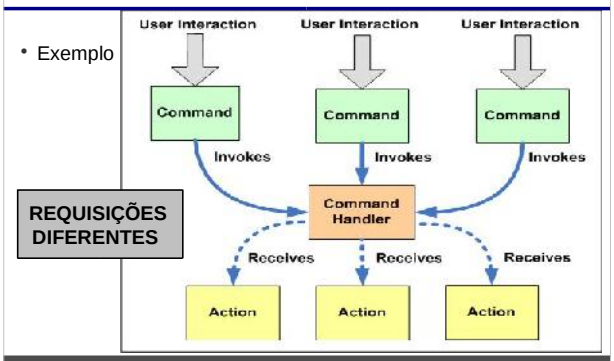
Projeto Detalhado de Software

Padrões de Projetos
(GoF Patterns)

Command

Padrão – Command

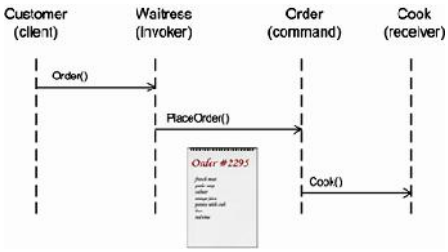
- Exemplo



REQUISIÇÕES DIFERENTES

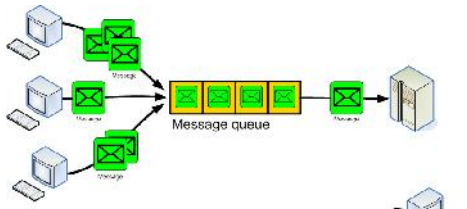
Padrão – Command

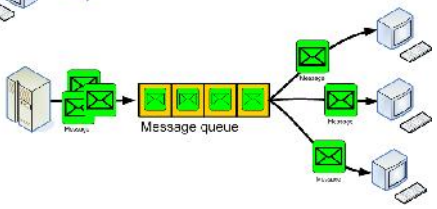
- **Intenção:** Encapsular requisições como objetos. Permite flexibilizar a interação entre objeto cliente (*invoker*) e objeto servidor (*receiver*).



Padrão – Command

- Exemplo





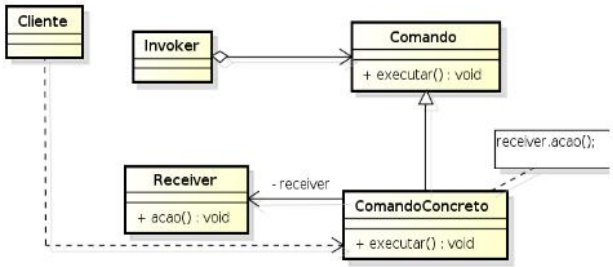
FILA DE REQUISIÇÕES

Padrão – Command

- **Aplicabilidade**
  - Requisições diferentes e futuras
    - Permitir fácil inclusão de novas requisições
  - Fila de requisições
    - Permitir tratamento de requisições concorrentes
  - Registro de requisições
    - Manter histórico
    - Desfazer requisições

Padrão – Command

- **Diagrama conceitual**



Padrão – Command

- **Participantes**
  - ***Comando*** é a interface para execução de uma ação
  - ***ComandoConcreto*** define a ligação entre a ação e o ***Receiver***.
  - ***Cliente*** *(aplicação)* cria o comando concreto e configura o *Receiver*.
  - ***Invoker*** dispara o comando.

Padrão – Command

```
public class Controle {
    private Command[] commands;
    private Command ultimoCommand;

    public Controle(Command abrirCommand, Command fecharCommand){
        this.commands = new Command[2];
        commands[0] = abrirCommand;
        commands[1] = fecharCommand;
    }

    public void abrirPortao(){
        commands[0].execute();
        ultimoCommand = commands[0];
    }

    public void fecharPortao(){
        commands[1].execute();
        ultimoCommand = commands[1];
    }

    public void desfazer(){
        ultimoCommand.undo();
    }
}
```


Padrão – Command

```
public class Portao {
    public final static int ABERTO = 0;
    public final static int FECHADO = 1;

    private int estado = 0;

    public int getEstado() {
        return estado;
    }

    public void setEstado(int estado) {
        this.estado = estado;
    }

    public void abrir(){
        this.estado = ABERTO;
        System.out.println("O portão abriu.");
    }

    public void fechar(){
        this.estado = FECHADO;
        System.out.println("O portão fechou.");
    }
```


Padrão – Command

```
public class AbrirCommand implements Command {
    private Portao portao;

    public AbrirCommand(Portao portao){
        this.portao = portao;
    }

    public void execute(){
        portao.abrir();
    }

    public void undo(){
        portao.fechar();
    }
}
```

```
public interface Command {
    public void execute();
    public void undo();
}
```


Padrão – Command

```
public class AbrirCommand implements Command {
    private Portao portao;

    public AbrirCommand(Portao portao){
        this.portao = portao;
    }

    public void execute(){
        portao.abrir();
    }

    public void undo(){
        portao.fechar();
    }
}
```

```
public interface Command {
    public void execute();
    public void undo();
}
```


Conclusão

- Command

TÍPICA INVENÇÃO PURA PARA EVITAR ACOPLAMENTO DIRETO ENTRE OBJETOS E OS SERVIÇOS

EM SISTEMAS CONCORRENTES, É PEÇA ESSENCIAL NA ARQUITETURA

Referências

- “**Padrões de Projeto: Soluções Reutilizáveis...**” Erich Gamma et al. Padrão Builder e Abstract Factory.
- **“Command Pattern”**
  http://sourcemaking.com/design_patterns/command
- “**Padrões GoF**” Cin/UFPE
  http://www.cin.ufpe.br/~if718/transparencias/pdf/05-padroesGoF.pdf
- http://www.csi.uneb.br/padroes_de_projetos/command_2.html