

Gabarito da Lista de Exercícios 3

```
1 package br.projecao.ed.linear;
2
3 /**
4  * Implementação de uma lista sequencial linear.
5  */
6 public class Lista
7 {
8     private Object[] elementos;
9
10    private Integer cursor;
11
12    public Lista(Integer capacidade)
13    {
14        this.elementos = new Object[capacidade];
15        this.cursor = 0;
16    }
17
18    /**
19     * Insere um dado valor na lista.
20     *
21     * @param value
22     *         Valor a ser inserido na lista.
23     */
24    public void inserir(Object value)
25    {
26        if (tamanho() < elementos.length)
27            this.elementos[cursor++] = value;
28    }
29
30    /**
31     * Remove a primeira ocorrência de um dado valor na {@link Lista}.
32     *
33     * @param value
34     *         Valor a ter a sua primeira ocorrência removida da lista.
35     */
36    public void remover(Object value)
37    {
38        int d = pesquisar(value);
39        if (d != -1)
40        {
41            for (int i = d; i < this.cursor; i++)
42            {
43                this.elementos[i] = this.elementos[i + 1];
44            }
45            this.cursor--;
46        }
47    }
48
49    /**
50     * Pesquisa por um dado valor na {@link Lista} e retorna a sua posição, caso
51     * esteja presente.
52     */
53 }
```

```

53  * @param value
54  *          Valor a ser pesquisado.
55  * @return A posição da primeira ocorrência de um dado valor na lista ou -1
56  *         se o valor não
57  *         estiver presente.
58  */
59 public int pesquisar(Object value)
60 {
61     int i = 0;
62     while (i <= this.cursor && this.elementos[i] != value)
63         i++;
64
65     return i > this.cursor ? -1 : i;
66 }
67
68 /**
69  * Inverte e retorna a ordem dos elementos dessa {@link Lista}, retornando
70  * uma nova
71  * {@link Lista}.
72  *
73  * @return Uma lista com os elementos dessa {@link Lista} na ordem inversa.
74  */
75 public Lista inverter()
76 {
77     Lista listaInvertida = new Lista(this.cursor);
78
79     Pilha p = new Pilha(this.cursor);
80
81     for (int i = 0; i < this.cursor; i++)
82     {
83         p.push(this.elementos[i]);
84     }
85
86     while (!p.isEmpty())
87         listaInvertida.inserir(p.pop());
88
89     return listaInvertida;
90 }
91
92 /**
93  * Retorna uma sublista dessa {@link Lista} a partir da posição p.
94  *
95  * @param p
96  *         Posição de início da nova sublista.
97  * @return Uma {@link Lista} que representa a sublista dessa {@link Lista} a
98  *         partir da posição
99  *         p.
100  */
101 public Lista subLista(int p)
102 {
103     Lista sublista = new Lista(this.cursor);
104     for (int i = p; i < this.cursor; i++)
105         sublista.inserir(this.elementos[i]);

```

```

103     return sublista;
104 }
105
106 /**
107  * Intercala os elementos dessa {@link Lista} com os de uma dada {@link
108   * Lista}.
109  *
110  * @param l2
111  *      {@link Lista} a ter os seus elementos intercalados com essa {
112   *      @link Lista}.
113  * @return Uma {@link Lista} com todos os elementos da {@link Lista} dada
114   * com os dessa
115   *      {@link Lista} de forma intercalada.
116  */
117 public Lista intercalar(Lista l2)
118 {
119     Lista intercalada = new Lista(this.cursor + l2.cursor);
120     for (int i = 0, j = 0, k = 0; k < (this.cursor + l2.cursor); k++)
121     {
122         if (k % 2 == 0)
123         {
124             if (i < this.cursor)
125                 intercalada.inserir(this.elementos[i++]);
126             else if (j < l2.cursor)
127                 intercalada.inserir(l2.elementos[j++]);
128         } else
129         {
130             if (j < l2.cursor)
131                 intercalada.inserir(l2.elementos[j++]);
132             else if (i < this.cursor)
133                 intercalada.inserir(this.elementos[i++]);
134         }
135     }
136
137     return intercalada;
138 }
139
140 /**
141  * Concatena essa {@link Lista} com uma dada {@link Lista}.
142  *
143  * @param l2
144  *      Lista a ser concatenada (conjunto união) com essa {@link Lista}
145   *
146  * @return Uma {@link Lista} com todos os elementos dessa {@link Lista} mais
147   * com os da
148   *      {@link Lista} dada.
149  */
150 public Lista concatenar(Lista l2)
151 {
152     Lista l3 = new Lista(this.cursor + l2.cursor);
153
154     for (int i = 0; i < this.cursor; i++)
155         l3.inserir(this.elementos[i]);

```

```

153     for (int i = 0; i < l2.cursor; i++)
        l3.inserir(l2.elementos[i]);

155     return l3;
157 }

159 /**
    * Retorna uma {@link Lista} representando o conjunto interseção entre essa
    * {@link Lista} e uma
    * dada {@link Lista}.
161     *
    * @param l2
163     *         A {@link Lista} a ser obtido o conjunto interseção com essa {
    *         {@link Lista}.
    * @return Uma {@link Lista} representando o conjunto interseção entre essa
    *         {@link Lista} e uma
165     *         dada {@link Lista}
    */
167 public Lista interseccao(Lista l2)
{
169     Lista interseccao = new Lista(this.cursor < l2.cursor ? l2.cursor : this.
        cursor);

171     for (int i = 0; i < this.cursor; i++)
    {
173         int d = l2.pesquisar(this.elementos[i]);
        if (d > -1)
175         {
            interseccao.inserir(elementos[i]);
177         }
    }
179     return interseccao;
}

181 /**
    * Retorna uma {@link Lista} representando o conjunto diferença entre essa {
    * {@link Lista} e uma
    * dada {@link Lista}.
185     *
    * @param l2
187     *         A {@link Lista} a ser obtido o conjunto diferença com essa {
    *         {@link Lista}.
    * @return Uma {@link Lista} representando o conjunto diferença entre essa {
    *         {@link Lista} e uma
189     *         dada {@link Lista}
    */
191 public Lista diferenca(Lista l2)
{
193     Lista diff = new Lista(this.cursor < l2.cursor ? l2.cursor : this.cursor);

195     for (int i = 0; i < this.cursor; i++)
    {
197         int d = l2.pesquisar(this.elementos[i]);
        if (d == -1)

```

```

199     {
200         diff.inserir(this.elementos[i]);
201     }
202 }
203
204 for (int i = 0; i < l2.cursor; i++)
205 {
206     int d = this.pesquisar(l2.elementos[i]);
207     if (d == -1)
208     {
209         diff.inserir(l2.elementos[i]);
210     }
211 }
212
213 return diff;
214 }
215
216 /**
217  * Retorna o tamanho da {@link Lista}.
218  *
219  * @return O tamanho da {@link Lista}.
220  */
221 public int tamanho()
222 {
223     return this.cursor;
224 }
225 }

```

Listing 1: Implementação de Lista Estática Linear em Java

1. Construa uma função que retorne o tamanho de uma lista estática.

```

1  /**
2   * Retorna o tamanho da {@link Lista}.
3   *
4   * @return O tamanho da {@link Lista}.
5   */
6  public int tamanho()
7  {
8      return this.cursor;
9  }

```

2. Dada uma lista estática de tamanho m , construa uma função que inverta a lista.

```

1  /**
2   * Inverte e retorna a ordem dos elementos dessa {@link Lista},
3   * retornando uma nova
4   * {@link Lista}.
5   *
6   * @return Uma lista com os elementos dessa {@link Lista} na ordem
7   * inversa.

```

```

7  */
  public Lista inverter()
  {
9      Lista listaInvertida = new Lista(this.cursor);

11     Pilha p = new Pilha(this.cursor);

13     for (int i = 0; i < this.cursor; i++)
        {
15         p.push(this.elementos[i]);
        }

17     while (!p.isEmpty())
19         listaInvertida.inserir(p.pop());

21     return listaInvertida;
  }

```

3. Dada uma lista estática de tamanho m , construir uma função que retorne uma *sublista* de tamanho n a partir da posição p . Compute a complexidade de tempo da sua função.

```

2  /**
   * Retorna uma sublista dessa {@link Lista} a partir da posição p.
   *
4   * @param p
   *         Posição de início da nova sublista.
6   * @return Uma {@link Lista} que representa a sublista dessa {@link
   *         Lista} a partir da posição
   *         p.
   */
  public Lista subLista(int p)
  {
10     Lista sublista = new Lista(this.cursor);
12     for (int i = p; i < this.cursor; i++)
        sublista.inserir(this.elementos[i]);
14
16     return sublista;
  }

```

4. Dada duas listas estáticas de tamanho m_1 e m_2 , respectivamente, construir uma função para intercalar as duas listas, gerando uma terceira.

```

2  /**
   * Intercala os elementos dessa {@link Lista} com os de uma dada {
   *     {@link Lista}.
   *
4   * @param l2

```

```

6      *      {@link Lista} a ter os seus elementos intercalados com
      *      essa {@link Lista}.
8      * @return Uma {@link Lista} com todos os elementos da {@link Lista}
      *      dada com os dessa
      *      {@link Lista} de forma intercalada.
      */
10     public Lista intercalar(Lista l2)
    {
12         Lista intercalada = new Lista(this.cursor + l2.cursor);
        for (int i = 0, j = 0, k = 0; k < (this.cursor + l2.cursor); k++)
        {
14             if (k % 2 == 0)
                {
16                 if (i < this.cursor)
                    intercalada.inserir(this.elementos[i++]);
18                 else if (j < l2.cursor)
                    intercalada.inserir(l2.elementos[j++]);
20                 } else
                {
22                 if (j < l2.cursor)
                    intercalada.inserir(l2.elementos[j++]);
24                 else if (i < this.cursor)
                    intercalada.inserir(this.elementos[i++]);
26                 }
                }
28         }

        return intercalada;
30     }

```

5. Dada duas listas estáticas de tamanho m_1 e m_2 , respectivamente, construir uma função para concatenar as duas listas, gerando uma terceira.

```

2     /**
      * Concatena essa {@link Lista} com uma dada {@link Lista}.
      *
4      * @param l2
      *      Lista a ser concatenada (conjunto união) com essa {@link
      *      Lista}
6      * @return Uma {@link Lista} com todos os elementos dessa {@link Lista}
      *      } mais com os da
      *      {@link Lista} dada.
      */
8     /**
10    public Lista concatenar(Lista l2)
    {
12        Lista l3 = new Lista(this.cursor + l2.cursor);

14        for (int i = 0; i < this.cursor; i++)
            l3.inserir(this.elementos[i]);

16        for (int i = 0; i < l2.cursor; i++)
            l3.inserir(l2.elementos[i]);
18    }

```

```
20     return l3;
    }
```

6. Uma maneira usual de representar conjuntos é listando seus elementos. Implemente uma aplicação que ofereça as operações usuais de conjuntos (união, intersecção e diferença), considerando que cada um dos conjuntos é representado por uma lista linear.

```
2  /**
3   * Retorna uma {@link Lista} representando o conjunto diferença entre
4   * essa {@link Lista} e uma
5   * dada {@link Lista}.
6   *
7   * @param l2
8   *      A {@link Lista} a ser obtido o conjunto diferença com
9   *      essa {@link Lista}.
10  * @return Uma {@link Lista} representando o conjunto diferença entre
11  *      essa {@link Lista} e uma
12  *      dada {@link Lista}
13  */
14 public Lista diferenca(Lista l2)
15 {
16     Lista diff = new Lista(this.cursor < l2.cursor ? l2.cursor : this.
17         cursor);
18
19     for (int i = 0; i < this.cursor; i++)
20     {
21         int d = l2.pesquisar(this.elementos[i]);
22         if (d == -1)
23         {
24             diff.inserir(this.elementos[i]);
25         }
26     }
27
28     for (int i = 0; i < l2.cursor; i++)
29     {
30         int d = this.pesquisar(l2.elementos[i]);
31         if (d == -1)
32         {
33             diff.inserir(l2.elementos[i]);
34         }
35     }
36
37     return diff;
38 }
```