

The background image shows a collaborative work environment. In the foreground, a person's hand is pointing at a detailed architectural floor plan spread across a wooden table. To the right, another person's hand is holding a black pen, poised to write on the plan. On the left side of the frame, a white hard hat is partially visible. In the background, other individuals are working, and a laptop is open on a desk. The scene is lit with warm, natural light, creating a professional and focused atmosphere.

PATRONES DE DISEÑO

Patrones de Comportamiento –
Chain of Responsibility

PATRON DE COMPORTAMIENTO - MEDIATOR



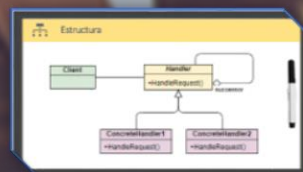
Propósito



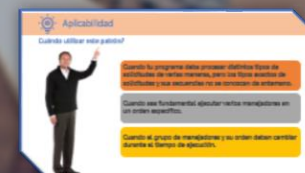
Problema



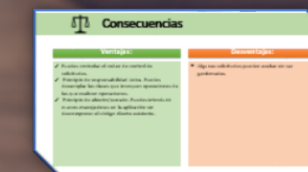
Solución



Estructura



Aplicabilidad



Consecuencias

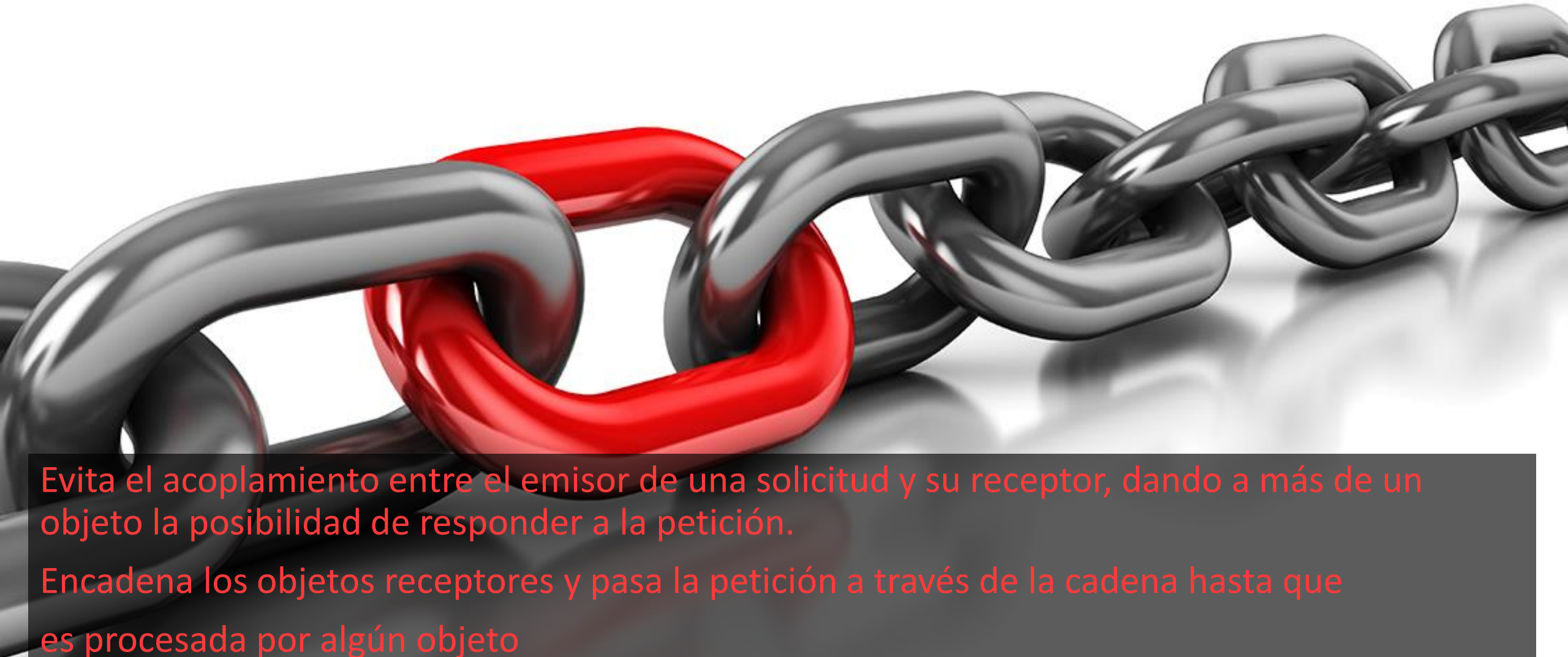


Ejercicio



Tarea



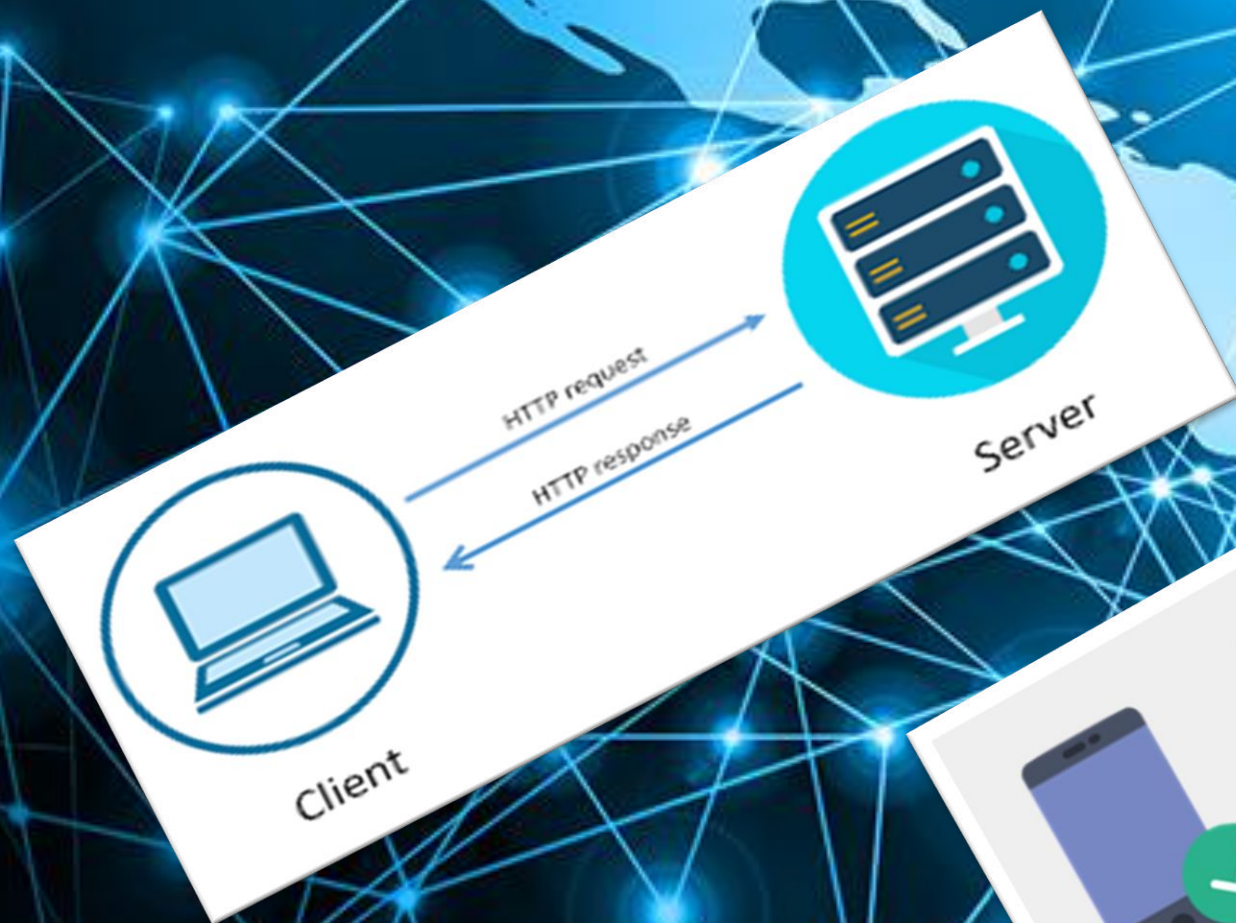


Evita el acoplamiento entre el emisor de una solicitud y su receptor, dando a más de un objeto la posibilidad de responder a la petición.

Encadena los objetos receptores y pasa la petición a través de la cadena hasta que es procesada por algún objeto



Problema



Demo

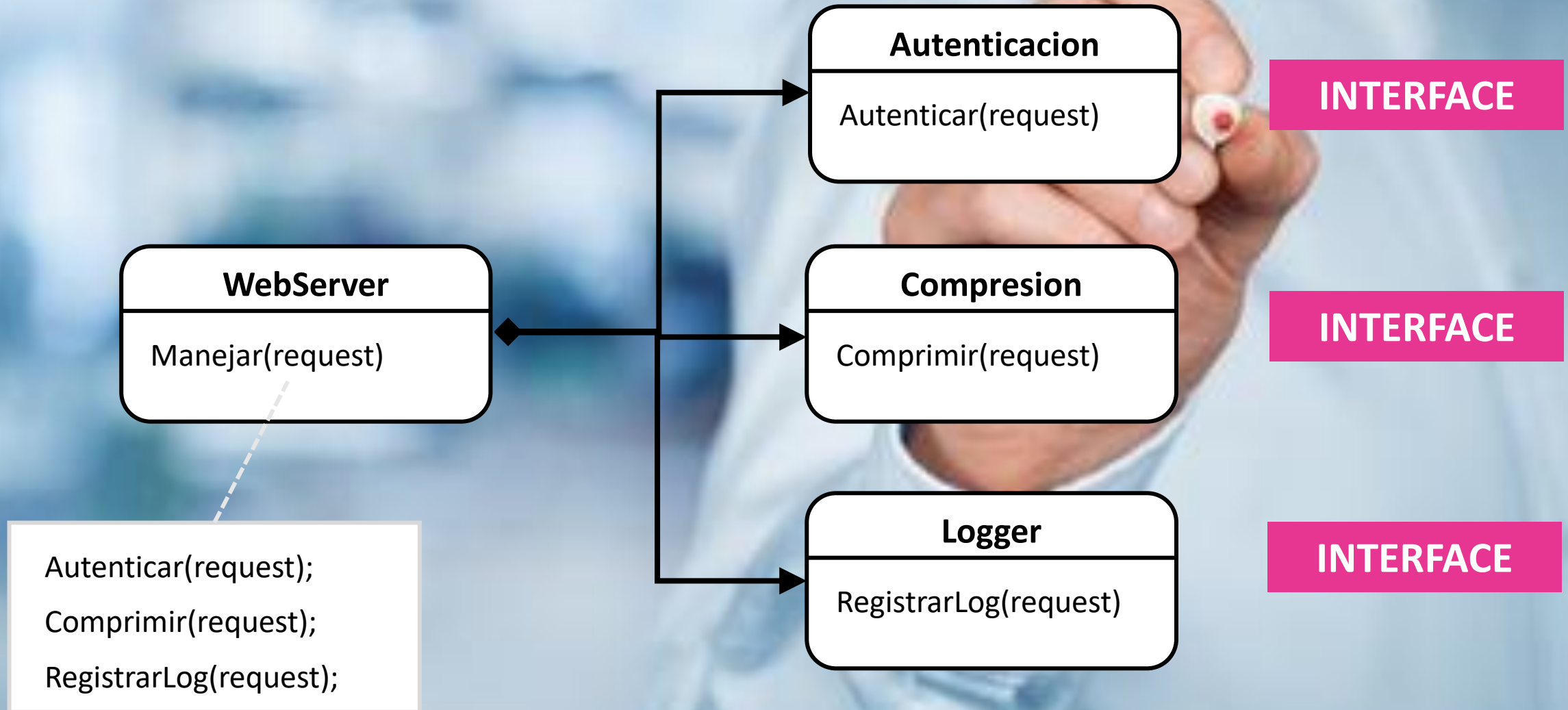
REVISEMOS EN CÓDIGO LA IMPLEMENTACIÓN DEL PROBLEMA

```
val getTokNum = fn : string -> int
val getTokSym = fn : string -> token
val discardL = fn : TextIO.instream -> unit
val tokenizeInput = fn : TextIO.instream -> token
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_
    then parse_unaryop tok = (tok = TK_NOT);
    else 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))
```

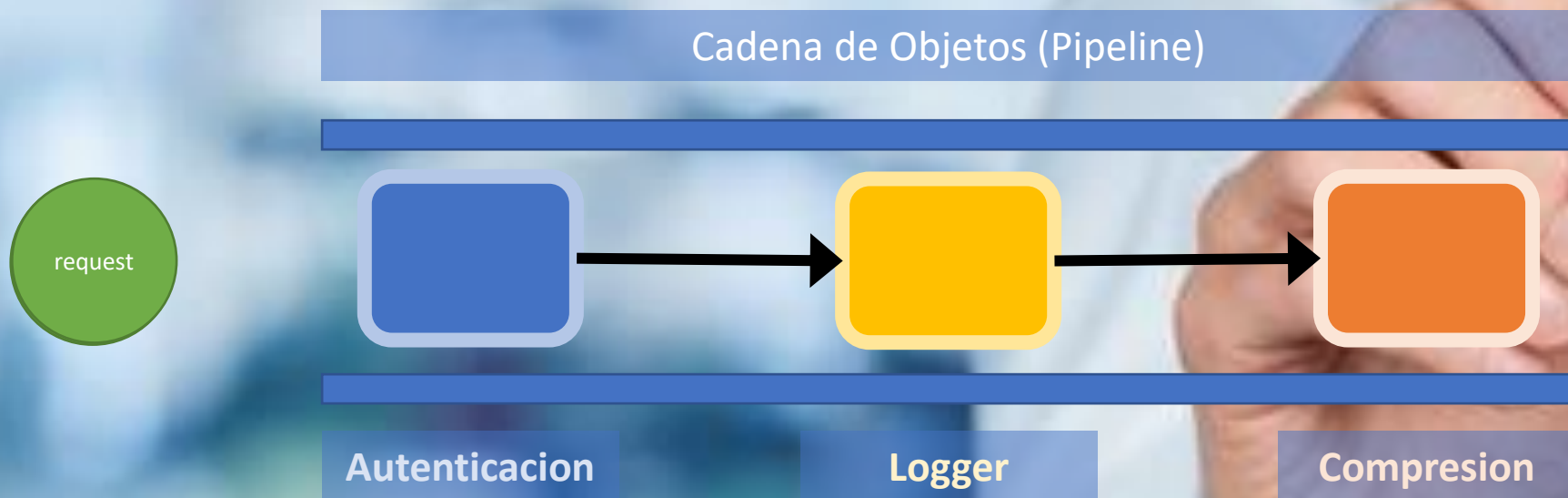


Solución





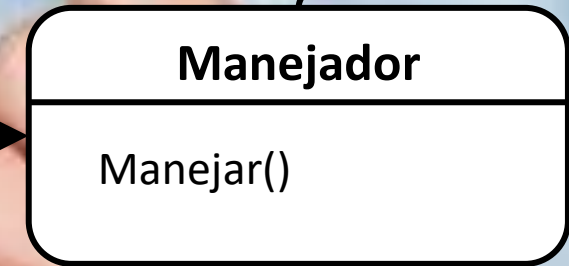
Solución



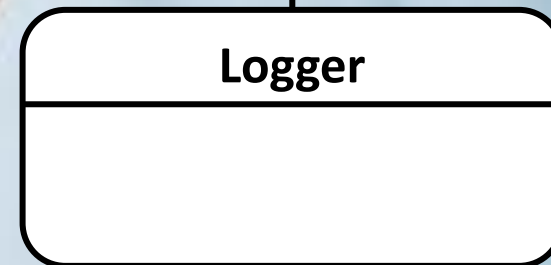
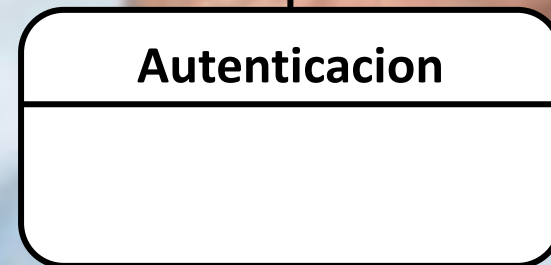
ABSTRACT CLASS



`manejador.Manejar(request);`



Siguiente



Solución

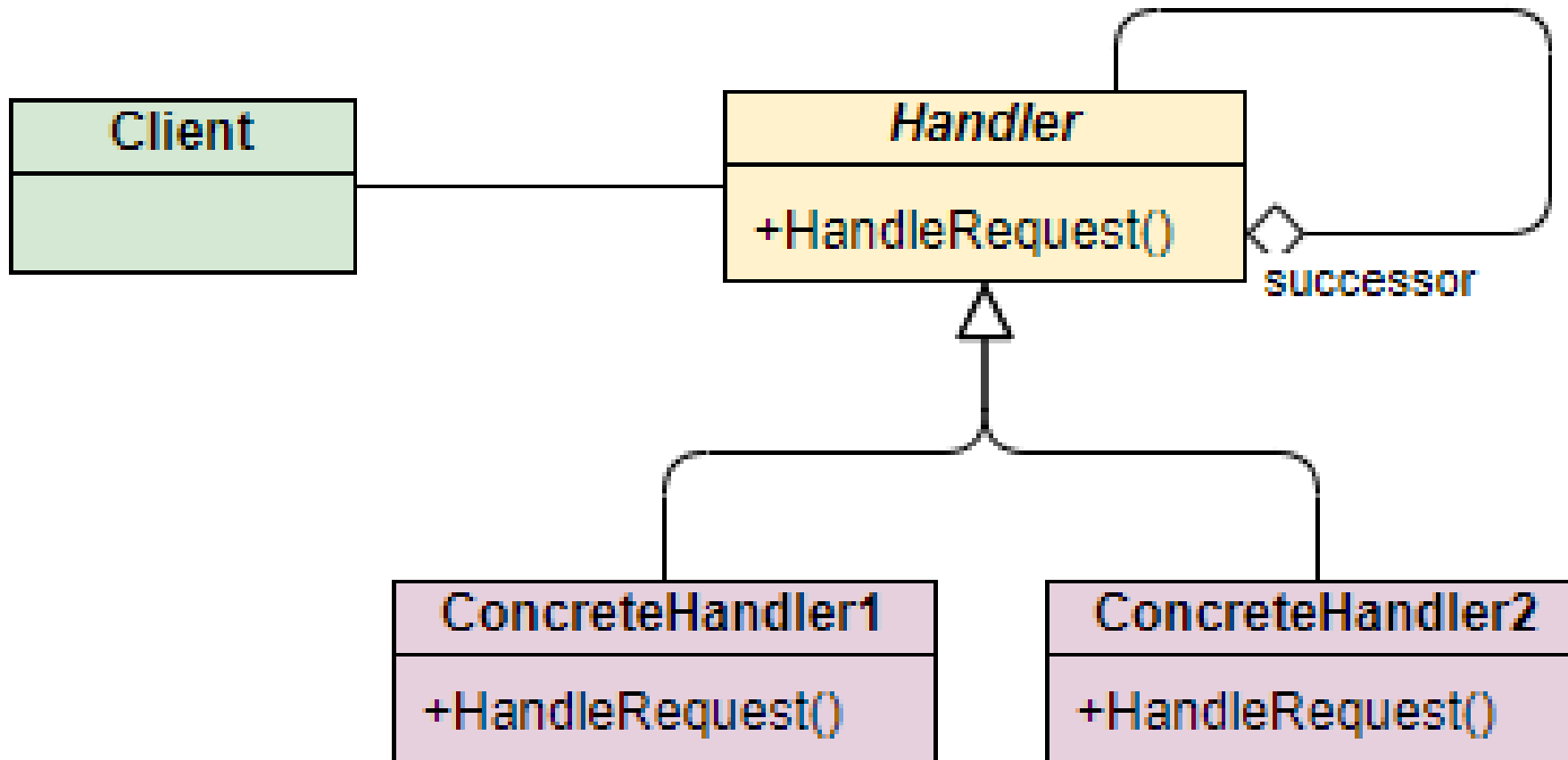

```
def operation == "MIRROR_X":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = True  
    mirror_mod.use_z = False  
elif operation == "MIRROR_Z":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = False  
    mirror_mod.use_z = True  
  
#selection at the end -add back the deselected 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  
obj = bpy.context.selected_objects[0]  
obj.data.modifiers[mirror_ob.name].select = 1
```

Demo

REVISEMOS EN CÓDIGO LA IMPLEMENTACIÓN DE LA SOLUCIÓN



Estructura





Estructura

Una estructura de objetos típica podría parecerse a esta:





Aplicabilidad

Cuándo utilizar este patrón?



Cuando tu programa deba procesar distintos tipos de solicitudes de varias maneras, pero los tipos exactos de solicitudes y sus secuencias no se conozcan de antemano.

Cuando sea fundamental ejecutar varios manejadores en un orden específico.

Cuando el grupo de manejadores y su orden deban cambiar durante el tiempo de ejecución.



Consecuencias

Ventajas:

- ✓ Puedes controlar el orden de control de solicitudes.
- ✓ Principio de responsabilidad única. Puedes desacoplar las clases que invoquen operaciones de las que realicen operaciones.
- ✓ Principio de abierto/cerrado. Puedes introducir nuevos manejadores en la aplicación sin descomponer el código cliente existente.

Desventajas:

- ✗ Algunas solicitudes pueden acabar sin ser gestionadas.



Ejercicio

Aprobar pedidos según el cargo

Utilizar el patrón de la cadena de responsabilidad para simular el proceso de aprobación de pedidos en una organización.

Para el ejemplo tenemos una estructura organizacional en el que varios gerentes y ejecutivos vinculados pueden responder a una solicitud de compra o pasarla a un superior.

Cada puesto puede tener su propio conjunto de reglas sobre los pedidos que pueden aprobar.

TIPS:

- *Cada pedido debe tener atributos para el numero de pedido, descripción del producto y monto*
- *Diseñar el conjunto de reglas tomando en cuenta el monto a ser aprobado.*



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                 elif types == 'international_bachelor_program':
34                     domain = [('course_id.is_international_bachelor_program', '=')]
35                 elif types == 'international_masters_program':
36                     domain = [('course_id.is_international_masters_program', '=')]
```



Tarea

Calculadora de operaciones básicas

Vamos a simular una pequeña calculadora, para esto debemos crear 4 objetos que se encarguen de:
sumar, restar, multiplicar o dividir.

En el cliente podremos enviar 2 números y una operación y debemos permitir que estos 4 objetos decidan cuál puede manejar el cálculo solicitado



Muchas Gracias...
los espero la siguiente clase



