02

Melhorando a organização

Um código organizado facilita muito na hora da manutenção e, por mais que façamos o possível para mantê-lo organizado, uma hora ou outra é importante fazer uma faxina. Organizar o código vai desde mantê-lo em uma formatação consistente até aplicar bem a orientação a objetos, fazendo sistemas menos acoplados.

Uma das maneiras de facilitar a leitura do código é mantendo a indentação correta e inserindo linhas entre elementos da classe, para deixar bem claro onde começam e onde terminam. Vamos usar como exemplo o código da classe Importador, responsável por ler as linhas de um arquivo de texto e convertê-las em uma lista de gastos:

```
public class Importador {
    public List<Gasto> importa(InputStream entrada) throws ParseException
    {
        Scanner leitor = new Scanner(entrada);
        List<Gasto> gastos = new ArrayList<Gasto>();
        while (leitor.hasNextLine())    {
                SimpleDateFormat df = new SimpleDateFormat("ddMMyyyy");
                String line = leitor.nextLine();
                String tp = line.substring(0, 6);
                String dt = line.substring(6, 14);
                String vl = line.substring(14, 23);
                String mat = line.substring(23,26);
                String nome = line.substring(26,56);
                String dataNascTxt = line.substring(56);
                double valor = Double.parseDouble(vl);
                int matricula = Integer.parseInt(mat);
                Calendar dataNascimento = Calendar.getInstance();
                dataNascimento.setTime(df.parse(dataNascTxt));
                Calendar dataDespesa = Calendar.getInstance();
                dataDespesa.setTime(df.parse(dt));
                Funcionario funcionario = new Funcionario(nome, matricula, dataNascimento);
                gastos.add(new Gasto(valor,tp,funcionario,dataDespesa));}
                leitor.close();
                return gastos;
}
}
```

O código dessa classe tem vários problemas que dificultam sua leitura e compreensão. Onde termina o método importa? a linha `return gastos;` está dentro ou fora do `while` ? o que é a variável `tp` ? Todos esses problemas podem levar a erros na hora de realizar a manutenção dessa classe.

Para conseguirmos entender esse método, teriamos que organizá-lo primeiro, de preferência sempre seguindo as convenções de código (http://www.oracle.com/technetwork/java/codeconvtoc-136057.html) da linguagem Java. Por sorte, o Eclipse já vem com uma ferramenta capaz de organizar nosso código de acordo com as boas práticas sem muito esforço. Dentro da classe Importador, pressione o atalho ctrl + shift + F, para formatar e indentar seu código automaticamente. Podemos melhorar ainda mais o resultado acrescentando algumas linhas ao nosso método, separando os passos das lógicas que estamos executando. O resultado final seria:

```
    public List<Gasto> importa(InputStream entrada) throws ParseException {
        Scanner leitor = new Scanner(entrada);
        List<Gasto> gastos = new ArrayList<Gasto>();

        while (leitor.hasNextLine()) {
            SimpleDateFormat df = new SimpleDateFormat("ddMMyyyy");

            String line = leitor.nextLine();
            String tp = line.substring(0, 6);
            String dt = line.substring(6, 14);
            String vl = line.substring(14, 23);
            String mat = line.substring(23, 26);
            String nome = line.substring(26, 56);
            String dataNascTxt = line.substring(56);

            double valor = Double.parseDouble(vl);
            int matricula = Integer.parseInt(mat);

            Calendar dataNascimento = Calendar.getInstance();
            dataNascimento.setTime(df.parse(dataNascTxt));
```

```
            Calendar dataDespesa = Calendar.getInstance();
            dataDespesa.setTime(df.parse(dt));

            Funcionario funcionario = new Funcionario(nome, matricula,
                    dataNascimento);
            gastos.add(new Gasto(valor, tp, funcionario, dataDespesa));
        }
        return gastos;
    }
```

Com o código já formatado e indentado, fica muito mais simples perceber por exemplo que a linha `return gastos` está fora do `while` , além de organizar as chaves e parênteses de modo a deixar mais claro o que está sendo fechado/aberto por eles.

Será que podemos melhorar mais nosso código?

Escolher bem os nomes de cada variável, método ou classe (http://blog.caelum.com.br/em-busca-do-nome-adequado-metodos-variaveis-e-classes/) também é uma arte e tem grande importância na manutenabilidade do código. O problema é que, frequentemente, enquanto desenvolvemos não temos uma idéia completa do que aquele trecho de código se tornará depois de pronto. E, principalmente, com um design evolutivo de código, é praticamente impossível dar nomes que se mantenham bons por toda a vida de um método ou classe.

Mudar nomenclaturas para que elas exprimam melhor o que um código faz hoje é uma das formas mais simples de facilitar a leitura de um código. Alterar um simples nome, no entanto, implica em uma infinidade de mudanças em cada código que usa aquele a ser mudado. Pense nas implicações de mudar um nome de classe, por exemplo: é preciso mudar o nome da classe, o do arquivo, toda variável do tipo dessa classe e cada criação de objeto.

Felizmente, exatamente por ser um trabalho mais mecânico, o desenvolvedor só precisa pensar em um bom nome e a IDE o ajudará com a consistência do sistema após as alterações. O atalho alt + shift + R é o responsável por renomear consistentemente de classes a simples variáveis locais.

Vamos começar com um caso mais simples, a variável `tp` em nosso método `importa` . Nomes abreviados devem ser evitados sempre que possível, pois podem levar a interpretações erradas. Vamos renomear essa variável para um nome melhor, como `tipoDeGasto` por exemplo.

Leve o cursor até a variável `tp` , pode ser tanto na declaração da variável quanto em algum local onde ela é utilizada e pressione alt + shift + R. O Eclipse vai contornar a variável e todos os seus usos, agora basta escrever o novo nome, e o Eclipse substituirá todas as referências à variável enquanto você digita. Ao final da edição, aperte enter para finalizar. Podemos repetir esse processo para todas as outras variáveis e teremos uma classe muito mais legível no final.

Além de renomear variáveis, também podemos fazer isso com classes, que terão todas as suas referências atualizadas automaticamente. Vamos alterar o nome de nossa classe `Importador` para `ImportadorDeGastos` , utlizando o atalho .

Continuando a análise do nosso método, podemos perceber que o objeto do tipo `SimpleDateFormat` é criado uma vez para cada iteração do nosso laço, o que é totalmente desnecessário.

```
while (leitor.hasNextLine()) {
    SimpleDateFormat df = new SimpleDateFormat("ddMMyyyy");
    // ... resto do código
```

Poderíamos instanciar o formatador apenas uma vez, fora do loop e utilizar o mesmo objeto sempre. Mas em vez de recortarmos a linha da criação do objeto e colar-mos fora do loop, vamos usar um atalho do Eclipse que nos permite mover linhas dentro de nosso código.

Como queremos mover a linha do `SimpleDateFormat` para cima, leve o cursor para qualquer ponto desta linha, segure a tecla alt e aperte a seta para cima. O conteúdo da linha toda será movida para fora do loop, e já será indentada de acordo com sua nova posição.

```
SimpleDateFormat df = new SimpleDateFormat("ddMMyyyy");
while (leitor.hasNextLine()) {
    // ... resto do código
```

Podemos também levar qualquer linha para baixo, basta manter a tecla alt pressionada e apertar a seta para baixo.

No caso deste `SimpleDateFormat` , poderiamos inclusive movê-lo para fora do método, transformando-o em um atributo da classe. Nesse caso, mesmo que o método `importa` seja chamado várias vezes, ainda assim utilizaremos apenas um objeto. Vamos movê-lo utilizando o mesmo atalho de antes, alt + seta para cima, não esquecendo do encapsulamento do atributo, que será private.

```
public class ImportadorDeGastos {

    private SimpleDateFormat df = new SimpleDateFormat("ddMMyyyy");

    public List<Gasto> importa(InputStream entrada) throws ParseException {
        Scanner leitor = new Scanner(entrada);
        List<Gasto> gastos = new ArrayList<Gasto>();

        while (leitor.hasNextLine()) {
            //.. resto do código
```

Mas esse método ainda está com muitas responsabilidades. Ele realiza o parse do arquivo, converte os valores de `String` para os tipos corretos e adiciona na lista de gastos. Podemos quebrar esse método em vários outros menores, cada um com uma responsabilidade menor e bem definida. Vamos começar eliminando a repetição de código dentro do método, na criação dos `Calendar`s:

```
            Calendar dataNascimento = Calendar.getInstance();
            dataNascimento.setTime(formatador.parse(dataNascTxt));

            Calendar dataDespesa = Calendar.getInstance();
            dataDespesa.setTime(formatador.parse(dataDespesaTxt));
```

Em vez de repetir esse trecho, podemos extraí-lo para um método, e chamar esse metodo todas as vezes que precisamos criar um calendar a partir de uma String. Esse processo de alterar um código já escrito sem modificar seu comportamento, visando melhorar a clareza ou deixar nosso código mais orientado a objetos é chamado de refactoring ou refatoração.

Para atingirmos nosso objetivo, basta criar um método, copiar o conteúdo de um dos trechos para esse novo método, acertar os retornos e parâmetros e substituir os trechos pelo novo método criado. É uma tarefa trabalhosa e que deve ser repetida toda vez que quisermos extrair um método. Novamente, não temos que nos preocupar com isso, já que o Eclipse tem um menu com várias opções de refactoring, sendo que algumas delas estão acessíves através de atalhos, como é o caso do Extract Method..., cujo atalho é shift + alt + M.

Vamos extrair a geração dos `Calendar`s para um método. Primeiro, selecione as duas linhas que deseja extrair, e pressione shift + alt + M. O Eclipse vai abrir uma tela, perguntando qual será o nome do novo método, vamos colocar `converteDataTxtParaCalendar`. Existem várias outras opções que podemos escolher para nosso novo método, como a visibilidade, lista de parâmetros, etc., mas por enquanto não temos que nos preocupar com isso, apenas aperte .

O Eclipse gerará o novo método `converteDataTxtParaCalendar` logo abaixo do método `importa`. Além disso, ele substituirá o trecho de código selecionado anteriormente por uma chamada ao `converteDataTxtParaCalendar`. Contudo, a grande vantagem da refatoração pela IDE é que ela identifica dentro da classe todos os trechos de código parecidos e os substitui por chamadas ao método criado. O resultado final do código será:

```
public List<Gasto> importa(InputStream entrada) throws ParseException {
//...
        Calendar dataNascimento = converteDataTxtParaCalendar(dataNascTxt);

        Calendar dataDespesa = converteDataTxtParaCalendar(dataDespesaTxt);
//resto do método importa...
}

private Calendar converteDataTxtParaCalendar(String dataNascTxt)
            throws ParseException {
        Calendar dataNascimento = Calendar.getInstance();
        dataNascimento.setTime(formatador.parse(dataNascTxt));
        return dataNascimento;
}
```

Podemos ainda pegar outros pedaços do método `importa` e também extraí-los, melhorando ainda mais a legibilidade de nosso código.

Para saber mais: Atalhos equivalentes no Eclipse versão Mac OS: Formatar e indentar: command + shift + F Renomear: alt + command + R Extrair método: command + option + M