**Mais Swing: layout managers, mais componentes e detalhes**

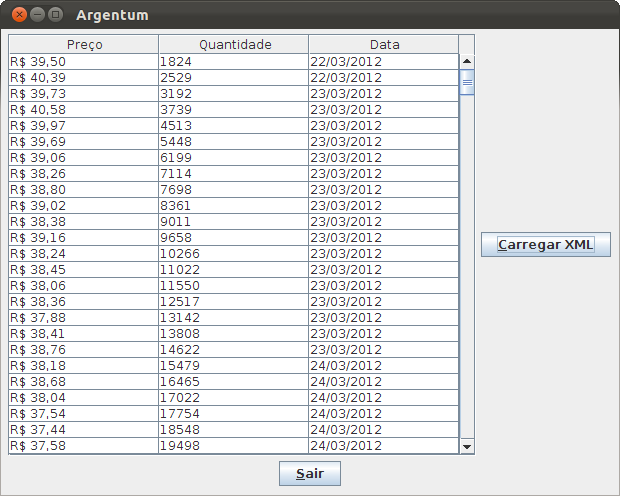
**8.1 - Gerenciadores de Layout**

Até agora, vínhamos adiciona do novos componentes sem dar a informação da localização deles no código. Contudo... como o Java soube onde posicioná-los? Por que sempre são adicionados do lado direito? Se redimensionamos a tela (teste isso), os elementos *fluem* para a linha de cima. Por quê?

As partes do Swing responsáveis por dispor os elementos na tela são os **Layout Managers**, os gerenciadores de layout do Swing/AWT. O Java já vem com uma série de **Layouts** diferentes, que determinam como os elementos serão dispostos na tela, seus tamanhos preferenciais, como eles se comportarão quando a janela for redimensionada e muitos outros aspectos.

Ao escrever uma aplicação Swing, você deve indicar qual Layout Manager você deseja utilizar. Por padrão, é utilizado o FlowLayout que especifica que os elementos devem ser justapostos, que eles devem "fluir" um para baixo do outro quando não couberem lado a lado na tela redimensionada e etc.

Usando o FlowLayout padrão, teste redimensionar a janela de várias formas. Podemos acabar com disposições como essa:

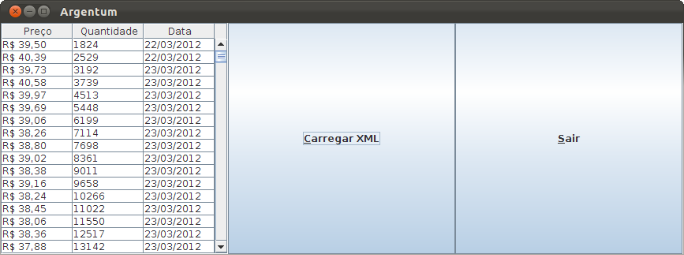


*ArgentumUi com FlowLayout*

Poderíamos usar um outro Layout Manager como o GridLayout, por exemplo. Fazer a mudança é simples. Adicione no método preparaPainelPrincipal:

painelPrincipal.setLayout(new GridLayout());

Mas repare como nossa aplicação fica totalmente diferente:



*ArgentumUi com GridLayout*

Agora os componentes tem tamanho igual (repare que o tamanho que colocamos para a tabela não é respeitado). Note como os elementos parecem estar dispostos em uma grade (um grid). Ao redimensionar essa tela, por exemplo, os elementos não fluem como antes; eles são redimensionados para se adaptarem ao novo tamanho do grid.

Ou ainda, usando o BoxLayout pelo eixo y:

painelPrincipal.setLayout(

new BoxLayout(painelPrincipal, BoxLayout.Y\_AXIS));



*ArgentumUi com BoxLayout*

Agora os botões não são mais redimensionados e a tabela tem seu tamanho redimensionado junto com a janela. Os componentes são dispostos um abaixo do outro pelo eixo Y.

Há uma série de Layout Managers disponíveis no Java, cada um com seu comportamento específico. Há inclusive Layout Managers de terceiros (não-oficiais do Java) que você pode baixar. O projeto **JGoodies**, por exemplo, tem um excelente Layout Manager otimizado para trabalhar com formulários, o FormLayout:

<http://www.jgoodies.com/>

**8.2 - Layout managers mais famosos**

Vimos algumas aplicações de Layout Manager diferentes antes. Vamos ver brevemente as principais características dos layout managers mais famosos:

**FlowLayout**

É o mais simples e o padrão de todos os JPanels. Organiza os componentes um ao lado do outro em linha, da esquerda para a direita, usando o tamanho que você definiu ou, se não houver definição, seus tamanhos preferenciais. Quando a linha fica cheia, uma nova linha é criada.

**BoxLayout**

Organiza os componentes sequencialmente pelo eixo X ou eixo Y (indicamos isso no construtor) usando os tamanhos preferenciais de cada componente.

**GridLayout**

Organiza os componentes em um grid (tabela) com várias linhas e colunas (podemos definir no construtor). Os componentes são colocados um por célula e com tamanho que ocupe a célula toda.

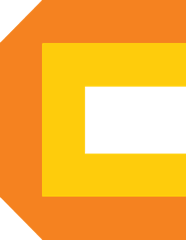
**GridBagLayout**

É o mais complexo layout e é baseado no GridLayout. A idéia também é representar a tela como um grid com linhas e colunas. Mas no GridBagLayout podemos posicionar elementos ocupando várias células em qualquer direção, o que permite layouts mais customizados, mas também causa um custo muito alto de manutenção. A definição de onde deve ser colocado cada componente é feita através de restrições (GridBagConstraints) passadas ao método add.

**BorderLayout**

Divide o container em cinco regiões: Norte, Sul, Leste, Oeste e Centro. Ao adicionar um componente, indicamos a região onde queremos adicioná-lo. Na hora de renderizar, o BorderLayout primeiro coloca os componentes do Norte e Sul em seus tamanhos preferenciais; depois, coloca os do Leste e Oeste também nos tamanhos preferenciais; por último, coloca o componente do Centro ocupando todo o restante do espaço.

**[Nova editora Casa do Código com livros de uma forma diferente](http://www.casadocodigo.com.br?utm_source=Apostila_HTML&utm_campaign=FJ-16&utm_medium=referral" \o "Editora Casa do Código)**

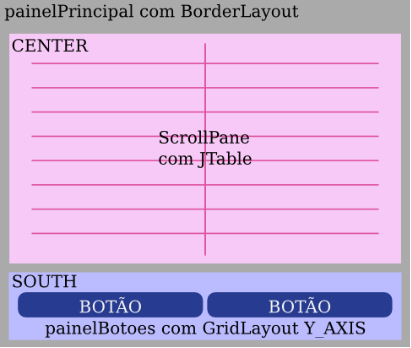
[[](http://www.casadocodigo.com.br/?utm_source=Apostila_HTML&utm_campaign=FJ-16&utm_medium=referral)](http://www.casadocodigo.com.br?utm_source=Apostila_HTML&utm_campaign=FJ-16&utm_medium=referral" \o "Editora Casa do Código)

[Editoras tradicionais pouco ligam para ebooks e novas tecnologias. Não conhecem programação para revisar os livros tecnicamente a fundo. Não têm anos de experiência em didáticas com cursos.  
Conheça a](http://www.casadocodigo.com.br?utm_source=Apostila_HTML&utm_campaign=FJ-16&utm_medium=referral" \o "Editora Casa do Código) **[Casa do Código](http://www.casadocodigo.com.br?utm_source=Apostila_HTML&utm_campaign=FJ-16&utm_medium=referral" \o "Editora Casa do Código)**[, uma editora diferente, com curadoria da](http://www.casadocodigo.com.br?utm_source=Apostila_HTML&utm_campaign=FJ-16&utm_medium=referral" \o "Editora Casa do Código) **[Caelum](http://www.casadocodigo.com.br?utm_source=Apostila_HTML&utm_campaign=FJ-16&utm_medium=referral" \o "Editora Casa do Código)** [e obsessão por livros de qualidade a preços justos.](http://www.casadocodigo.com.br?utm_source=Apostila_HTML&utm_campaign=FJ-16&utm_medium=referral" \o "Editora Casa do Código)

[Casa do Código, ebook com preço de ebook.](http://www.casadocodigo.com.br?utm_source=Apostila_HTML&utm_campaign=FJ-16&utm_medium=referral" \o "Editora Casa do Código)

**8.3 - Exercícios: usando layout managers**

1. Vamos organizar melhor nossos componentes usando alguns *layout managers* que vimos. Veja o esquema:



1. Na classe ArgentumUI, **altere** o método montaTela para chamar nosso novo método (cuidado com a ordem):
2. public void montaTela() {
3. preparaJanela();
4. preparaPainelPrincipal();
5. preparaTabela();
6. preparaPainelBotoes(); // adicione essa linha!
7. preparaBotaoCarregar();
8. preparaBotaoSair();
9. mostraJanela();
10. }

A chamada ao preparaPainelBotoes vai ficar sublinhada em vermelho, indicando que o método ainda não existe. Na linha do erro, use o ctrl + 1 e escolha a opção de **criar o método**preparaPainelBotoes. Adicione as seguintes linhas de implementação:

private void preparaPainelBotoes() {

painelBotoes = new JPanel(new GridLayout());

painelPrincipal.add(painelBotoes);

}

Agora, quem vai indicar erro é o atributo painelBotoes. Basta usar o Ctrl + 1 e escolher a opção de colocar a variável num novo atributo (*Create field painelBotoes*). Ele vai inserir **automaticamente** na lista de atributos o código:

private JPanel painelBotoes;

**Altere** os métodos preparaBotaoCarregar e preparaBotaoSair que adicionam os botões para adicionar ambos ao novo painelBotoes. Por exemplo, troque:

painelPrincipal.add(botaoCarregar);

Por:

painelBotoes.add(botaoCarregar);

Rode a classe ArgentumUI. Repare que, ao redimensionar, os botões não mais "escorregam" separadamente - eles agora estão agrupados.

1. Próximo passo: usar o BorderLayout para posicionar a tabela e os botões corretamente.

**Altere** o método preparaPainelPrincipal adicionando a chamada ao layout:

private void preparaPainelPrincipal() {

painelPrincipal = new JPanel();

painelPrincipal.setLayout(new BorderLayout());

janela.add(painelPrincipal);

}

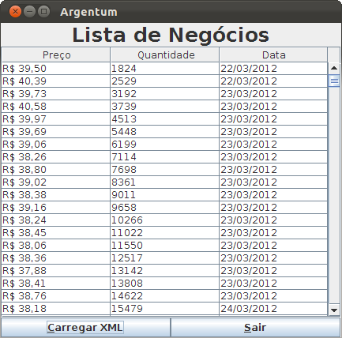
**Altere** o método preparaTabela para indicar que queremos adicioná-la ao centro da tela. Basta **adicionar um parâmetro** na chamada ao método add:

painelPrincipal.add(scroll, BorderLayout.CENTER);

**Altere** o método preparaPainelBotoes para indicar que queremos adicioná-lo na região sul. Como o acima, basta apenas **adicionar um parâmetro** na chamada ao método add:

painelPrincipal.add(painelBotoes, BorderLayout.SOUTH);

Rode a aplicação novamente e veja a diferença em relação a nossa tela anterior.



1. (opcional) Se você fez o título no exercício anterior, você deve ter notado que o título sumiu! Para trazê-lo de volta, acrescente a opção BorderLayout.NORTH similarmente ao que foi feito nos itens anteriores.

**8.4 - Integrando JFreeChart**

No capítulo anterior, desvendamos o JFreeChart e criamos toda a infraestrutura necessária para criar gráficos complexos para nossa análise técnica. Agora, vamos integrar esses gráficos à nossa aplicação.

Lembre que, na classe GeradorDeGrafico, já criamos um método getPanel que devolve um componente pronto para ser exibido no Swing. O que vamos fazer é gerar o gráfico logo após a leitura do XML.

E, para exibir o gráfico em nossa interface, vamos organizar tudo com abas (*tabs*). Usando um JTabbedPane vamos organizar a tabela e o gráfico, cada um em uma aba diferente. Usar um JTabbedPane é muito fácil:

JTabbedPane abas = new JTabbedPane();

abas.add("Label 1", componenteDa1aAba);

abas.add("Label 2", componenteDa2aAba);

Para gerar o gráfico baseado na lista de negócios, usaremos toda a infraestrutura que fizemos até agora: obteremos a lista de negócios do XML e mandaremos a CandlestickFactory construir as candles para nós. Então, criaremos uma SerieTemporal com elas e a passaremos para o GeradorDeGrafico. Depois de plotar os indicadores, conseguiremos o painel com gráfico para adicionar à aba!

O código disso tudo ficará parecido com esse:

List<Negocio> negocios = ...

CandlestickFactory fabrica = new CandlestickFactory();

List<Candle> candles = fabrica.constroiCandles(negocios);

SerieTemporal serie = new SerieTemporal(candles);

GeradorDeGrafico gg =

new GeradorDeGrafico(serie, 2, serie.getTotal() - 1);

gg.plotaIndicador(new MediaMovelSimples());

gg.plotaIndicador(new IndicadorFechamento());

JPanel grafico = gg.getPanel();

**8.5 - Exercícios: completando a tela da nossa aplicação**

1. Na classe ArgentumUI, **adicione** a chamada ao preparaAbas no método montaTela (cuidado com a ordem):
2. public void montaTela() {
3. preparaJanela();
4. preparaPainelPrincipal();
5. preparaAbas();
6. preparaTabela();
7. preparaPainelBotoes();
8. preparaBotaoCarregar();
9. preparaBotaoSair();
10. mostraJanela();
11. }

Note que **a linha ficará vermelha**. Novamente, use o ctrl + 1 para criar o método automaticamente.

1. **Adicione** a implementação ao método preparaAbas() recém-criado.
2. private void preparaAbas() {
3. abas = new JTabbedPane();
4. abas.addTab("Tabela", null);
5. abas.addTab("Gráfico", null);
6. painelPrincipal.add(abas);
7. }

**Altere** o método preparaTabela para colocar a tabela com scroll na primeira aba do painel abas. Comente (ou remova) a linha que adiciona o scroll ao painelPrincipal e adicione a chamada ao abas.setComponentAt como mostrado abaixo:

private void preparaTabela() {

tabela = new JTable();

JScrollPane scroll = new JScrollPane();

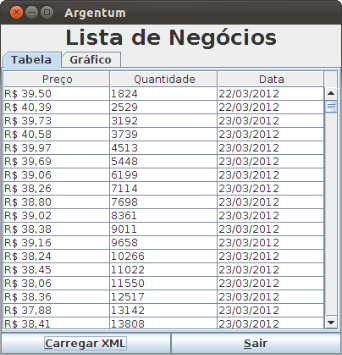
scroll.getViewport().add(tabela);

// painelPrincipal.add(scroll, BorderLayout.CENTER);

abas.setComponentAt(0, scroll);

}

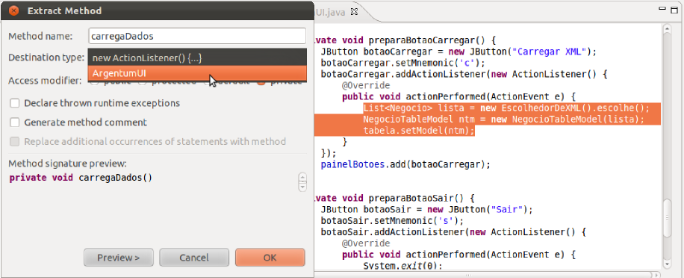
Rode novamente e observe a montagem das abas.



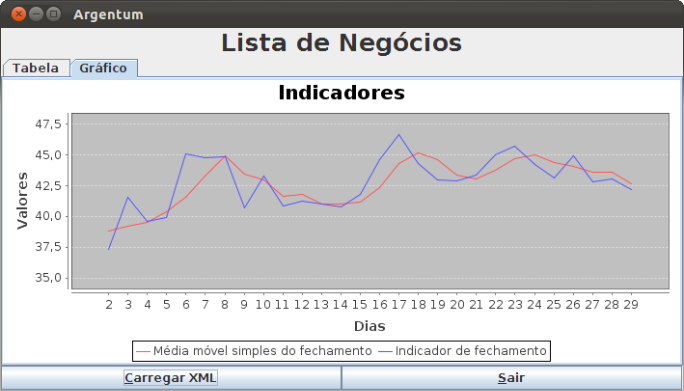
1. Precisamos alterar a classe anônima que trata o evento do botão de carregar o XML na classe ArgentumUI para gerar o gráfico através das classes que criamos. Mas se colocarmos todo esse código dentro da classe anônima, com certeza perderemos legibilidade.

Vamos então criar um **método auxiliar** chamado carregaDados na classe ArgentumUI, que será responsável por tratar o evento de carregar tanto a **tabela** quanto o **grafico**.

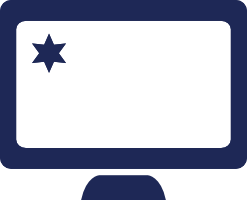
Comecemos isolando as linhas que já existem no método actionPerformed do botaoCarregar em um método da classe ArgentumUI. Para isso, selecione as linhas de dentro desse método e faça **alt + shift + M**. Na janelinha que abrir, dê o nome para esse método (carregaDados) e **não esqueça** de mudar o *Destination type* para que ele seja declarado na própria ArgentumUI.



1. Agora, só falta preencher o método carregaDados criado com a construção das candles, do gerador de gráfico e adicionar o painel à segunda aba! **Complete** o método como indicado abaixo (note que você já criou o método e já tem a implementação até a linha 4:
2. private void carregaDados() {
3. List<Negocio> lista = new EscolhedorDeXML().escolhe();
4. NegocioTableModel ntm = new NegocioTableModel(lista);
5. tabela.setModel(ntm);
7. CandlestickFactory fabrica = new CandlestickFactory();
8. List<Candle> candles = fabrica.constroiCandles(lista);
9. SerieTemporal serie = new SerieTemporal(candles);
11. GeradorDeGrafico gerador = new GeradorDeGrafico(serie, 2, serie.getTotal() - 1);
12. gerador.plotaIndicador(new MediaMovelSimples());
13. gerador.plotaIndicador(new IndicadorFechamento());
15. abas.setComponentAt(1, gerador.getPanel());
16. }
17. Rode novamente e teste o gráfico e as abas!



**[Já conhece os cursos online Alura?](http://www.alura.com.br" \o "Cursos online Alura)**

[[](http://www.alura.com.br/)](http://www.alura.com.br" \o "Cursos online Alura)

[A](http://www.alura.com.br" \o "Cursos online Alura) **[Alura](http://www.alura.com.br" \o "Cursos online Alura)** [oferece dezenas de](http://www.alura.com.br" \o "Cursos online Alura) **[cursos online](http://www.alura.com.br" \o "Cursos online Alura)** [em sua plataforma exclusiva de ensino que favorece o aprendizado com a](http://www.alura.com.br" \o "Cursos online Alura) **[qualidade](http://www.alura.com.br" \o "Cursos online Alura)** [reconhecida da Caelum. Você pode escolher um curso nas áreas de Java, Ruby, Web, Mobile, .NET e outros, com uma](http://www.alura.com.br" \o "Cursos online Alura) **[assinatura](http://www.alura.com.br" \o "Cursos online Alura)** [que dá acesso a todos os cursos.](http://www.alura.com.br" \o "Cursos online Alura)

[Conheça os cursos online Alura.](http://www.alura.com.br" \o "Cursos online Alura)

**8.6 - Indicadores mais Elaborados e o Design Pattern Decorator**

Vimos no capítulo anterior que os analistas financeiros fazem suas análises sobre indicadores mais elaborados, como por exemplo *Médias Móveis*, que são *calculadas* a partir de outros indicadores. No momento, nossos algoritmos de médias móveis sempre calculam seus valores sobre o preço de fechamento. Mas, e se quisermos calculá-las a partir de outros indicadores? Por exemplo, o que faríamos se precisássemos da *média móvel simples do preço máximo*, da abertura ou de outro indicador qualquer?

Criaríamos classes como MediaMovelSimplesAbertura e MediaMovelSimplesMaximo? Que código colocaríamos lá? Provavelmente, copiaríamos o código que já temos e apenas trocaríamos a chamada do getFechamento pelo getAbertura e getVolume.

A maior parte do código seria a mesma e não estamos reaproveitando código - copiar e colar código não é reaproveitamento, é uma forma de nos dar dor de cabeça no futuro ao ter que manter 2 códigos idênticos em lugares diferentes.

Queremos calcular médias móveis de fechamento, abertura, volume, etc, sem precisar copiar essas classes de média. Na verdade, o que queremos é calcular a média móvel baseado em algum *outro indicador*. Já temos a classe IndicadorFechamento e é trivial implementar outros como IndicadorAbertura, IndicadorMinimo, etc.

A MediaMovelSimples é um Indicador que vai depender de algum *outro*Indicador para ser calculada (por exemplo o IndicadorFechamento). Queremos chegar em algo assim:

MediaMovelSimples mms = new MediaMovelSimples(

new IndicadorFechamento());

// ... ou ...

MediaMovelSimples mms = new MediaMovelPonderada(

new IndicadorFechamento());

Repare na flexibilidade desse código. O cálculo de média fica totalmente independente do dado usado e, toda vez que criarmos um novo indicador, já ganhamos a média móvel desse novo indicador de brinde. Vamos fazer então nossa classe de média receber algum outro Indicador:

public class MediaMovelSimples implements Indicador {

private final Indicador outroIndicador;

public MediaMovelSimples(Indicador outroIndicador) {

this.outroIndicador = outroIndicador;

}

// ... calcula ...

}

E, dentro do método calcula, em vez de chamarmos o getFechamento, delegamos a chamada para o outroIndicador:

@Override

public double calcula(int posicao, SerieTemporal serie) {

double soma = 0.0;

for (int i = posicao - 2; i <= posicao; i++) {

soma += outroIndicador.calcula(i, serie);

}

return soma / 3;

}

Nossa classe MediaMovelSimples recebe um outro indicador que **modifica um pouco** os valores de saída - ele complementa o algoritmo da média! Passar um objeto que modifica um pouco o comportamento do seu é uma solução clássica para ganhar em **flexibilidade** e, como muitas soluções clássicas, ganhou um nome nos design patterns de **Decorator**.

**Também é um composite!**

Note que, agora, nossa MediaMovelSimples**é um**Indicador e também **tem um** outro Indicador. Já vimos antes outro tipo que se comporta da mesma forma, você se lembra?

Assim como os componentes do Swing, nossa MediaMovelSimples se tornou **também** um exemplo de *Composite*.

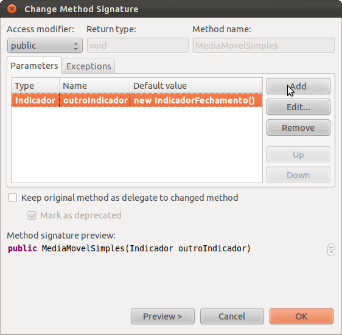
**8.7 - Exercícios: Indicadores mais espertos e o Design Pattern Decorator**

1. Faremos uma grande mudança agora: nossas médias móveis devem receber como argumento um outro indicador, formando o *design pattern* Decorator, como visto no texto.

Para isso, crie o construtor padrão (sem parâmetros) da MediaMovelSimples usando o atalho de sua preferência:

* 1. Comece a escrever o nome da classe e mande autocompletar: Med**<ctrl + espaço>**;
  2. Use o criador automatico de construtores: **ctrl + 3***Constructor*.

1. Modifique o método usando o atalho **alt + shift + C** para adicionar o parâmetro do tipo Indicador chamado outroIndicador e com valor padrão new IndicadorFechamento().



Agora, com o cursor sobre o parâmetro outroIndicador, faça **ctrl + 1** e guarde esse valor em um novo atributo, selecionando *assign parameter to new field*.

1. Troque a implementação do método calcula para chamar o calcula do outroIndicador:
2. public double calcula(int posicao, SerieTemporal serie) {
3. double soma = 0.0;
4. for (int i = posicao - 2; i <= posicao; i++) {
5. soma += outroIndicador.calcula(i, serie);
6. }
7. return soma / 3;
8. }
9. Lembre que toda refatoração **deve** ser acompanhada dos testes correspondentes. Mas ao usar a refatoração do Eclipse no construtor da nossa classe MediaMovelSimples, a IDE evitou que quebrassemos os testes já passando o IndicadorFechamento como parâmetro padrão para todos eles!

Agora, rode os testes novamente e tudo **deve** continuar se comportando exatamente como antes da refatoração. Caso contrário, nossa refatoração não foi bem sucedida e seria bom reverter o processo todo.

1. Agora, para que o JFreeChart consiga plotar uma linha do gráfico para cada uma das médias, ainda é preciso que o toString da MediaMovelSimples seja, também, decorado, isto é, é preciso modificá-lo para:
2. public String toString() {
3. return "Média móvel simples do " + outroIndicador;
4. }
5. (opcional) Se você fez a MediaMovelPonderada, modifique também essa outra classe para também ter um *Decorator*.

Isto é, faça ela também receber um outroIndicador no construtor e delegar a chamada a esse indicador no seu método calcula, assim como fizemos com a MediaMovelSimples.

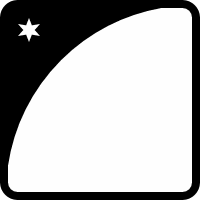
**Não esqueça** de alterar também o toString da MediaMovelPonderada.

1. (opcional) Crie uma outra classe IndicadorAbertura e faça um teste unitário na classe MediaMovelSimplesTest que use esse novo indicador em vez do de fechamento.

Faça também para máximo, mínimo ou outros que desejar.

**8.8 - Discussão em sala de aula: uso de IDEs para montar a tela**

**[Você não está nessa página a toa](http://www.caelum.com.br/curso-java-testes-xml-design-patterns/?utm_source=Apostila_HTML&utm_campaign=FJ-16&utm_medium=referral" \o "Curso Laboratório Java com Testes, XML e Design Patterns)**

[[](http://www.caelum.com.br/curso-java-testes-xml-design-patterns/?utm_source=Apostila_HTML&utm_campaign=FJ-16&utm_medium=referral)](http://www.caelum.com.br/curso-java-testes-xml-design-patterns/?utm_source=Apostila_HTML&utm_campaign=FJ-16&utm_medium=referral" \o "Curso Laboratório Java com Testes, XML e Design Patterns)

[Você chegou aqui porque a Caelum é referência nacional em cursos de Java, Ruby, Agile, Mobile, Web e .NET.  
Faça curso com](http://www.caelum.com.br/curso-java-testes-xml-design-patterns/?utm_source=Apostila_HTML&utm_campaign=FJ-16&utm_medium=referral" \o "Curso Laboratório Java com Testes, XML e Design Patterns) **[quem escreveu essa apostila](http://www.caelum.com.br/curso-java-testes-xml-design-patterns/?utm_source=Apostila_HTML&utm_campaign=FJ-16&utm_medium=referral" \o "Curso Laboratório Java com Testes, XML e Design Patterns)**[.](http://www.caelum.com.br/curso-java-testes-xml-design-patterns/?utm_source=Apostila_HTML&utm_campaign=FJ-16&utm_medium=referral" \o "Curso Laboratório Java com Testes, XML e Design Patterns)

[Consulte as vantagens do curso](http://www.caelum.com.br/curso-java-testes-xml-design-patterns/?utm_source=Apostila_HTML&utm_campaign=FJ-16&utm_medium=referral" \o "Curso Laboratório Java com Testes, XML e Design Patterns) *[Laboratório Java com Testes, XML e Design Patterns](http://www.caelum.com.br/curso-java-testes-xml-design-patterns/?utm_source=Apostila_HTML&utm_campaign=FJ-16&utm_medium=referral" \o "Curso Laboratório Java com Testes, XML e Design Patterns)*[.](http://www.caelum.com.br/curso-java-testes-xml-design-patterns/?utm_source=Apostila_HTML&utm_campaign=FJ-16&utm_medium=referral" \o "Curso Laboratório Java com Testes, XML e Design Patterns)

Capítulo anterior:

[Gráficos com JFreeChart](http://www.caelum.com.br/apostila-java-testes-xml-design-patterns/graficos-com-jfreechart/)

Próximo capítulo:

[Reflection e Annotations](http://www.caelum.com.br/apostila-java-testes-xml-design-patterns/reflection-e-annotations/)

**Você encontra a Caelum também em:**

**[Blog Caelum](http://blog.caelum.com.br/?utm_source=Apostila_HTML&utm_campaign=FJ-16&utm_medium=referral)**

**[Cursos Online](http://www.alura.com.br/)**

**[Facebook](http://facebook.com/caelumbr)**

**[Newsletter](http://www.caelum.com.br/newsletter/?utm_source=Apostila_HTML&utm_campaign=FJ-16&utm_medium=referral)**

**[Casa do Código](http://casadocodigo.com.br/?utm_source=Apostila_HTML&utm_campaign=FJ-16&utm_medium=referral)**

**[Twitter](http://twitter.com/caelum)**

<img height="1" width="1" border="0" alt="" src="http://googleads.g.doubleclick.net/pagead/viewthroughconversion/995678627/?frame=0&random=1395954469600&cv=7&fst=1395954469600&num=1&fmt=1&label=DYcgCO3T\_gMQo7Pj2gM&guid=ON&u\_h=1080&u\_w=1920&u\_ah=1040&u\_aw=1920&u\_cd=24&u\_his=1&u\_tz=-180&u\_java=true&u\_nplug=13&u\_nmime=54&frm=0&url=http%3A//www.caelum.com.br/apostila-java-testes-xml-design-patterns/mais-swing-layout-managers-mais-componentes-e-detalhes/&ref=http%3A//www.google.com.br/url%3Fsa%3Dt%26rct%3Dj%26q%3D%26esrc%3Ds%26source%3Dweb%26cd%3D1%26cad%3Drja%26uact%3D8%26ved%3D0CCwQFjAA%26url%3Dhttp%253A%252F%252Fwww.caelum.com.br%252Fapostila-java-testes-xml-design-patterns%252Fmais-swing-layout-managers-mais-componentes-e-detalhes%252F%26ei%3DZ5M0U4aaLObRsQTw04CoDQ%26usg%3DAFQjCNEfn7GIobcNMHV9RvLSxH9L6xxQiQ%26bvm%3Dbv.63808443%2Cd.cWc" />