Instituto Politécnico de Beja
Escola Superior de Tecnologia e Gestão

Regras de Estilo para Programação em Java™

João Paulo Barros

Versão de 29 de Setembro de 2007

A linguagem de programação Java™ tem uma sintaxe bastante complicada. Como permite uma grande liberdade no posicionamento de cada carácter, torna muito fácil a criação de programas pouco legíveis. Estas duas características são comuns a muitas outras linguagens de programação. Por essa razão opta-se, quase sempre (quer em meios académicos quer em meios industriais), por impor um conjunto de regras adicionais relativas quer à formatação do código a produzir, quer à utilização da linguagem. Tal ajuda o programador pois este deixa de se preocupar com pormenores triviais (formato dos nomes, espaços, etc.) para se centrar no problema a resolver. Por outro lado, facilita a compreensão do programa e portanto também o trabalho em equipa, dado que cada programador pode mais facilmente compreender o código produzido por outros.

As regras que se seguem procuram seguir práticas comuns na programação com Java™ e, nalguns casos, em outras linguagens semelhantes. Claro que para a maior parte existem alternativas para as quais é fácil encontrar argumentos pró e contra. No entanto, o objectivo das regras é exactamente especificar com rigor quais dessas alternativas devem ser utilizadas evitando discussões estéreis.

Quem já programou, e principalmente quem já programou em C, C++, C# ou Java™, terá provavelmente os seus hábitos e gostos quanto ao estilo a utilizar. No entanto, é uma prova de profissionalismo abdicar dos hábitos pessoais e gostos quando se trata de alcançar um compromisso que contribua para o sucesso do código a desenvolver.

1 Gerais

1. Todas as regras devem ser aplicadas de forma coerente ao longo de todo o programa.
2. O código deve ser escrito em inglês. Tal é fundamental para que quem não entende a mesma língua do programador possa perceber o código.

2 Nomes

3. Todos os nomes devem ser significativos. Por exemplo, `nStudents` é preferível a `ns`. Como excepção temos os nomes que sejam utilizados "muito localmente". Por exemplo, variáveis que apenas são utilizadas numa ou poucas mais linhas consecutivas.

4. Todos os nomes de classes começam por uma letra maiúscula.

5. Os nomes das classes são sempre substantivos simples ou compostos. Por exemplo: `Book`, `Car`, `AlarmButton`.

6. Todos os nomes de variáveis (ou referências para objectos) e métodos começam por uma letra minúscula.

7. Todos os nomes de constantes se escrevem utilizando exclusivamente letras maiúsculas e, se necessário, o *underscore* para separar palavras. Por exemplo:

```
final double GRAVITATIONAL_CONSTANT = 6.673e-11;
final int EARTH_SATELLITES = 1;
```

8. As constantes não devem ser utilizadas "literalmente"no programa. Em vez disso deve ser definido um nome simbólico com a palavra reservada **final** e utilizar esse nome. Por exemplo, em vez de

```
 // WRONG!
double force = 6.673e-11 * m1 * m2 / (r * r);
```

deve utilizar

```
// RIGHT!
double force = GRAVITATIONAL_CONSTANT * m1 * m2 / (r * r);
```

9. Nos nomes constituídos por mais do que uma palavra, ou abreviatura, capitaliza-se a primeira letra da segunda palavra (ou abreviatura) e seguintes. Por exemplo: `int dayOfTheMonth = 2;`

3 Alinhamentos e Espaçamentos

10. Deve existir uma linha em branco entre a lista de definições de atributos e os construtores.

11. Deve existir uma linha em branco entre a definição de cada construtor e de cada método.

12. As chavetas devem ser alinhadas à esquerda:

```
class Book
{
     public String getAuthor()
     {
         statements
     }
}

while (expression)
{
   statements
}

do
{
   statements
} while (expression)

for (initialization; condition; update)
{
   statements
}

if (expression)
{
   statements
}
else
{
   statements
}

if (expression)
{
   statements
}
else if (expression)
{
   statements
}
else
{
   statements
}
```

```
switch (expression)
{
   case n:
      statements
      /* falls through */
   case m:
      statements
      break;
   . . .
   default:
      statements;
      break;
}
```

O **default** é obrigatório. O último **break** não faz falta mas é boa ideia lá estar para que não fique esquecido se o **default** passar a **case**.

13. As chavetas nunca são omitidas mesmo quando apenas incluem uma instrução. Por exemplo, deve escrever

```
if (expression)
{
   statements
}
```

e não

```
if (expression)
   statements; // WRONG!
```

14. Para os métodos *getter* e *setter* é permitido utilizar a seguinte formatação e indentação:

```
class Calendar
{
   private DayOfTheWeek day;

   public DayOfTheWeek getDay() { return day; }
   public void setDay(DayOfTheWeek day) { this.day = day; }
}
```

15. É obrigatório indentar cada bloco. A indentação dever ser de 3 espaços. O nível de indentação deve ser igual ao longo de todo o programa.

16. Nunca deve utilizar *tabs*.

17. Nenhuma linha do programa deve apresentar mais do que 80 colunas. Se for tentado a tal, quebre a linha e faça uma indentação "lógica". Por exemplo:

```
public void methodName(int firstParameter,  int  secondParameter,
                       int thirdParameter,  int  fourthParameter)
{
//...
}
```

18. Devem ser colocados espaços antes e depois de cada operador binário. Por exemplo:

```
x = (-b + Math.sqrt(b * b - 4 * a * c)) / (2 * a); // RIGHT!
```

em vez de:

```
x=(-b+Math.sqrt(b*b-4*a*c))/(2*a); // WRONG!
```

19. Os parêntesis devem ser considerados como se fizessem parte do nome do método. Por essa razão colocam-se sempre juntos ao nome do método. Desta forma é mais fácil distinguir o nome de um método do nome de uma variável pois a variável não tem parêntesis associados.

```
z = f(x, y);  // RIGHT!
z = f (x, y); // WRONG!
```

20. Não deverá ter atributos e métodos com igual nome.

21. Nas instruções **if**, **while**, **for** e **switch** deve existir um espaço entre o nome da instrução e os parêntesis:

```
if (n > max) //...
while (n > max) //...
for (int i = 0; i < max; i++) //...
switch (n) //...
```

22. Nas expressões não se deve confiar na ordem de precedência como forma de evitar parêntesis. Ninguém deve ser obrigado a consultar a tabela de precedências dos operadores. Por exemplo:

```
if ( 0 < n && n < x || x < n && n < y )  // no!

if ( ((0 < n) && (n < x)) || ((x < n) && (n < y)) )  // better!
```

Claro que nos casos da multiplicação e divisão versus soma e subtracção poder-se-á assumir que todos os leitores conhecem as regras de precedência. Dessa forma evitar-se-ão alguns parêntesis:

```
x = (-b + Math.sqrt(b * b - 4 * a * c)) / (2 * a);  // enough
x = (-b + Math.sqrt((b * b) -
    (4 * a * c))) / (2 * a);  // probably too much
```

4 Utilização da Linguagem

23. Na utilização de atributos do próprio objecto deve sempre especificar-se a referência **this**.

24. Na chamada de métodos do próprio objecto deve sempre especificar-se a referência **this**.

25. Na utilização de atributos da classe (**static**) deve sempre especificar-se o nome da classe.

26. Na chamada de métodos da classe (**static**) deve sempre especificar-se o nome da classe.

27. Todas as variáveis locais (dentro de métodos) devem ser explicitamente inicializadas. Por exemplo, deve escrever-se `int i = 0;` e nunca `int i;`. Note que esta regra não é sempre seguida no livro mas deve ser seguida no código que produzir.

28. Todas as variáveis devem ser definidas, e simultaneamente inicializadas, apenas quando necessárias e nunca antes. Note que esta regra não é sempre seguida no livro recomendado. Por exemplo:

```
int x;   // ERRADO!
int y;
do_something(2, 3);
x = obj.getX();
y = obj.getY();


do_something(2, 3);
int x = obj.getX(); // CERTO!
int y = obj.getY();
```

29. Nenhum método deve conter mais do que 30 linhas. Esta regra destina-se a incentivar a criação de métodos curtos ou a sua eventual decomposição em métodos **private** mais curtos.

30. Os métodos *get* e *set* (*getter*s e *setter*s) devem seguir a norma *setAtribName* e *getAttribName*. Por exemplo:

```
class Calendar
{
   private DayOfTheWeek day;

   public DayOfTheWeek getDay()
   {
      return day;
   }
   public void setDay(DayOfTheWeek day)
   {
      this.day = day;
   }
}
```

ou

```
class Calendar
{
   private DayOfTheWeek day;

   public DayOfTheWeek getDay() { return day; }
   public void setDay(DayOfTheWeek day) { this.day = day; }
}
```

31. Por regra não deve definir *setters*. Apenas excepcionalmente os deve definir e só para os atributos para os quais sejam mesmo necessários.

32. As declarações **import** não devem utilizar o asterisco. `import java.util.*;` não deve ser utilizado. Em vez disso, os pacotes (*packages*) devem ser indicados explicitamente, um em cada linha:

```
import java.util.ArrayList;
import java.util.HashSet;
```

Veja também a excepção na regra seguinte.

33. As seguintes declarações, são excepções à regra anterior:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
```

34. Os atributos nunca são **public**. Os atributos constantes (atributos **final**) são uma excepção e podem ser **public**.

35. Todos os métodos e atributos devem obrigatoriamente ter um modificador de visibilidade (**public**, **private**, ou **protected**).

36. A ordem das declarações na classe é sempre a seguinte:

    (a) Atributos constante **public**;

    (b) Atributos constante **private**;

    (c) Atributos **private**;

    (d) Construtores;

    (e) Métodos **public**;

    (f) Métodos **private**.

37. Todas as classes devem definir explicitamente pelo menos um construtor, mesmo que este tenha um corpo vazio.

38. Todos os atributos devem ser inicializados no construtor. Por exemplo:

```
public class Book
{
   private String author;
   private String title;
   private int    nCopies;

   public Book(String author, String title)
   {
      this.author  = author;
      this.title   = title;
      this.nCopies = 1;
   }
}
```

39. Para percorrer objectos nas classes *Collections* devem ser utilizados ciclos *foreach* ou iteradores e nunca variáveis **int** (índices).

5 Documentação

40. Todos os métodos devem indicar, no seu início, em comentário `/** */`, qual o seu objectivo, nomeadamente as suas pré-condições e/ou pós-condições.

41. Todas as classes devem conter um comentário `/** */` nas linhas que precedem a sua definição (a palavra class). Este comentário deve incluir:

    - Uma breve descrição da classe;
    - Os nomes de todos os autores;
    - Um número de versão, que pode corresponder à data (e eventualmente hora) da última modificação.

42. O restante código deve ser comentado apenas quando necessário para explicar algo que não seja óbvio para quem conhece a linguagem Java. Por exemplo, devem ser evitados comentários como os seguintes:

    ```
    int nClasses = 0; // nClasses is an integer
    int ns = 30;      // number of students
    ```

    Em seu lugar deve optar-se por:

    ```
    int nClasses  = 0;
    int nStudents = 30;
    ```

Este texto baseia-se nas regras de estilo usuais para a linguagem Java, em trabalhos anteriores do autor e no Program Style Guide *escrito por Michael Kölling e David Barnes para o livro* Objects First With Java - A Practical Introduction Using BlueJ.

As sugestões para melhorar este texto são bem-vindas.