

The background image shows a close-up of people's hands working on architectural blueprints spread out on a wooden table. One person is pointing at a specific section of the plan, while another is holding a pen, ready to draw or mark. A white hard hat is visible on the left side of the frame. The scene is lit with warm, golden light, creating a professional and collaborative atmosphere. A semi-transparent rectangular box is centered over the image, containing the title and subtitle text.

# PATRONES DE DISEÑO

Patrones de Comportamiento - Mediator

# PATRON DE COMPORTAMIENTO - MEDIATOR



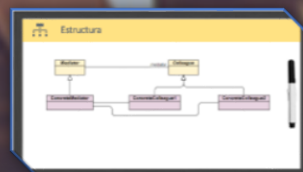
Propósito



Problema



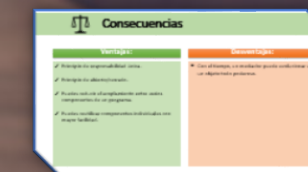
Solución



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



# Problema

Caja de dialogo para editar artículos

Artículo 1  
Artículo 2  
Artículo 3

Artículo 1

Guardar



# Problema

Caja de dialogo para editar artículos

Artículo 1  
Artículo 2  
Artículo 3

Guardar

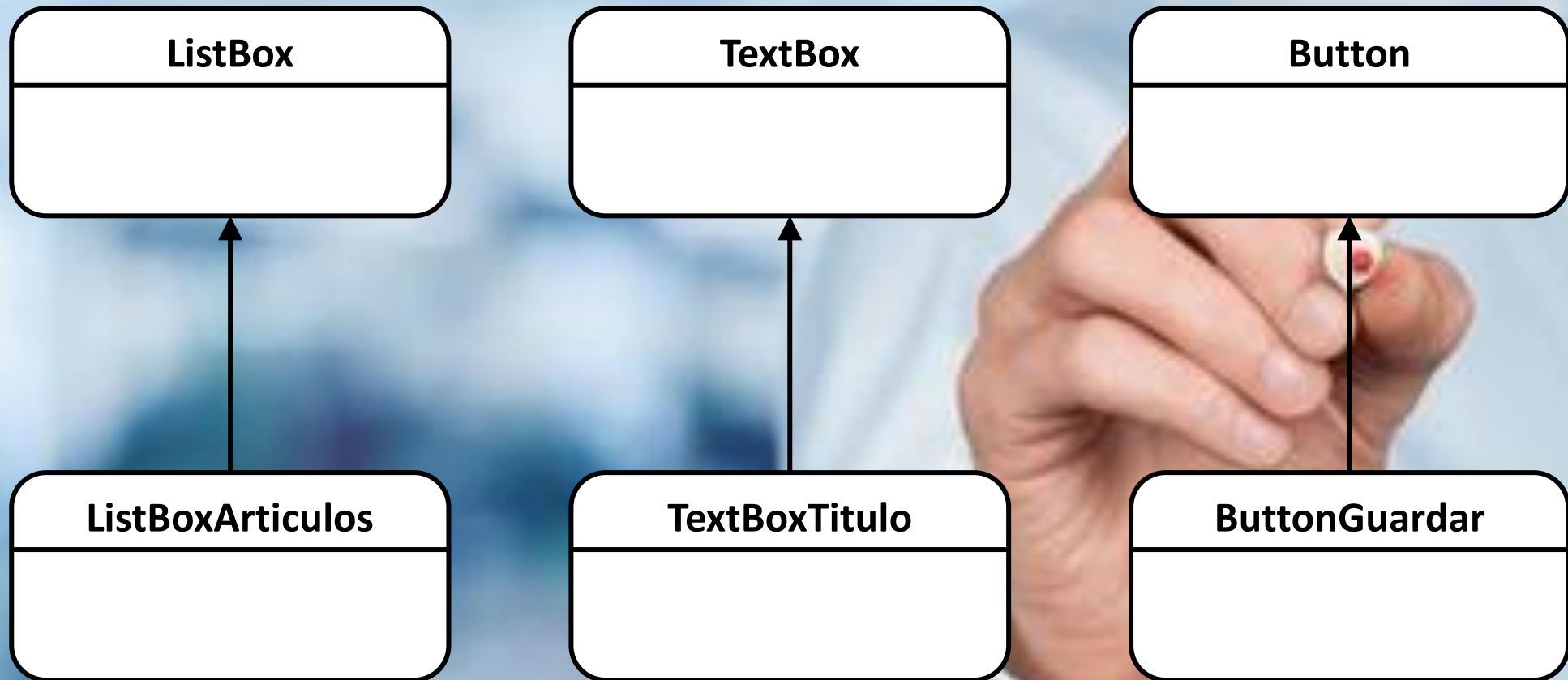


# Demo

REVISEMOS EN CÓDIGO LA IMPLEMENTACIÓN DEL PROBLEMA

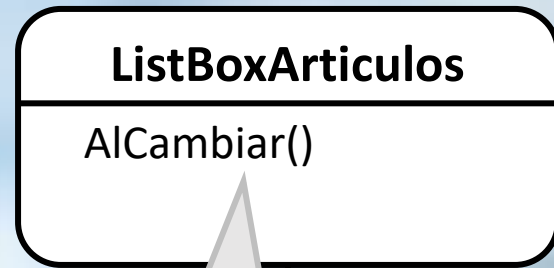
```
val getTokNum = fn : string -> int
val getTokSym = fn : string -> token
val discardL = fn : TextIO.instream -> unit
val tokenizeInput = fn : TextIO.instream -> token list
val nextToken = fn : TextIO.instream -> token
val it = () : unit
val ntk = ref TK_NONE : token ref
exception BigError of string
val parse_factor = fn : 'a -> token
val is_unaryop = fn : token -> bool
val parse_unaryop = fn : 'a -> token
val parse_unary = fn : TextIO.instream -> token
val is_multop = fn : token -> bool
val parse_multop = fn : 'a -> token
val parse_term = fn : TextIO.instream -> token
val is_addop = fn : token -> bool
val parse_addop = fn : 'a -> token
val parse_simple = fn : TextIO.instream -> token
val is_relop = fn : token -> bool
val parse_relop = fn : 'a -> token
val parse_boolop = fn : TextIO.instream -> token
val is_boolop = fn : token -> bool
val parse_boolop = fn : 'a -> token
val parse_expression = fn : TextIO.instream -> token
val parse_program = fn : 'a -> token
val parse = fn : string -> token
```

```
ntk := nextToken file
else
  raise BigError("Expected 'lparen'.")
else if Intk = TK_ID then
  (ntk := nextToken file; (* Skip TK_ID *)
   if Intk = TK_
    unaryop tok = (tok = TK_NOT);
  parse_unaryop file =
    if (is_unaryop (Intk)) then
      Intk
    else
      raise BigError("Expected 'unaryop'.");
  parse_unary -> {unaryop}_opt factor
  if (is_unaryop (Intk)) then
    (parse_unaryop file;
     ntk := nextToken file;
     parse_factor file)
  if (is_unaryop (Intk)) then
    (parse_factor file;
     ntk := nextToken file;
     parse_factor file)
  tok = ((tok = TK_TIMES) orelse (tok = TK_DIV))
```



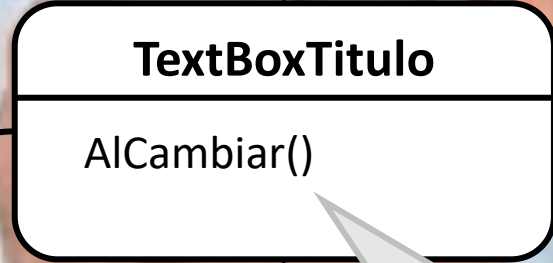
Problema



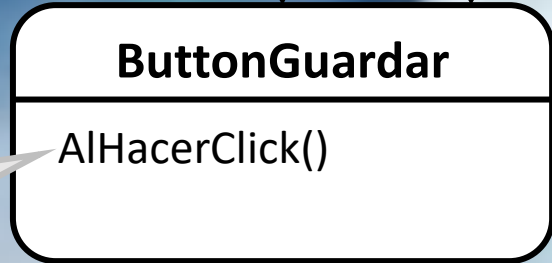


TextBox.Contenido = ...  
Boton.Habilitado = true

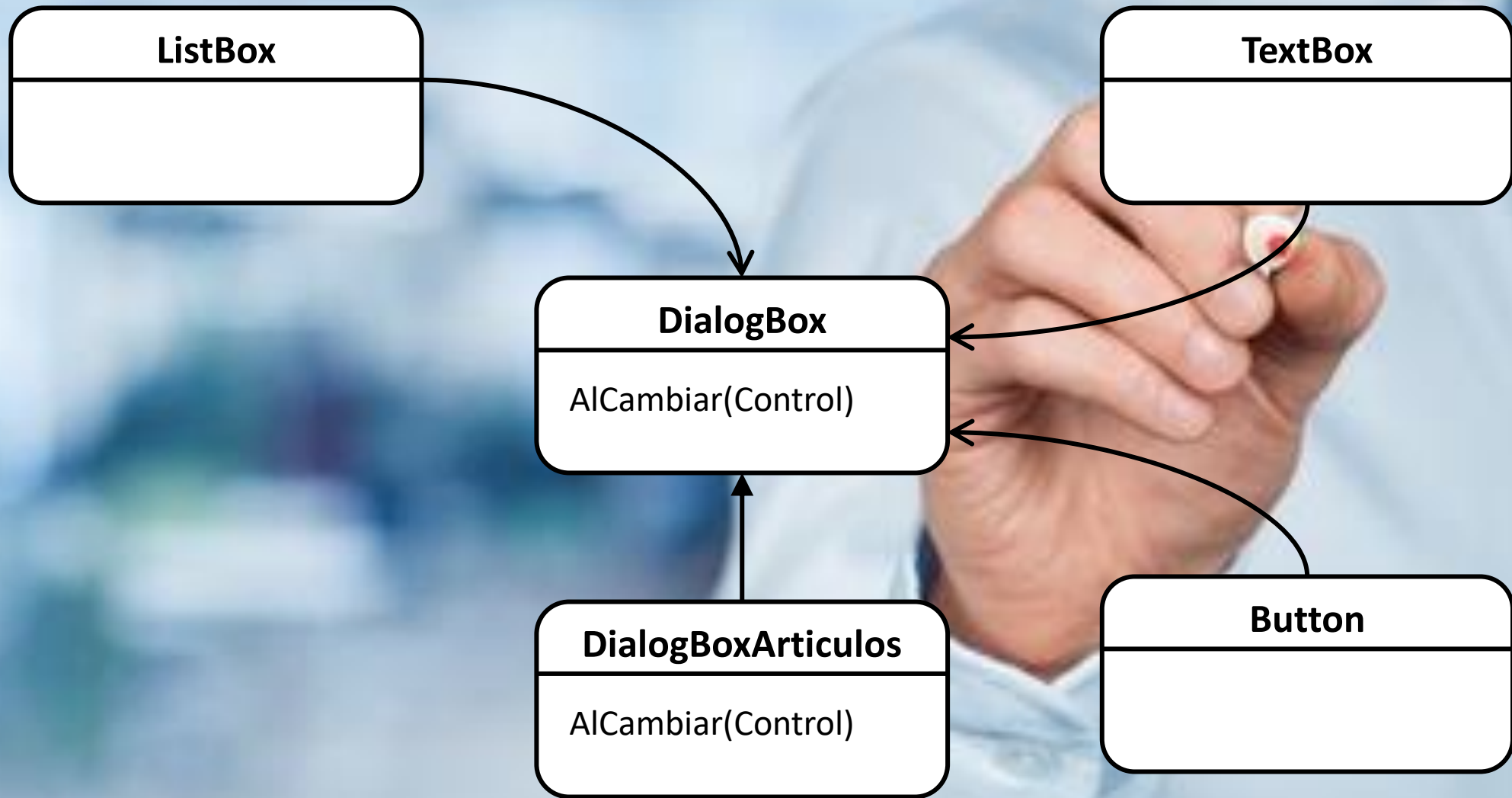
// Actualizar BD  
Guardar(TextBox.Contenido)



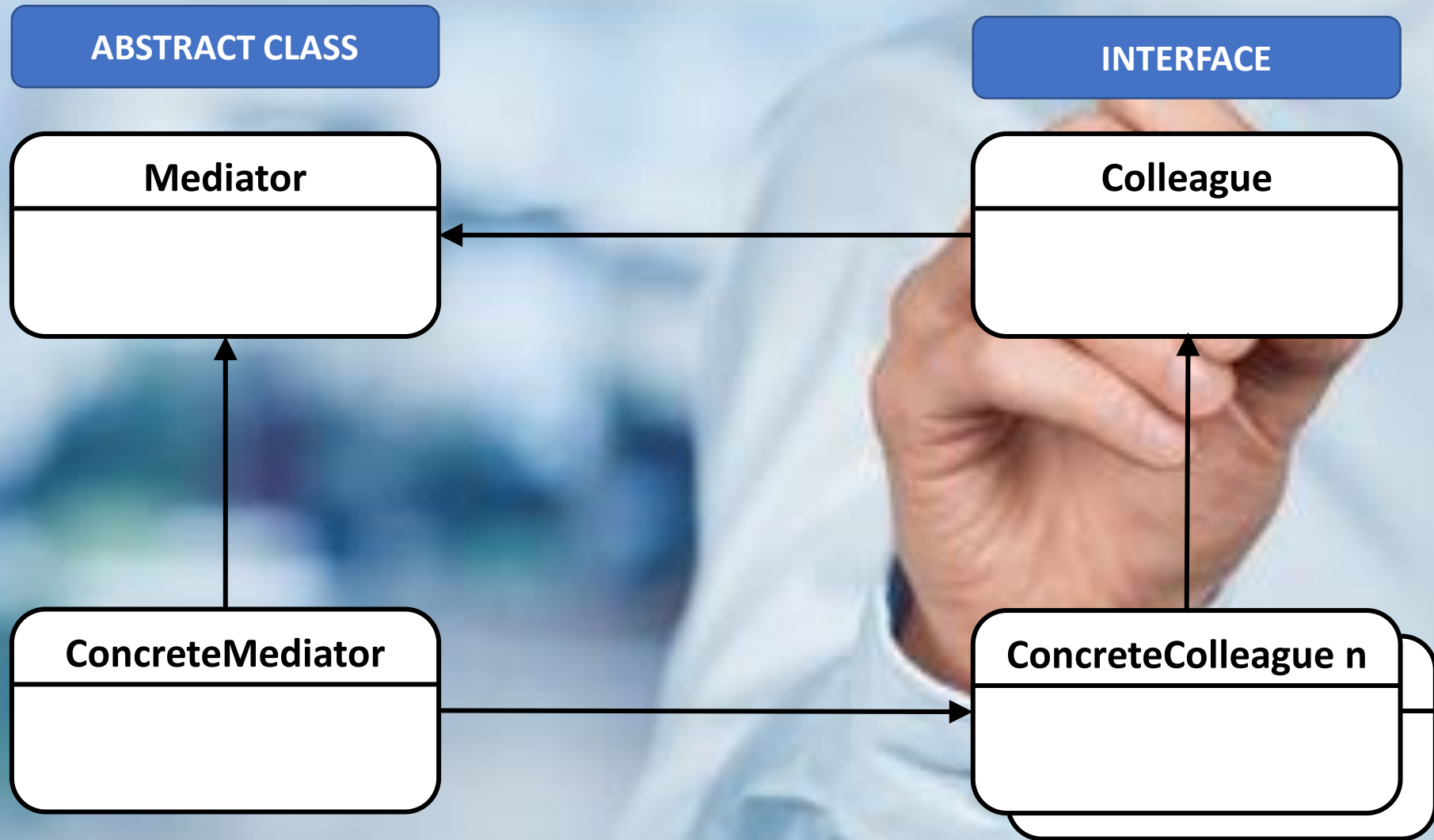
Boton.Habilitado = false



Problema



**Solución**



Solución



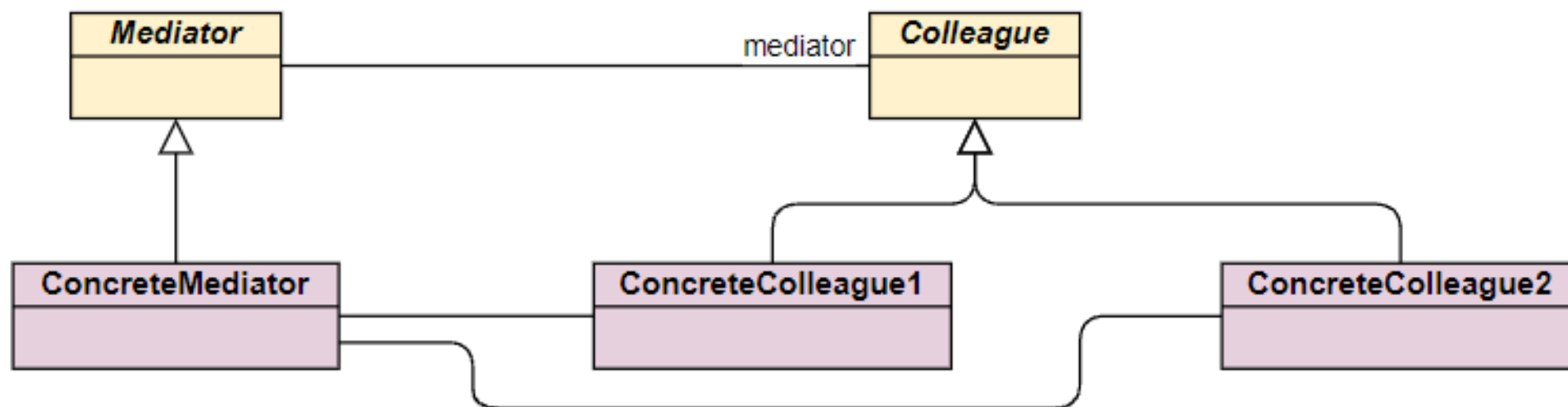
```
def operation == "MIRROR_X":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = True  
    mirror_mod.use_z = False  
elif operation == "MIRROR_Z":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = False  
    mirror_mod.use_z = True  
  
#selection at the end -add back the deselected mirror modifier  
mirror_ob.select= 1  
modifier_ob.select=1  
bpy.context.scene.objects.active = modifier_ob  
print("Selected" + str(modifier_ob)) # modifier ob is the active ob  
mirror_ob.select = 0  
bpy.context.selected_objects[0]  
bpy.data.objects[mirror_ob.name].select = 1
```

## Demo

REVISEMOS EN CÓDIGO LA IMPLEMENTACIÓN DE LA SOLUCIÓN



# Estructura





# 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*

Formulario de registro de usuarios

Usuario

Contraseña

[Texto ilegible]

☐ Acepto las condiciones de servicio



# Demo

REVISEMOS EN CÓDIGO LA IMPLEMENTACIÓN LA PRÁCTICA

```
4 from calendar import calendar
5 import math
6 import pytz
7 import io, base64
8
9
10 class AdmissionExtensionOnlineController(http.Controller):
11
12     @http.route('/get/type_wise_program', website=True, auth='none',
13               def type_wise_program(self, **kwargs):
14                 if len(kwargs['types']) <= 0:
15                     return "None"
16
17                 types = kwargs['types']
18                 program_list = []
19                 domain = []
20
21                 if types == 'local_bachelor_program_hsc':
22                     domain = [('course_id.is_local_bachelor_program_hsc', '=')]
23                 elif types == 'local_bachelor_program_a_level':
24                     domain = [('course_id.is_local_bachelor_program_a_level', '=')]
25                 elif types == 'local_bachelor_program_diploma':
26                     domain = [('course_id.is_local_bachelor_program_diploma', '=')]
27                 elif types == 'local_masters_program_bachelor':
28                     domain = [('course_id.is_local_masters_program_bachelor', '=')]
29                 elif types == 'international_bachelor_program':
30                     domain = [('course_id.is_international_bachelor_program', '=')]
31                 elif types == 'international_masters_program':
32                     domain = [('course_id.is_international_masters_program', '=')]
33
34                 domain = [domain]
```





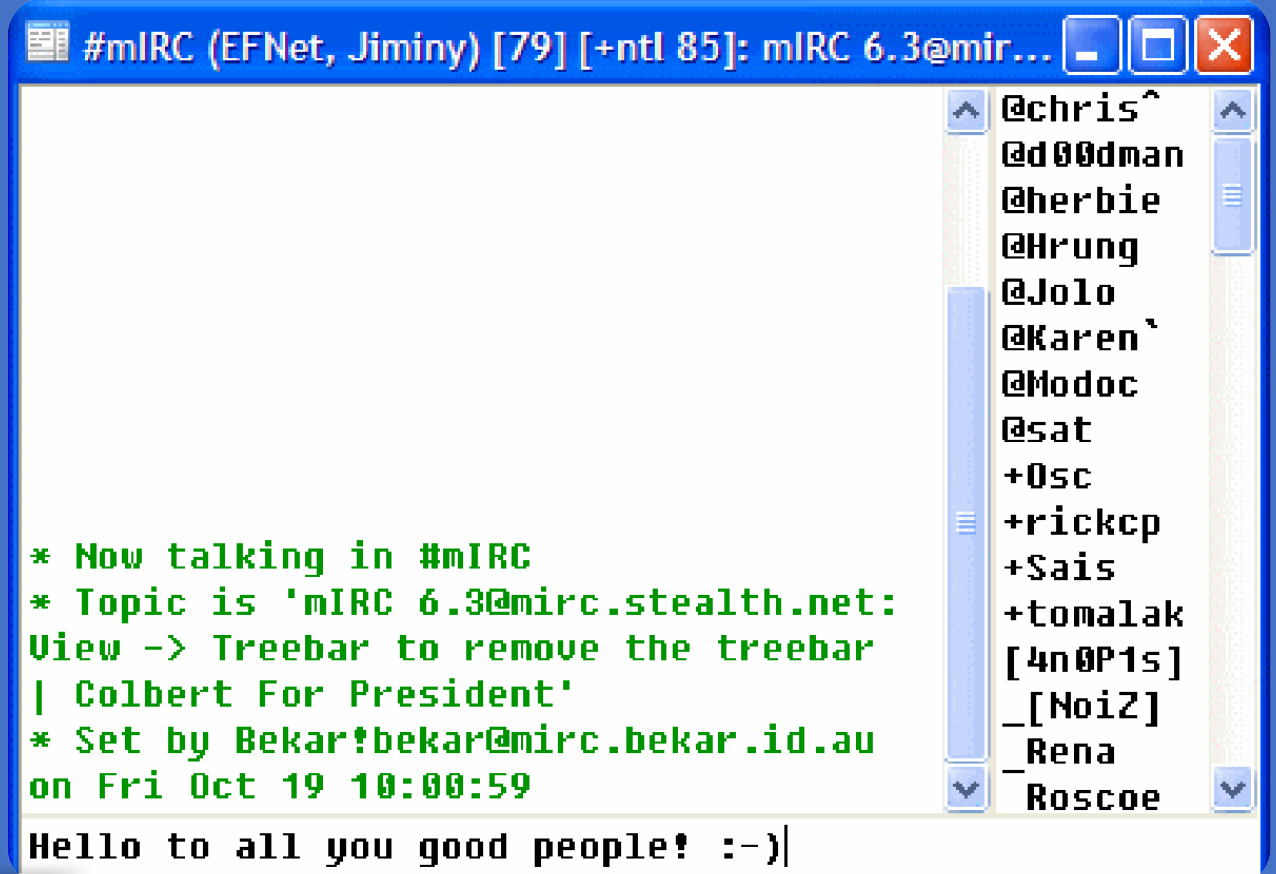
# Tarea

## 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**



