# PATRÓN MEMENTO

GEANCARLO RIVERA HERNÁNDEZ C06516
JULIO ALEJANDRO RODRÍGUEZ SALGUERA C16717

# PROBLEMA

- El patrón Memento se utiliza para guardar y restaurar el estado de un objeto sin violar su encapsulación.
- Se utiliza para recordar y restaurar el estado de un objeto en momentos específicos.
- Es especialmente útil para corrección de errores y mantener historial de versiones.

- Evita la violación de la encapsulación al restringir el acceso a la interfaz de modificación del estado guardado.
- Almacena solo ciertos estados del objeto "Originator" en componentes llamadas "Memento".
- Útil para implementar funcionalidades como "deshacer" en aplicaciones.

# EJEMPLO

- Un ejemplo del patrón Memento se encuentra en la plataforma de Correo Institucional de la Universidad de Costa Rica.
- La cual permite volver al estado previo de un correo al presionar "Ctrl + z".
- El patrón Memento se utiliza para almacenar los diferentes estados del correo mientras se redacta.
- Se puede retroceder al estado anterior o incluso al estado inicial del correo.
- Es decir, proporciona la funcionalidad de deshacer y retroceder en el proceso de redacción del correo, restaurando un estado previo.

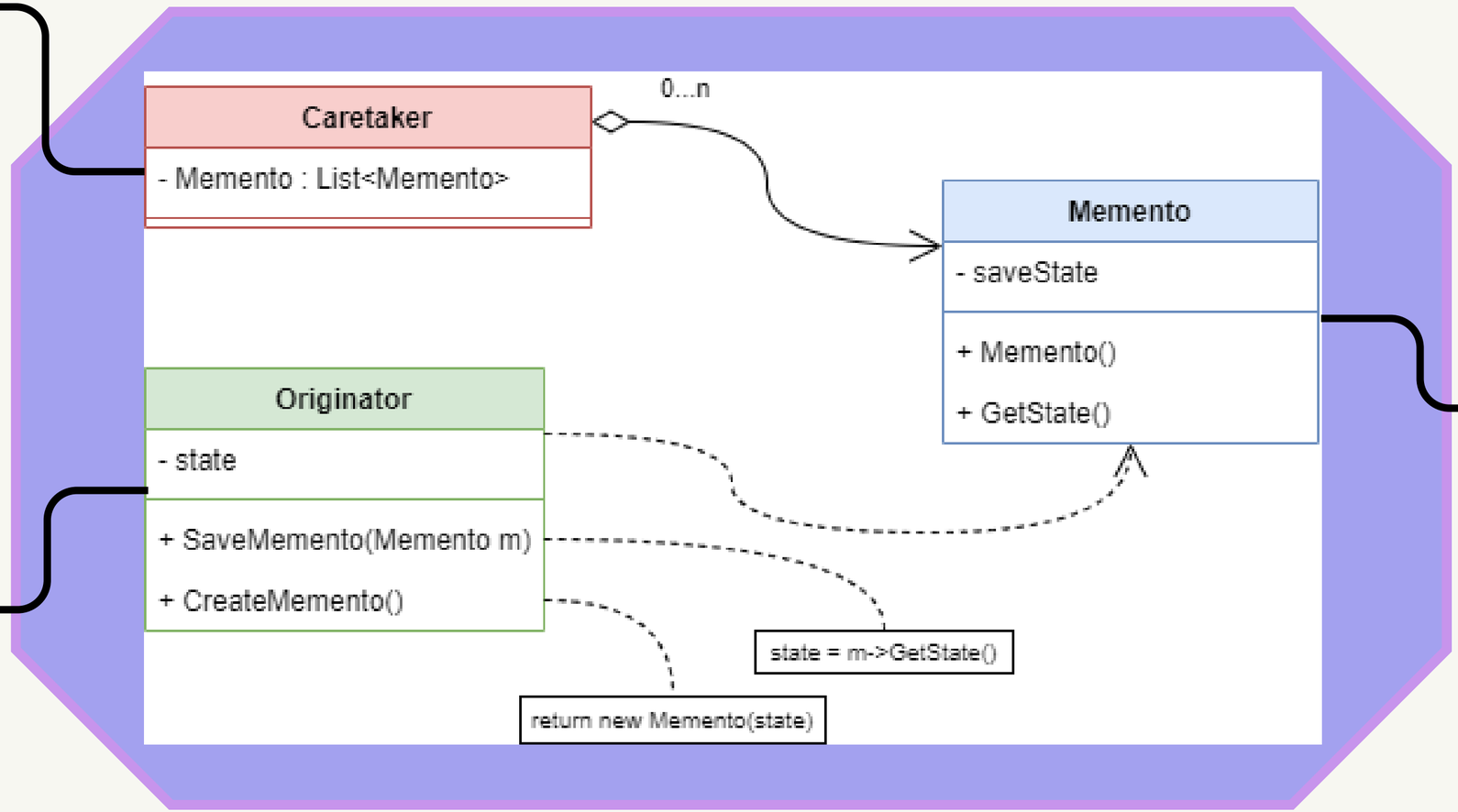
# SOLUCIÓN

**Este patrón tiene 3 componentes:**

**Editor del Correo**

**Correo**

**Estado del Correo**



# SOLUCIÓN

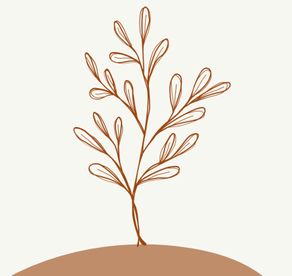
## Originator

- La clase es responsable de tomar una "snapshot" del estado actual, los cuales son los objetos Memento

- La clase utiliza objetos Memento para recuperar un estado anterior

- La clase interactúa directamente con los objetos Memento para leer su contenido y restaurar los datos

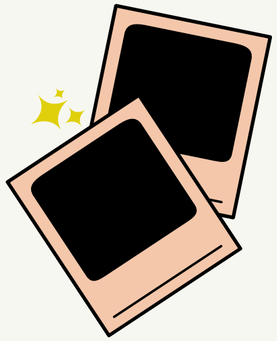
- Es componente el componente Caretaker el que determina cuándo guardar y restaurar un estado



# SOLUCIÓN

## Memento

- La clase Memento almacena la información del estado del componente "Originator" en un momento específico.
- Se conoce comúnmente como "snapshot" y contiene toda la información necesaria para volver a versiones anteriores
- Los objetos Memento son guardados en una estructura de datos del "Caretaker"
- El acceso a los datos del Memento es limitado para el componente Caretaker para respetar el encapsulamiento
- El componente Originator tiene un acceso más amplio a los Mementos y los utiliza para recuperar estados anteriores

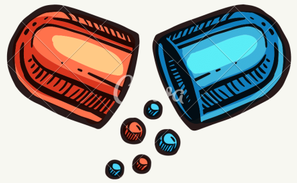


# SOLUCIÓN

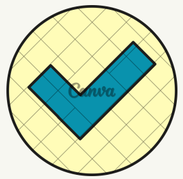
## Caretaker

- Es responsable de almacenar cada snapshot del estado actual, es decir los objetos Memento
- Provee interfaces para comunicar al Originator la necesidad de guardar o restaurar un estado previo
- No accede directamente al estado del Origen ni a la información de los Mementos. Únicamente los gestiona

# CONSECUENCIAS



**Preservación de la Encapsulación**

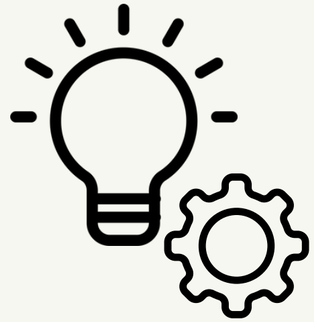


**Simplificación del Originator**

**Definición de interfaces**

**Costos asociados a gestión de Mementos**

# IMPLEMENTACIÓN



- Considerar el soporte de niveles de interfaces a la hora de elegir un lenguaje para implementar el patrón
- Definir adecuadamente las responsabilidades del Caretaker
- Considerar granularidad para almacenar cambios incrementales y no el estado completo del objeto afectado
- Considerar el impacto y rendimiento a la hora de guardar estados de objetos muy grandes o complejos
- Combinar con otros patrones como el Command u Observer permiten a este patrón ofrecer una funcionalidad más completa y flexible

# EJEMPLO DE CÓDIGO

## Originator

```java
class Correo {

  //  Atributo que define el estado actual del correo
  private String contenido;

  // --Otra información del correo que no es guardada en el Memento--

  public Correo(String contenido) {
    this.contenido = contenido;
  }

  public void CambiarContenido(String texto) {
    this.contenido = texto;
  }

  public EstadoCorreo GuardarEstado() {
    System.out.println("~Guardando el estado del correo~");
    return new EstadoCorreo(this.contenido);
  }

  public void RestaurarEstado(EstadoCorreo estadoCorreo) {
    System.out.println("\n~Recuperando un estado del correo~");
    this.contenido = estadoCorreo.RecuperarDatosGuardados();
  }

  public String toString() {
    return "\n\nContenido: " + this.contenido + "\n";
  }

```

# EJEMPLO DE CÓDIGO

## Memento

```
public static class EstadoCorreo {

  private String contenidoCorreo;

  public EstadoCorreo(String guardadoContenidoCorreo) {
    this.contenidoCorreo = guardadoContenidoCorreo;
  }

  private String RecuperarDatosGuardados( ) {
    return this.contenidoCorreo;
  }
}
```

# EJEMPLO DE CÓDIGO

## Caretaker

```java
import java.util.Stack;

class EditorCorreo {

  private Stack<Correo.EstadoCorreo> EstadosDelCorreo;

  private Correo correo;

  public EditorCorreo() {
    EstadosDelCorreo = new Stack<>();
    correo = new Correo("");
  }

  public void CambiarContenido(String texto) {
    this.correo.CambiarContenido(texto);
  }

  public void RealizarGuardado() {
    EstadosDelCorreo.push(correo.GuardarEstado());
  }

  public void RetrocederEstado() {
    if (!EstadosDelCorreo.isEmpty()) {
      correo.RestaurarEstado(EstadosDelCorreo.pop());
    }
  }

  public void ImprimirEstadoActual() {
    System.out.println(this.correo);
  }
}
```

# EJEMPLO DE CÓDIGO

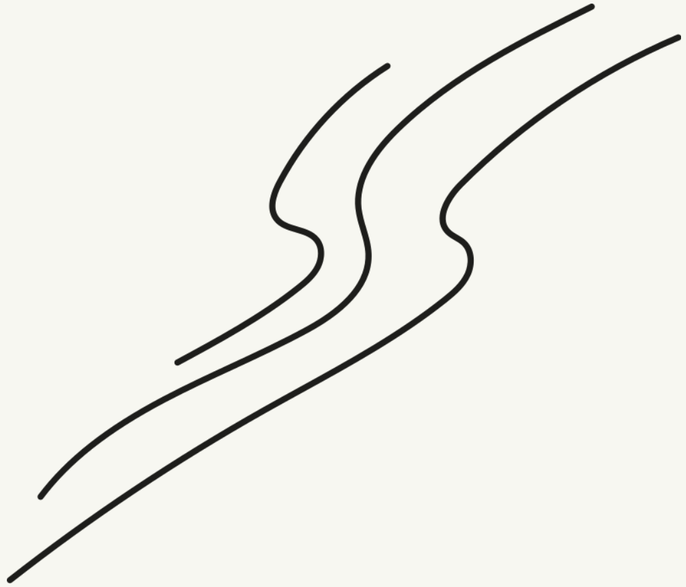
## Main

```java
class Main {
  public static void main(String[] args) {

    System.out.println("\n-----Creando un nuevo correo vacio-----");

    EditorCorreo editorCorreo = new EditorCorreo();
    editorCorreo.ImprimirEstadoActual();

    System.out.println("\n\n-----Escribiendo en el correo y guardando el estado-----");

    editorCorreo.CambiarContenido("Este patrón de comportamiento es muy útil " +
                                  "cuando se desea guardar el estado de un objeto.");
    editorCorreo.ImprimirEstadoActual();

    editorCorreo.RealizarGuardado();

    System.out.println("\n\n\n-----Cambiando datos al correo-----");

    editorCorreo.CambiarContenido("Los peces globos son muy bonitos.");
    editorCorreo.ImprimirEstadoActual();

    System.out.println("\n\n-----Retrocediendo al estado anterior-----");

    editorCorreo.RetrocederEstado();
    editorCorreo.ImprimirEstadoActual();
  }
}
```

```
-----Creando un nuevo correo vacio-----

Contenido:


-----Escribiendo en el correo y guardando el estado-----

Contenido: Este patrón de comportamiento es muy útil cuando se desea guardar el estado de un objeto.

~Guardando el estado del correo~


-----Cambiando datos al correo-----

Contenido: Los peces globos son muy bonitos.


-----Retrocediendo al estado anterior-----

~Recuperando un estado del correo~

Contenido: Este patrón de comportamiento es muy útil cuando se desea guardar el estado de un objeto.
```

# PATRONES RELACIONADOS

- **Command**
- **Observer**
- **Iterator**
- **State**



# PRINCIPIOS

## SOLID

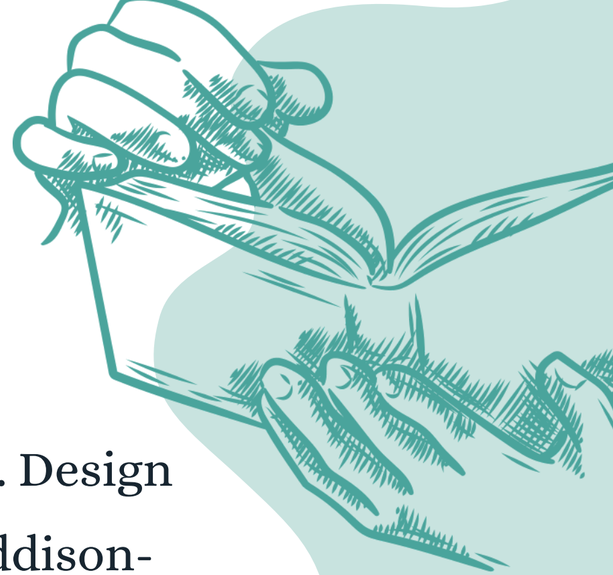
- **Single-responsibility Principle:** El patrón separa la responsabilidad de guardar los estados de la clase Originator y se la delega a la clase Memento, que es responsable únicamente de encapsular el estado y no debe tener ninguna lógica adicional.

KISS

OOP

DRY

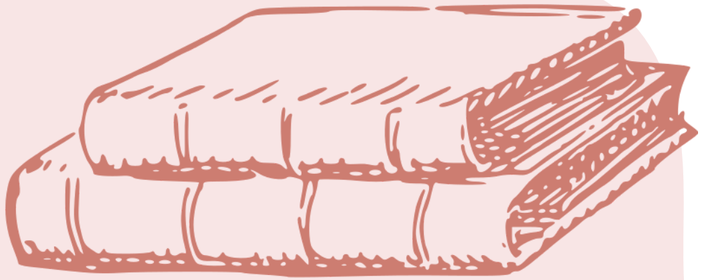
# REFERENCIAS




Haythornwaite, C. (2002). Gamma, E., Helm, R., Johnson, R. & Vlissides, J. Design Patterns: Elements of Reusable Object Oriented Software. New York: Addison-Wesley, 1995. ADDISON-WESLEY.

Savoir, L.A. (2007) Pattern design. Beverly, MA: Rockport Publishers.

Smith, J.M. (2012) Elemental Design Patterns. Upper Saddle River, NJ: Addison-Wesley.

# Muchas Gracias!