Sprint 1

Santiago Zuluaga Guerrero

Grupo 13, ciclo 4 G29.

ADAPTER

Este patrón de diseño permite la colaboración entre diferentes interfaces.

Básicamente, cuando nosotros nos vamos de un país a otro, y queremos cargar nuestros dispositivos nos percatamos que el tomacorriente tiene un enchufe diferente al de nuestro país. Por lo cual, debemos usar un adaptador, este nos permite realizar la tarea de cargar nuestros dispositivos sin que estos se den cuenta que hay un adaptador.

Por otra parte, nos encontramos en una sala de teleconferencias donde se suelen usar dispositivos Windows y se tiene una entrada al proyector de tipo HDMI normal. Resulta que un compañero llevo su macbook, la cual no tiene una entrada hdmi normal, por lo cual, el siempre carga con su adaptador de video, el cual, permite la conexión y la transmisión de video. El adaptador no es detectado ni por el proyector ni por el macbook y de igual forma, el video es transmitido.

Código fuente:

```
// Digamos que tienes dos clases con interfaces compatibles:
// RoundHole (HoyoRedondo) y RoundPeg (PiezaRedonda).
class RoundHole is
    constructor RoundHole(radius) { ... }

    method getRadius() is
        // Devuelve el radio del agujero.

    method fits(peg: RoundPeg) is
        return this.getRadius() >= peg.getRadius()

class RoundPeg is
    constructor RoundPeg(radius) { ... }

    method getRadius() is
        // Devuelve el radio de la pieza.


// Pero hay una clase incompatible: SquarePeg (PiezaCuadrada).
class SquarePeg is
    constructor SquarePeg(width) { ... }

    method getWidth() is
        // Devuelve la anchura de la pieza cuadrada.
```

Figura 1. (Refactoring Guru, 2021)

```
// Una clase adaptadora te permite encajar piezas cuadradas en
// hoyos redondos. Extiende la clase RoundPeg para permitir a
// los objetos adaptadores actuar como piezas redondas.
class SquarePegAdapter extends RoundPeg is
    // En realidad, el adaptador contiene una instancia de la
    // clase SquarePeg.
    private field peg: SquarePeg

    constructor SquarePegAdapter(peg: SquarePeg) is
        this.peg = peg

    method getRadius() is
        // El adaptador simula que es una pieza redonda con un
        // radio que pueda albergar la pieza cuadrada que el
        // adaptador envuelve.
        return peg.getWidth() * Math.sqrt(2) / 2


// En algún punto del código cliente.
hole = new RoundHole(5)
rpeg = new RoundPeg(5)
hole.fits(rpeg) // verdadero

small_sqpeg = new SquarePeg(5)
large_sqpeg = new SquarePeg(10)
hole.fits(small_sqpeg) // esto no compila (tipos incompatibles)

small_sqpeg_adapter = new SquarePegAdapter(small_sqpeg)
large_sqpeg_adapter = new SquarePegAdapter(large_sqpeg)
hole.fits(small_sqpeg_adapter) // verdadero
hole.fits(large_sqpeg_adapter) // falso
```

Figura 2. (Refactoring Guru, 2021)

COMANDO

Este patrón de diseño me permite parametrizar todos los métodos con diferentes solicitudes. Es decir, poner en cola, priorizar, adelantar, borrar, frenar las solicitudes convertidas a objetos independientes.

El ejemplo más tradicional es: vas a un restaurante y le haces un pedido al mesero que anota tu pedido en un papel que posteriormente pasa a cocina donde es preparado el pedido, luego el mesero una vez listo el pedido revisa que todo este en orden y te lleva el pedido solicitado.

Otro ejemplo sería: Cuando solicitas un pedido por un comercio digital, tan pronto le das al botón de comprar envías un comando, una solicitud que llega al comercio, ellos revisan el pedido, revisan inventario, alistan el pedido y lo envían. Todo esto gracias al el comando de pedido.

Código fuente:

```
// La clase base comando define la interfaz común a todos los
// comandos concretos.
abstract class Command is
    protected field app: Application
    protected field editor: Editor
    protected field backup: text

    constructor Command(app: Application, editor: Editor) is
        this.app = app
        this.editor = editor

    // Realiza una copia de seguridad del estado del editor.
    method saveBackup() is
        backup = editor.text

    // Restaura el estado del editor.
    method undo() is
        editor.text = backup

    // El método de ejecución se declara abstracto para forzar a
    // todos los comandos concretos a proporcionar sus propias
    // implementaciones. El método debe devolver verdadero o
    // falso dependiendo de si el comando cambia el estado del
    // editor.
    abstract method execute()
```

Figura 3. (Refactoring Guru, 2021)

```
// Los comandos concretos van aquí.
class CopyCommand extends Command is
    // El comando copiar no se guarda en el historial ya que no
    // cambia el estado del editor.
    method execute() is
        app.clipboard = editor.getSelection()
        return false

class CutCommand extends Command is
    // El comando cortar no cambia el estado del editor, por lo
    // que debe guardarse en el historial. Y se guardará siempre
    // y cuando el método devuelva verdadero.
    method execute() is
        saveBackup()
        app.clipboard = editor.getSelection()
        editor.deleteSelection()
        return true

class PasteCommand extends Command is
    method execute() is
        saveBackup()
        editor.replaceSelection(app.clipboard)
        return true

// La operación deshacer también es un comando.
class UndoCommand extends Command is
    method execute() is
        app.undo()
        return false
```

Figura 4. (Refactoring Guru, 2021)

```
// El historial global de comandos tan solo es una pila.
class CommandHistory is
    private field history: array of Command

    // El último dentro...
    method push(c: Command) is
        // Empuja el comando al final de la matriz del
        // historial.

    // ...el primero fuera.
    method pop():Command is
        // Obtiene el comando más reciente del historial.


// La clase editora tiene operaciones reales de edición de
// texto. Juega el papel de un receptor: todos los comandos
// acaban delegando la ejecución a los métodos del editor.
class Editor is
    field text: string

    method getSelection() is
        // Devuelve el texto seleccionado.

    method deleteSelection() is
        // Borra el texto seleccionado.

    method replaceSelection(text) is
        // Inserta los contenidos del portapapeles en la
        // posición actual.
```

Figura 5. (Refactoring Guru, 2021)

```
// La clase Aplicación establece relaciones entre objetos. Actúa
// como un emisor: cuando algo debe hacerse, crea un objeto de
// comando y lo ejecuta.
class Application is
    field clipboard: string
    field editors: array of Editors
    field activeEditor: Editor
    field history: CommandHistory

    // El código que asigna comandos a objetos UI puede tener
    // este aspecto.
    method createUI() is
        // ...
        copy = function() { executeCommand(
            new CopyCommand(this, activeEditor)) }
        copyButton.setCommand(copy)
        shortcuts.onKeyPress("Ctrl+C", copy)

        cut = function() { executeCommand(
            new CutCommand(this, activeEditor)) }
        cutButton.setCommand(cut)
        shortcuts.onKeyPress("Ctrl+X", cut)

        paste = function() { executeCommand(
            new PasteCommand(this, activeEditor)) }
        pasteButton.setCommand(paste)
        shortcuts.onKeyPress("Ctrl+V", paste)

        undo = function() { executeCommand(
            new UndoCommand(this, activeEditor)) }
        undoButton.setCommand(undo)
        shortcuts.onKeyPress("Ctrl+Z", undo)
```

Figura 6. (Refactoring Guru, 2021)

```
// Ejecuta un comando y comprueba si debe añadirse al
// historial.
method executeCommand(command) is
    if (command.execute)
        history.push(command)

// Toma el comando más reciente del historial y ejecuta su
// método deshacer. Observa que no conocemos la clase de ese
// comando. Pero no tenemos por qué, ya que el comando sabe
// cómo deshacer su propia acción.
method undo() is
    command = history.pop()
    if (command != null)
        command.undo()
```

Figura 7. (Refactoring Guru, 2021)

Referencias

Refactoring Guru. (6 de 11 de 2021). *Refactoring Guru*. Obtenido de https://refactoring.guru/es/design-patterns/adapter