✓ **Implementación**:
  - ✓ El contexto debe tener en su protocolo métodos que permitan cambiar la estrategia
  - ✓ Parámetros entre el contexto y la estrategia