Lucía Karlen

ADAPTER: DEFINICIÓN

Es un patrón de diseño estructural que permite la colaboración entre objetos con interfaces incompatibles

Es un objeto especial que convierte la interfaz de un objeto, de forma que otro objeto pueda comprenderla

Convierten datos en varios formatos y ayudan a objetos con distintas interfaces a colaborar

ADAPTER: FUNCIONAMIENTO

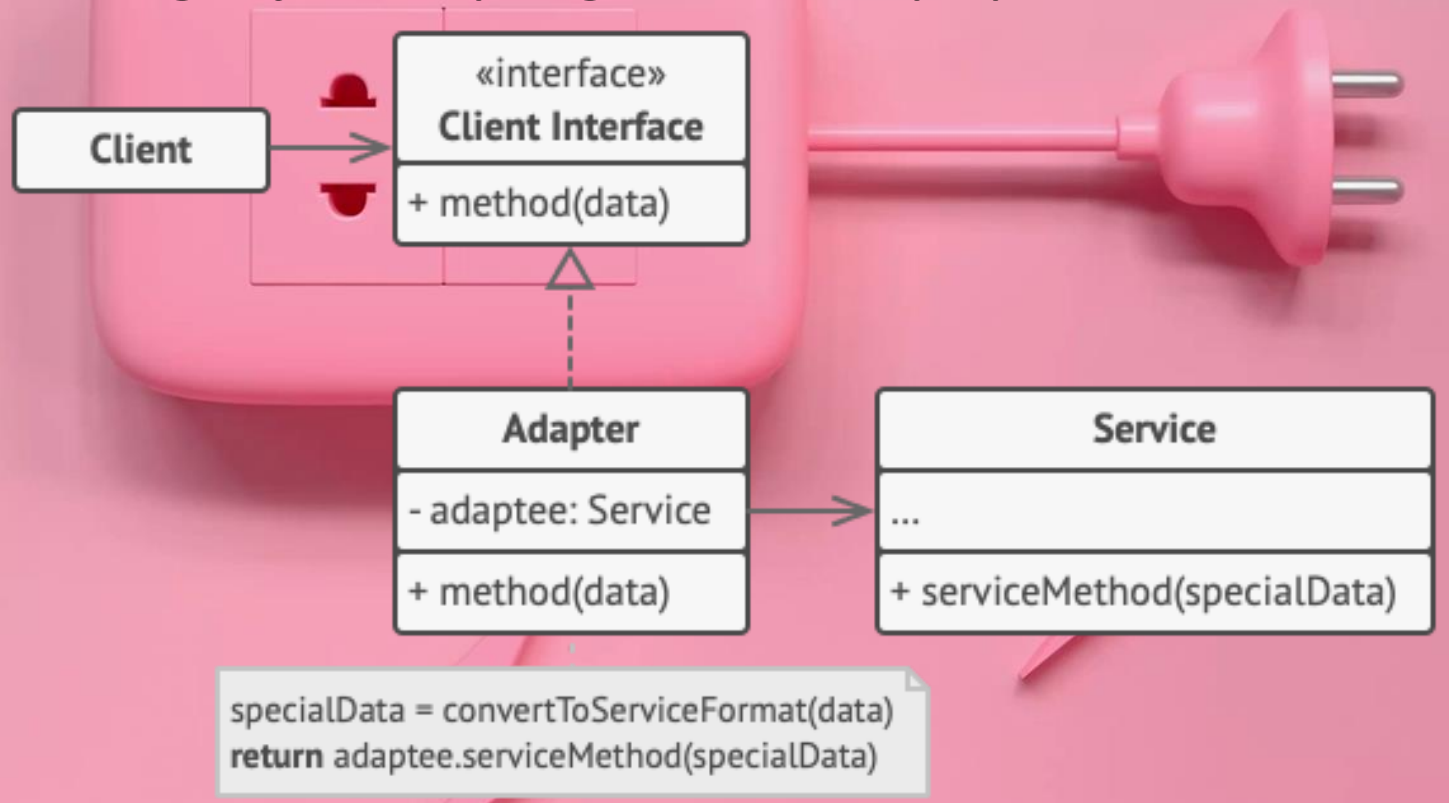
1) Obtiene una interfaz compatible con uno de los objetos existentes

2) Con esa interfaz, el objeto existente puede invocar con seguridad los métodos del adaptador.

3) Al recibir una llamada, el adaptador pasa la solicitud al segundo objeto, pero en un formato y orden que ese segundo objeto espera.

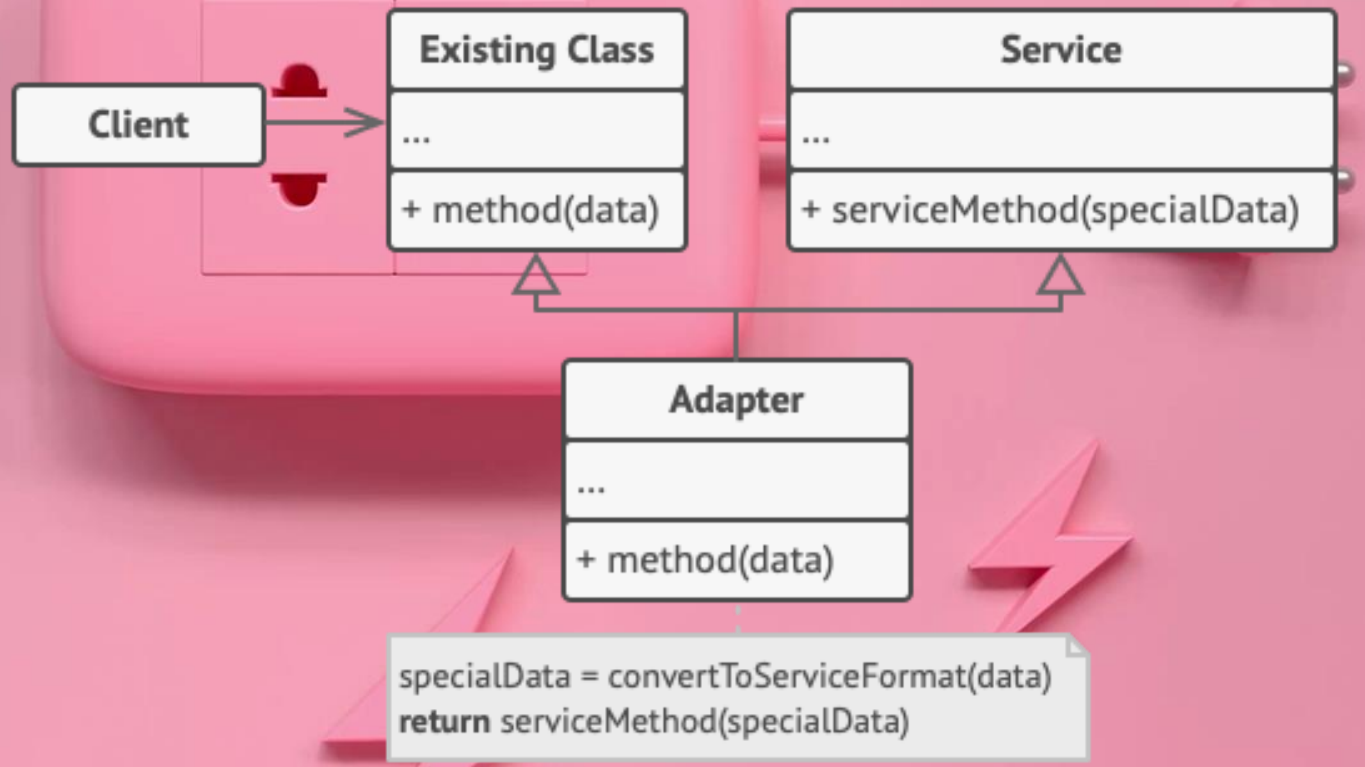
ADAPTER: ESTRUCTURA

- Adaptador de objetos: usa el principio de composición de objetos
- En todos los lenguajes de programación populares.



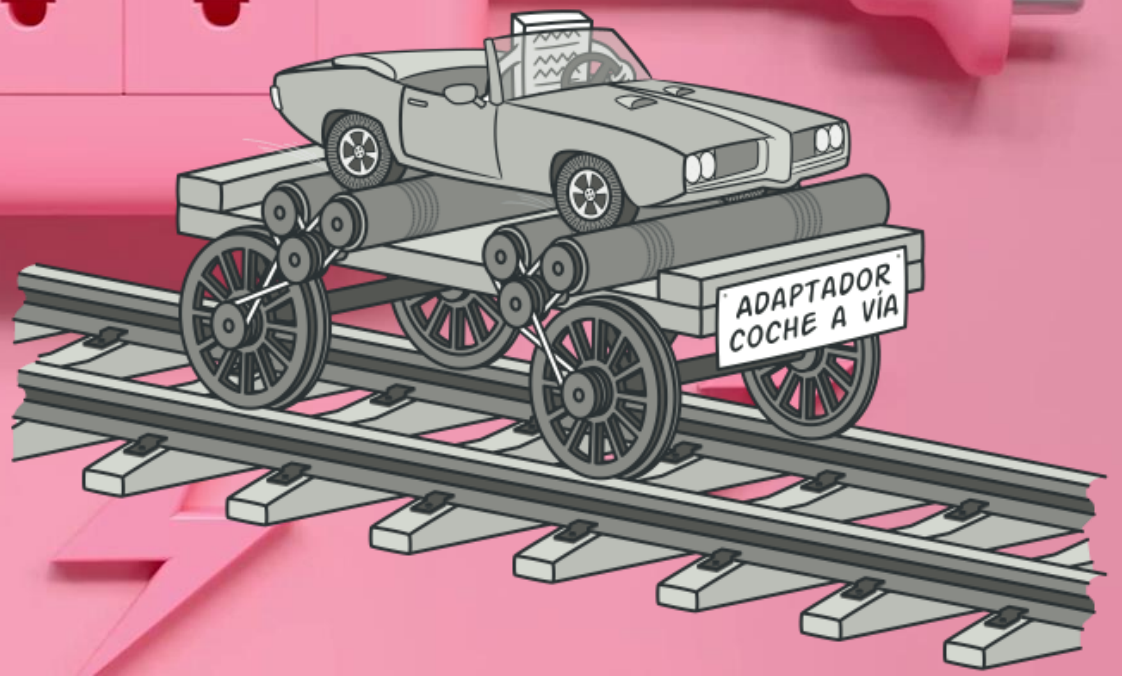
ADAPTER: ESTRUCTURA

- Clase adaptadora: usa la herencia
- En lenguajes de programación que soporten la herencia múltiple.



ADAPTER: APLICABILIDAD

- Usa la clase adaptadora cuando quieras usar una clase existente, pero su interfaz no sea compatible con el resto del código
- Usa el patrón cuando quieras reutilizar varias subclases existentes que no tengan alguna funcionalidad común que no pueda añadirse a la superclase



ADAPTER: VENTAJAS Y DESVENTAJAS

VENTAJAS

- Principio de responsabilidad única
- Principio de abierto/cerrado

DESVENTAJAS

- La complejidad general del código aumenta, ya que debes introducir un grupo de nuevas interfaces y clases

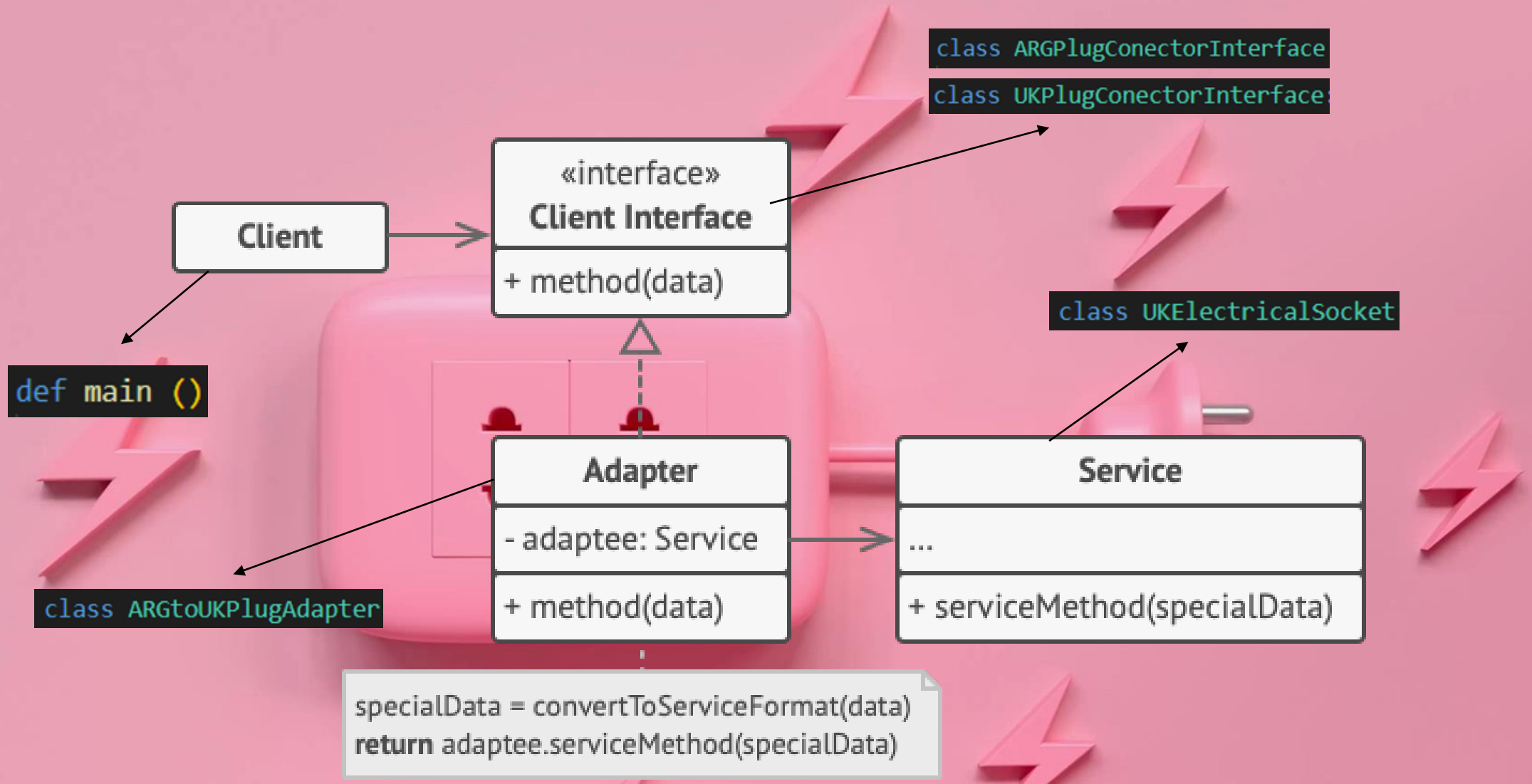
ADAPTER: EJEMPLO DE CÓDIGO

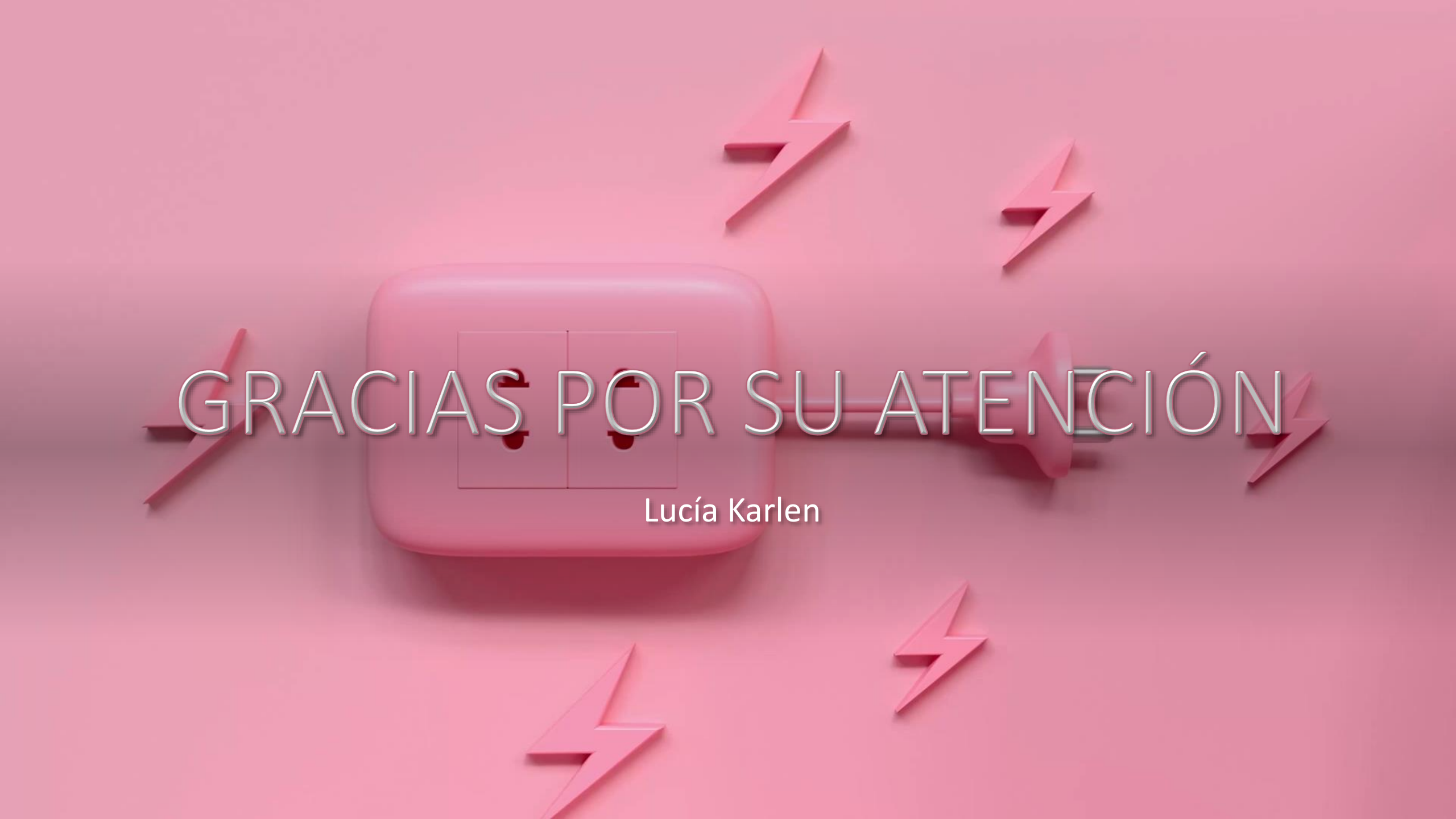
```python
# Interfaz para los enchufes de Argentina
class ARGPlugConectorInterface:
    def give_electricity(self):
        pass

# Clase que representa un enchufe de Argentina
class ARGPlugConector (ARGPlugConectorInterface)
    def give_electricity(self):
        print("This is an ARG plug")

# Interfaz para los enchufes del Reino Unido
class UKPlugConectorInterface:
    def provide_electricity(self):
        pass

# Clase que representa un enchufe eléctrico del
# Reino Unido
class UKElectricalSocket:
    def plug_in(self, uk_plug):
        print("This is a UK electrical socket")
        uk_plug.provide_electricity()
```

```python
# Clase adaptadora que permite encajar un enchufe de
# Argentina en un enchufe del Reino Unido
class ARGtoUKPlugAdapter(UKPlugConectorInterface):
    def __init__(self, arg_plug):
        self.arg_plug = arg_plug

    def provide_electricity(self):
        self.arg_plug.give_electricity()

def main ():
    # Crear un enchufe de Argentina
    arg_plug = ARGPlugConector()
    # Crear un enchufe eléctrico del Reino Unido
    uk_electrical_socket = UKElectricalSocket()
    # Crear un adaptador para el enchufe de Argentina
    uk_adapter = ARGtoUKPlugAdapter(arg_plug)
    # Conectar el adaptador al enchufe eléctrico
    # del Reino Unido
    uk_electrical_socket.plug_in(uk_adapter)

if __name__ == "__main__":
    main()
```

```
This is a UK electrical socket
This is an ARG plug
```

Código


class ARGPlugConectorInterface
class UKPlugConectorInterface
«interface»
Client Interface
+ method(data)
Client
class UKElectricalSocket
def main ()
Adapter
- adaptee: Service
+ method(data)
Service
...
+ serviceMethod(specialData)
class ARGtoUKPlugAdapter
specialData = convertToServiceFormat(data)
return adaptee.serviceMethod(specialData)


GRACIAS POR SU ATENCIÓN
Lucía Karlen