Unidade 7: Documentação de Código

A Importância dos Comentários

Prof. Daniel Caetano

Objetivo: Desenvolver a habilidade de comentar código adequadamente

INTRODUÇÃO

Até o momento a preocupação do curso tem sido com o aprendizado da linguagem Java e o desenvolvimento de código em si. Estes tópicos são extremamente importantes, mas não estão completos sem que os códigos gerados sejam corretamente documentados.

A documentação de código pode ser feita em documentos à parte ou no próprio código. A documentação à parte pode facilitar a consulta para referência no projeto de mudanças futuras e a que fica no código é fundamental para a rápida compreensão do funcionamento do código durante sua leitura, seja porque ela ocorre muitos anos após a criação do código, seja porque a pessoa que o está lendo não foi a pessoa que o criou.

Assim, esta aula se debruça sobre a necessidade de documentar o código no próprio código e, de quebra, trata como fazê-lo de maneira que a documentação à parte possa ser gerada a partir do próprio código.

1. DOCUMENTAR CÓDIGO?

Uma vez que a função do código é explicar para o computador quais são as tarefas que ele deve cumprir, é natural que o código nem sempre seja facilmente compreendido pelos seres humanos, em especial aqueles que não estão totalmente "por dentro" do funcionamento do programa.

Tendo isto em mente, documentar um código é a técnica/arte de inserir comentários no código de maneira que ele seja mais facilmente compreendido por quem quer que o venha a ler. Trata-se de técnica porque existem algumas regras básicas a se seguir; trata-se também de uma arte, pois não há regras para definir tudo que precisa ser comentado, sendo essa uma tarefa deixada para o bom senso do desenvolvedor.

Os comentários, em java, podem ser especificados de três formas:

1) Comentários de uma linha: usa-se duas barras no início da linha:

```
// Este é um comentário de uma linha
```

2) Comentários de várias linhas: usa-se /* para iniciar e */ para finalizá-lo:

```
/* Esta é a primeira linha
   de um comentário de múltiplas
   linhas. Simples não?
 */
```

Por questões estéticas, é comum que os programadores coloquem alguns asteriscos a mais:

```
/* Esta é a primeira linha
 * de um comentário de múltiplas
 * linhas. Simples não?
 */
```

3) Comentários do tipo JavaDoc: usa-se /** para iniciar e */ para finalizá-lo:

```
/** Esta é a primeira linha
 *  de um comentário de múltiplas
 *  linhas em formato JavaDoc. Simples não?
 */
```

NOTA: A razão para se usar comentários do tipo JavaDoc será vista mais adiante.

Agora que já foram apresentadas as formas de se inserir comentários em um programa Java, precisamos entender quais são os tipos de comentários que precisaremos fazer.

Existem, basicamente, dois tipos de comentário em um código:

A) Comentários que descrevem O QUE o código faz (sempre necessários)
B) Comentários que descrevem COMO o código faz (nem sempre necessários).

Examinemos cada um deles com maior profundidade.

1.1 Comentários do tipo "O QUÊ"

Os comentários do tipo "o que" são aqueles que explicam em linhas gerais o que um trecho de código faz. É comum ter comentários deste tipo para as classes e para cada um de seus métodos, descrevendo em detalhes para que essa classe/método serve e como devem ser utilizado em um programa, como mostra o exemplo a seguir.

Main.java

```
/*  Esta classe Main é a principal do programa.
 *  Ela é responsável por inicializar o programa
 *  E executar suas tarefas mais básicas.
 *  Esta classe não depende de nenhuma outra.
 *
 *  Criada em: 10/04/2010
 *  Autor: Daniel Caetano (daniel@caetano.eng.br)
 */
public class Main {

   /*  Este método imprime um texto em uma janela.
    *  Este método depende do parâmetro "texto", do tipo String,
    *  que é o texto que será impresso na janela.
    */
   public static void imprime(String texto) {
      JOptionPane.showMessageDialog(null,texto,"Mensagem!", JOptionPane.PLAIN_MESSAGE);
      }

   /*  Este método abre uma janela e pede que o usuário digite um número.
    *  Este método depende do parâmetro "texto", do tipo String,
    *  que é a pergunta que o usuário deverá responder no campo.
    *  Este método retorna o número digitado pelo usuário, como um valor
    *  do tipo "double".
    */
   public static double entrada(String texto) {
      String valor = JOptionPane.showInputDialog(texto);
      return (Double.parseDouble(valor));
      }
   }
```

Estes comentários são obrigatórios e são os mais importantes para que nosso código possa ser USADO por alguém que não o conhece perfeitamente ou não se lembra como se usava o código. Faça com cuidado, o usuário da documentação pode ser você mesmo, no futuro, e os minutos gastos documentando o código poderão lhe poupar horas de aborrecimento futuro.

1.2 Comentários do tipo "COMO"

Os comentários do tipo "como" são aqueles que explicam em detalhe o que alguns trechos do código mais complexos fazem. Como não existe uma definição clara de o que é um "trecho complexo", a criação desse tipo de comentário acaba sendo um pouco de "arte", já deve-se evitar comentar demais - o que polui o código, assim como se deve evitar comentários de menos - o que não ajuda ninguém. Este tipo de comentário é mostrado no exemplo a seguir.

```
   public static boolean processa(double av1, double av2, double av3, double freq) {

      double media; // usada como variável auxliar para o cálculo da média

      // Alunos com AV2 menor que 4 e frequencia menor que 75% são reprovados
      if (av2 < 4.0 || freq < 75.0) return false;
      // Para o aluno ser aprovado, pelo menos a AV1 ou AV3 precisa ser >= 4
      if (av1 < 4.0 && av3 < 4.0) return false;
      // Se AV1 é a maior, ela que entra na média com AV2
```

```
        if (av1 >= av3) media = (av1 + av2) / 2;
        // Caso contrário, usa-se AV3 para compor a média com AV2
        else media = (av3 + av2) / 2;
        // Finalmente... se a média for menor que 6, aluno reprovado!
        if (media < 6.0) return false;
        // Se todos os critérios forem atendidos, aluno aprovado!
        return true;
        }
```

2. COMENTÁRIOS JAVADOC

Como é muito complicado manter as duas documentações - a do código e a em papel - ao mesmo tempo, seria interessante se pudéssemos fazer uma documentação única e, dela, extrair a outra. A Sun Microsystems pensou nisso e criou a aplicação JavaDoc, que usa os comentários do tipo "o que" para gerar a documentação externa.

O JavaDoc é um programa que lê o código das classes que escrevemos e gera um arquivo HTML par cada uma delas, resumindo todas as informações importantes que colocamos nos comentários de nosso código. O JavaDoc é uma ferramente muito versátil e permite gerar diferentes tipos de documentação "externa" com base em nosso código:

Só Públicos: com o parâmetro **-public** , o JavaDoc documenta apenas as classes, métodos e atributos públicos de um programa.
Públicos e Protegidos: com o parâmetro **-protected** , o JavaDoc documenta as classes, métodos e atributos públicos e protegidos de um programa. Este é o comportamento padrão.
Públicos, Protegidos e Pacotes: com o parâmetro **-package** , o JavaDoc documenta as classes, métodos e atributos públicos e protegidos de um programa, além de especificar os pacotes.
Tudo: com o parâmetro **-private** , o JavaDoc documenta as classes integralmente, incluindo métodos e atributos públicos, protegidos e privados de um programa, além de especificar os pacotes.

Mas, como devemos especificar os comentários para que o aplicativo JavaDoc os compreenda e possa gerar a documentação externa para nós?

2.1. Sintaxe JavaDoc

Como já foi visto, os comentários JavaDoc tem uma especificação levemente diferente dos comentários de múltiplas linhas, começando com o sinal /** e terminando com o sinal */. Este comentário só será reconhecido pelo JavaDoc se vier imediatamente ANTES da classe, interface, construtor, método ou campo/atributo (daqui em diante chamados apenas de **entidades**).

A primeira linha de um comentário JavaDoc deve ser sempre uma descrição clara e concisa do que a entidade faz, pois esta linha será usada como referência. Um ponto final ou um "tab" indica o "fim" dessa linha para o JavaDoc.

Observe o exemplo:

Main.java

```
/**  Classe principal, responsável pela inicialização e gerenciamento.
 *  Esta classe é responsável por inicializar o programa
 *  E executar suas tarefas mais básicas.
 *  Esta classe não depende de nenhuma outra.
 */
public class Main {

   /** Este método imprime um texto em uma janela.


    */
   public static void imprime(String texto) {
      JOptionPane.showMessageDialog(null,texto,"Mensagem!", JOptionPane.PLAIN_MESSAGE);
      }

   /** Este método abre uma janela e pede que o usuário digite um número.
    */
   public static double entrada(String texto) {
      String valor = JOptionPane.showInputDialog(texto);
      return (Double.parseDouble(valor));
      }
   }
```

Dentro destes comentários pode-se usar tags HTML se considerado interessante (<B>, <EM>, <I>...). Evite usar tags estruturadores, como <P>, <H1>, <HR> e outros.

Depois da primeira linha, pode-se fazer uma explicação mais extensa sobre a entidade sendo documentada. Observe o exemplo:

Main.java

```
/**  Classe principal, responsável pela inicialização e gerenciamento.
 */
public class Main {

   /** Este método imprime um texto em uma janela.
    */
   public static void imprime(String texto) {
      JOptionPane.showMessageDialog(null,texto,"Mensagem!", JOptionPane.PLAIN_MESSAGE);
      }

   /** Este método abre uma janela e pede que o usuário digite um número.
    */
   public static double entrada(String texto) {
      String valor = JOptionPane.showInputDialog(texto);
      return (Double.parseDouble(valor));
      }
   }
```

Existem, ainda, diversos indicadores que podemos e alguns que devemos usar dentro dos comentários. Estes indicadores são feitos com o uso de *tags* especiais, que devem vir no início da linha do comentário. Eles estão descritos a seguir:

@author
@deprecated
@exception
{@link}
@param
@return
@see
@serial / @serialData / @serialField
@since
@throws
@version

É interessante que todos os *tags* de um mesmo tipo venham agrupados, pois isso facilita o trabalho do aplicativo JavaDoc. Por exemplo, se um método ou classe tem mais de um autor, cada um deles deve ser especificado em uma linha iniciando com @author, mas todas essas linhas devem ser agrupadas.

Foge ao escopo deste curso estudar em profundidade todas as tags do JavaDoc, mas você pode encontrar informações sobre elas na Internet. Aqui iremos falar das mais comuns e importantes neste ponto do curso: @author, @deprecated, @param, @return, @version.

@author serve para identificar o autor de um trecho de código. Usa-se assim:

@author Nome do Autor

@deprecated serve para identificar uma classe/método que existe por compatibilidade (e, portanto, não deve ser usada em nada novo). Usa-se assim:

@deprecated Evite usar esta classe. Use a classe XXXXX no lugar desta.

@param serve para identificar para que serve um dos parâmetros de um método. Usa-se assim:

```
/** Este método imprime um texto em uma janela.
 *  @param texto Indica o texto que deve ser impresso na janela de mensagem.
 */
public static void imprime(String texto) {
   JOptionPane.showMessageDialog(null,texto,"Mensagem!", JOptionPane.PLAIN_MESSAGE);
   }
```

Observe que o "nome" depois do @param deve ser o mesmo que aparece na declaração do parâmetro do método!

@return serve para indicar o que um método retorna. Usa-se assim:

```
    /** Este método abre uma janela e pede que o usuário digite um número.
     *  @return O número digitado pelo usuário.
     */
    public static double entrada(String texto) {
       String valor = JOptionPane.showInputDialog(texto);
       return (Double.parseDouble(valor));
       }
```

@version serve para indicar a versão de uma classe, pacote ou método. Usa-se assim:

@version 1.10.17

3. EXERCÍCIOS

A) Comente o código abaixo, usando a sintaxe vista para o JavaDoc:

```
import javax.swing.*;

public class ContraCheque {
    public static void imprime(String texto) {
        JOptionPane.showMessageDialog(null,texto,"Atenção!",JOptionPane.PLAIN_MESSAGE);
    }

    public static double entrada(String texto) {
       return (Double.parseDouble(JOptionPane.showInputDialog(texto)));
    }

    public static double calculaImpostoRetido(double salario) {
        if (salario <= 1499.15) return 0;
        else if (salario <= 2246.75) return arDinheiro((salario-112.43)*0.075);
        else if (salario <= 2995.70) return arDinheiro((salario-280.94)*0.150);
        else if (salario <= 3743.19) return arDinheiro((salario-505.62)*0.225);
        else return arDinheiro((salario-692.78)*0.275);
    }

    public static double calculaInssRetido(double salario) {
        if (salario <= 1024.197) return arDinheiro(0.08*salario);
        else if (salario <= 1708.27) return arDinheiro(0.09*salario);
        else if (salario <= 3416.54) return arDinheiro(0.11*salario);
        else return 375.82;
    }

    public static double arDinheiro(double valor) {
        return (Math.rint(valor*100)/100);
    }

    public static void processa() {
        double bruto, inss, irrf;
        String saida;
        bruto = entrada("Digite o Salário Bruto");
        inss = calculaInssRetido(bruto);
        irrf = calculaImpostoRetido(bruto-inss);
        saida  = "Salário Bruto:   R$ " + bruto +"\n";
        saida += "Desconto INSS:   R$ " + inss +"\n";
        saida += "Salário-IRRF:   R$ " + arDinheiro(bruto-inss) +"\n";
        saida += "Desconto IRRF:   R$ " + irrf +"\n";
        saida += "Salário Líquido:   R$ " + arDinheiro(bruto-inss-irrf);
        imprime(saida);
    }

    public static void main(String[] args) {
        processa();
        System.exit(0);
    }
}
```