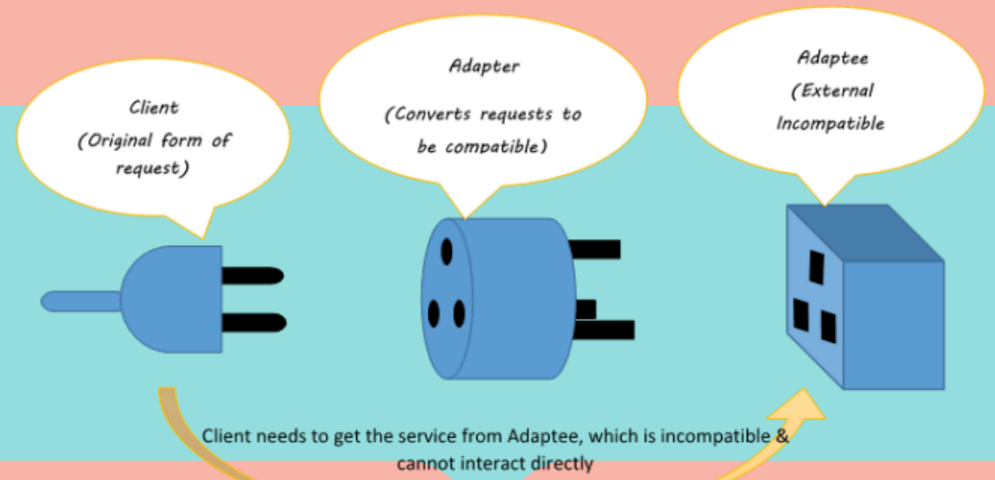
ANALISIS Y DISEÑO DE SOFTWARE

Patrones adapter y command

Grupo 1:
Mateo Llumigusin
Richard Gualotuña
David Asmal

PATRON ADAPTER

Permite que dos clases incompatibles trabajen juntas adaptando la interfaz de una clase a lo que espera otra clase



Analogía del mundo real:

Un adaptador de enchufe que te permite conectar un dispositivo con clavija europea en un tomacorriente americano.

Uso típico:

- Cuando se necesite usar una clase existente, pero su interfaz no coincide con la que se requiere
- Para integrar sistemas antiguos con nuevos sin modificar su código original

Estructura básica:

- **Target:** La interfaz esperada por el cliente.
- **Adaptee:** La clase existente con una interfaz incompatible
- **Adapter:** Traduce la interfaz de Adaptee a la de Target.

Ejemplo en software:

Supón que tienes una API de pagos antigua que usa el método procesarPago(), pero tu nuevo sistema espera un método llamado pagar(). Puedes crear un AdaptadorPago que tenga el método pagar() y por dentro llame a procesarPago()

Codigo Adapter

```java
public class EstudianteExterno {  1 usage
    private String identificador;  2 usages
    private String nombreCompleto;  2 usages
    private String edadTexto;  2 usages

    public EstudianteExterno(String identificador, String nombreCompleto, String edadTexto) {  no usages
        this.identificador = identificador;
        this.nombreCompleto = nombreCompleto;
        this.edadTexto = edadTexto;
    }

    public String getIdentificador() { return identificador; }  1 usage
    public String getNombreCompleto() { return nombreCompleto; }  1 usage
    public String getEdadTexto() { return edadTexto; }  1 usage
}
```

```java
public class EstudianteAdapter extends Estudiante {  no usages
    public EstudianteAdapter(EstudianteExterno externo) {  no usages
        super(Integer.parseInt(externo.getIdentificador()), externo.getNombreCompleto(), Integer.parseInt(externo.getEdadTexto()));
    }
}
```

Ventajas y Desventajas del Adapter

- Reutilización de código existente
- Desacoplamiento
- Facilita la integración de sistemas heredados
- Mejora la legibilidad y mantenibilidad
- Aplicable tanto en tiempo de compilación como de ejecución

- Puede aumentar la complejidad del diseño
- Posible sobrecarga de rendimiento
- No siempre se puede adaptar completamente
- Mantenimiento de adaptadores múltiples
- Riesgo de ocultar una mala arquitectura

Command.

Es un patrón de diseño de comportamiento que encapsula una solicitud como un objeto, permitiendo parametrizar métodos con diferentes solicitudes, almacenar operaciones para ejecutarlas más tarde o deshacer acciones.

Codigo Command

```java
public interface Comando {  4 usages  4 implementations
    void ejecutar();  4 usages  4 implementations
}
```

```java
public class ComandoAgregar implements Comando {  1 usage
    private List<Estudiante> lista;  2 usages
    private Estudiante estudiante;  2 usages

    public ComandoAgregar(List<Estudiante> lista, Estudiante estudiante) {
        this.lista = lista;
        this.estudiante = estudiante;
    }

    public void ejecutar() {  4 usages
        lista.add(estudiante);
    }
}
```

Ventajas y Desventajas del COMMAND

- Desacoplamiento: Separa el objeto que invoca la acción del que la realiza.
- Flexibilidad: Permite encolar, deshacer o reprogramar comandos.
- Extensibilidad: Es fácil añadir nuevos comandos sin modificar el código existente.

- Sobrecarga en el diseño: Si se abusa, puede complicar la estructura del código.
- Más clases: Requiere definir una clase o estructura por cada acción que se desea encapsular.