PATRONES DE DISEÑO

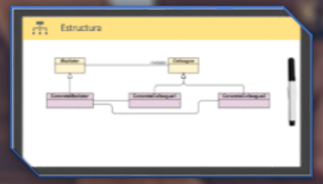
Patrones de Comportamiento - Mediator

PATRON DE COMPORTAMIENTO - MEDIATOR

Propósito

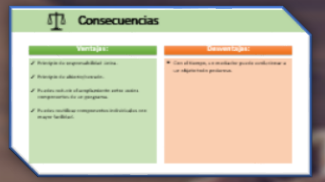
Problema

Solución

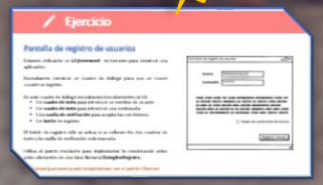


Estructura

Aplicabilidad



Consecuencias



Ejercicio

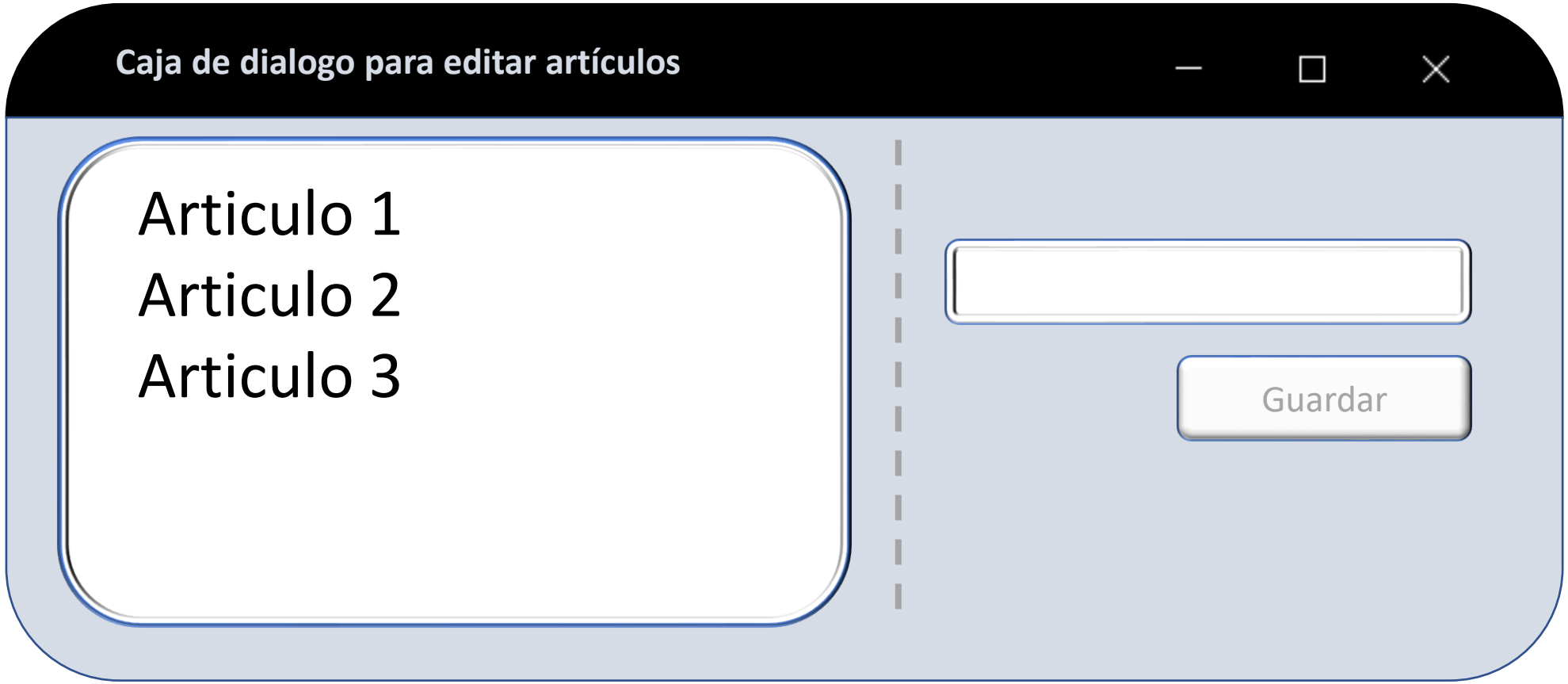
Tarea

Propósito

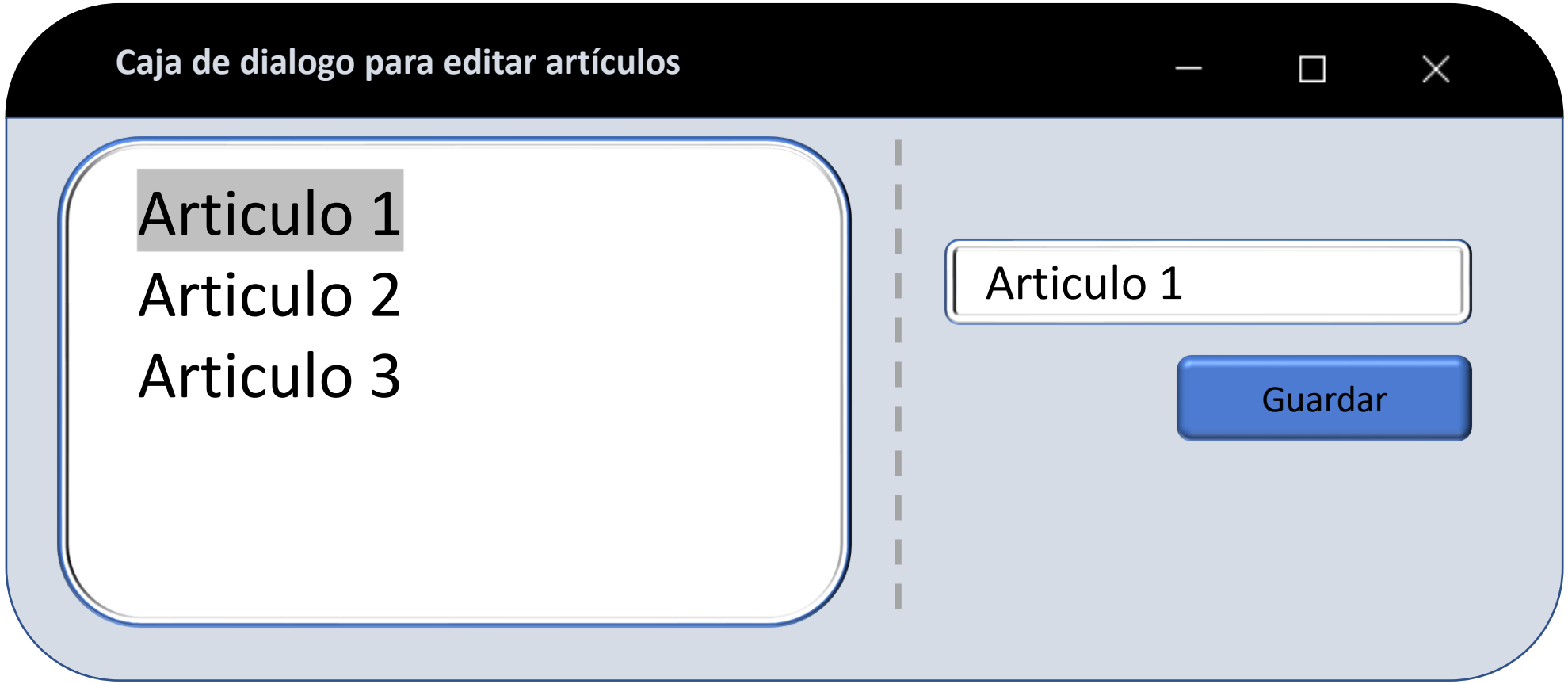
Define un objeto que encapsula la manera en cómo interactúan una serie de objetos.

Promueve un-bajo acoplamiento al evitar que los objetos se refieran unos a otros explícitamente, y permite variar la interacción entre ellos de forma independiente.

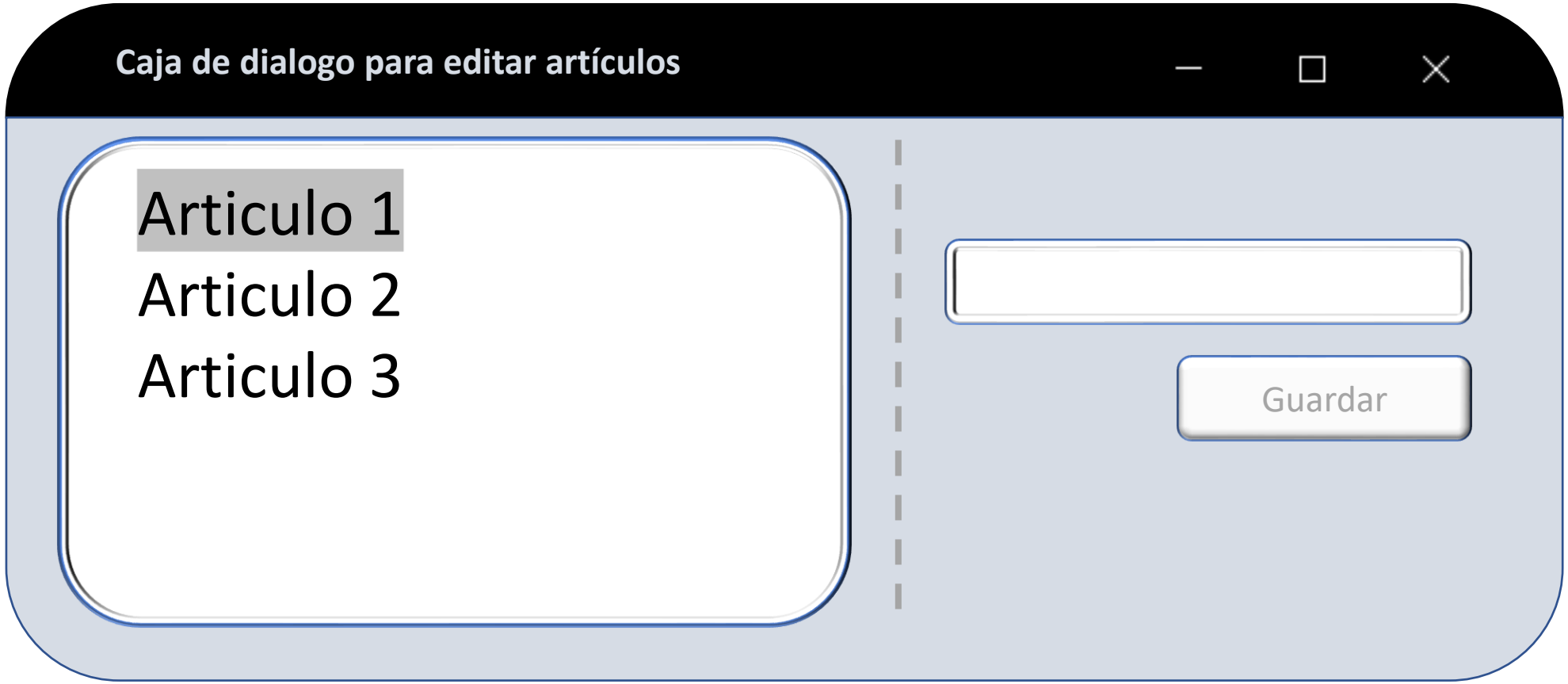
Problema


Caja de dialogo para editar artículos
Articulo 1
Articulo 2
Articulo 3
Guardar

Problema

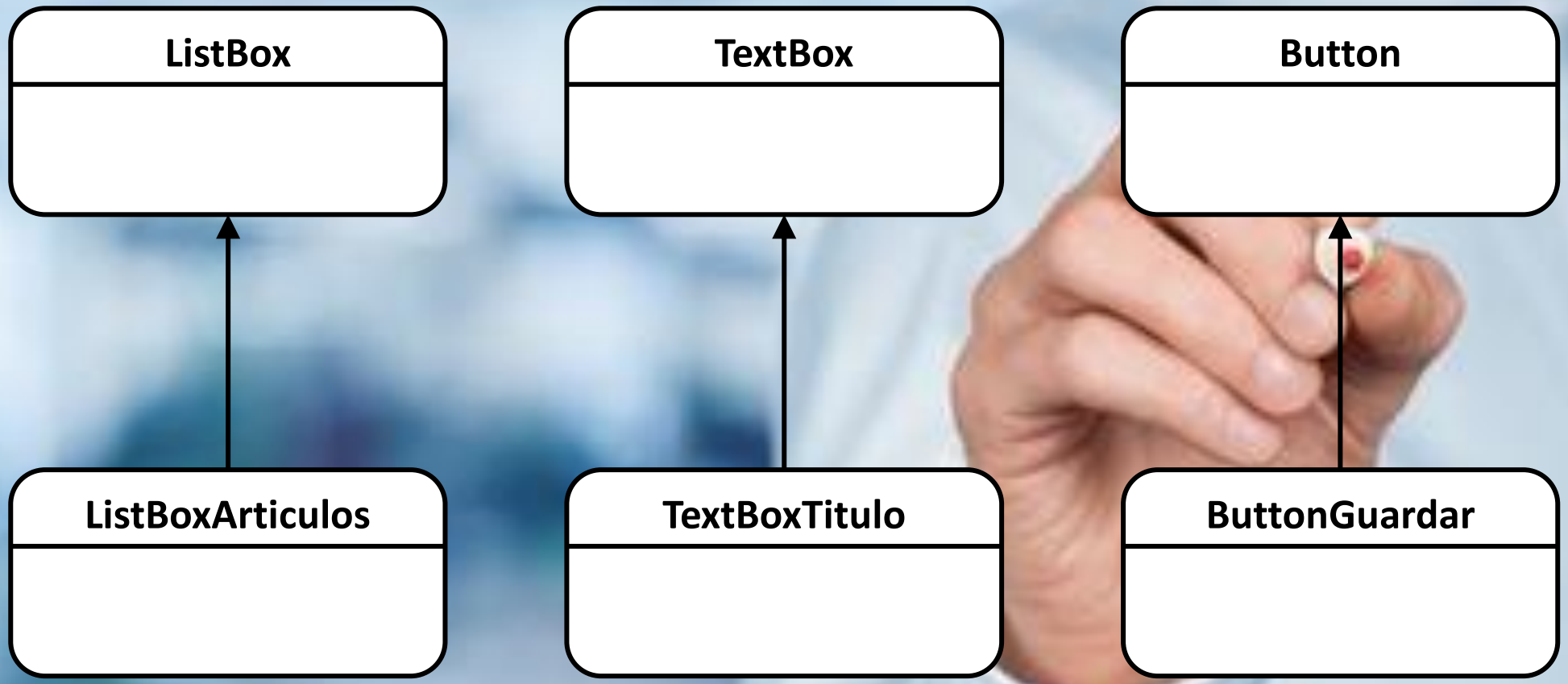

Caja de dialogo para editar artículos
Articulo 1
Articulo 2
Articulo 3
Articulo 1
Guardar

Problema

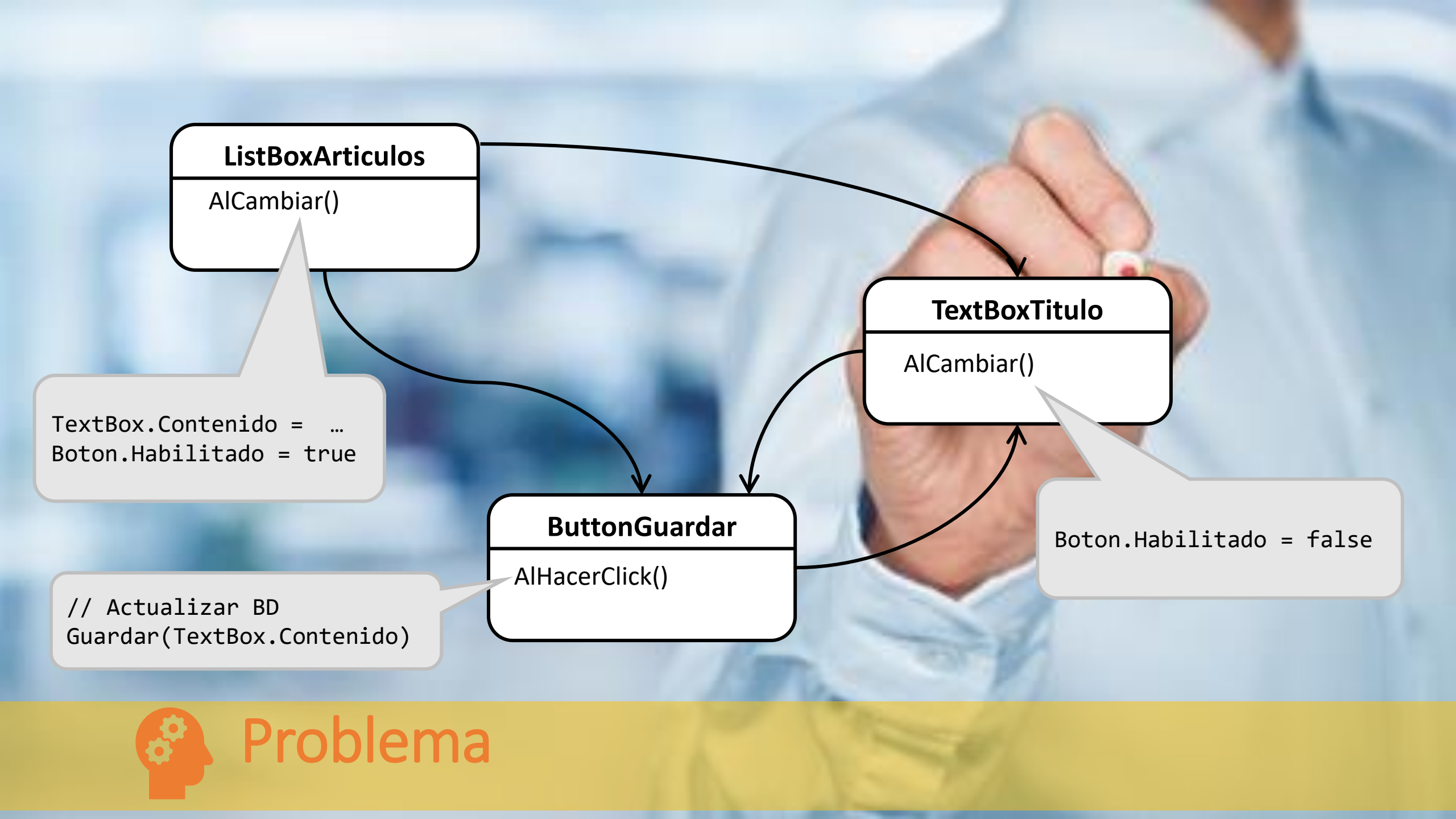

Caja de dialogo para editar artículos
Articulo 1
Articulo 2
Articulo 3
Guardar

Demo

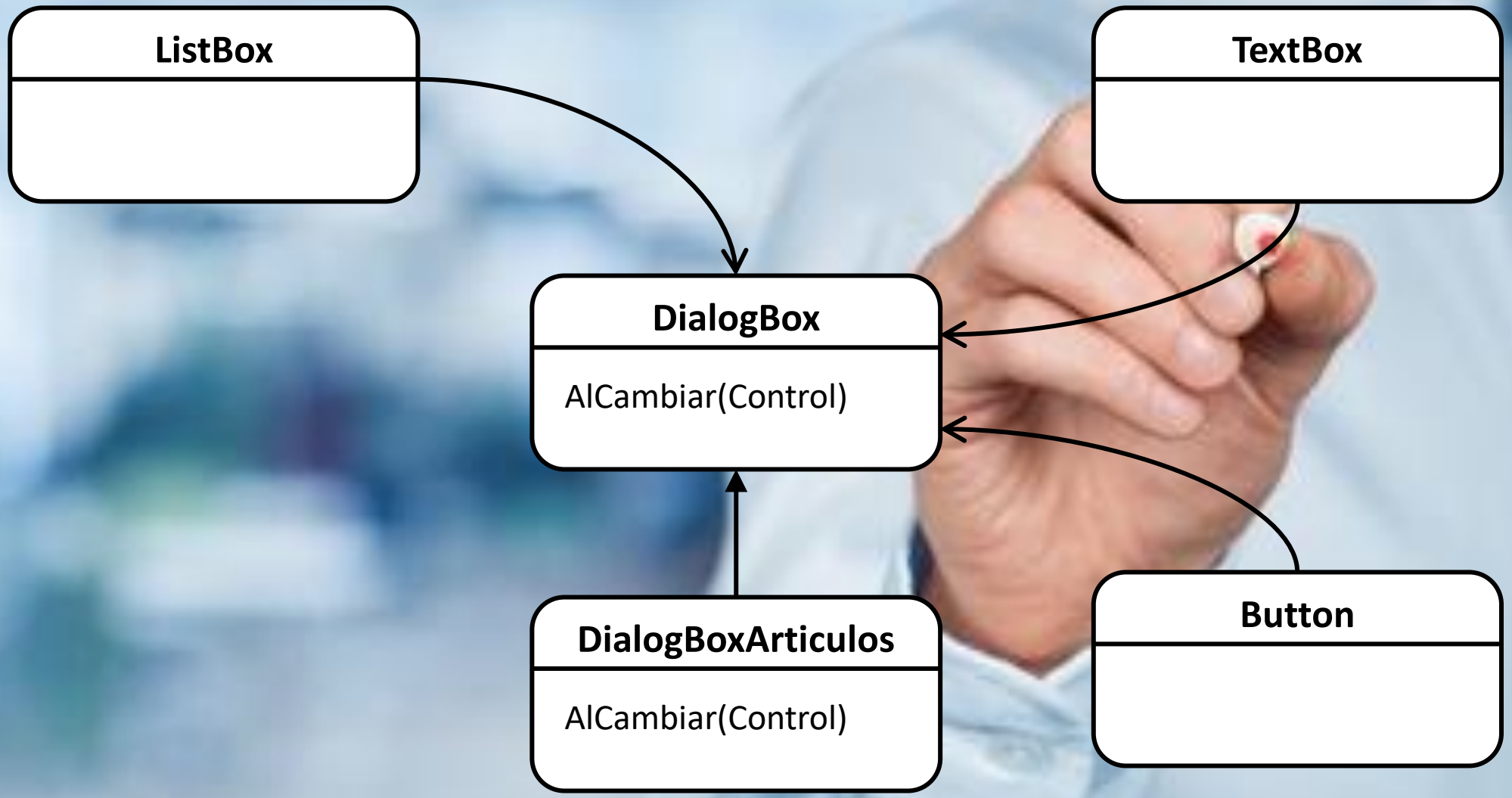
REVISEMOS EN CÓDIGO LA IMPLEMENTACIÓN DEL PROBLEMA


ListBox
TextBox
Button
ListBoxArticulos
TextBoxTitulo
ButtonGuardar

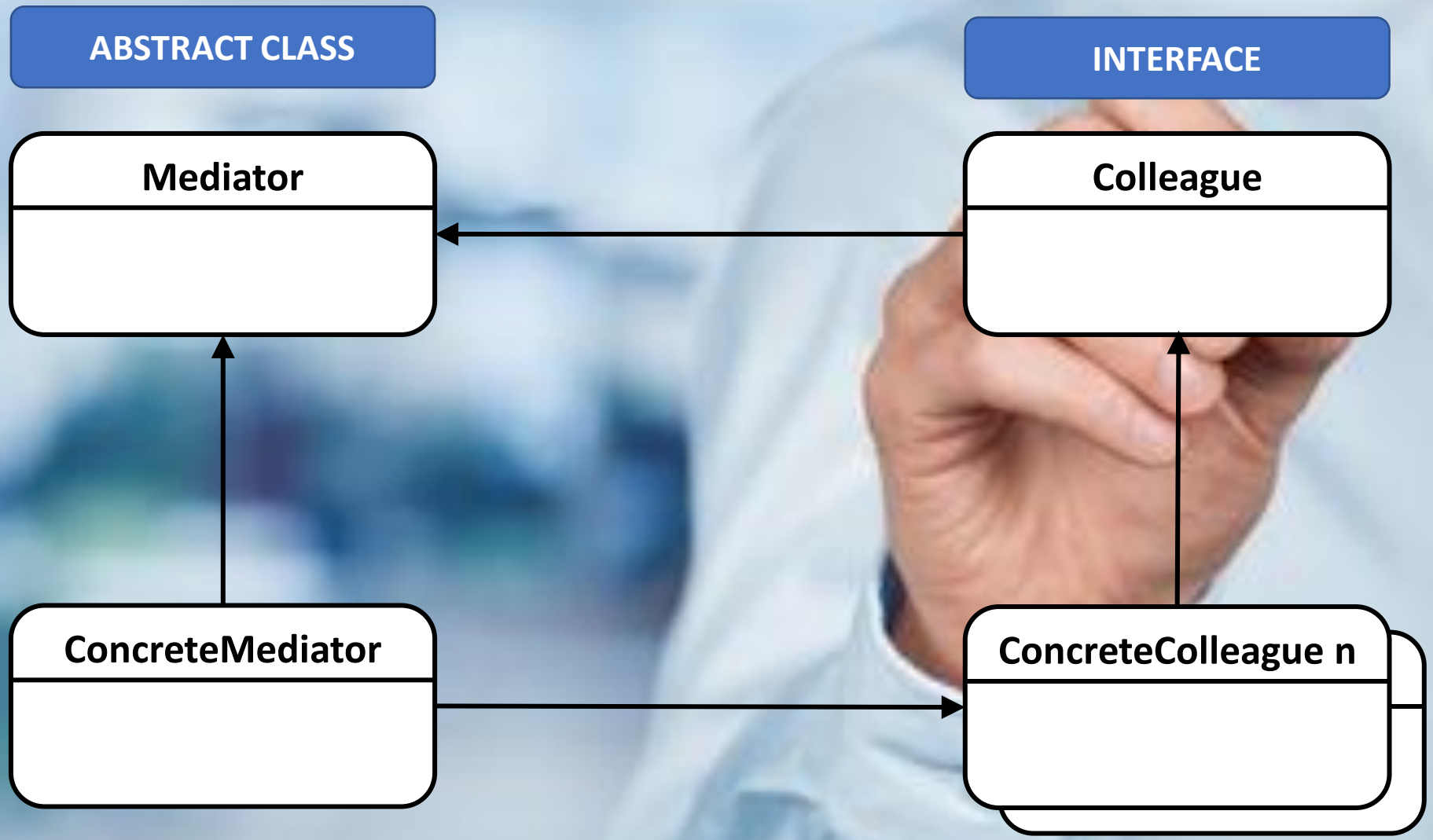
Problema


ListBoxArticulos
AlCambiar()
TextBox.Contenido = …
Boton.Habilitado = true
TextBoxTitulo
AlCambiar()
Boton.Habilitado = false
ButtonGuardar
AlHacerClick()
// Actualizar BD
Guardar(TextBox.Contenido)

Problema


ListBox
TextBox
DialogBox
AlCambiar(Control)
DialogBoxArticulos
AlCambiar(Control)
Button

Solución

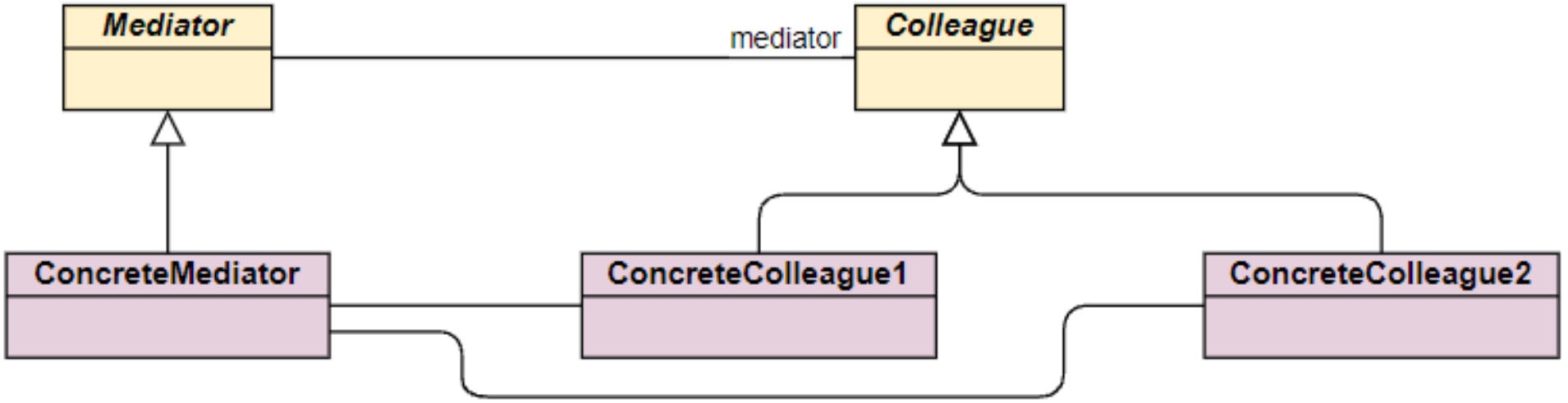

ABSTRACT CLASS
INTERFACE
Mediator
Colleague
ConcreteMediator
ConcreteColleague n

Solución

Demo

REVISEMOS EN CÓDIGO LA IMPLEMENTACIÓN DE LA SOLUCIÓN

Estructura


Mediator
mediator
Colleague
ConcreteMediator
ConcreteColleague1
ConcreteColleague2

Aplicabilidad

Cuándo utilizar este patrón?

Utiliza el patrón Mediator cuando resulte difícil cambiar algunas de las clases porque están fuertemente acopladas a un puñado de otras clases.

Utiliza el patrón cuando no puedas reutilizar un componente en un programa diferente porque sea demasiado dependiente de otros componentes.

Utiliza el patrón Mediator cuando te encuentres creando cientos de subclases de componente sólo para reutilizar un comportamiento básico en varios contextos.

Consecuencias

Ventajas:

- ✓ Principio de responsabilidad única.
- ✓ Principio de abierto/cerrado.
- ✓ Puedes reducir el acoplamiento entre varios componentes de un programa.
- ✓ Puedes reutilizar componentes individuales con mayor facilidad.

Desventajas:

- ✗ Con el tiempo, un mediador puede evolucionar a un objeto todo poderoso.

Ejercicio

Pantalla de registro de usuarios

Estamos utilizando un ***UI-framework*** de terceros para construir una aplicación.

Necesitamos construir un cuadro de diálogo para que un nuevo usuario se registre.

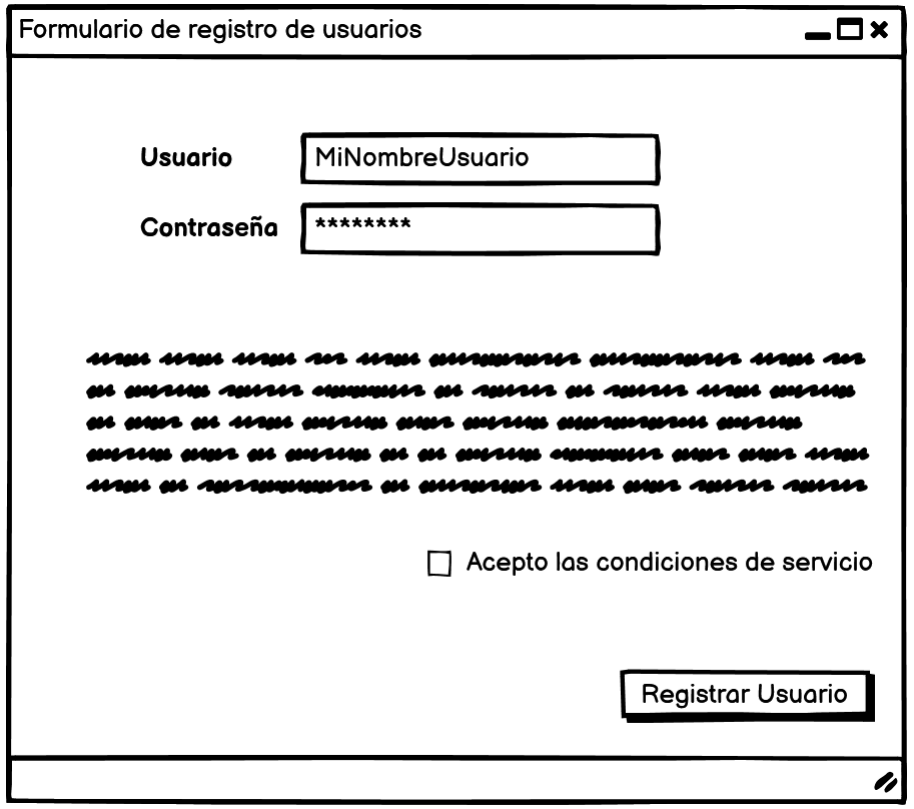
En este cuadro de diálogo necesitamos tres elementos de UI:

- Un **cuadro de texto** para introducir un nombre de usuario
- Un **cuadro de texto** para introducir una contraseña
- Una **casilla de verificación** para aceptar las condiciones
- Un **botón** de registro

El botón de registro sólo se activa si se rellenan los dos cuadros de texto y la casilla de verificación está marcada.

Utiliza el patrón mediador para implementar la coordinación entre estos elementos en una clase llamada **DialogBoxRegistro**.

Plus: *Investigue como puede complementar con el patrón Observer*



Demo

REVISEMOS EN CÓDIGO LA IMPLEMENTACIÓN LA PRÁCTICA

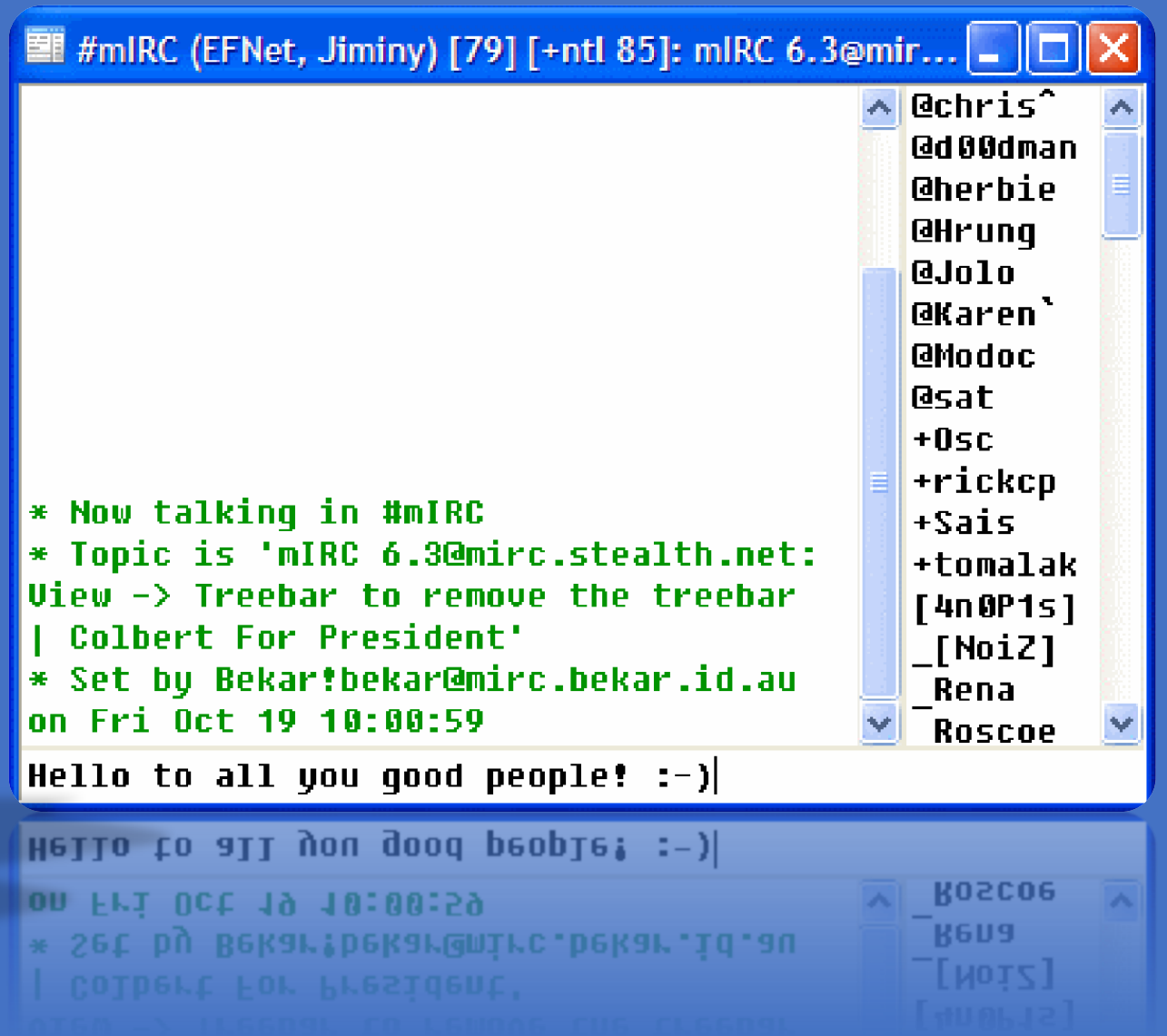
Tarea

Sala de Chat estilo mIRC

Debemos desarrollar una aplicación que represente una sala de chat similar a las que se utilizaban antes Ej: mIRC

Hagámoslo utilizando el patrón Mediator

Tip: La sala de chat es el mediador y los usuarios los colegas



Muchas Gracias…

los espero la siguiente clase