PATRÓN ADAPTER

UNIVERSIDAD CATÓLICA
SAN ANTONIO
UCAM

Ricardo Blanco Pérez
48615801-B
Curso 2017/2018

¿Qué es el patrón Adapter?

- Busca una manera estandarizada de adaptar un objeto a otro. Se utiliza para transformar una interfaz en otra, de tal modo que una clase que no pudiera utilizar la primera haga uso de ella a través de la segunda.
- Una clase adapter implementa una interfaz que conoce a sus clientes y proporciona acceso a una instancia de una clase que no conoce a sus clientes, es decir convierte la interfaz de una clase en una interfaz que el cliente espera.

¿Cuándo debe ser utilizado?

- Se quiere utilizar una clase que llame a un método a través de una interface, pero se busca utilizarlo con una clase que no implementa ese interface.

¿Qué hace el patrón?

- Convierte la interfaz de una clase en otra interfaz que el cliente espera. Esto permite a las clases trabajar juntas, lo que de otra manera no podrían hacerlo debido a sus interfaces incompatibles.
- Este patrón debe ser aplicado cuando se debe transformar una estructura a otra, pero sin tocar la original, ya sea porque no se puede o se desea cambiarla.

Diagrama de ejemplo de patrón Adapter

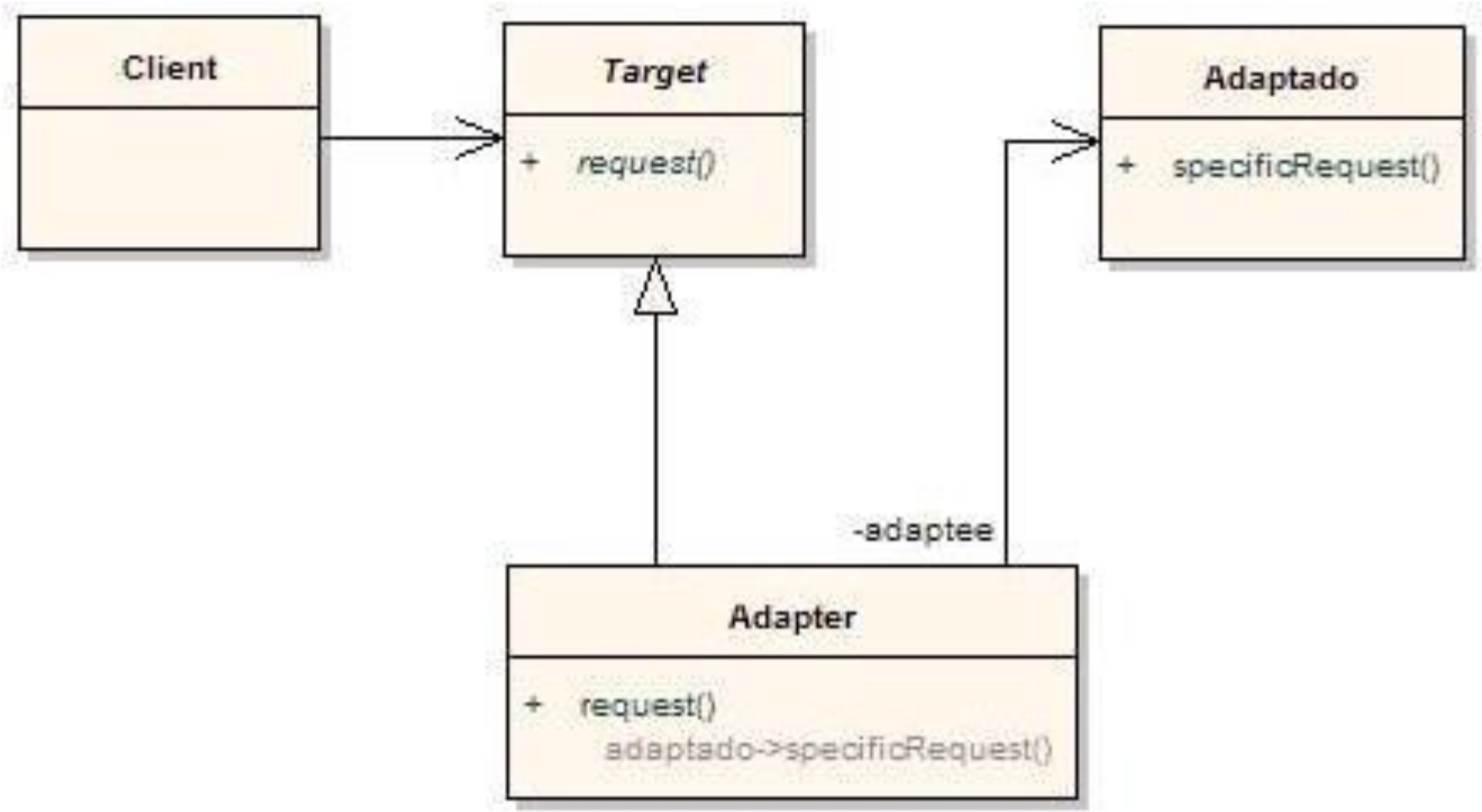

Diagrama de ejemplo de patrón Adapter

- **Target:** define la interfaz específica del dominio que el cliente usa.
- **Cliente:** colabora con la conformación de objetos para la interfaz Target.
- **Adaptado:** define una inferfaz existente que necesita adaptarse.
- **Adapter:** adapta la inferfaz de adapter a la interfaz Target.

Ejemplo de patrón Adapter

```java
public class ViejaToNuevaAdapter implements IPersonaNueva {
    private PersonaVieja vieja;

    public ViejaToNuevaAdapter(PersonaVieja vieja) {
        super();
        this.vieja = vieja;
    }

    public int getEdad() {
        GregorianCalendar c = new GregorianCalendar();
        GregorianCalendar c2 = new GregorianCalendar();
        c2.setTime(vieja.getFechaDeNacimiento());
        return c2.get(1) - c2.get(2);
    }

    public String getNombre() {
        return vieja.getNombre() + " " + vieja.getApellido();
    }

    public void setEdad(int edad) {
        GregorianCalendar c = new GregorianCalendar();
        int anioActual = c.get(1);
        c.set(1, anioActual - edad);
        vieja.setFechaDeNacimiento(c.getTime());
    }

    public void setNombre(String nombreCompleto) {
        String[] name = nombreCompleto.split(" ");
        String firstName = name[0];
        String lastName = name[1];
        vieja.setNombre(firstName);
        vieja.setApellido(lastName);
    }
```

Ejemplo de patrón Adapter

```
<terminated> main [Java Application] C:\Program Files\Java\jre1.8.0_111\bin\javaw.exe (3 jun. 2018 17:46:37)
1999
Ricardo Blanco
2003
Juan Perez
```

Ahora vemos lo que devuelve el programa por pantalla, siendo la persona Ricardo Blanco la clase personavieja y Juan Perez la personanueva.

Conclusión

- El cliente y las clases adapter permanecen independiente unas de las otras.
- Aunque puede hacer que el programa sea menos fácil de entender.
- Permite que un único adapter trabaje con muchos adapter, es decir, por si mismo y las subclases que tenga.