Patrón Command

Mª Ángeles Guillén Zapata
48692752-G
Angeles.guilen.zapata@gmail.com
Convocatoria Junio 2017-2018

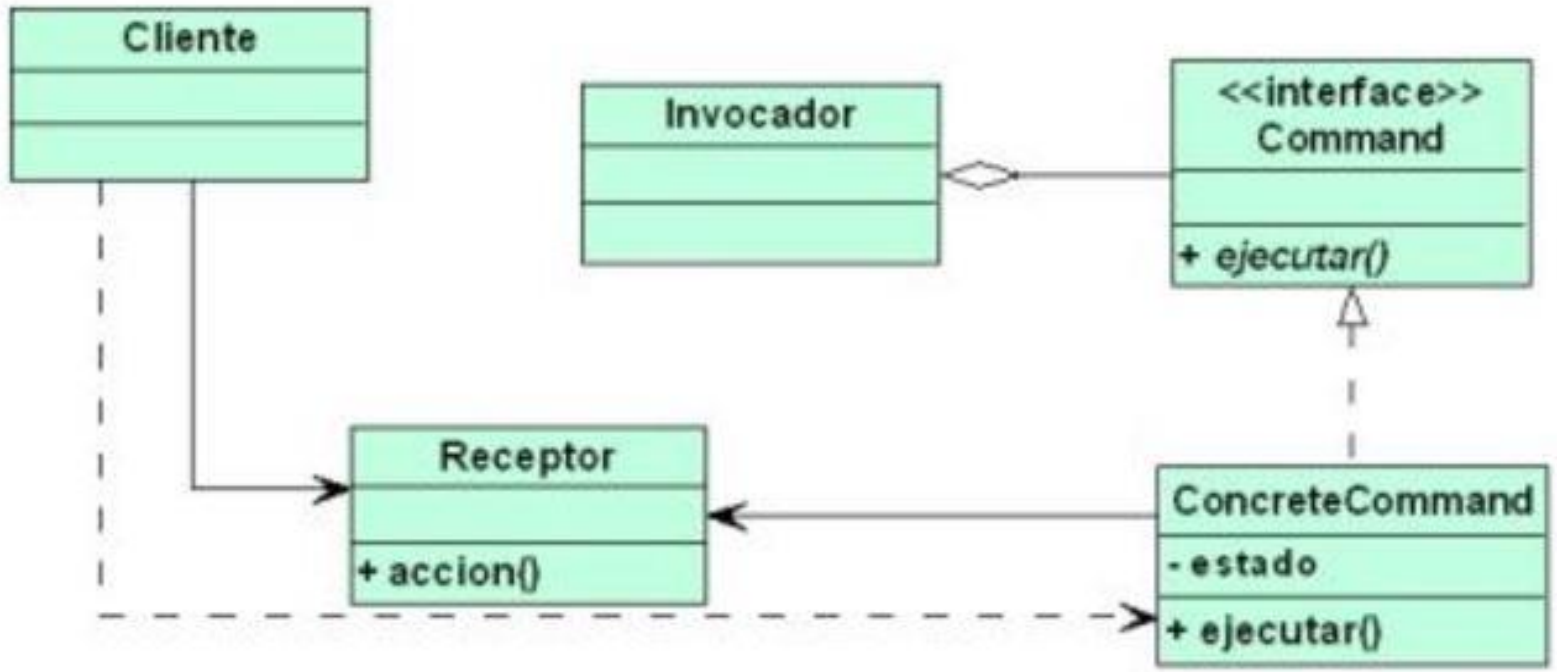
UNIVERSIDAD CATÓLICA
SAN ANTONIO
UCAM

Patrón Command

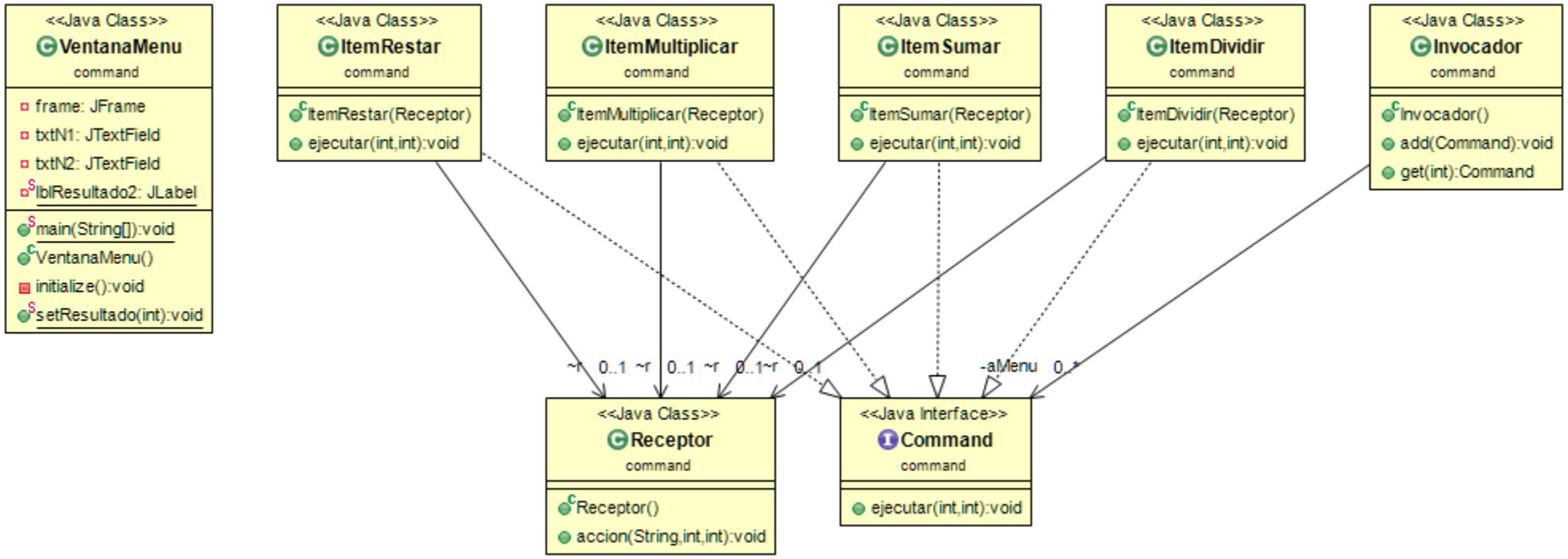
- Propósito: permite encapsular un mensaje como un objeto (de modo que pueda parametrizar otros objetos con distintas peticiones y ofrecer una interfaz común que permite invocar las acciones de forma uniforme) y extender el sistema con nuevas acciones de forma más sencilla.

- Motivación: en ocasiones es necesario enviar un mensaje a un objeto sin conocer el selector del mensaje ni el objeto receptor, por ejemplo los widgets realizan una acción como respuesta a la interacción del usuario.

Patrón Command



Patrón Command. Ejemplo



Patrón Command. Ejemplo

```
private void initialize() {

    //Creamos el invocador
    Invocador objMenu = new Invocador();

    //Creamos el receptor
    Receptor objReceptor = new Receptor();

    //Creamos los distintos comandos concretos
    Command objOpcionSumar = new ItemSumar( objReceptor );
    Command objOpcionRestar = new ItemRestar( objReceptor );
    Command objOpcionMultiplicar = new ItemMultiplicar( objReceptor );
    Command objOpcionDividir = new ItemDividir( objReceptor );

    //Los añadimos al array de opciones del invocador
    objMenu.add( objOpcionSumar );
    objMenu.add( objOpcionRestar );
    objMenu.add( objOpcionMultiplicar );
    objMenu.add( objOpcionDividir );
```

```
public class Invocador
{
    private ArrayList<Command> aMenu = new ArrayList<Command>();

    // --------------------------

    public Invocador() {
    }

    // --------------------------

    public void add( Command objMenuItem )
    {
        this.aMenu.add( objMenuItem );
    }

    // --------------------------

    public Command get( int nOpcion )
    {
        return this.aMenu.get( nOpcion );
    }
}
```

Patrón Command. Ejemplo

```java
package command;
 public interface Command {
     public void ejecutar(int n1, int n2);
}
```

```java
package command;
public class ItemSumar implements Command
{
     Receptor r;
    // --------------------------
    public ItemSumar( Receptor r ) {
        this.r = r;
    }
    // --------------------------
    @Override
     public void ejecutar(int n1, int n2) {
         r.accion("SUMA", n1, n2);
    }
}
```

```java
public class Receptor
{
    public Receptor() {
    }

     public void accion( String accion, int n1, int n2)
    {
        if( accion.compareTo("SUMA") == 0 ) {
            VentanaMenu.setResultado(n1+n2);
        } else if( accion.compareTo("RESTA") == 0 ) {
            VentanaMenu.setResultado(n1-n2);
        } else if( accion.compareTo("MULTIPLICACION") == 0 ) {
            VentanaMenu.setResultado(n1*n2);
        } else if( accion.compareTo("DIVISION") == 0 ){
            VentanaMenu.setResultado(n1/n2);
        } else {
            System.out.println("Opción no válida");
        }
    }
}
```

Patrón Command. Ejemplo

```
btnSumar.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        if(!txtN1.getText().equals("") && !txtN2.getText().equals("")){
            objMenu.get(0).ejecutar(Integer.parseInt(txtN1.getText()), Integer.parseInt(txtN2.getText()));
            lblResultado1.setVisible(true);
        }
    }
});
```