CAPÍTULO 8

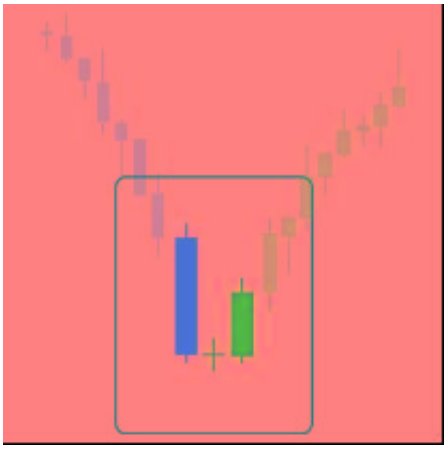
Refatoração: os Indicadores da bolsa

"Nunca confie em um computador que você não pode jogar pela janela."
— Steve Wozniak

8.1 - ANÁLISE TÉCNICA DA BOLSA DE VALORES

Munehisa Homna, no século 18, foi quem começou a pesquisar os preços antigos do arroz para reconhecer padrões. Ele fez isso e começou a criar um catálogo grande de figuras que se repetiam.

A estrela da manhã, *Doji*, da figura abaixo, é um exemplo de figura sempre muito buscada pelos analistas:



Ela indica um padrão de reversão. Dizem que quando o preço de abertura e fechamento é praticamente igual (a estrela), essa é uma forte indicação de que o mercado se inverta, isto é, se estava em uma grande **baixa**, tenderá a **subir** e, se estava em uma grande **alta**, tenderá a **cair**.

Baseada nessas ideias, surgiu a **Análise Técnica Grafista**: uma escola econômica que tem como objetivo avaliar o melhor momento para compra e venda de ações através da análise histórica e comportamental do ativo na bolsa.

Essa forma de análise dos dados gerados sobre dados das negociações (preço, volume, etc), usa gráficos na busca de padrões e faz análise de tendências para tentar prever o comportamento futuro de uma ação.

A análise técnica surgiu no início do século 20, com o trabalho de Charles Dow e Edward Jones. Eles criaram a empresa Dow Jones & Company e foram os primeiros a inventarem índices para tentar prever o comportamento do mercado de ações. O primeiro índice era simplesmente uma média ponderada de 11 ativos famosos da época, que deu origem ao que hoje é conhecido como Dow-Jones.

A busca de padrões nos candlesticks é uma arte. Através de critérios subjetivos e formação de figuras, analistas podem determinar, com algum grau de acerto, como o mercado se comportará dali para a frente.

8.2 - Indicadores Técnicos

Uma das várias formas de aplicar as premissas da análise técnica grafista é através do uso de indicadores técnicos. **Indicadores** são fórmulas que manipulam dados das negociações e tiram valores deles em busca de informações interessantes para recomendar as próximas ações para um ativo. Esse novo número, é determinístico e de fácil cálculo por um computador. É até de praxe que analistas financeiros programem diversas dessas fórmulas em macros VBScript, para vê-las dentro do Excel.

É comum, na mesma visualização, termos uma combinação de gráficos, indicadores e até dos *candles*:

Diversos livros são publicados sobre o assunto e os principais *homebrokers* fornecem softwares que traçam esses indicadores e muitos outros. Além disso, você encontra uma lista com os indicadores mais usados e como calculá-los em: http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators

8.3 - As médias móveis

Há diversos tipos de médias móveis usadas em análises técnicas e elas são frequentemente usadas para investidores que fazem compras/vendas em intervalos muito maiores do que o intervalo de recolhimento de dados para as *candles*. As médias mais famosas são a simples, a ponderada, a exponencial e a Welles Wilder.

Vamos ver as duas primeiras, a média móvel simples e a média móvel ponderada.

Média móvel simples

A média móvel simples calcula a média aritmética de algum dos valores das candlesticks do papel para um determinado intervalo de tempo -- em geral, o valor de fechamento. Basta pegar todos os valores, somar e dividir pelo número de dias.

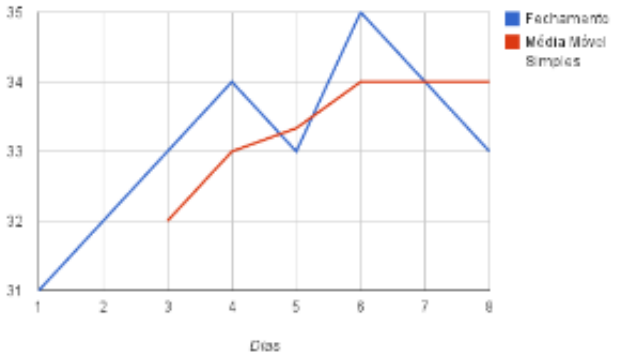
A figura a seguir mostra duas médias móveis simples: uma calculando a média dos últimos 50 dias e outra dos últimos 200 dias. O gráfico é do valor das ações da antiga Sun Microsystems em 2001.

Repare que a média móvel mais 'curta', a de 50 dias, responde mais rápido aos movimentos atuais da ação, mas pode gerar sinais pouco relevantes a médio prazo.

Usualmente, estamos interessados na média móvel dos **últimos** `N` dias e queremos definir esse dia inicial. Por exemplo, para os dados de fechamento abaixo:

```
DIA     FECHAMENTO
dia 1:     31
dia 2:     32
dia 3:     33
dia 4:     34
dia 5:     33
dia 6:     35
dia 7:     34
dia 8:     33
```

Vamos fazer as contas para que o indicador calcule a média para os 3 dias anteriores ao dia que estamos interessados. Por exemplo: se pegamos o **dia 6**, a média móvel simples para os últimos 3 dias é a soma do dia 4 ao dia 6: (34 + 33 + 35) / 3 = **34**. A média móvel do dia 3 para os últimos 3 dias é 2: (31 + 32 + 33) / 3. E assim por diante.



O gráfico anterior das médias móveis da Sun pega, para cada dia do gráfico, a média dos 50 dias anteriores.

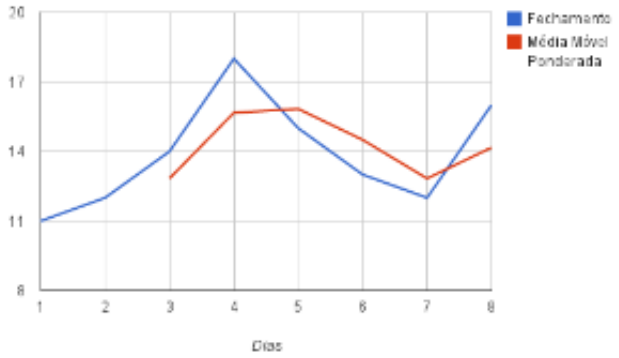
Média móvel ponderada

Outra média móvel muito famosa é a ponderada. Ela também leva em conta os últimos `N` dias a partir da data a ser calculada. Mas, em vez de uma média aritmética simples, faz-se uma média ponderada onde damos mais *peso* para o valor mais recente e vamos diminuindo o peso dos valores conforme movemos para valores mais antigos.

Por exemplo, para os dias a seguir:

```
DIA     FECHAMENTO
dia 1:     11
dia 2:     12
dia 3:     14
dia 4:     18
dia 5:     15
dia 6:     13
dia 7:     12
dia 8:     16
```

Vamos calcular a média móvel para os últimos 3 dias, onde hoje tem peso 3, ontem tem peso 2 e anteontem tem peso 1. Se calcularmos a média móvel ponderada para o dia 6 temos: (13*3 + 15*2 + 18*1) / 6 = **14.50**.



A média ponderada nos dá uma visão melhor do que está acontecendo no momento com a cotação da minha ação, mostrando com menos importância os resultados "atrasados", portanto menos relevantes para minhas decisões de compra e venda atuais. Essa média, contudo, não é tão eficiente quando estudamos uma série a longo prazo.

8.4 - Exercícios: criando indicadores

1. O cálculo de uma média móvel é feito a partir de uma lista de resumos do papel na bolsa. No nosso caso, vamos pegar vários Candlesticks, um para cada dia, e usar seus valores de fechamento.

Para encapsular a lista de candles e aproximar a nomenclatura do código à utilizada pelo cliente no dia a dia, vamos criar a classe `SerieTemporal` no pacote `br.com.caelum.argentum.modelo`:

```
public class SerieTemporal {

  private final List<Candlestick> candles;

  public SerieTemporal(List<Candlestick> candles) {
    this.candles = candles;
  }

  public Candlestick getCandle(int i) {
    return this.candles.get(i);
  }

  public int getUltimaPosicao() {
    return this.candles.size() - 1;
  }
}
```

2. Vamos criar a classe `MediaMovelSimples`, dentro do novo pacote `br.com.caelum.argentum.indicadores`. Essa classe terá o método `calcula` que

recebe a posição a ser calculada e a `SerieTemporal` que proverá os candles. Então, o método devolverá a média simples dos fechamentos dos dois dias anteriores e o atual.

Comece fazendo apenas o cabeçalho desse método:

```
public class MediaMovelSimples {

  public double calcula(int posicao, SerieTemporal serie) {
    return 0;
  }

}
```

A ideia é passarmos para o método calcula a `SerieTemporal` e o dia para o qual queremos calcular a média móvel simples. Por exemplo, se passarmos que queremos a média do dia 6 da série, ele deve calcular a média dos valores de fechamento dos dias 6, 5 e 4 (já que nosso intervalo é de 3 dias).

Como essa é uma lógica um pouco mais complexa, **começaremos essa implementação pelos testes**.

3. Seguindo a ideia do TDD, faremos o teste antes mesmo de implementar a lógica da média. Assim como você já vem fazendo, use o `ctrl + N` -> *JUnit Test Case* para criar a classe de teste `MediaMovelSimplesTest` na *source folder* `src/test/java`, pacote `br.com.caelum.argentum.indicadores`.

Então, crie um teste para verificar que a média é calculada corretamente para a sequência de fechamentos `1, 2, 3, 4, 3, 4, 5, 4, 3`.

Note que, para fazer tal teste, será necessário criar uma série temporal com candles cujo fechamento tenha tais valores. Criaremos uma outra classe para auxiliar nesses testes logo em seguida. Por hora, não se preocupe com o erro de compilação da 5a. linha do código abaixo:

```
1  public class MediaMovelSimplesTest {
2
3    @Test
4    public void sequenciaSimplesDeCandles() throws Exception {
5      SerieTemporal serie =
6          GeradorDeSerie.criaSerie(1, 2, 3, 4, 3, 4, 5, 4, 3);
7      MediaMovelSimples mms = new MediaMovelSimples();
8
9      Assert.assertEquals(2.0, mms.calcula(2, serie), 0.00001);
10     Assert.assertEquals(3.0, mms.calcula(3, serie), 0.00001);
11     Assert.assertEquals(10.0/3, mms.calcula(4, serie), 0.00001);
12     Assert.assertEquals(11.0/3, mms.calcula(5, serie), 0.00001);
13     Assert.assertEquals(4.0, mms.calcula(6, serie), 0.00001);
```

```
14      Assert.assertEquals(13.0/3, mms.calcula(7, serie), 0.00001);
15      Assert.assertEquals(4.0, mms.calcula(8, serie), 0.00001);
16    }
17  }
```

4. Ainda é necessário fazer esse código compilar! Note que, pelo que escrevemos, queremos chamar um método estático na classe `GeradorDeSerie` que receberá diversos valores e devolverá a série com Candles respectivas.

Deixe que o Eclipse o ajude a criar essa classe: use o `ctrl + 1` e deixe que ele crie a classe para você e, então, adicione a ela o método `criaSerie`, usando o recurso de *varargs* do Java para receber diversos `doubles`.

> **Varargs**
>
> A notação `double... valores` (com os três pontinhos mesmo!) que usaremos no método a seguir é indicação do uso de *varargs*. Esse recurso está presente desde o Java 5 e permite que chamemos o método passando de zero a quantos doubles quisermos! Dentro do método, esses argumentos serão interpretados como um array.
>
> Varargs vieram para oferecer uma sintaxe mais amigável nesses casos. Antigamente, quando queríamos passar um número variável de parâmetros de um mesmo tipo para um método, era necessário construir um array com esses parâmetros e passá-lo como parâmetro.
>
> Leia mais sobre esse recurso em:
> http://docs.oracle.com/javase/1.5.0/docs/guide/language/varargs.html

Na classe `GeradorDeSerie`, faça o seguinte método:

Atenção: não é necessário copiar o JavaDoc.

```
1  /**
2   * Serve para ajudar a fazer os testes.
3   *
4   * Recebe uma sequência de valores e cria candles com abertura,
fechamento,
5   * minimo e maximo iguais, mil de volume e data de hoje. Finalmente,
devolve
6   * tais candles encapsuladas em uma Serie Temporal.
7   **/
8  public static SerieTemporal criaSerie(double... valores) {
9    List<Candlestick> candles = new ArrayList<Candlestick>();
10   for (double d : valores) {
```

```
    candles.add(new Candlestick(d, d, d, d, 1000,
                    Calendar.getInstance()));
  }
  return new SerieTemporal(candles);
}
```

Agora que ele compila, rode a classe de teste. **Ele falha**, já que a implementação padrão simplesmente devolve zero!

5. **Volte** à classe principal `MediaMovelSimples` e **implemente** agora a lógica de negócio do método `calcula`, que já existe. O método deve ficar parecido com o que segue:

```
public class MediaMovelSimples {

  public double calcula(int posicao, SerieTemporal serie) {
    double soma = 0.0;
    for (int i = posicao; i > posicao - 3; i--) {
      Candlestick c = serie.getCandle(i);
      soma += c.getFechamento();
    }
    return soma / 3;
  }
}
```

Repare que iniciamos o `for` com `posicao` e iteramos com `i--`, **retrocedendo** nas posições enquanto elas forem **maiores** que os três dias de intervalo. Isso significa que estamos calculando a média móvel apenas dos últimos 3 dias.

Mais para frente, existe um exercício opcional para parametrizar esse valor.

6. Crie a classe `MediaMovelPonderada` análoga a `MediaMovelSimples`. Essa classe dá peso 3 para o dia atual, peso 2 para o dia anterior e o peso 1 para o dia antes desse. O código interno é muito parecido com o da média móvel simples, só precisamos multiplicar sempre pela quantidade de dias passados.

A implementação do método `calcula` deve ficar bem parecida com isso:

```
public class MediaMovelPonderada {

  public double calcula(int posicao, SerieTemporal serie) {
    double soma = 0.0;
    int peso = 3;

    for (int i = posicao; i > posicao - 3; i--) {
      Candlestick c = serie.getCandle(i);
      soma += c.getFechamento() * peso;
      peso--;
    }
    return soma / 6;
```

```
13     }
14 }
```

Repare que o `peso` começa valendo 3, o tamanho do nosso intervalo, para o dia atual e vai reduzindo conforme nos afastamos do dia atual, demonstrando a maior importância dos valores mais recentes.

A divisão por 6 no final é a **soma dos pesos** para o intervalo de 3 dias: (3 + 2 + 1 = 6).

7. Depois a classe `MediaMovelPonderada` deve passar pelo seguinte teste:

```
public class MediaMovelPonderadaTest {

  @Test
  public void sequenciaSimplesDeCandles() {
    SerieTemporal serie =
            GeradorDeSerie.criaSerie(1, 2, 3, 4, 5, 6);
    MediaMovelPonderada mmp = new MediaMovelPonderada();

    //ex: calcula(2): 1*1 + 2*2 +3*3 = 14. Divide por 6, da 14/6
    Assert.assertEquals(14.0/6, mmp.calcula(2, serie), 0.00001);
    Assert.assertEquals(20.0/6, mmp.calcula(3, serie), 0.00001);
    Assert.assertEquals(26.0/6, mmp.calcula(4, serie), 0.00001);
    Assert.assertEquals(32.0/6, mmp.calcula(5, serie), 0.00001);
  }
}
```

Rode o teste e veja se passamos!

8. (opcional) Crie um teste de unidade em uma nova classe `SerieTemporalTest`, que verifique se essa classe pode receber uma lista nula. O que não deveria poder.

Aproveite o momento para pensar quais outros testes poderiam ser feitos para essa classe.

8.5 - Refatoração

"Refatoração é uma técnica controlada para reestruturar um trecho de código existente, alterando sua estrutura interna sem modificar seu comportamento externo. Consiste em uma série de pequenas transformações que preservam o comportamento inicial. Cada transformação (chamada de refatoração) reflete em uma pequena mudança, mas uma sequência de transformações pode produzir uma significante reestruturação. Como cada refatoração é pequena, é menos provável que se introduza um erro. Além disso, o sistema continua em pleno funcionamento depois de cada pequena refatoração, reduzindo as chances do

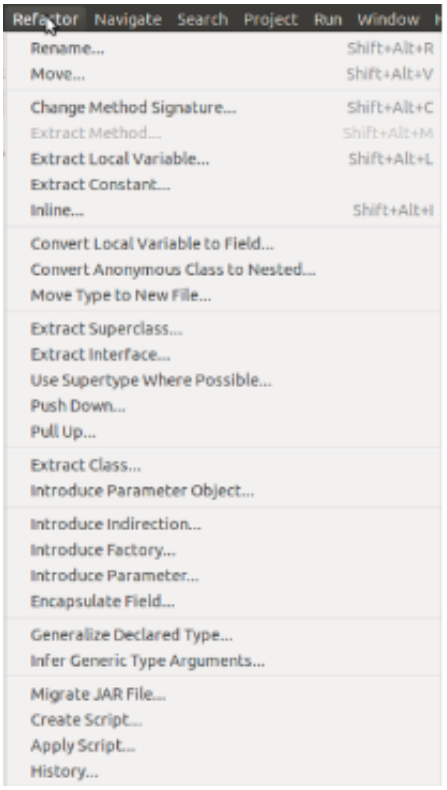
sistema ser seriamente danificado durante a reestruturação. -- **Martin Fowler**

Em outras palavras, refatoração é o processo de modificar um trecho de código já escrito, executando pequenos passos (**baby-steps**) sem modificar o comportamento do sistema. É uma técnica utilizada para melhorar a clareza do código, facilitando a leitura ou melhorando o design do sistema.

Note que para garantir que erros não serão introduzidos nas refatorações, bem como para ter certeza de que o sistema continua se comportando da mesma maneira que antes, a presença de testes é fundamental. Com eles, qualquer erro introduzido será imediatamente apontado, facilitando a correção a cada passo da refatoração imediatamente.

Algumas das refatorações mais recorrentes ganharam nomes que identificam sua utilidade (veremos algumas nas próximas seções). Além disso, **Martin Fowler** escreveu o livro **Refactoring: Improving the Design of Existing Code**, onde descreve em detalhes as principais.

Algumas são tão corriqueiras, que o próprio Eclipse inclui um menu com diversas refatorações que ele é capaz de fazer por você:

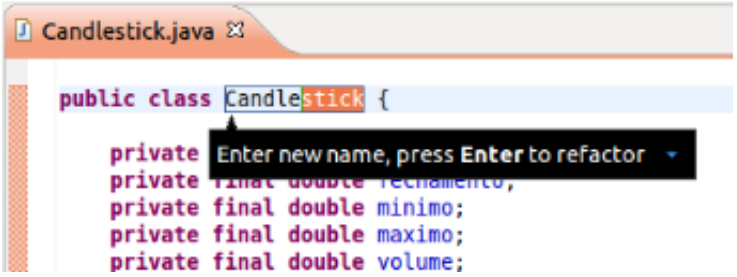


8.6 - Exercícios: Primeiras refatorações

1. Temos usado no texto sempre o termo **candle** em vez de **Candlestick**. Essa nomenclatura se propagou no nosso dia-a-dia, tornando-se parte do nosso modelo. Refatore o nome da classe `Candlestick` para `Candle` no pacote `br.com.caelum.argentum.modelo`.

Use `ctrl + shift + T` para localizar e abrir a classe `Candlestick`.

Seja no *Package Explorer* ou na classe aberta no editor, coloque o cursor sobre o nome da classe e use o atalho `alt + shift + R`, que renomeia. Esse atalho funciona para classes e também para métodos, variáveis, etc.



2. No método `calcula` da classe `MediaMovelSimples`, temos uma variável do tipo `Candle` que só serve para que peguemos o fechamento desse objeto. Podemos, então, deixar esse método uma linha menor fazendo o *inline* dessa variável!

Na linha 4 do método abaixo, coloque o cursor na variável `c` e use o `alt + shift + I`. (Alternativamente, use `ctrl + 1` *Inline local variable*)

```
1 public double calcula(int posicao, SerieTemporal serie) {
2   double soma = 0.0;
3   for (int i = posicao; i > posicao - 3; i--) {
4     Candle c = serie.getCandle(i);
5     soma += c.getFechamento();
6   }
7   return soma / 3;
8 }
```

O novo código, então ficará assim:

```
public double calcula(int posicao, SerieTemporal serie) {
  double soma = 0.0;
  for (int i = posicao; i > posicao - 3; i--) {
    soma += serie.getCandle(i).getFechamento();
  }
  return soma / 3;
}
```

Não esqueça de tirar os imports desnecessários (`ctrl + shift + O`)!

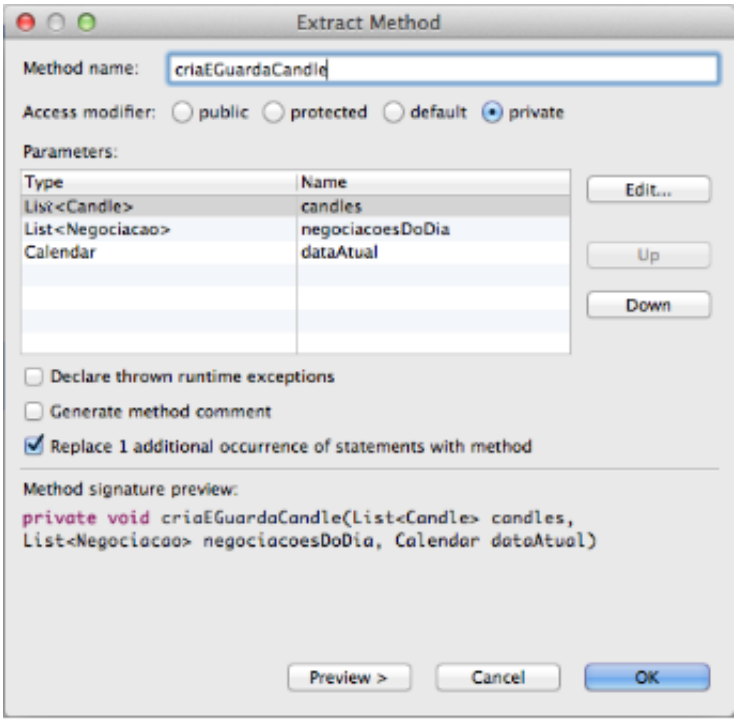
3. Finalmente, abra a classe `CandlestickFactory` e observe o método `constroiCandles`. Escrevemos esse método no capítulo de XML com um algoritmo para separar todos os negócios em vários candles.

No meio desse código, contudo, há um pequeno bloco de código que se repete duas vezes, dentro e fora do `for`:

```
Candle candleDoDia = constroiCandleParaData(dataAtual, negociacoesDoDia);
candles.add(candleDoDia);
```

Se encontramos códigos iguais pelo nosso código, as boas práticas de orientação a objetos nos dizem para isolar então essa parte repetida em um novo método que, além de poder ser chamado várias vezes, ainda tem um nome que ajuda a compreender o algoritmo final.

No Eclipse, podemos aplicar a refatoração **Extract Method**. Basta ir até a classe `CandlestickFactory` e selecionar essas linhas de código dentro do método `constroiCandles` e usar o atalho `alt + shift + M`, nomeando o novo método de `criaEGuardaCandle` e clique em OK.



Repare como a IDE resolve os parâmetros e ainda substitui as chamadas ao código repetido pela chamada ao novo método.

4. No método `constroiCandleParaData`, observe que no bloco de código dentro do `for` invocamos `negociacao.getPreco()` quatro vezes.

```
// ...
for (Negociacao negociacao : negociacoes) {
  volume += negociacao.getVolume();

  if (negociacao.getPreco() > maximo) {
    maximo = negociacao.getPreco();
  }
  if (negociacao.getPreco() < minimo) {
    minimo = negociacao.getPreco();
  }
}
// ...
```

Uma forma de deixar o código mais limpo e evitar chamadas desnecessárias seria extrair para uma variável local.

Para isso, usaremos a refatoração **Extract Local Variable** através do Eclipse. Selecione a primeira chamada para `negociacao.getPreco()` e pressione `alt + shift + L`.

Um box vai aparecer perguntando o nome da variável que será criada, mantenha o nome aconselhado pelo Eclipse e pressione OK.

O Eclipse vai alterar o código de forma que fique parecido com:

```
// ...
for (Negociacao negociacao : negociacoes) {
  volume += negociacao.getVolume();

  double preco = negociacao.getPreco();
  if (preco > maximo) {
    maximo = preco;
  }
  if (preco < minimo) {
    minimo = preco;
  }
}
// ...
```

Observe que o Eclipse automaticamente substituiu as outras chamadas à `negociacoes.getPreco()` por `preco`.

5. (Opcional) Aproveite e mude os nomes das outras classes que usam a palavra `Candlestick`: a Factory e os Testes.

> **Refatorações disponíveis**
>
> Para quem está começando com a usar o Eclipse, a quantidade de atalhos pode assustar. Note, contudo, que os atalhos de refatoração começam sempre com `alt + shift`!
>
> Para ajudar um pouco, comece memorizando apenas o atalho que mostra as refatorações disponíveis dado o trecho de código selecionado: `alt + shift + T`.

8.7 - Refatorações maiores

As refatorações que fizemos até agora são bastante simples e, por conta disso, o Eclipse pôde fazer todo o trabalho para nós!

Há, contudo, refatorações bem mais complexas que afetam diversas classes e podem mudar o design da aplicação. Algumas refatorações ainda mais complexas chegam a modificar até a arquitetura do sistema! Mas, quanto mais complexa a mudança para chegar ao resultado final, mais importante é **quebrar essa refatoração em pedaços pequenos e rodar os testes a cada passo**.

Nos próximos capítulos faremos refatorações que flexibilizam nosso sistema e

tornam muito mais fácil trabalhar com os indicadores técnicos que vimos mais cedo.

Exemplos de refatorações maiores são:

- Adicionar uma camada a um sistema;
- Tirar uma camada do sistema;
- Trocar uma implementação doméstica por uma biblioteca;
- Extrair uma biblioteca de dentro de um projeto;
- etc...

8.8 - Discussão em aula: quando refatorar?

CAPÍTULO ANTERIOR:

Introdução ao JSF e Primefaces

PRÓXIMO CAPÍTULO:

Gráficos interativos com Primefaces

Blog Caelum

Cursos Online

Facebook

Newsletter

Casa do Código

Twitter