**Funcionamento**

O padrão Command nos diz como criar “objetos de comando” que encapsula uma solicitação para fazer algo em um objeto específico.

Basicamente todos os objetos de Comando implementam a mesma interface, que consiste em um único método normalmente chamado de execute(). Segue abaixo um exemplo desta implementação.

**Listagem 1:** Exemplo de implementação da interface do Padrão Command

public interface Command {

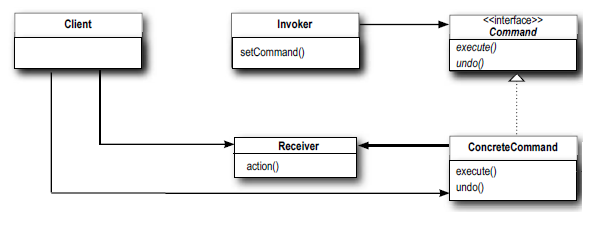
public void execute();

}

O Padrão Command tem como definição encapsular uma solicitação como um objeto, o que lhe permite parametrizar outros objetos com diferentes solicitações, enfileirar ou registrar solicitações e implementar recursos de cancelamento de operações.

Analisando a definição do Padrão Command acima, observa-se que ele cita primeiramente o encapsulamento de uma solicitação, ou seja, o método execute() é o responsável por encapsular essa solicitação a um objeto real, mas isso somente ocorre quando invoca-se o método execute(). Externamente os objetos não sabem quais ações estão sendo executadas no receptor, eles apenas visualizam o método execute() que irá executar as suas solicitações. Quando a definição do padrão fala sobre a parametrização é como se pudéssemos setar diferentes objetos de comando no objeto, isso ficará mais claro quando estudarmos o exemplo de uma implementação nas próximas seções do artigo. A implementação de filas, criação de registros de ações ou o suporte a operações de “refazer” também são suportados pelo padrão Command.

O Diagrama de classe abaixo mostra mais detalhes sobre o funcionamento do padrão Command.



**Figura 1:** Diagrama de classe do Padrão Command

No diagrama de classe acima tem-se o Client que é quem cria um Comando concreto e quem define o receptor, mas devemos atentar para o fato de que o cliente nunca chama o receptor diretamente. O Invoker é quem contém um comando e a qualquer momento chamará o método execute() para que o comando faça alguma ação. A interface Command é padrão para todos os Comandos e define o método execute() que será implementado pelos comandos concretos. O ConcreteCommand tem uma ligação com o Receiver, pois o ConcreteCommand é quem chamará uma ou mais ações do Receiver. O ConcreteCommand será chamado pelo Invoker, mas é ele quem faz as chamadas ao Receiver para executar as ações, Invoker apenas vê o seu método execute(). Por fim, o Receiver é quem contém as ações necessárias para atender as solicitações.

**Exemplo de Implementação**

Segue abaixo um exemplo de implementação em Java utilizando o Padrão Command.

**Listagem 2:** Exemplo de implementação do Padrão Coomand

public class LightOnCommand implements Command {

Light light;

public LightOnCommand(Light light) {

this.light = light;

}

public void execute() {

light.on();

}

}

Acima temos uma simples implementação do padrão Command. Nesse exemplo nota-se a implementação da interface Command que define um método padrão que será sobrescrito pelo Comando e invocado pelo cliente. O método execute() nada mais faz do que simplesmente chamar o método on() no objeto receptor que é a luz que está sendo controlada. Também nota-se que o construtor recebe esse objeto receptor (Luz) que este comando deverá controlar. E se quiséssemos um comando que desligue a luz? Agora é a sua vez, tente implementar se baseando no comando acima que liga a luz.

Após a implementação do objeto de Comando podemos implementar o objeto de controle remoto que receberá um comando que após acionado fará as ações necessárias como ligar uma luz. Segue abaixo a implementação desse controle remoto.

**Listagem 3:** Exemplo de implementação Controle Remoto que invocará o Comando.

public class ControleRemoto {

Command slot;

public ControleRemoto() {

}

public void setCommand(Command command) {

slot = command;

}

public void botaoPressionado() {

slot.execute();

}

}

Podemos notar pela implementação acima (Invoker) que tem-se um atributo slot que armazenará o comando que irá controlar o dispositivo específico. Também tem-se o método setCommand() que serve para definir o comando que o slot irá “controlar“. Se o cliente quiser mudar o comportamento do botão ele poderá chamar esse método setando outro comando dentro. Salientamos que o cliente é quem vai criar o Receiver, o Command e seta-los no Invoker.

**Conclusão**

O Padrão Command desconecta o objeto que faz uma solicitação do objeto que sabe como atender a essa solicitação, sendo que o objeto que está no centro dessa desconexão é um objeto de comando que encapsula um receptor que possui uma determinada ação. Também viu-se que o Invocador chama um Comando através do método execute() que fará as a invocação das ações do Receptor.

Referencia:  
 [Padrão de Projeto Command em Java](http://www.devmedia.com.br/padrao-de-projeto-command-em-java/26456#ixzz3s7tWELnQ) <http://www.devmedia.com.br/padrao-de-projeto-command-em-java/26456#ixzz3s7tWELnQ> .