Design Patterns / Padrões de Projeto
Exercícios – Padrões de Comportamento

1. Chain of Responsibility

1.1. Exercício:

Crie um programa que simule uma máquina de vendas (de refrigerante, salgadinhos, etc.). A máquina possui diversos "slots", cada um capaz de receber um tipo de moeda diferente: 1, 5, 10 centavos, etc. A máquina deve receber moedas e delegar aos slots que capturem-nas. Quando chegar ao valor do produto (ex.: $ 1,00 o refrigerante, $ 2,50 o chips, etc.), a máquina deve entregar o produto e informar o troco.

1.2. Exercício:

Crie uma Chain of Responsibility formada por 15 objetos Handlers, cada um com um número de 1 a 15, cujo método handleRequest() deve receber um número de requisição e imprimir uma mensagem caso o handler decida tratar aquela requisição (imprima o número do handler). A requisição deve ser tratada caso o handler não tenha tratado uma requisição nos últimos 200 milissegundos. Para verificar isso, use o método System.currentTimeMillis(), que retorna o número de milissegundos passados desde o "momento zero" (01/01/1970 00:00:00:000). Caso o handler esteja ainda ocupado com a requisição anterior, passar a requisição para frente na cadeia.

2. Observer

2.1. Exercício:

Em arquivo anexo (pacote br.ufes.inf.designpatterns.behavioral.observer.janelas) está implementado um programa gráfico que contém quatro componentes que manipulam uma mesma classe de modelo. Implemente o padrão Observer utilizando as classes da API Java para este padrão. Você precisará dos seguintes métodos:

- Alterar o valor de uma caixa de seleção (JComboBox): getModel().setSelectedItem(valor);
- Alterar o valor de uma lista (JList): setSelectedIndex(valor);
- Alterar o valor de um campo texto (JTextField): setText("" + valor);
- Alterar o valor de um slider (JSlider): setValue(valor).

2.2. Exercício:

Monte uma estrutura multi-níveis de observadores e observáveis. Crie uma classe que representa um sistema de alarme que monitora diversos sensores. O sistema de alarme, por sua vez, é observado por uma classe que representa a delegacia de polícia e outra que representa a companhia de seguros. Quando um sensor detecta o movimento deve alertar o sistema que, em cadeia, alerta a delegacia e a cia. seguros.

3. Strategy

3.1. Exercício:

Escreva um programa que exiba uma mensagem diferente para cada dia da semana usando o padrão Strategy.

3.2. Exercício:

Em arquivo anexo (pacote br.ufes.inf.designpatterns.behavioral.strategy.sort) estão implementadas quatro formas bastante conhecidas de ordenação: bubble sort, insertion sort, merge sort e quick sort. Coloque-as no padrão Strategy e escreva um cliente que alterna de estratégia de ordenação livremente. Se estiver curioso, cronometre a execução de cada método para verificar qual é o mais eficiente (deve ser usada uma quantidade grande de números no vetor para perceber a diferença).

4. Template Method

4.1. Exercício:

Exercite o padrão Template Method criando uma classe abstrata que lê uma String do console, transforma-a e imprime-a transformada. A transformação é delegada às subclasses. Implemente quatro subclasses, uma que transforme a string toda para maiúsculo, outra que transforme em tudo minúsculo, uma que duplique a string e a última que inverta a string.

4.2. Exercício:

Os Comparators de Java podem ser considerados uma variação do Template Method, apesar de não serem feitos via herança. Monte um vetor de doubles e escreva um comparator que compare os números de ponto-flutuante pelo valor decimal (desconsidere o valor antes da vírgula). Em seguida, use Arrays.sort() para ordenar o vetor e Arrays.toString() para imprimi-la.