1 - Lista Encadeada Circular

Uma lista encadeada circular ou simplesmente lista circular difere de uma lista encadeada apenas pelo fato do último nó da lista apontar para o 1º nó da lista, assim como está exemplificado na Figura 1. Cada nó é formado por 1 conteúdo e 1 endereço. Quando a lista possui apenas 1 nó, o próximo do nó será ele mesmo, assim como está representado na Figura 2.

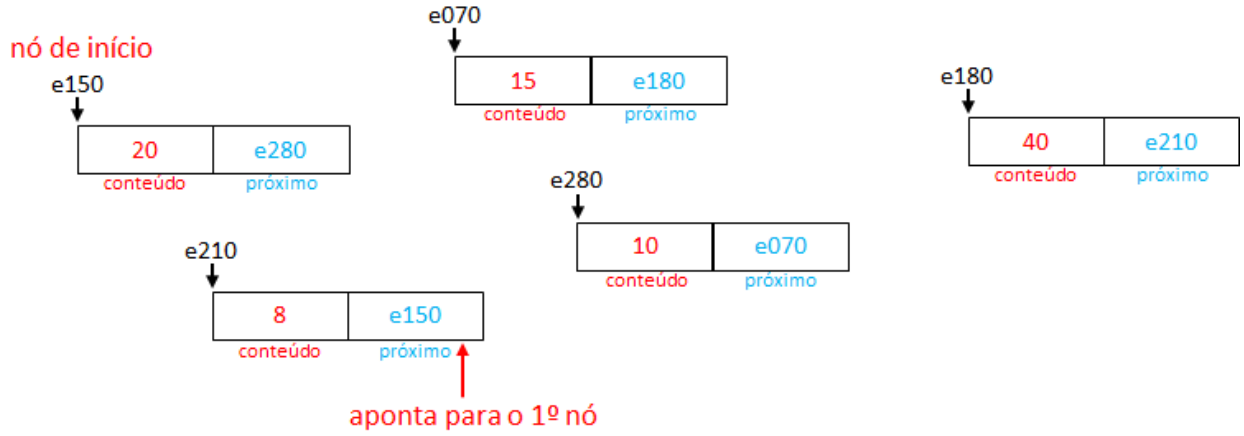

Figura 1 – Representação de uma lista encadeada circular na memória.

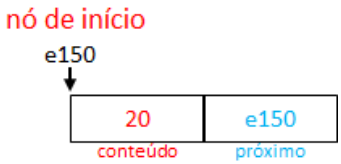

Figura 2 – Representação de uma lista encadeada circular com apenas 1 nó.

A Figura 3 possui uma implementação de lista encadeada circular e a Figura 4 possui o código da classe No, veja que a classe No não possui alterações com relação a lista encadeada. A Figura 5 possui um código para testar a classe Lista e a Figura 6 mostra o resultado. Na lista encadeada circular marcar o 1º nó como início é mera formalidade, pois qualquer nó pode ser o início de uma lista circular.

```
public class Lista {
   No inicio;

   Lista(){
      /* a lista está vazia */
      inicio = null;
   }

   void inserir(int nro){
      /* criar um nó */
      No no = new No();
      no.conteudo = nro;

      /* checa se a lista está vazia */
      if( inicio == null ){
         no.proximo = no;
         inicio = no;
      }
      else{
```

```
public class No {
   int conteudo;
   No proximo;
}
```

Figura 4 – Código da classe No.

```
public class Principal {
   public static void main(String[] args) {
      Lista lista = new Lista();
      lista.remover(10);
      lista.imprimir();
      lista.inserir(20);
      lista.imprimir();
      lista.inserir(10);
      lista.imprimir();
```

```
        /* percorrer a lista até encontrar o último nó */
        No ultimo = inicio;
        while( ultimo.proximo != inicio ){
           ultimo = ultimo.proximo;
        }
        /* alterar o próximo do último para o endereço do no */
        no.proximo = ultimo.proximo;
        ultimo.proximo = no;
     }
  }

  void imprimir(){
     /* checa se a lista está vazia */
     if( inicio == null ){
        System.out.println("\nLista vazia");
     }
     else{
        System.out.println(); /* quebra de linha na tela */
        /* percorrer a lista até voltar ao nó de início */
        No ultimo = inicio;
        do{
           System.out.print( ultimo.conteudo +" ");
           ultimo = ultimo.proximo;
        }while( ultimo != inicio );
     }
  }

  /*irá retornar o nó que possui o nro ou null caso contrário*/
  No buscar(int nro){
     if( inicio == null ){
        return null;
     }
     else{
        No atual = inicio;
        /* percorre até encontrar o nro ou o fim da lista */
        while( atual.proximo != inicio &&
               atual.conteudo != nro ){
           atual = atual.proximo;
        }
        return atual.conteudo == nro? atual : null;
     }
  }

  void remover(int nro){
     if( inicio != null ){
        /* ponteiro para o elemento anterior*/
        No anterior = inicio;
        /* ponteiro para o elemento atual*/
        No atual = inicio.proximo;

        /* percorre até encontrar o nro ou o fim da lista */
        while( atual != inicio && atual.conteudo != nro ){
           anterior = atual;
           atual = atual.proximo;
        }

        if( atual == anterior ){
           /* remove o único nó da lista */
           inicio = null;
        }
        else if( atual == inicio ){
           /* remove o 1o nó da lista */
           inicio = inicio.proximo;
           anterior.proximo = inicio;
        }
        else{
           /* remove do meio ou final da sequência */
           anterior.proximo = atual.proximo;
        }
     }
  }
}
```

Figura 3 – Implementação da lista encadeada circular.

```
      lista.inserir(15);
      lista.imprimir();
      lista.inserir(40);
      lista.imprimir();
      lista.inserir(8);
      lista.imprimir();
      lista.remover(8);
      lista.imprimir();
      lista.remover(20);
      lista.imprimir();
      lista.remover(15);
      lista.imprimir();
      lista.remover(40);
      lista.imprimir();
      lista.remover(10);
      lista.imprimir();
   }
}
```

Figura 5 – Código da classe Principal para testar a lista encadeada circular.

```
Lista vazia

20
20 10
20 10 15
20 10 15 40
20 10 15 40 8
20 10 15 40
10 15 40
10 40
10
Lista vazia
```

Figura 6 – Resultado do código da Figura 5.

2 - Lista Duplamente Encadeada

Uma lista duplamente encadeada é uma lista encadeada que possui o endereço do nó anterior e do próximo nó. A Figura 7 mostra uma representação do nó e a Figura 8 mostra uma representação da lista na memória.

Figura 7 – Representação de um nó da lista duplamente encadeada.

Uma lista duplamente encadeada possui a vantagem que partindo de qualquer nó da lista pode-se ir para à esquerda ou direita.

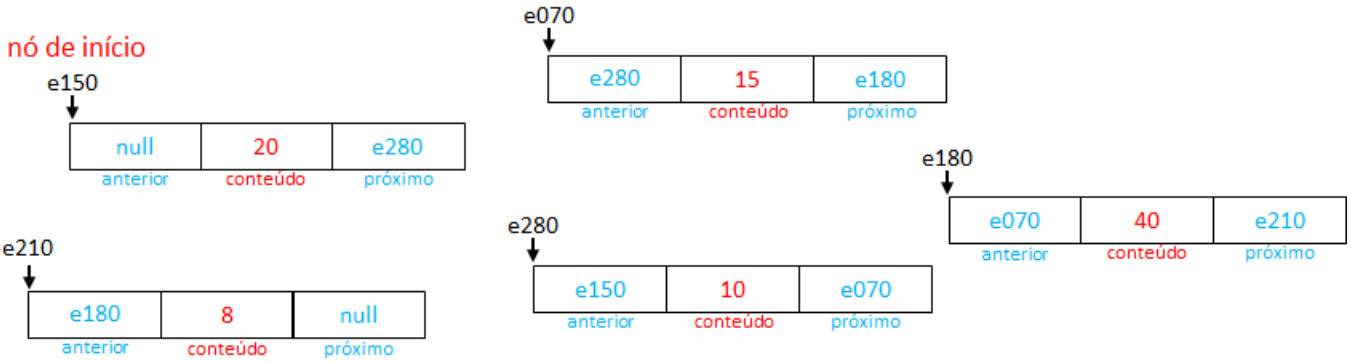

Figura 8 – Representação de uma lista duplamente encadeada na memória.

A Figura 9 possui uma implementação de lista duplamente encadeada e a Figura 10 possui o código da classe No, veja que a classe No possui os atributos anterior e proximo, esses atributos receberão endereços de objetos do tipo No. A Figura 11 possui um código para testar a classe Lista e a Figura 12 mostra o resultado.

Na lista duplamente encadeada qualquer nó pode ser o início, mas se o nó de início não for o mais à esquerda, então o processo de navegação precisará envolver o uso do atributo anterior e não apenas do proximo, assim como foi usado no exemplo da Figura 9.

```
public class Lista {
   No inicio;

   Lista(){
      inicio = null; /* a lista está vazia */
   }

   /* inserir no final da lista */
   void inserir(int nro){
      /* criar um nó */
      No no = new No();
      no.conteudo = nro;
      no.anterior = null;
      no.proximo = null;

      /* checa se a lista está vazia */
      if( inicio == null ){
         inicio = no;
      }
      else{
         /* percorrer a lista até encontrar o último nó */
         No ultimo = inicio;
         while( ultimo.proximo != null ){
            ultimo = ultimo.proximo;
         }
```

```
public class No {
   int conteudo;
   No anterior, proximo;
}
```

Figura 10 – Código da classe No.

```
public class Principal {
   public static void main(String[] args) {
      Lista lista = new Lista();
      lista.remover(10);
      lista.imprimir();
      lista.inserir(20);
      lista.imprimir();
      lista.inserir(10);
      lista.imprimir();
      lista.inserir(15);
      lista.imprimir();
      lista.inserir(40);
      lista.imprimir();
      lista.inserir(8);
```

```
            /* alterar o próximo do último para o endereço do no */
            ultimo.proximo = no;
            no.anterior = ultimo;
        }
    }

    void imprimir(){
        /* checa se a lista está vazia */
        if( inicio == null ){
            System.out.println("\nLista vazia");
        }
        else{
            System.out.println(); /* quebra de linha na tela */
            /* percorrer a lista até encontrar o último nó */
            No ultimo = inicio;
            while( ultimo != null ){
                System.out.print( ultimo.conteudo +" ");
                ultimo = ultimo.proximo;
            }
        }
    }

    /*irá retornar o nó que possui o nro ou null caso contrário*/
    No buscar(int nro){
        No atual = inicio;
        /* percorre até encontrar o nro ou o fim da lista */
        while( atual != null && atual.conteudo != nro ){
            atual = atual.proximo;
        }
        return atual;
    }

    void remover(int nro){
        /* checa se a lista está vazia */
        if( inicio != null ){
            No atual = inicio;
            /* procura o nó que possui o nro */
            while( atual != null && atual.conteudo != nro ){
                atual = atual.proximo;
            }
            /* checa se o nro foi encontrado */
            if( atual != null ){
                /* checa se é o 1o nó */
                if( atual == inicio ){
                    /* checa se existe somente 1 nó */
                    if( atual.proximo == null ){
                        inicio = null;
                    }
                    else{
                        inicio = atual.proximo;
                        atual.proximo.anterior = atual.anterior;
                    }
                }
                /* checa se é o último nó */
                else if( atual.proximo == null ){
                    atual.anterior.proximo = atual.proximo;
                }
                else{ /* o nó está no meio da lista */
                    atual.anterior.proximo = atual.proximo;
                    atual.proximo.anterior = atual.anterior;
                }
            }
            else{
                System.out.println(nro + " não existe");
            }
        }
    }
}
```

Figura 9 – Implementação da lista duplamente encadeada.

```
        lista.imprimir();
        lista.remover(8);
        lista.imprimir();
        lista.remover(20);
        lista.imprimir();
        lista.remover(15);
        lista.imprimir();
        lista.remover(40);
        lista.imprimir();
        lista.remover(10);
        lista.imprimir();
    }
}
```

Figura 11 – Código da classe Principal para testar a lista duplamente encadeada.

```
Lista vazia

20
20 10
20 10 15
20 10 15 40
20 10 15 40 8
20 10 15 40
10 15 40
10 40
10
Lista vazia
```

Figura 12 – Resultado do código da Figura 11.

3 - Lista Circular Duplamente Encadeada

Uma lista circular duplamente encadeada é semelhante a lista duplamente encadeada, a diferença é que o último nó possui como próximo o 1º nó e o 1º nó possui como anterior o último. A Figura 13 mostra a estrutura de um nó e a Figura 14 mostra a representação de uma lista circular duplamente encadeada na memória, veja que o último nó se conecta ao 1º e o 1º ao último.

Figura 13 – Representação de um nó da lista circular duplamente encadeada.

Uma lista circular duplamente encadeada possui a vantagem que partindo de qualquer nó da lista pode-se ir para à esquerda ou direita bem como voltar ao ponto de partida, uma vez que ela é circular.

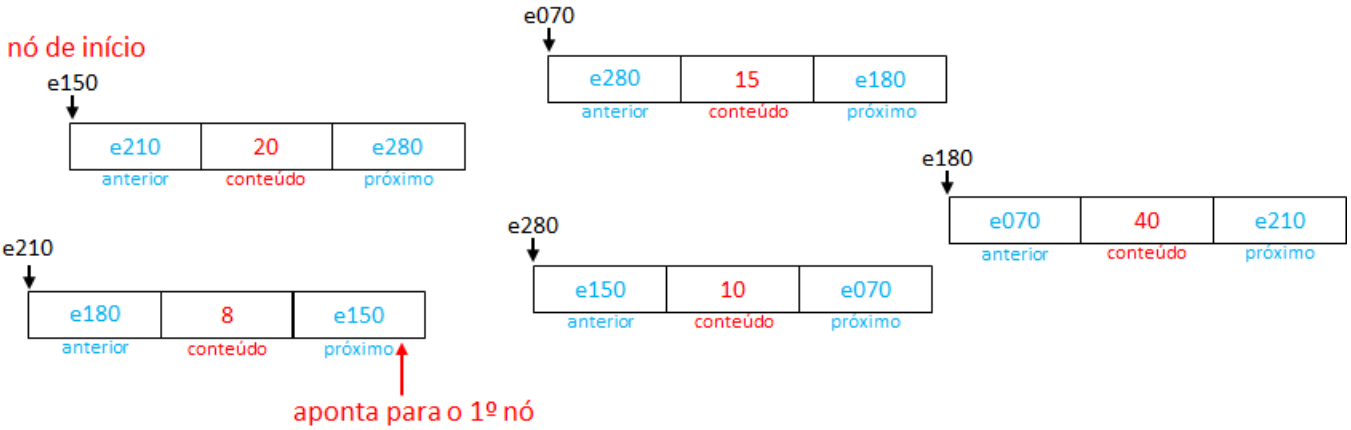

Figura 14 – Representação de uma lista circular duplamente encadeada na memória.

```java
public class Lista {
   No inicio;

   Lista(){
      /* a lista está vazia */
      inicio = null;
   }

   /* inserir no final da lista */
   void inserir(int nro){
      /* criar um nó */
      No no = new No();
      no.conteudo = nro;
      no.anterior = null;
      no.proximo = null;

      /* checa se a lista está vazia */
      if( inicio == null ){
         inicio = no;
         no.proximo = no;
         no.anterior = no;
      }
      else{
         /* percorrer a lista até encontrar o último nó */
         No ultimo = inicio;
         while( ultimo.proximo != inicio ){
            ultimo = ultimo.proximo;
         }
         /* alterar o próximo do último para o endereço do no */
         ultimo.proximo.anterior = no;
         no.proximo = ultimo.proximo;
         no.anterior = ultimo;
         ultimo.proximo = no;
```

```java
public class No {
   int conteudo;
   No anterior, proximo;
}
```

Figura 16 – Código da classe No.

```java
public class Principal {
   public static void main(String[] args) {
      Lista lista = new Lista();
      lista.remover(10);
      lista.imprimir();
      lista.inserir(20);
      lista.imprimir();
      lista.inserir(10);
      lista.imprimir();
      lista.inserir(15);
      lista.imprimir();
      lista.inserir(40);
      lista.imprimir();
      lista.inserir(8);
      lista.imprimir();
      lista.remover(8);
      lista.imprimir();
      lista.remover(20);
      lista.imprimir();
      lista.remover(15);
      lista.imprimir();
      lista.remover(40);
      lista.imprimir();
```

```
      }
   }

   void imprimir(){
      /* checa se a lista está vazia */
      if( inicio == null ){
         System.out.println("\nLista vazia");
      }
      else{
         System.out.println(); /* quebra de linha na tela */
         /* percorrer a lista até voltar ao nó de início */
         No ultimo = inicio;
         do{
            System.out.print( ultimo.conteudo +" ");
            ultimo = ultimo.proximo;
         }while( ultimo != inicio );
      }
   }


   /*irá retornar o nó que possui o nro ou null caso contrário*/
   No buscar(int nro){
      if( inicio == null ){
         return null;
      }
      else{
         No atual = inicio;
         /* percorre até encontrar o nro ou o fim da lista */
         while( atual.proximo != inicio &&
               atual.conteudo != nro ){
            atual = atual.proximo;
         }
         return atual.conteudo == nro? atual : null;
      }
   }


   void remover(int nro){
      /* checa se a lista está vazia */
      if( inicio != null ){
         No atual = inicio;
         /* procura o nó que possui o nro */
         while( atual.proximo != inicio &&
                atual.conteudo != nro){
            atual = atual.proximo;
         }
         /* checa se o nro foi encontrado */
         if( atual.conteudo == nro ){
            /* checa se é o 1o nó */
            if( atual == inicio ){
               /* checa se existe somente 1 nó */
               if( atual.proximo == atual ){
                  inicio = null;
               }
               else{
                  inicio = atual.proximo;
                  atual.anterior.proximo = atual.proximo;
                  atual.proximo.anterior = atual.anterior;
               }
            }
            else{ /* é do meio ou final da lista */
               atual.anterior.proximo = atual.proximo;
               atual.proximo.anterior = atual.anterior;
            }
         }
         else{
            System.out.println(nro + " não existe");
         }
      }
   }
}
```

Figura 15 – Implementação da lista circular duplamente encadeada.

```
      lista.remover(10);
      lista.imprimir();
   }
}
```

Figura 17 – Código da classe Principal para testar a lista circular duplamente encadeada.

```
Lista vazia

20
20 10
20 10 15
20 10 15 40
20 10 15 40 8
20 10 15 40
10 15 40
10 40
10
Lista vazia
```

Figura 18 – Resultado do código da Figura 17.

4 - Exercícios

1 – Programar o método inserirSort(int) na classe Lista da Figura 3. O método deverá inserir os elementos na lista mantendo a ordem. Observação: os demais métodos **não** podem ser alterados.

2 – Programar o método inserirSort(int) na classe Lista da Figura 9. O método deverá inserir os elementos na lista mantendo a ordem. Observação: os demais métodos **não** podem ser alterados.

3 – Programar o método inserirSort(int) na classe Lista da Figura 15. O método deverá inserir os elementos na lista mantendo a ordem. Observação: os demais métodos **não** podem ser alterados.