Gabarito da Lista de Exercícios 3

```java
package br.projecao.ed.linear;

/**
 * Implementação de uma lista sequencial linear.
 */
public class Lista
{
  private Object[] elementos;

  private Integer cursor;

  public Lista(Integer capacidade)
  {
    this.elementos = new Object[capacidade];
    this.cursor = 0;
  }

  /**
   * Insere um dado valor na lista.
   *
   * @param value
   *            Valor a ser inserido na lista.
   */
  public void inserir(Object value)
  {
    if (tamanho() < elementos.length)
      this.elementos[cursor++] = value;
  }

  /**
   * Remove a primeira ocorrência de um dado valor na {@link Lista}.
   *
   * @param value
   *            Valor a ter a sua primeira ocorrência removida da lista.
   */
  public void remover(Object value)
  {
    int d = pesquisar(value);
    if (d != -1)
    {
      for (int i = d; i < this.cursor; i++)
      {
        this.elementos[i] = this.elementos[i + 1];
      }
      this.cursor--;
    }
  }

  /**
   * Pesquisa por um dado valor na {@link Lista} e retorna a sua posição, caso
       esteja presente.
   *
```

```
     * @param value
     *            Valor a ser pesquisado.
     * @return A posição da primeira ocorrência de um dado valor na lista ou -1 se o valor não
     *         estiver presente.
     */
    public int pesquisar(Object value)
    {
      int i = 0;
      while (i <= this.cursor && this.elementos[i] != value)
        i++;

      return i > this.cursor ? -1 : i;
    }

    /**
     * Inverte e retorna a ordem dos elementos dessa {@link Lista}, retornando uma nova
     * {@link Lista}.
     *
     * @return Uma lista com os elementos dessa {@link Lista} na ordem inversa.
     */
    public Lista inverter()
    {
      Lista listaInvertida = new Lista(this.cursor);

      Pilha p = new Pilha(this.cursor);

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

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

      return listaInvertida;
    }

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

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

```
    for (int i = 0; i < l2.cursor; i++)
153   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 intersecção 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
  /**
183  * 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         {
              diff.inserir(this.elementos[i]);
201         }
          }
203
          for (int i = 0; i < l2.cursor; i++)
205       {
            int d = this.pesquisar(l2.elementos[i]);
207         if (d == -1)
            {
209           diff.inserir(l2.elementos[i]);
            }
211       }

213       return diff;
        }
215
        /**
217      * Retorna o tamanho da {@link Lista}.
         *
219      * @return O tamanho da {@link Lista}.
         */
221     public int tamanho()
        {
223       return this.cursor;
        }
225 }
```

Listing 1: Implementação de Lista Estática Linear em Java

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

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

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

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

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

    Pilha p = new Pilha(this.cursor);

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

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

    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.

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

    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.

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

```
     *            {@link Lista} a ter os seus elementos intercalados com
         essa {@link Lista}.
     * @return Uma {@link Lista} com todos os elementos da {@link Lista}
         dada com os dessa
     *         {@link Lista} de forma intercalada.
     */
    public Lista intercalar(Lista l2)
    {
      Lista intercalada = new Lista(this.cursor + l2.cursor);
      for (int i = 0, j = 0, k = 0; k < (this.cursor + l2.cursor); k++)
      {
        if (k % 2 == 0)
        {
          if (i < this.cursor)
            intercalada.inserir(this.elementos[i++]);
          else if (j < l2.cursor)
            intercalada.inserir(l2.elementos[j++]);
        } else
        {
          if (j < l2.cursor)
            intercalada.inserir(l2.elementos[j++]);
          else if (i < this.cursor)
            intercalada.inserir(this.elementos[i++]);
        }
      }

      return intercalada;
    }
```

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

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

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

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

```

```
    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.

```
  /**
   * Retorna uma {@link Lista} representando o conjunto diferença entre
       essa {@link Lista} e uma
   * dada {@link Lista}.
   *
   * @param l2
   *            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
   *         dada {@link Lista}
   */
  public Lista diferenca(Lista l2)
  {
    Lista diff = new Lista(this.cursor < l2.cursor ? l2.cursor : this.
        cursor);

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

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

    return diff;
  }
```