STATE

PATRÓN DE COMPORTAMIENTO

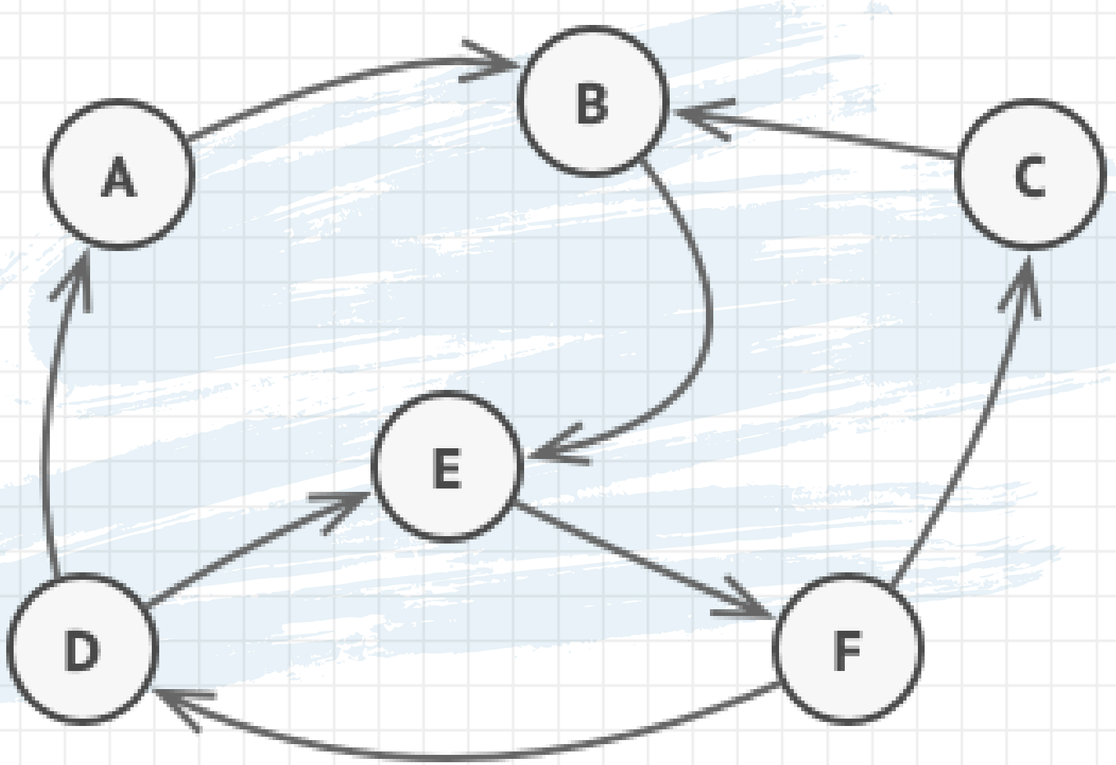
MAURICIO DELGADO LEANDRO B82553

DEFINICIÓN

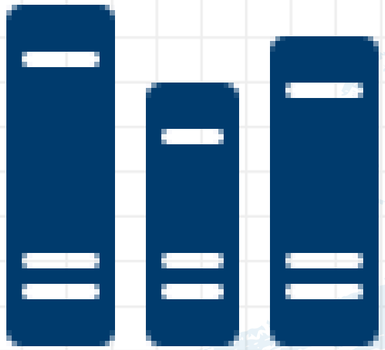
Patrón de comportamiento

Cambia comportamiento de un objeto

PROBLEMA



PROBLEMA

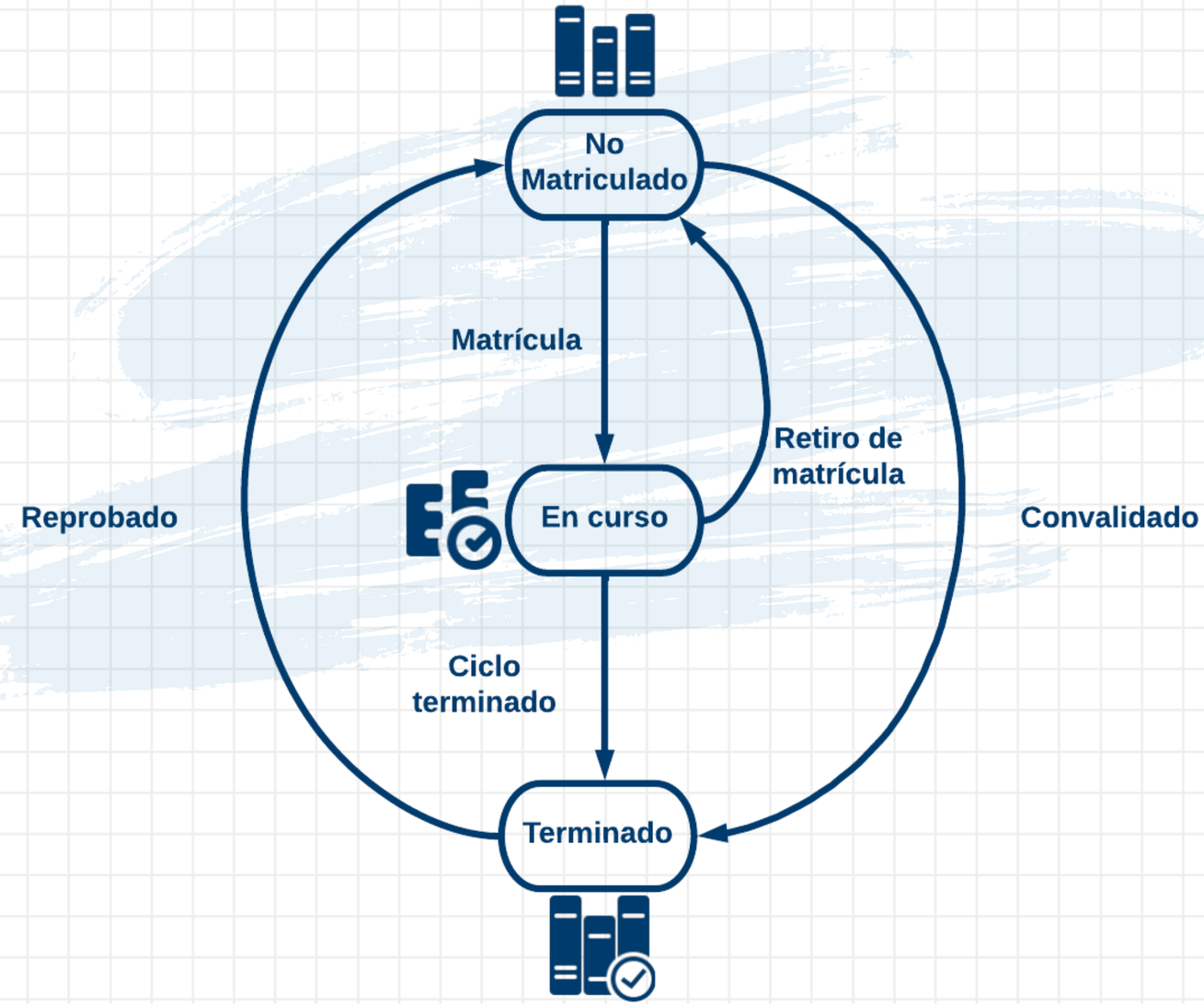


CURSO NO MATRICULADO

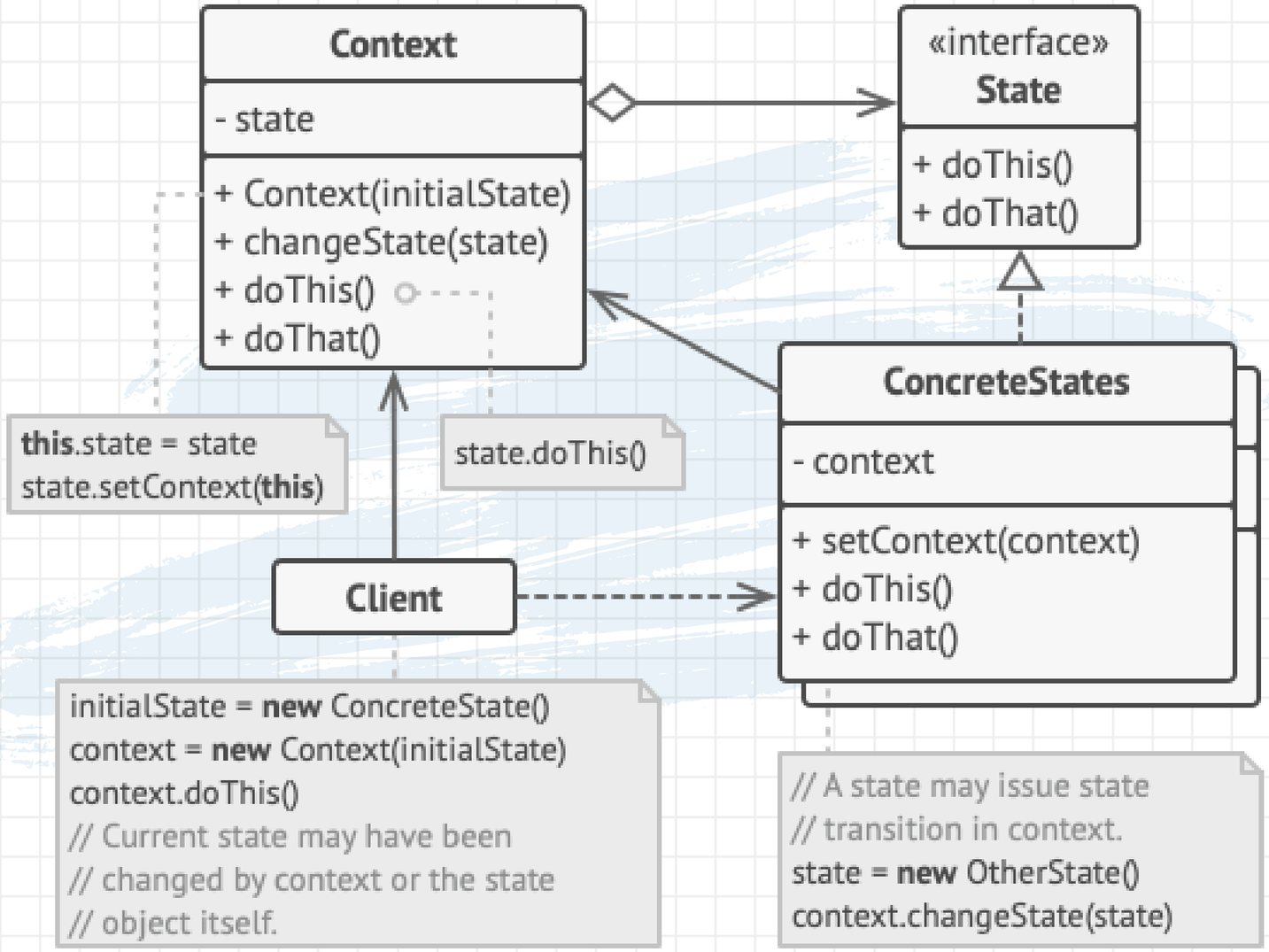
CURSO EN PROCESO

CURSO TERMINADO

MÁQUINA DE ESTADOS FINITA



SOLUCIÓN


Context
- state
+ Context(initialState)
+ changeState(state)
+ doThis()
+ doThat()
«interface»
State
+ doThis()
+ doThat()
ConcreteStates
- context
+ setContext(context)
+ doThis()
+ doThat()
Client
this.state = state
state.setContext(this)
state.doThis()
initialState = new ConcreteState()
context = new Context(initialState)
context.doThis()
// Current state may have been
// changed by context or the state
// object itself.
// A state may issue state
// transition in context.
state = new OtherState()
context.changeState(state)

INTERFAZ ESTADO

```python
# Clase abstracta para definir la interfaz de los estados
class Estado(ABC):

    # init method or constructor
    def __init__(self, contexto):
        self._contexto = contexto

    @abstractmethod
    def accion_x(self):
        pass

    @abstractmethod
    def accion_y(self):
        pass
```

ESTADOS CONCRETOS

```python
# Estado concreto
class estado_concreto1(Estado):
    def accion_x(self):
        print("Estado concreto 1")
        self._contexto.cambiar_estado(estado_concreto2(self._contexto))

    def accion_y(self):
        print("Transición incorrecta")

# Estado concreto
class estado_concreto2(Estado):
    def accion_y(self):
        print("Estado concreto 2")
        self._contexto.cambiar_estado(estado_concreto1(self._contexto))

    def accion_x(self):
        print("Transición incorrecta")
```

ESTADOS CONCRETOS

```python
# Estado concreto
class estado_concreto1(Estado):
    def accion_x(self):
        print("Estado concreto 1")
        self._contexto.cambiar_estado(estado_concreto2(self._contexto))

    def accion_y(self):
        print("Transición incorrecta")

# Estado concreto
class estado_concreto2(Estado):
    def accion_y(self):
        print("Estado concreto 2")
        self._contexto.cambiar_estado(estado_concreto1(self._contexto))

    def accion_x(self):
        print("Transición incorrecta")
```

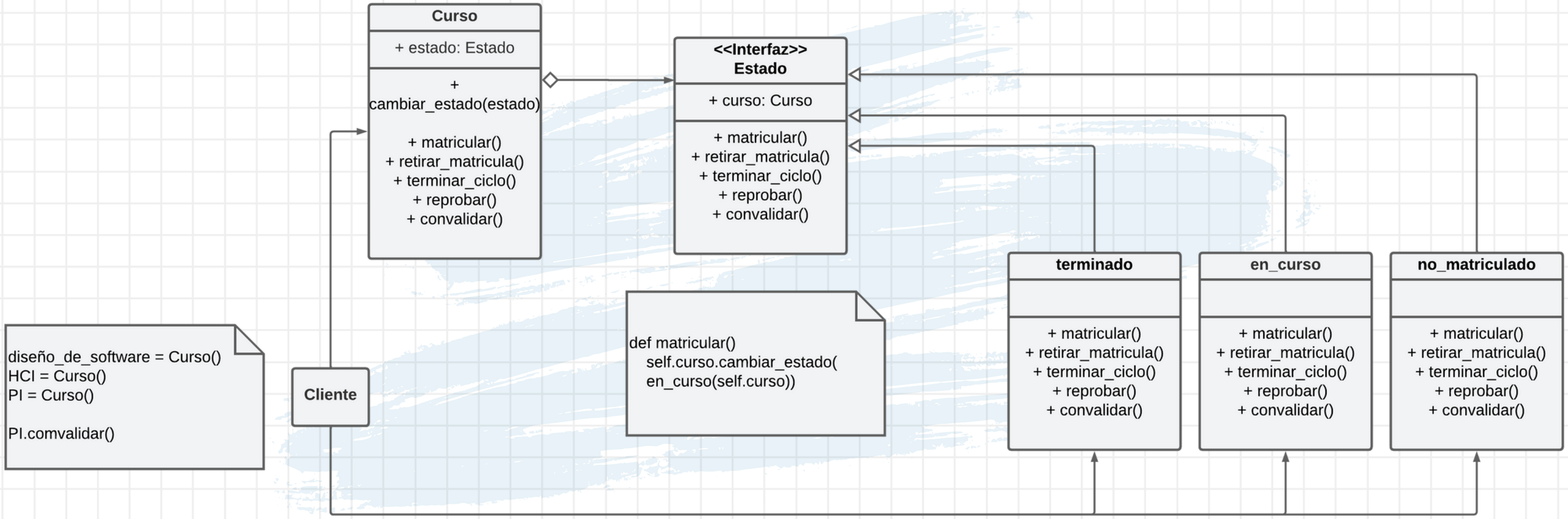
CONTEXTO

```
# Contexto
class Contexto:
    def __init__(self):
        # Estado inicial
        self.estado_actual = estado_concreto1(self)

    def cambiar_estado(self,estado):
        self.estado_actual = estado

    def accion_x(self):
        self.estado_actual.accion_x()

    def accion_y(self):
       self.estado_actual.accion_y()
```

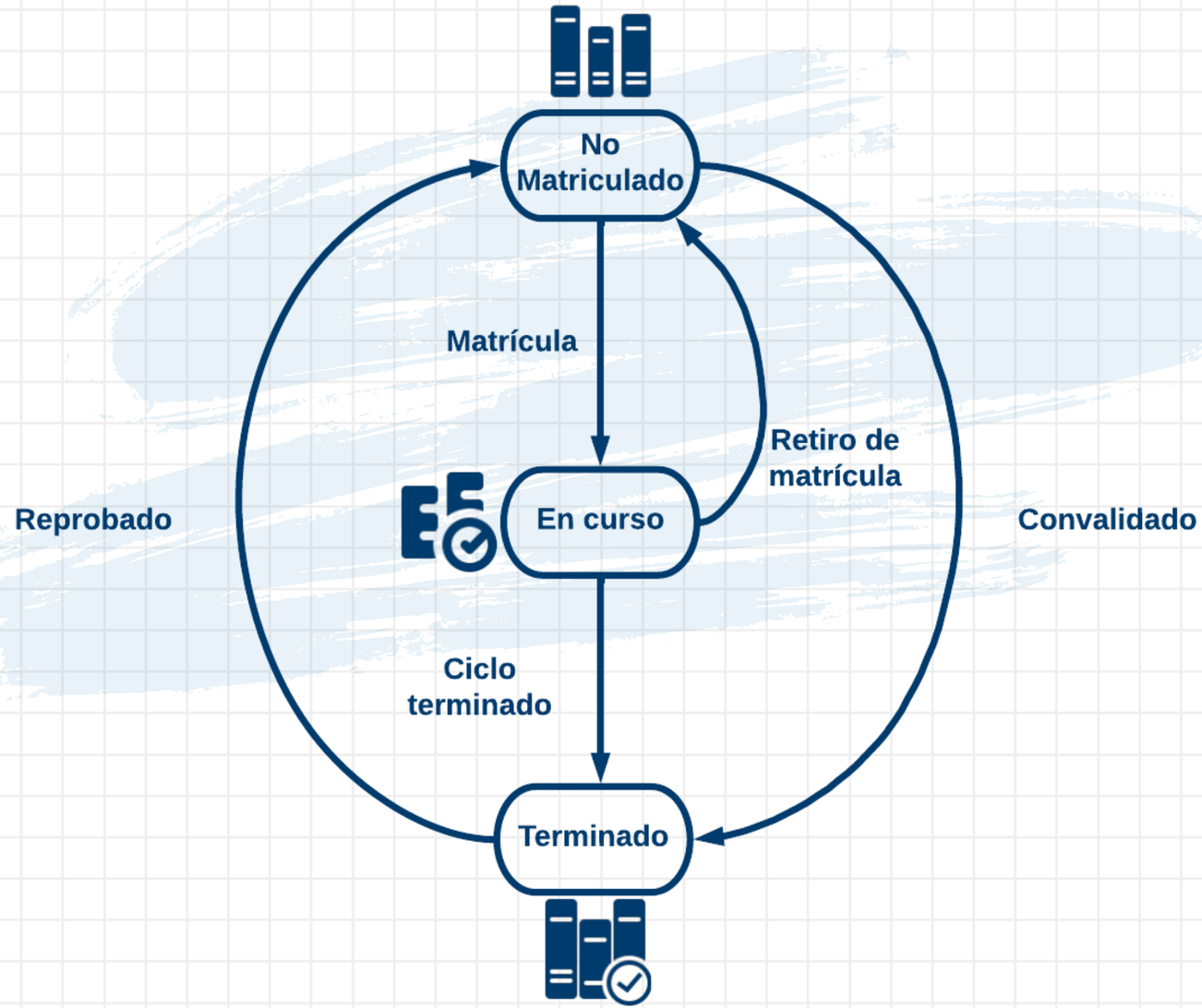
"CLIENTE"

```python
if __name__ == "__main__":

    contexto = Contexto()
    contexto.accion_x()
    contexto.accion_y()


    contexto.cambiar_estado(estado_concreto2(contexto))
```

EJEMPLO


Curso
+ estado: Estado
+ cambiar_estado(estado)
+ matricular()
+ retirar_matricula()
+ terminar_ciclo()
+ reprobar()
+ convalidar()
<<Interfaz>>
Estado
+ curso: Curso
+ matricular()
+ retirar_matricula()
+ terminar_ciclo()
+ reprobar()
+ convalidar()
terminado
+ matricular()
+ retirar_matricula()
+ terminar_ciclo()
+ reprobar()
+ convalidar()
en_curso
+ matricular()
+ retirar_matricula()
+ terminar_ciclo()
+ reprobar()
+ convalidar()
no_matriculado
+ matricular()
+ retirar_matricula()
+ terminar_ciclo()
+ reprobar()
+ convalidar()
Cliente
diseño_de_software = Curso()
HCI = Curso()
PI = Curso()
PI.comvalidar()
def matricular()
self.curso.cambiar_estado(
en_curso(self.curso))

MÁQUINA DE ESTADOS FINITA



Código en github

SIMULACIÓN

```
PS C:\Users\Gigabyte\Desktop\3er año\Diseño\software-design\docs\5.patterns\comportamiento\state\src> pytho
n3 .\mediacion_virtual_ejemplo.py
Matriculo Diseño de Software:
Curso matriculado exitosamente.
Estado: <class '__main__.en_curso'>
Matriculo HCI:
Curso convalidado exitosamente.
Estado: <class '__main__.terminado'>
Matriculo PI:
Curso matriculado exitosamente.
Estado: <class '__main__.en_curso'>

Fin de ciclo lectivo
Termino Diseño de Software:
Curso terminado.
Estado: <class '__main__.terminado'>
Termino HCI:
No se puede terminar el curso
Termino PI:
Curso terminado.
Estado: <class '__main__.terminado'>

Reprobé Diseño de Software:
Curso reprobado.
Estado: <class '__main__.no_matriculado'>
PS C:\Users\Gigabyte\Desktop\3er año\Diseño\software-design\docs\5.patterns\comportamiento\state\src>
```

Código en github

CONSECUENCIAS

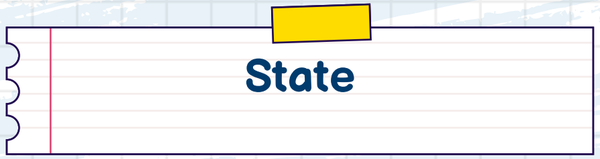
- ✓ Single Resposibility Principle
- ✓ Open/Closed Principle (escalabilidad)
- ✓ Chao condiciones
- ✗ Overkill

IMPLEMENTACIÓN

- Objetos con muchos estados cambiantes
- Clases con condicionales que alteran el comportamiento
- Codigo duplicado en presencia de una maquina de estados finita

RELACIÓN CON OTROS PATRONES

Similitud



BIBLIOGRAFIA

Refactoring.Guru. (2023). State. Refactoring.Guru.
https://refactoring.guru/design-patterns/state

Design Patterns - State Pattern. (s. f.).
https://www.tutorialspoint.com/design_pattern/state_pattern.htm

GeeksforGeeks. (2023). State Design Pattern. GeeksforGeeks.
https://www.geeksforgeeks.org/state-design-pattern/

Yenigün, O. (2023, 22 marzo). Design Patterns in Python: State Pattern - Dev Genius. Medium.
https://blog.devgenius.io/design-patterns-in-python-state-pattern-e646423969d5

MUCHAS GRACIAS!

ACTIVIDAD

Puerta de acceso

Máquina de Café

Estados de la materia