





State Pattern Design

Fundamentos:

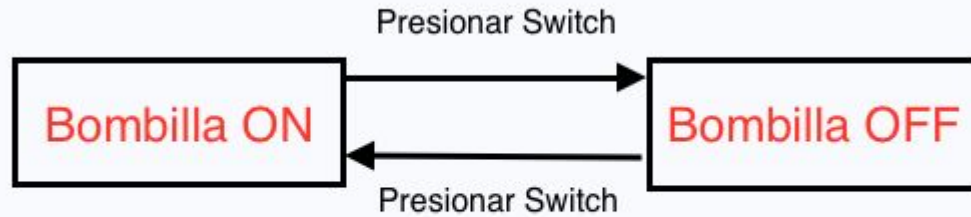
- Se encuentra dentro de la categoría de patrones de comportamiento.
- El patrón de estado permite a un objeto cambiar su comportamiento dependiendo del valor o estado actual del mismo.



¿Tiene nuestro
proyecto/problema múltiples
estados,transiciones,eventos?



Ejemplo Rápido



Ejemplo sin State:

```
func main() {  
    myApp := app.New()  
  
    myWindow := myApp.NewWindow( title: "Control de Bombilla")  
  
    bombillaText := widget.NewLabel( text: "Bombilla OFF")  
  
    boton := widget.NewButton( label: "Encender/Apagar", func() {  
        if bombillaText.Text == "Bombilla OFF" {  
            bombillaText.SetText( text: "Bombilla ON")  
        } else {  
            bombillaText.SetText( text: "Bombilla OFF")  
        }  
    })  
  
    content := container.New(layout.NewVBoxLayout(), boton, bombillaText)  
    myWindow.SetContent(content)  
  
    myWindow.ShowAndRun()  
}
```

Ejemplo simple con State:

```
func main() {  
    myApp := app.New()  
  
    var widgetLabel *widget.Label  
    var bombilla *state.BombillaSimple  
  
    myWindow := myApp.NewWindow( title: "Control de Bombilla")  
  
    bombilla = state.NewBombillaSimple()  
  
    widgetLabel = widget.NewLabel( text: "Bombilla OFF")  
  
    boton := widget.NewButton( label: "Cambiar estado", func() {  
        bombilla.PressButton(widgetLabel)  
    })  
  
    content := container.New(layout.NewVBoxLayout(), boton, widgetLabel)  
    myWindow.SetContent(content)  
  
    myWindow.ShowAndRun()  
}
```

Ejemplo simple con State:

```
const (
    Off      int = 0
    OnMode1  int = 1
    OnMode2  int = 2
)

// BombillaSimple mantiene el estado actual y proporciona una forma de cambiarlo.
type BombillaSimple struct {
    state int
}

// NewBombillaSimple inicializa una nueva Bombilla con un estado inicial
func NewBombillaSimple() *BombillaSimple {
    return &BombillaSimple{state: Off}
}

func (b *BombillaSimple) PressButton(widgetLabel *widget.Label) {
    switch b.state {
    case Off:
        widgetLabel.SetText( text: "Bombilla ON -> Modo 1")
        b.state = OnMode1
    case OnMode1:
        widgetLabel.SetText( text: "Bombilla ON -> Modo 2")
        b.state = OnMode2
    case OnMode2:
        widgetLabel.SetText( text: "Bombilla OFF")
        b.state = Off
    }
}
```

Ejemplo difícil con State:

```
func main() {  
    myApp := app.New()  
  
    var widgetLabel *widget.Label  
    var bombilla *state.Bombilla  
  
    myWindow := myApp.NewWindow( title: "Control de Bombilla")  
  
    bombilla = state.NewBombilla()  
  
    widgetLabel = widget.NewLabel( text: "Bombilla OFF")  
  
    boton := widget.NewButton( label: "Cambiar estado", func() {  
        bombilla.PressButton(widgetLabel)  
    })  
  
    content := container.New(layout.NewVBoxLayout(), boton, widgetLabel)  
    myWindow.SetContent(content)  
  
    myWindow.ShowAndRun()  
}
```


Ejemplo difícil con State:

```
// State es una interfaz para los diferentes estados de la bombilla
type State interface {
    PressButton(bombilla *Bombilla, widgetLabel *widget.Label)
}

// Bombilla mantiene el estado actual y proporciona una forma de cambiarlo.
type Bombilla struct {
    state State
}

// NewBombilla inicializa una nueva Bombilla con un estado inicial
func NewBombilla() *Bombilla {
    return &Bombilla{state: &OffState{}}
}

// PressButton permite que el estado actual maneje el evento del botón
func (b *Bombilla) PressButton(widgetLabel *widget.Label) {
    b.state.PressButton(b, widgetLabel)
}
```

```
// OffState representa el estado de la bombilla apagada
type OffState struct{}

func (s *OffState) PressButton(bombilla *Bombilla, widgetLabel *widget.Label) {
    widgetLabel.SetText( text: "Bombilla ON -> Modo 1")
    bombilla.state = &OnStateModel1{}
}

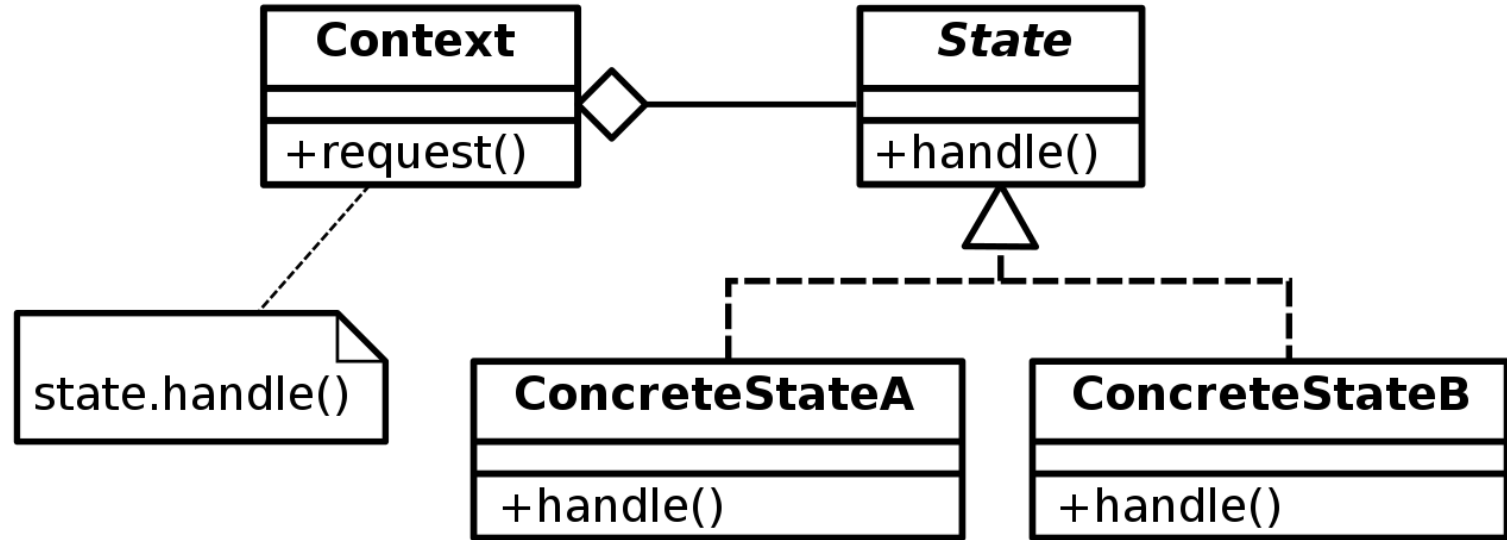
// OnStateModel1 representa el estado de la bombilla encendida en Modo 1
type OnStateModel1 struct{}

func (s *OnStateModel1) PressButton(bombilla *Bombilla, widgetLabel *widget.Label) {
    widgetLabel.SetText( text: "Bombilla ON -> Modo 2")
    bombilla.state = &OnStateMode2{}
}

// OnStateMode2 representa el estado de la bombilla encendida en Modo 2
type OnStateMode2 struct{}

func (s *OnStateMode2) PressButton(bombilla *Bombilla, widgetLabel *widget.Label) {
    widgetLabel.SetText( text: "Bombilla OFF")
    bombilla.state = &OffState{}
}
```

Estructura UML:



Relación con otros patrones:

Los patrones State y Strategy son similares uno del otro ya que ambos permiten cambiar el comportamiento del objeto y la principal diferencia es la siguiente:

Por un lado el patrón de Strategy hace que los objetos sean completamente independiente y desconocidos entre uno y otro. Por el otro lado , el patrón State no restringe dependencias que existan entre estados, lo que permite que los objetos alteren ese estado a voluntad.

Pros:

- Principio de única responsabilidad. Código organizado relacionado a estados particulares en estructuras separadas.
- Ayuda a implementar el principio de Abierto-Cerrado. Lo que permite introducir nuevos estados sin cambiar el estado actual de la estructura o contexto.
- Permite simplificar el código al remover condicionales grandes.

Contras:

- Aplicar el patrón podría ser perjudicial si un objeto con estado tiene unos pocos estados o casi nunca cambio entre ellos.