

METODOLOGÍAS DE PROGRAMACIÓN I

Patrón de comportamiento *Command*

Situación de ejemplo

- En una sale de espera de un hospital, un enfermero es el encargado de darle un número de atención a cada paciente que ingresa al hospital, así como de avisarle al próximo paciente a atender que el médico lo va a atender.
- En pocas palabras, el enfermero actúa como una "cola" de pacientes.

```
class Enfermero
    List pacientesEnEspera

    void recibirPaciente(paciente)
        // Método push
    Paciente proximoAAtender()
        // Método pop
```

Situación de ejemplo

- Según el jefe de turno de la sala los enfermeros tienen distintas órdenes al momento de recibir un paciente (push) o hacer pasar al consultorio al siguiente en espera (pop).
- Hay un jefe de sala que ordena a los enfermeros que tomen la presión arterial a los pacientes al entrar, mientras que otro ordena que se le ofrezca una taza de café caliente.
- Otro jefe de sala ordena que al momento de avisar al próximo paciente a atender le de una palmada en la espalda deseándole suerte, mientras que otro jefe ordena que se desinfecte el asiento donde estaba sentado el paciente que entró al consultorio.
- Un quinto jefe no da ninguna orden en particular y los enfermeros no hacen nada especial cuando llega un paciente o al momento de invitarlos a pasar al consultorio.

Problema

- Todo enfermero actúa como una "cola", pero en ocasiones tienen que hacer tareas extras al momento de hacer un "push" o un "pop".
- ¿Cómo se implementan los métodos "push" y "pop"?

```
class Enfermero
    List pacientesEnEspera

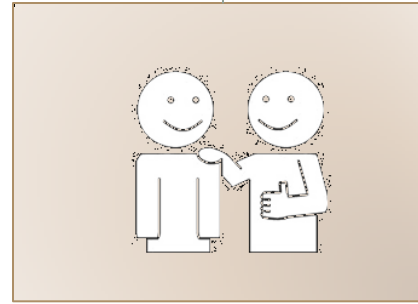
    void recibirPaciente(paciente)
        // Método push
    Paciente proximoAAtender()
        // Método pop
```

Problema

¿Esta es una solución?

Comportamiento
"push" y "pop"

Enfermero



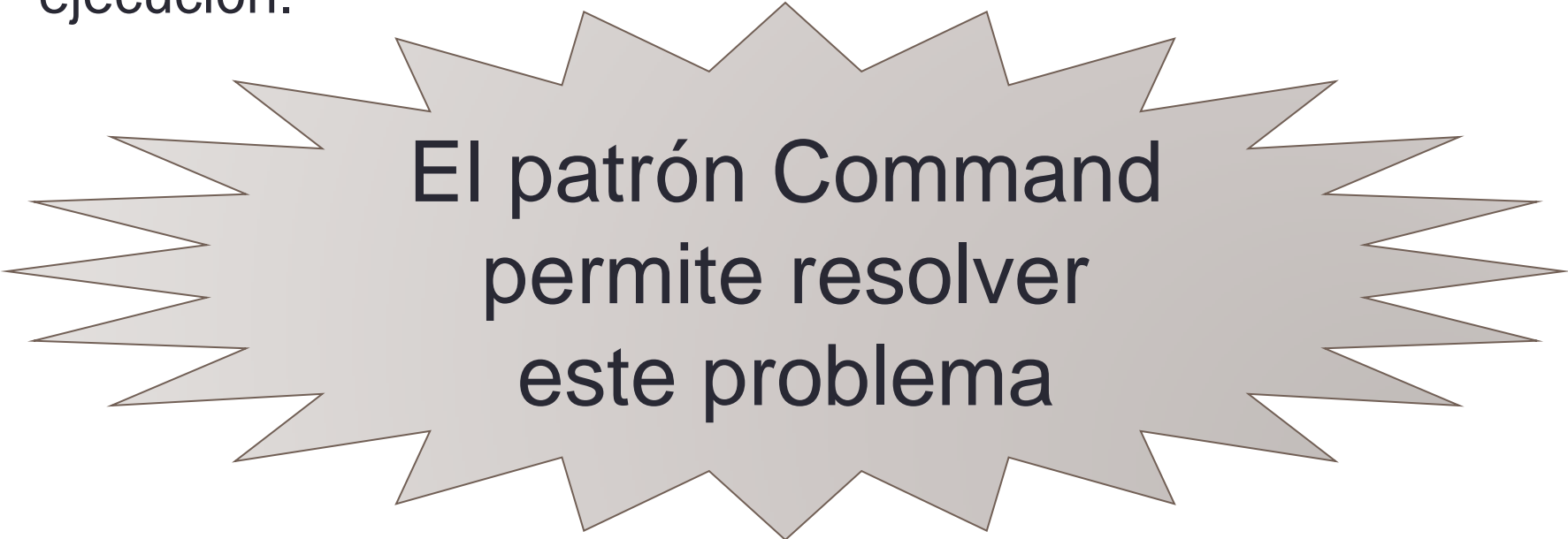
Comportamiento específico "extra"

Problema

- Si un enfermero hace un turno con dos jefes distintos ¿cómo cambiamos su comportamiento en tiempo de ejecución?
- ¿Cómo combinamos distintas acciones al hacer un "push" o un "pop"? ¿Cómo podemos hacer que un enfermero haga dos tareas extras al hacer "push" o "pop"?
- Si la clase Enfermero es una simple "cola". ¿Cómo logramos que pueda ser reutilizado y trabaje en otro lugar otorgando números ("push") y avisando al próximo a atender ("pop")?

Motivación

Buscamos un mecanismo que permita que un objeto tenga un comportamiento básico ("push" y "pop") y que también permita configurar comportamiento extra. Y además que ese comportamiento sea configurable en tiempo de ejecución.



**El patrón Command
permite resolver
este problema**

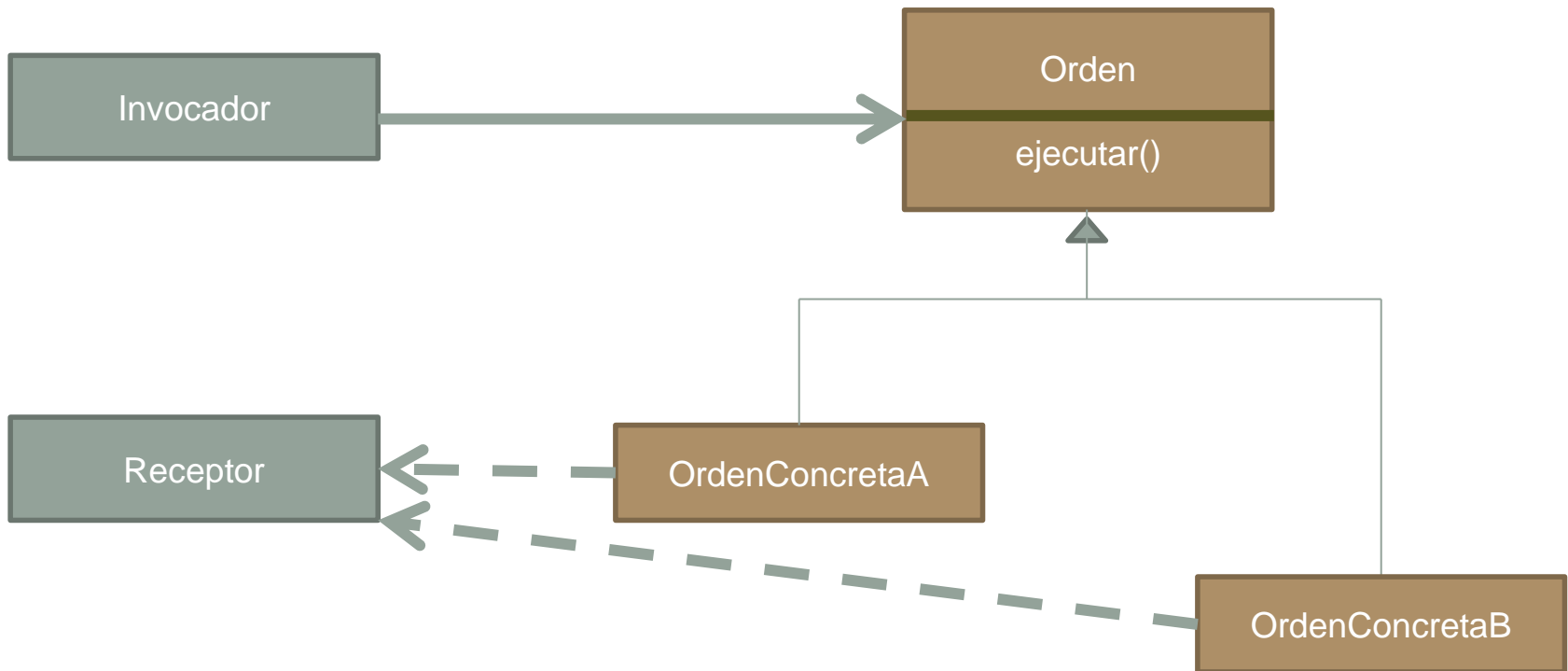
Command

Propósito: Encapsula una petición en un objeto, permitiendo así parametrizar a los clientes con diferentes peticiones, hacer cola o llevar un registro de las peticiones y poder deshacer las operaciones.

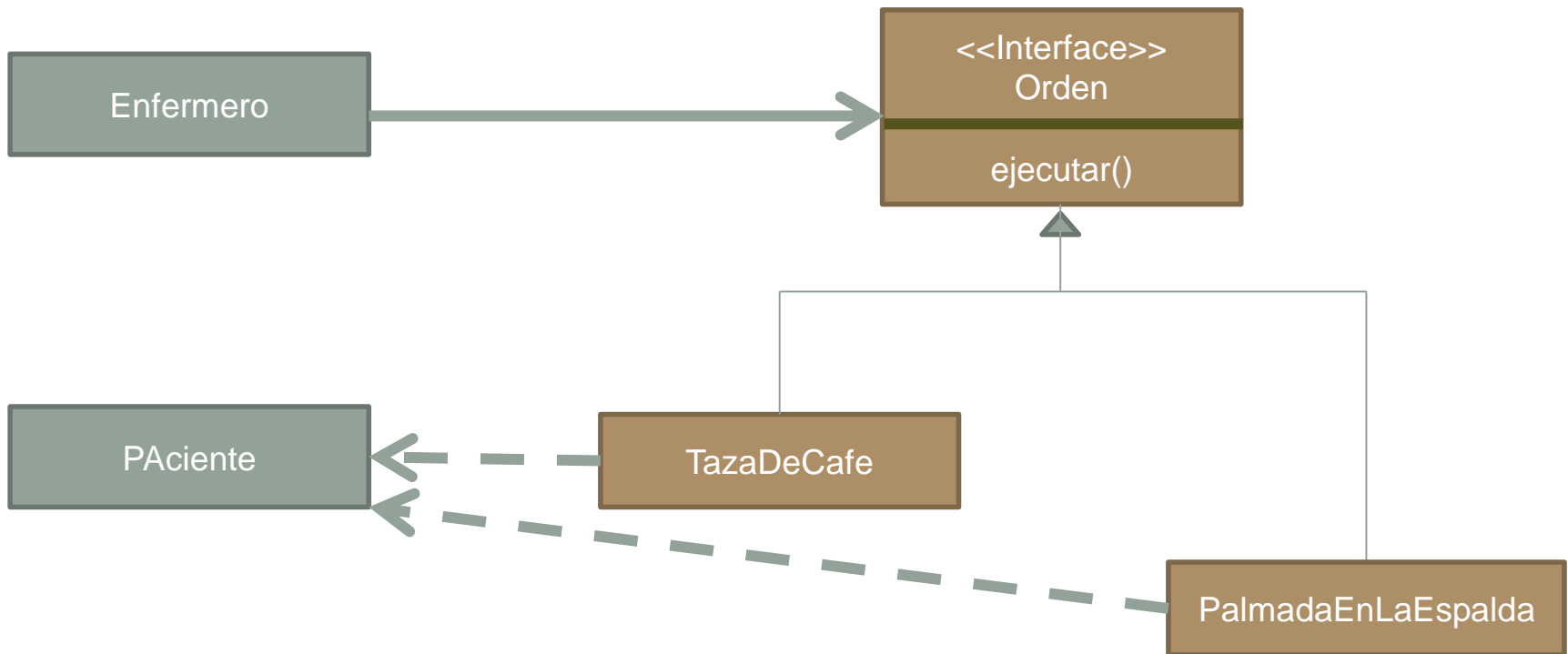
Aplicabilidad: usarlo cuando

- Se quiera parametrizar objetos con una acción a realizar
- Se necesite especificar, poner en cola y ejecutar peticiones en diferentes instantes de tiempo.
- Se requiera deshacer las acciones. Este patrón puede guardar el estado anterior de un objeto al momento de aplicar una acción para luego poder deshacer la acción.
- Se quiera registrar los cambios.

Command - Estructura



Command - Estructura



Command - Implementación

```
interface IOrden
```

```
void ejecutar()
```

La interface *IOrden* la podemos implementar como una interface con un único método ejecutar

Command - Implementación

```
class Enfermero
    IOrden ordenPush, ordenPop

    constructor(IOrden oPush, IOrden oPop)
        ordenPush = oPush
        ordenPop = oPop

    void setOrdenPush(IOrden oPush)
        ordenPush = oPush

    void setOrdenPop(IOrden oPop)
        ordenPop = oPop
```

La clase Enfermero tiene una orden para ejecutar al momento de hacer un "push" y otra para ejecutar al momento de hacer un "pop".

Command - Implementación

```
class Enfermero
    List pacientesEnEspera

    void recibirPaciente(paciente)
        // Método push
        pacientesEnEspera.Add(paciente)
        if(ordenPush != null)
            ordenPush.ejecutar()

    Paciente proximoAAtener()
        // Método pop
        p = pacientesEnEspera.Remove(0)
        if(ordenPop != null)
            ordenPop.ejecutar()
        return p
```

Al momento de hacer un "push" o un "pop" se ejecuta la orden correspondiente.

Command - Implementación

```
class TazaDeCafe : IOrden
```

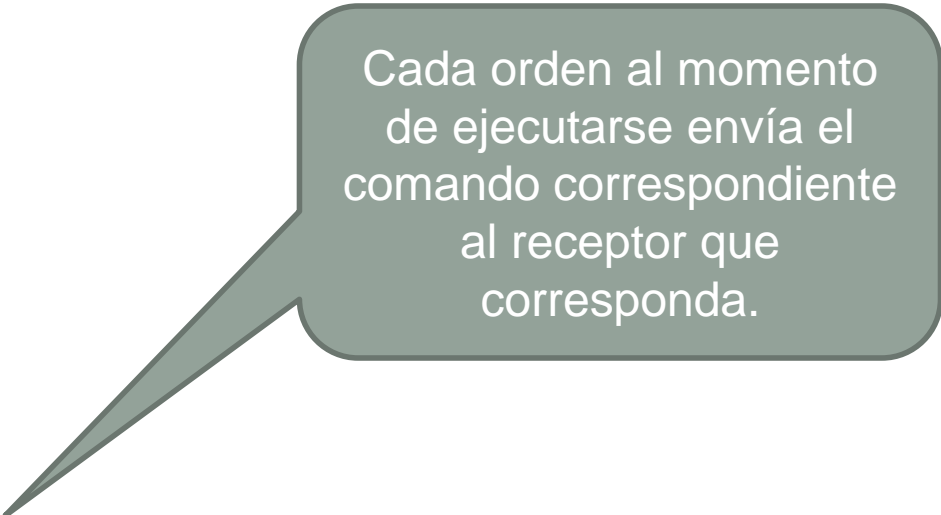
```
    paciente
```

```
    constructor(p)
```

```
        paciente = p
```

```
    void ejecutar()
```

```
        paciente.ofrecerTazaDeCafe()
```



Cada orden al momento de ejecutarse envía el comando correspondiente al receptor que corresponda.

Command - Implementación

```
void metodoCliente  
    paciente = new Paciente()  
  
    orden = new TazaDeCafe(paciente)  
  
    enfermero = new Enfermero()  
  
    enfermero.setOrdenPush(orden)  
  
    enfermero.recibirPaciente(paciente)
```

Al momento de crear un enfermero se setea cual es la orden que se desea ejecutar cuando reciba a un paciente.

Command – Ventajas

- Desacopla el objeto que invoca la operación de aquel que sabe como realizarla.
- Se pueden ensamblar órdenes en una orden compuesta (patrón Composite)
- Es fácil cambiar órdenes de manera dinámica en tiempo de ejecución