

# Exercício-Programa 1

Adição e multiplicação de precisão ilimitada

1º Semestre 2003

## 1 Introdução

Visando aplicar o conceito de listas ligadas, este exercício-programa implementa as operações de adição e multiplicação de inteiros positivos.

As operações são de certa forma “ilimitadas”, tendo em vista a aplicação das listas ligadas. Nesta implementação, cada argumento a ser submetido às operações citadas tem seu tamanho limitado pelo tamanho máximo de uma *string*.

Na prática, temos que as operações são virtualmente ilimitadas, dada a grande quantidade de conteúdo de armazenamento por *string*.

## 2 Classes e descrições

Arquivos componentes:

- `Celula.java`
- `ListaLigada.java`
- `AdicaoMultiplicacao.java`

### 2.1 Classe Celula

A `Celula` é o elemento fundamental da `ListaLigada`, e cada uma encapsula um tipo primitivo `byte`.

### 2.2 Classe ListaLigada

Implementação de lista simplesmente ligada de objetos do tipo `Celula`.

### 2.3 Classe AdicaoMultiplicacao

Classe principal do exercício-programa.

Implementa as operações de adição e multiplicação de forma virtualmente ilimitada.

## 3 Códigos-fonte

Estão apresentados a seguir os códigos-fonte das classes relacionadas.

## 3.1 Celula.java

```

1  /* Celula.java
2     Copyright (C) 2004 Rodolpho Iemini Atoji
3
4     This program is free software; you can redistribute it and/or modify
5     it under the terms of the GNU General Public License as published by
6     the Free Software Foundation; either version 2 of the License, or
7     (at your option) any later version.
8
9     This program is distributed in the hope that it will be useful,
10    but WITHOUT ANY WARRANTY; without even the implied warranty of
11    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
12    GNU General Public License for more details.
13
14    You should have received a copy of the GNU General Public License
15    along with this program; if not, write to the Free Software
16    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
17 */
18
19 // *****
20 // **  MAC323 - Estruturas de Dados                **
21 // **  IME-USP - Primeiro Semestre de 2004        **
22 // **  Turma 45 - Professor Siang Wun Song        **
23 // **                                             **
24 // **  Exercício-Programa 1 -- Adição e Multiplicação de **
25 // **                                     precisão ilimitada **
26 // **                                             **
27 // **  Arquivo: Celula.java                       **
28 // **                                             **
29 // **  Rodolpho Iemini Atoji           4894631    **
30 // **                                             **
31 // **  Data de entrega: 13/04/2004              **
32 // *****
33
34 /**
35  * A 'Celula' é o elemento fundamental da "lista ligada".
36  * Cada 'Celula' possui uma referência para a próxima da lista e pode
37  * conter informação na forma de um objeto de tipo qualquer.
38  *
39  * @author Rodolpho Iemini Atoji
40  * @version 0.3-byte
41  */
42 public class Celula {
43
44     /**
45     * Conteúdos de uma célula pertencente a uma lista ligada.
46     * informacao: byte que se deseje armazenar numa célula

```

```
47     * proxima:    referência para o próximo objeto 'Celula' da lista ligada
48     */
49 private byte informacao;
50 private Celula proxima;
51
52 /**
53  * Construtor de um só argumento, apontando para referência nula.
54  *
55  * @param info Inteiro não-negativo para inicializar a célula.
56  */
57 public Celula(byte info) {
58     this(info, null);
59     if (info < 0)
60         informacao = 0;
61 }
62
63 /**
64  * Construtor-padrão: cria objeto com informação e aponta para a
65  * próxima célula da lista ligada.
66  *
67  * @param info Objeto para inicializar a célula.
68  * @param prox Referência para a próxima célula.
69  */
70 public Celula(byte info, Celula prox) {
71     this.informacao = info;
72     this.proxima = prox;
73 }
74
75 /**
76  * Devolve a informação principal do conteúdo da Celula.
77  *
78  * @return Informação principal.
79  */
80 public byte devolveInfo() {
81     return informacao;
82 }
83
84 /**
85  * Devolve a referência para a próxima célula.
86  *
87  * @return Referência para a próxima célula.
88  */
89 public Celula devolveProxima() {
90     return proxima;
91 }
92
93 /**
```

```

94     * Muda a referência para a próxima célula.
95     *
96     * @param prox Referência para a próxima célula.
97     */
98     public void mudaProxima(Celula prox) {
99         this.proxima = prox;
100     }
101
102 }// class Celula

```

### 3.2 ListaLigada.java

```

1  /* ListaLigada.java
2     Copyright (C) 2004 Rodolpho Iemini Atoji
3
4     This program is free software; you can redistribute it and/or modify
5     it under the terms of the GNU General Public License as published by
6     the Free Software Foundation; either version 2 of the License, or
7     (at your option) any later version.
8
9     This program is distributed in the hope that it will be useful,
10    but WITHOUT ANY WARRANTY; without even the implied warranty of
11    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
12    GNU General Public License for more details.
13
14    You should have received a copy of the GNU General Public License
15    along with this program; if not, write to the Free Software
16    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
17 */
18
19 // *****
20 // **  MAC323 - Estruturas de Dados                **
21 // **  IME-USP - Primeiro Semestre de 2004        **
22 // **  Turma 45 - Professor Siang Wun Song        **
23 // **                                              **
24 // **  Exercício-Programa 1 -- Adição e Multiplicação de **
25 // **                                     precisão ilimitada **
26 // **                                              **
27 // **  Arquivo:ListaLigada.java                  **
28 // **                                              **
29 // **  Rodolpho Iemini Atoji           4894631    **
30 // **                                              **
31 // **  Data de entrega: 13/04/2004              **
32 // *****
33
34 /**
35  * Implementação de uma "lista ligada" composta de objetos do tipo 'Celula'.
36  * Cada objeto do tipo 'Celula' pode guardar como informação apenas variáveis

```

```
37 * do tipo 'byte'.
38 *
39 * @author Rodolpho Iemini Atoji
40 * @version 0.7.1-lite
41 */
42 public class ListaLigada {
43
44     /**
45      * Referências para a primeira e última célula da lista.
46      */
47     private Celula inicio;
48     private Celula fim;
49
50     /**
51      * Inicializa os atributos de classe.
52      */
53     public ListaLigada() {
54         inicio = null;
55         fim = null;
56     }
57
58     /**
59      * Indica se a lista está ou não vazia.
60      */
61     public boolean listaVazia() {
62         return (inicio == null);
63     }
64
65     /**
66      * Devolve referência para primeira célula.
67      *
68      * @return Referência para primeira célula da lista ligada.
69      */
70     public Celula devolveInicio() {
71         return inicio;
72     }
73
74     /**
75      * Devolve referência para a última célula.
76      *
77      * @return Referência para última célula da lista ligada.
78      */
79     public Celula devolveFinal() {
80         return fim;
81     }
82
83     /**
```

```
84     * "Limpa" a lista ligada.
85     */
86     public void limpaLista() {
87         inicio = null;
88         fim = null;
89     }
90
91     /**
92     * Insere uma célula no início da lista, mudando a sua referência de
93     * "próxima célula" para a célula inicial anterior.
94     *
95     * @param cel Objeto a ser inserido no início da lista.
96     */
97     public void insereNoInicio(Celula cel) {
98         if (listaVazia()) {
99             inicio = cel;
100            fim = inicio;
101        }
102        else {
103            Celula inicioTemp = inicio;
104            inicio = cel;
105            inicio.mudaProxima(inicioTemp);
106        }
107    }
108
109    /**
110    * Insere uma célula no final da lista, mudando a referência de
111    * "próxima célula" da agora penúltima célula para a última.
112    *
113    * @param cel Objeto a ser inserido no final da lista.
114    */
115    public void insereNoFinal(Celula cel) {
116        if (listaVazia()) {
117            inicio = cel;
118            fim = inicio;
119            cel.mudaProxima(null);
120        }
121        else {
122            Celula nova = cel;
123            cel.mudaProxima(null);
124            fim.mudaProxima(nova);
125            fim = nova;
126        }
127    }
128
129    /**
130    * Retira uma célula do início da lista.
```

```

131      *
132      * @return Extrai uma célula do começo da lista.
133      */
134      public Celula retiraComeco() {
135          if (listaVazia())
136              return null;
137          else {
138              Celula retirada = inicio;
139              inicio = inicio.devolveProxima();
140              return retirada;
141          }
142      }
143
144 } // class ListaLigada

```

### 3.3 AdicaoMultiplicacao.java

```

1  /* SomaMultiplicacao.java
2     Copyright (C) 2004 Rodolpho Iemini Atoji
3
4     This program is free software; you can redistribute it and/or modify
5     it under the terms of the GNU General Public License as published by
6     the Free Software Foundation; either version 2 of the License, or
7     (at your option) any later version.
8
9     This program is distributed in the hope that it will be useful,
10    but WITHOUT ANY WARRANTY; without even the implied warranty of
11    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
12    GNU General Public License for more details.
13
14    You should have received a copy of the GNU General Public License
15    along with this program; if not, write to the Free Software
16    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
17 */
18
19 // *****
20 // **  MAC323 - Estruturas de Dados                **
21 // **  IME-USP - Primeiro Semestre de 2004        **
22 // **  Turma 45 - Professor Siang Wun Song        **
23 // **                                             **
24 // **  Exercício-Programa 1 -- Adição e Multiplicação de **
25 // **                                             **
26 // **                                             **
27 // **  Arquivo:SomaMultiplicacao.java            **
28 // **                                             **
29 // **  Rodolpho Iemini Atoji          4894631    **
30 // **                                             **
31 // **  Data de entrega: 13/04/2004              **

```

```
32 // *****
33
34 /**
35  * Realiza adição e multiplicação de precisão "ilimitada", implementando um
36  * algoritmo inocente destas operações.
37  *
38  * @author Rodolpho Iemini Atoji
39  * @version 0.1
40  */
41 public class AdicaoMultiplicacao {
42
43     /**
44      * Pega argumentos da linha de comando e devolve o resultado.
45      * Assegura que os argumentos serão passados para os métodos na ordem
46      * mais conveniente para o cálculo.
47      */
48     public static void main(String[] args) {
49         if (args == null || args.length < 3) {
50             System.out.println("\n" +
51                 "MAC323 - EP1 - Adicao e multiplicacao de " +
52                 "precisao ilimitada\n\n" +
53                 "Uso: AdicaoMultiplicacao numero1 op numero2\n"+
54                 "op: + = soma, x = multiplicacao\n");
55         }
56         else {
57             AdicaoMultiplicacao sm = new AdicaoMultiplicacao();
58             try {
59                 String num1 = sm.excluiZeros(args[0]);
60                 String num2 = sm.excluiZeros(args[2]);
61
62                 if (args[1].equalsIgnoreCase("+")) {
63                     if (num1.length() < num2.length())
64                         sm.imprimeResultado(sm.soma(sm.criaLista(num2),
65                                                         sm.criaLista(num1)));
66                     else
67                         sm.imprimeResultado(sm.soma(sm.criaLista(num1),
68                                                         sm.criaLista(num2)));
69                 }
70                 else if (args[1].equalsIgnoreCase("x")) {
71                     if (num1.length() < num2.length())
72                         sm.imprimeResultado(sm.produto(sm.criaLista(num2),
73                                                         sm.criaLista(num1)));
74                     else
75                         sm.imprimeResultado(sm.produto(sm.criaLista(num1),
76                                                         sm.criaLista(num2)));
77                 }
78                 else
```

```
79         System.out.println("Operacao nao reconhecida.");
80     } catch (NumberFormatException erro) {
81         System.out.println("Erro nos dados de entrada. \n" +
82             "Verifique se foram entrados apenas " +
83             "inteiros positivos.");
84     }
85 }
86 }
87
88 /**
89  * Função de soma "ilimitada".
90  *
91  * @param arg1 Maior número dos a serem somados.
92  * @param arg2 Menor dos números a serem somados.
93  * @return Soma dos dois números.
94  */
95 public ListaLigada soma(ListaLigada arg1, ListaLigada arg2) {
96     ListaLigada resultado = new ListaLigada();
97     byte p1 = 0;
98     byte p2 = 0;
99     byte carry = 0;
100    int resultadoTemp = 0;
101    Celula num1 = arg1.devolveInicio();
102    Celula num2 = arg2.devolveInicio();
103
104    /*
105     * Soma coluna-a-coluna até que ocorra a soma com a última coluna
106     * do menor número (segundo argumento).
107     */
108    while (num2 != null) {
109        p1 = num1.devolveInfo();
110        p2 = num2.devolveInfo();
111        resultadoTemp = (int)(p1 + p2 + carry);
112        carry = (byte)(resultadoTemp / 10);
113        resultadoTemp %= 10;
114        resultado.insereNoFinal(new Celula((byte)resultadoTemp));
115        num1 = num1.devolveProxima();
116        num2 = num2.devolveProxima();
117    }
118
119    /*
120     * No caso de o segundo argumento ser menor que o primeiro,
121     * verifica-se se é necessário somar o "vai-um" (carry) às colunas
122     * restantes do maior número.
123     */
124    if (carry > 0) {
125        while (num1 != null) {
```

```
126         resultadoTemp = (int)(num1.devolveInfo() + carry);
127         carry = (byte)(resultadoTemp / 10);
128         resultadoTemp %= 10;
129         resultado.inserirNoFinal(new Celula((byte)resultadoTemp));
130         num1 = num1.devolveProxima();
131     }
132     if (carry > 0)
133         resultado.inserirNoFinal(new Celula((byte)(1)));
134 }
135
136 /*
137  * Se não for necessário somar o "vai-um" basta copiar as colunas
138  * restantes.
139  */
140 else {
141     while (num1 != null) {
142         resultado.inserirNoFinal(new Celula(num1.devolveInfo()));
143         num1 = num1.devolveProxima();
144     }
145 }
146
147     return resultado;
148 } // soma
149
150 /**
151  * Função de produto "ilimitado".
152  *
153  * @param arg1 Maior dos números a ser multiplicado.
154  * @param arg2 Menor dos números a ser multiplicado.
155  * @return Resultado da multiplicação.
156  */
157 public ListaLigada produto(ListaLigada arg1, ListaLigada arg2) {
158     ListaLigada resultado = criaLista("0");
159     ListaLigada listaTemp;
160     byte p1 = 0;
161     byte p2 = 0;
162     byte carry = 0;
163     int numZeros = 0;
164     int resultadoTemp = 0;
165     Celula num1;
166     Celula num2 = arg2.devolveInicio();
167
168     /*
169     * Para cada algarismo do segundo argumento, será executada a
170     * multiplicação algarismo-a-algarismo do primeiro argumento
171     * (maior número).
172     */
```

```
173     while (num2 != null) {
174         listaTemp = new ListaLigada();
175
176         for (long i = 0; i < numZeros; i++)
177             listaTemp.insereNoFinal(new Celula((byte)(0)));
178
179         p2 = num2.devolveInfo();
180
181         /*
182          * Caso um dos algarismos do segundo argumento seja zero, deve-se
183          * ignorar a multiplicação deste pelos demais algarismos do
184          * primeiro argumento.
185          */
186         if (p2 == 0) {
187             num1 = null;
188             listaTemp = criaLista("0");
189         }
190         else
191             num1 = arg1.devolveInicio();
192
193         /*
194          * Para cada algarismo do primeiro argumento é feita a
195          * multiplicação pelo algarismo atual do segundo.
196          */
197         while (num1 != null) {
198             p1 = num1.devolveInfo();
199             resultadoTemp = (int)((p1 * p2) + carry);
200             carry = (byte)(resultadoTemp / 10);
201             resultadoTemp %= 10;
202             listaTemp.insereNoFinal(new Celula((byte)(resultadoTemp)));
203             num1 = num1.devolveProxima();
204         }
205         if (carry > 0)
206             listaTemp.insereNoFinal(new Celula(carry));
207
208         carry = 0;
209         num2 = num2.devolveProxima();
210         numZeros++;
211
212         if (p2 != 0)
213             resultado = soma(listaTemp, resultado);
214     }
215     return resultado;
216 } // produto
217
218 /**
219  * Coloca os números entrados em forma de lista ligada.
```

```
220     *
221     * @see ListaLigada
222     * @param numero Número a ser convertido.
223     * @return Lista ligada com o número convertido (na lista ligada, os
224     * números são representados na ordem inversa à grafia).
225     */
226     public ListaLigada criaLista(String numero) {
227         int i = 0;
228         int tamNum = numero.length();
229         ListaLigada lista = new ListaLigada();
230
231         /*
232          * Verifica se uma das parcelas é nula.
233          */
234         if (i == tamNum)
235             lista.inserereNoInicio(new Celula((byte)(0)));
236         else {
237             while (i < tamNum) {
238                 lista.inserereNoInicio(new Celula
239                     (Byte.parseByte
240                     (new Character
241                         (numero.charAt(i)).toString())));
242                 i++;
243             }
244         }
245
246         return lista;
247     }
248
249     /**
250     * Exclui zeros de um número.
251     *
252     * @param num Número a ter os zeros excluídos
253     */
254     public String excluiZeros(String num) {
255         int tamanho = num.length();
256         int i = 0;
257
258         while (i < tamanho && num.charAt(i) == '0')
259             i++;
260
261         /*
262          * Devolve o restante da String a partir do ponto 'i', isto é, partir
263          * do ponto em que se achou um caractere não-nulo.
264          */
265         return num.substring(i);
266     }
```

```
267
268     /**
269     * Imprime o resultado na tela.
270     *
271     * @param lista Lista ligada a ser impressa.
272     */
273     public void imprimeResultado(ListaLigada lista) {
274         Celula atual = lista.devolveInicio();
275         ListaLigada resposta = new ListaLigada();
276
277         while (atual != null) {
278             resposta.insererNoInicio(new Celula(atual.devolveInfo()));
279             atual = atual.devolveProxima();
280         }
281         atual = resposta.devolveInicio();
282
283         while (atual != null) {
284             System.out.print(atual.devolveInfo());
285             atual = atual.devolveProxima();
286         }
287         System.out.println();
288     }
289
290 } // class SomaMultiplicacao
```

## 4 Testes

### 4.1 Adição

#### 4.1.1 Teste 1

```
[ratoji@atoji]~ $ java AdicaoMultiplicacao 1 + 1
```

```
2
```

#### 4.1.2 Teste 2

```
[ratoji@atoji]~ $ java AdicaoMultiplicacao 123456789 + 987654321
```

```
1111111110
```

#### 4.1.3 Teste 3

```
[ratoji@atoji]~ $ java AdicaoMultiplicacao 0293983478656713289746324055687793164
78945104987312765448795420983750928475984752876585162138450865 + 210982345098748
75692475761642534389420847578361013874498654209823476210834725764902375746523987
659876234652401273894287524865435876482678245
```

21098234509874875692475761642534389420847607759361740169983184455881779614042243  
847480733836753108671655636152202370272277742021038621129110

#### 4.1.4 Teste 4

```
[ratoji@atoji]~ $ java AdicaoMultiplicacao 8391740984798374913738128043982747386
53808487543971598031873829478321740983748031120392347678450012875746235764687416
47613010000137464465498475048750275087014234840428725387456456849325 + 018734894
65714657861542352378989842947857665165437812563421576157887561246839857249765476
72864502759465872658796238568246589246123243746009834683742384239862348768762348
76391746567921305812930848368651254234814087676523542820989843579495786345123782
34587969748569784568972365498568728634

18734894657146578615423523789898429478576651654378125634215761578875612468398572
49765476728645027594742643997810369431603273741672264933963727686178239558426675
07057131981327303159524262052785268186641299810498523639400189510898449541404413
2987426984863056762804624997697752955025577959
```

## 4.2 Multiplicação

### 4.2.1 Teste 1

```
[ratoji@atoji]~ $ java AdicaoMultiplicacao 1 x 0

0
```

### 4.2.2 Teste 2

```
[ratoji@atoji]~ $ java AdicaoMultiplicacao 9907123456789047 x 31415926536

311241462701567071001451192
```

### 4.2.3 Teste 3

```
[ratoji@atoji]~ $ java AdicaoMultiplicacao 9873497502107382437965982473659824659
265161823010217406376712365780634782369876348763249173246 x 13204810239874098374
9327498327476651253421897234103873247593214734856973614615234

13037766091919889581583661540802517197359927942753106884028539704153508022409917
19581648453947054357538962875831918163167799347437728802191613909116074160386498
145572896829564
```

#### 4.2.4 Teste 4

```
[ratoji@atoji]~ $ java AdicaoMultiplicacao 8391740984798374913738128043982747386
53808487543971598031873829478321740983748031120392347678450012875746235764687416
47613010000137464465498475048750275087014234840428725387456456849325 x 018734894
65714657861542352378989842947857665165437812563421576157887561246839857249765476
72864502759465872658796238568246589246123243746009834683742384239862348768762348
76391746567921305812930848368651254234814087676523542820989843579495786345123782
34587969748569784568972365498568728634

15721838334025704216821987478322850091544437162965114493466497041529562154305986
06810044757058984950353696106464747449961453190522739041196893108101984963817990
78209265614560257140838973647545899776391694773488405174959505135637047519662761
02780051444713323329592887264412914415320799038710209443897106439210349079044157
47459082232302612722799520820330926821791966039427728570824214870293262800086113
01524172486803963271877012970267467269119034863798178646027246551072050
```

## 5 Aferição dos resultados

Os resultados foram confrontados com os apresentados pelo utilitário `bc` (*An arbitrary precision calculator language*), presente na maioria das distribuições Linux, em combinação com o programa `diff`, que verifica diferença entre dois arquivos.

As saídas do exercício-programa foram redirecionadas para um arquivo, da mesma maneira que as saídas do `bc`, sendo então aplicado o `diff`, cujas saídas indicavam completo batimento de resultados.