Código 6 - Estruturas de dados genéricas personalizadas e lista encadeada

Resumo: O código apresentado exemplifica o uso de conceitos fundamentais de Java, como generics, encapsulamento, orientação a objetos e a implementação de listas duplamente encadeadas. A classe NoDuplo define a estrutura de um nó, armazenando dados genéricos e referências tanto para o nó anterior quanto para o próximo, o que permite a navegação bidirecional pela lista. A classe ListaDuplamenteEncadeada implementa operações essenciais como a adição de elementos no início e no final da lista, a remoção de nós tanto do início quanto do final, e a verificação da lista vazia. A classe Principal demonstra o uso da lista duplamente encadeada, manipulando elementos, removendo-os, e exibindo a estrutura da lista em diferentes momentos. O material foi elaborado com base no livro "Java: Como Programar" (Deitel) para conceitos de orientação a objetos e generics, e utiliza a documentação oficial da linguagem Java para a implementação técnica.

Será detalhado abaixo cada parte das classes ***NoDuplo***, **ListaDupla** e ***Principal***, explicando os conceitos usados nos códigos.

Classe “NoDuplo”

```java
public class NoDuplo<T> {
    private T dado;  // Armazena o valor genérico do nó
    private NoDuplo<T> proximoNo; // Referência ao próximo nó na lista.
    private NoDuplo<T> anteriorNo; // Referência ao nó anterior na lista.
    private int indice;  // Atributo para armazenar o índice
```

Essa classe define um nó **genérico NoDuplo<T>** para uma estrutura de dados. O **T** é um parâmetro de tipo genérico que permite que a classe funcione com qualquer tipo de dado. O atributo **dado** armazena o valor do nó; o **próximoNo** é uma referência ao próximo nó na lista; o **anteriorNo** é uma referência ao nó anterior da lista; e o **indice** é o atributo que armazena a posição (índice) do nó da lista.

```java
public NoDuplo(T dado, int indice) {
    this.dado = dado;
    this.indice = indice;
    this.proximoNo = null;
    this.anteriorNo = null;
}
```

Esse construtor Inicializa o nó com o valor **dado** e o **indice**. Já **proximoNo** e **anteriorNo** são iniciados como **null**, pois o nó recém-criado ainda não está conectado a outros nós.

```java
public T getDado() {
    return dado;
}

public void setDado(T dado) {
    this.dado = dado;
}
```

Esses são os métodos de acesso (**getters** e **setters**) para o atributo **dado**. Eles permitem ler e modificar o valor armazenado no nó.

```
public NoDuplo<T> getProximoNo() {
    return proximoNo;
}

public void setProximoNo(NoDuplo<T> proximoNo) {
    this.proximoNo = proximoNo;
}
```

Esses métodos são usados para definir e acessar o próximo nó na lista. **setProximoNo** permite vincular o nó atual a outro nó, e **getProximoNo** retorna a referência ao próximo nó.

```
public NoDuplo<T> getAnteriorNo() {
    return anteriorNo;
}

public void setAnteriorNo(NoDuplo<T> anteriorNo) {
    this.anteriorNo = anteriorNo;
}
```

Esses métodos são usados para definir e acessar o nó anterior na lista. **setAnteriorNo** permite vincular o nó atual a outro nó, e **getAnteriorNo** retorna a referência ao nó anterior.

```
public int getIndice() {
    return indice;
}

public void setIndice(int indice) {
    this.indice = indice;
}
```

Esses métodos são usados para definir e modificar o índice do nó.

```
@Override
public String toString() {
    return "{ Índice: " + getIndice() + ", Dado: " + getDado() + " }";
}
```

O método **toString** foi sobrescrito para fornecer uma representação em string do objeto **NoDuplo**. Ele retorna uma **string** como representação do nó, com seu índice e valor, útil para depuração e impressão da lista.

Classe “ListaDupla”

```
public class ListaDupla<T> {
    private NoDuplo<T> primeiroNo; // Referência para o primeiro nó da lista.
    private NoDuplo<T> ultimoNo; // Referência para o último nó da lista.
    private String nomeLista; // Nome da lista (pode ser personalizado).
    private int tamanho;  // Atributo para manter o tamanho da lista
```

A classe **ListaDupla** é uma implementação genérica de uma lista encadeada, onde **T** representa o tipo de dados que a lista armazenará. Os atributos **primeiroNo** e **ultimoNo** são referências ao primeiro e ao último nó da lista, respectivamente, enquanto **nomeLista** armazena o nome descritivo da lista, e **tamanho** armazena o número de nós na lista.

```
public ListaDupla() {
    this("Lista Duplamente Encadeada");
}
```

Esse é o construtor padrão que inicializa uma lista com o nome padrão "**Lista Duplamente Encadeada**". Ele chama outro construtor da classe, passando o nome como argumento.

```
public ListaDupla(String nomeLista) {
    this.nomeLista = nomeLista;
    this.primeiroNo = null;
    this.ultimoNo = null;
    this.tamanho = 0;
}
```

Esse construtor permite criar uma lista com um nome específico, definido pelo usuário. Inicialmente, a lista está vazia, então **primeiroNo** e **ultimoNo** são definidos como null e **tamanho** como 0.

```
public void addInicio(T dado) {
    NoDuplo<T> novoNo = new NoDuplo<T>(dado, 0);
    if (primeiroNo == null) {
        primeiroNo = ultimoNo = novoNo;
    } else {
        novoNo.setProximoNo(primeiroNo);
        primeiroNo.setAnteriorNo(novoNo);
        primeiroNo = novoNo;
    }
    atualizaIndices(); // Atualiza os índices após a inserção
    tamanho++;
}
```

O método **addInicio** insere um novo nó no início da lista. Se a lista estiver vazia (**primeiroNo** é **null**), o novo nó se torna tanto o **primeiro** quanto o **último** nó da lista. Caso contrário, o novo nó é vinculado ao nó que era o **primeiro**, e então se torna o novo **primeiroNo**. Depois, chama o método **atualizaIndices** para ajustar os índices e aumenta o **tamanho** da lista.

```
public void addFinal(T dado) {
    NoDuplo<T> novoNo = new NoDuplo<T>(dado, tamanho);
    if (ultimoNo == null) {
        primeiroNo = ultimoNo = novoNo;
    } else {
        novoNo.setAnteriorNo(ultimoNo);
        ultimoNo.setProximoNo(novoNo);
        ultimoNo = novoNo;
    }
    tamanho++;
}
```

O método **addFinal** insere um novo nó no **final** da lista. Se a lista estiver vazia (**ultimoNo** é **null**), o novo nó se torna tanto o **primeiro** quanto o **último**. Caso contrário, o novo nó é adicionado após o último nó, e então se torna o novo **ultimoNo**, ajustando as referências e o tamanho.

```
public void addMeio(T dado, int posicao) {
    if (posicao <= 0) {
        addInicio(dado);
        return;
    }
    if (posicao >= tamanho) {
        addFinal(dado);
        return;
    }
    NoDuplo<T> novoNo = new NoDuplo<>(dado, posicao);
    NoDuplo<T> atual = primeiroNo;
    int indice = 0;

    while (atual != null && indice < posicao) {
        atual = atual.getProximoNo();
        indice++;
    }

    NoDuplo<T> anterior = atual.getAnteriorNo();
    novoNo.setProximoNo(atual);
    novoNo.setAnteriorNo(anterior);

    if (anterior != null) {
        anterior.setProximoNo(novoNo);
    } else {
        primeiroNo = novoNo;
    }
    atual.setAnteriorNo(novoNo);
    atualizaIndices();
    tamanho++;
}
```

O método **addMeio** insere um novo nó em uma posição intermediária da lista. Caso a posição seja inválida, o nó é inserido no início (**posicao** menor ou igual a 0) ou final (**posicao** maior ou igual ao tamanho da lista). Caso contrário, percorre a lista até a posição desejada, ajusta as referências do nó anterior e do nó seguinte, insere o novo nó e atualiza os índices.

```
public void removeInicio() {
    if (primeiroNo == null) {
        System.out.println("Lista Vazia");
    } else {
        System.out.println("Dado: " + primeiroNo.getDado() + " removido da lista.");
        primeiroNo = primeiroNo.getProximoNo();
        if (primeiroNo != null) {
            primeiroNo.setAnteriorNo(null);
        } else {
            ultimoNo = null;
        }
        atualizaIndices();
        tamanho--;
    }
}
```

O método **removeInicio** remove o primeiro nó da lista. Se a lista estiver vazia, ele exibe uma mensagem indicando que a lista está vazia. Caso contrário, o nó atual **primeiroNo** é removido, e o próximo nó na sequência se torna o novo **primeiroNo**. O método ajusta os índices e decrementa o tamanho.

```
public void removeFinal() {
    if (ultimoNo == null) {
        System.out.println("Lista Vazia");
    } else {
        System.out.println("Dado: " + ultimoNo.getDado() + " removido da lista.");
        ultimoNo = ultimoNo.getAnteriorNo();
        if (ultimoNo != null) {
            ultimoNo.setProximoNo(null);
        } else {
            primeiroNo = null;
        }
        tamanho--;
    }
}
```

O método **removeFinal** remove o último nó da lista. Se a lista estiver vazia, ele exibe uma mensagem indicando que a lista está vazia. Caso contrário, ele percorre a lista para encontrar o **penúltimo** nó, define este nó como o novo **ultimoNo**, desvincula o último nó da lista e decrementa o **tamanho**.

```
public void removeMeio(int posicao) {
    if (posicao <= 0) {
        removeInicio();
        return;
    }
    if (posicao >= tamanho - 1) {
        removeFinal();
        return;
    }
    NoDuplo<T> atual = primeiroNo;
    int indice = 0;

    while (atual != null && indice < posicao) {
        atual = atual.getProximoNo();
        indice++;
    }
    NoDuplo<T> anterior = atual.getAnteriorNo();
    NoDuplo<T> proximo = atual.getProximoNo();

    if (anterior != null) {
        anterior.setProximoNo(proximo);
    }
    if (proximo != null) {
        proximo.setAnteriorNo(anterior);
    }
    System.out.println("Dado: " + atual.getDado() + " removido da lista.");
    atualizaIndices();
    tamanho--;
}
```

O método **removeMeio** remove um nó que se encontra no meio da lista, ou seja, em uma posição específica. Se a posição for menor ou igual a 0, o método chama **removeInicio()**, pois isso indica que o nó a ser removido está no começo da lista. Se a posição for maior ou igual ao índice do último nó (ou seja, tamanho - 1), o método chama **removeFinal()**, pois o nó a ser removido está no final da lista. O método percorre a lista começando pelo **primeiroNo**. Utiliza uma variável de controle **indice** que é comparada com a posição fornecida. A cada iteração do **while**, o método avança para o próximo nó (**atual = atual.getProximoNo()**) e incrementa o índice até alcançar o nó na posição desejada. Após localizar o nó a ser removido, ele armazena os nós anterior e próximo do nó atual. Se o nó anterior não for **null**, ele ajusta o ponteiro **proximoNo** do nó anterior para apontar para o próximo nó. Se o nó próximo não for **null**, ele ajusta o ponteiro **anteriorNo** do nó seguinte para apontar para o nó anterior. Isso efetivamente "desconecta" o nó da lista duplamente encadeada. O método imprime o valor do dado do nó removido com a mensagem "Dado: X removido da lista." Após a remoção do nó, o método chama **atualizaIndices()** para garantir que os índices dos nós restantes na lista sejam ajustados adequadamente. O tamanho da lista é decrementado em 1 após a remoção bem-sucedida do nó.

```
private void atualizaIndices() {
    NoDuplo<T> atual = primeiroNo;
    int indice = 0;
    while (atual != null) {
        atual.setIndice(indice);
        atual = atual.getProximoNo();
        indice++;
    }
}
```

O método **atualizaIndices** percorre a lista e atualizar os índices de todos os nós na lista. A inicialização começa com o nó **primeiroNo** e um índice **0**. A lista é percorrida pelo nó **atual** até que ele seja **null** (ou seja, enquanto houver nós na lista), com isso o índice do nó atual é definido com o valor atual do índice. Depois o nó atual é movimentado para o próximo nó da lista (**atual = atual.getProximoNo()**) e o índice é incrementado (**indice++**). Após a execução deste método, todos os nós da lista terão seus índices atualizados de acordo com suas posições na lista.

```
public void imprimeLista() {
    if (primeiroNo == null) {
        System.out.println("Lista Vazia");
    } else {
        System.out.printf("Dados da lista de %s:\n", nomeLista);
        NoDuplo<T> aux = primeiroNo;
        while (aux != null) {
            System.out.printf(" %s ", aux.toString());
            aux = aux.getProximoNo();
        }
        System.out.println();
    }
}
```

O método **imprimeLista** percorre a lista e imprime todos os dados armazenados. Se a lista estiver vazia, uma mensagem indicando isso é exibida. Caso contrário, ele percorre cada nó da lista, imprimindo o dado armazenado em cada um.

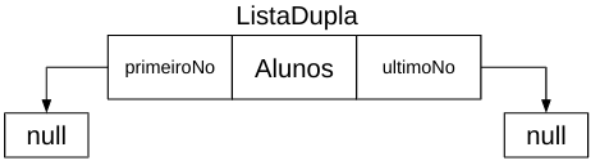
Classe “Principal”

```
public class Principal{
    public static void main(String[] args){
```

A classe **Principal** contém o método **main**, que é o ponto de entrada do programa.

```
ListaDupla<String> lista = new ListaDupla<String>("Alunos");
```

Essa linha cria um objeto da classe **ListaDupla** chamado lista, com o tipo genérico **String**, que armazena nomes de alunos. O nome da lista é definido como "**Alunos**".

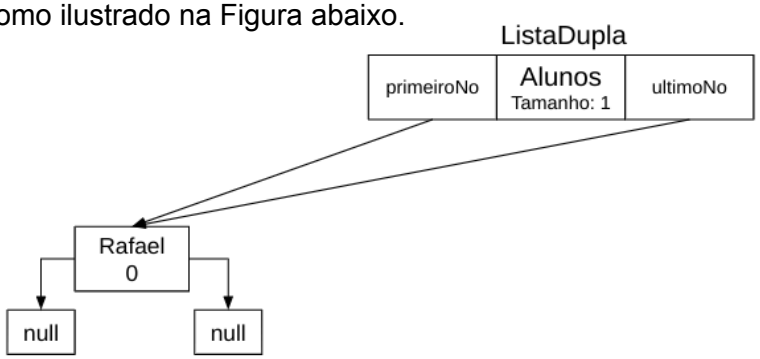



```
lista.imprimeLista();
```

Essa linha chama o método **imprimeLista** do objeto **lista** da classe **ListaDupla**. Esse método verifica se a lista está vazia. Se estiver, ele imprime "Lista Vazia". Caso contrário, ele percorre todos os nós da lista e imprime os dados armazenados em cada nó. Como essa é a primeira chamada a **imprimeLista**, e nenhum elemento foi adicionado ainda, a saída será "Lista Vazia".

```
lista.addInicio("Rafael");
```

O método **addInicio** é chamado com o argumento "Rafael". Esse método cria um novo nó contendo o **dado** "**Rafael**" e o insere no início da lista. Se a lista estava vazia, esse nó se torna o primeiro e o último nó da lista, como ilustrado na Figura abaixo.

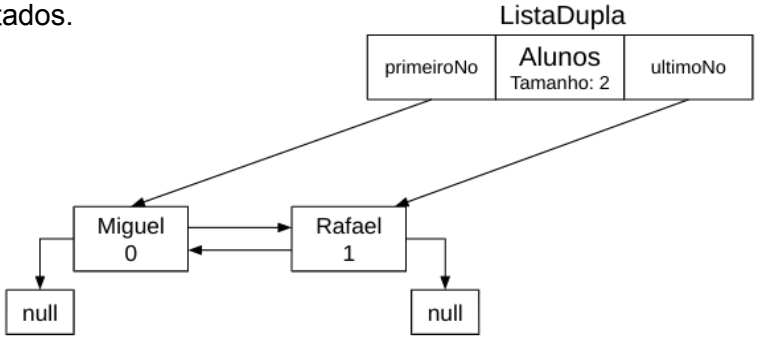



```
lista.imprimeLista();
```

Após adicionar "**Rafael**" à lista, o método **imprimeLista** é chamado novamente. Dessa vez, o método vai percorrer a lista e imprimir apenas os dados do nó "**Rafael**", que foi adicionado no início da lista.

```
lista.addInicio("Miguel");
```

Essa linha adiciona o dado "Miguel" no início da lista, assim como foi feito com "Rafael". Agora, "Miguel" se torna o primeiro nó da lista, e o nó que contém "Rafael" se torna o segundo, e os índices dos nós são ajustados.

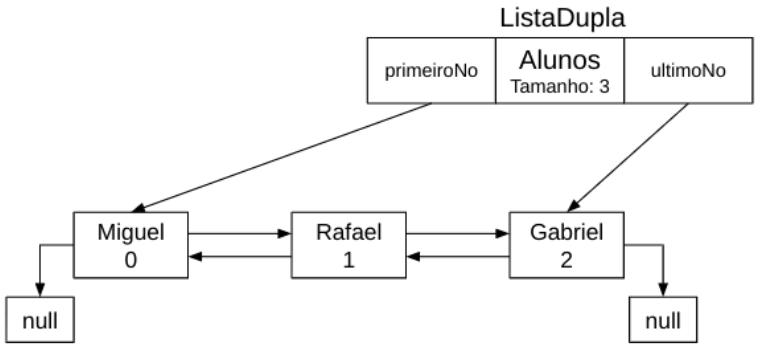


```
lista.imprimeLista();
```

Após adicionar "**Miguel**", a lista contém dois elementos. Ao chamar **imprimeLista**, o método imprime os dados dos nós em ordem: "**Miguel**" (primeiro) e "**Rafael**" (segundo).

```
lista.addFinal("Gabriel");
```

Nessa linha, o método **addFinal** é chamado com o parâmetro "**Gabriel**". Esse método adiciona um novo nó contendo "**Gabriel**" no final da lista. Se a lista estivesse vazia (o que não é o caso aqui), "**Gabriel**" se tornaria tanto o primeiro quanto o último nó. Como a lista já tem dois elementos, "**Gabriel**" se torna o **novo** último nó, e o nó anterior que continha "Rafael" se torna o penúltimo.

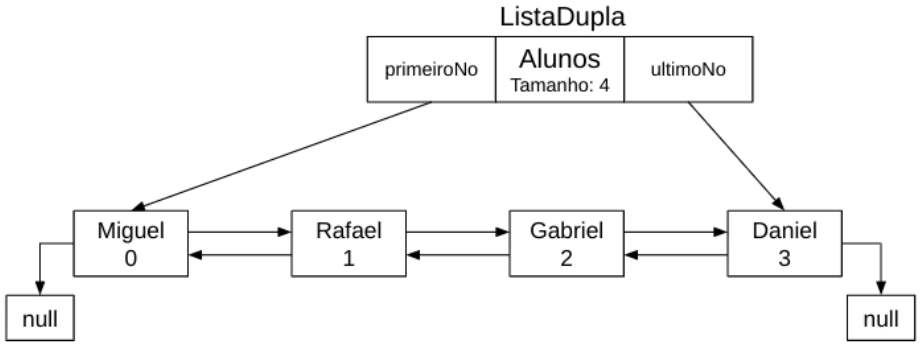



```
lista.imprimeLista();
```

Agora, ao chamar **imprimeLista**, a lista contém três elementos: "**Miguel**", "**Rafael**" e "**Gabriel**", nessa ordem. O método imprime os dados dos três nós.

```
lista.addFinal("Daniel");
```

Essa linha adiciona o dado "**Daniel**" ao final da lista, da mesma forma que foi feito com "**Gabriel**". Agora, "Daniel" se torna o último nó da lista, como ilustrado abaixo.

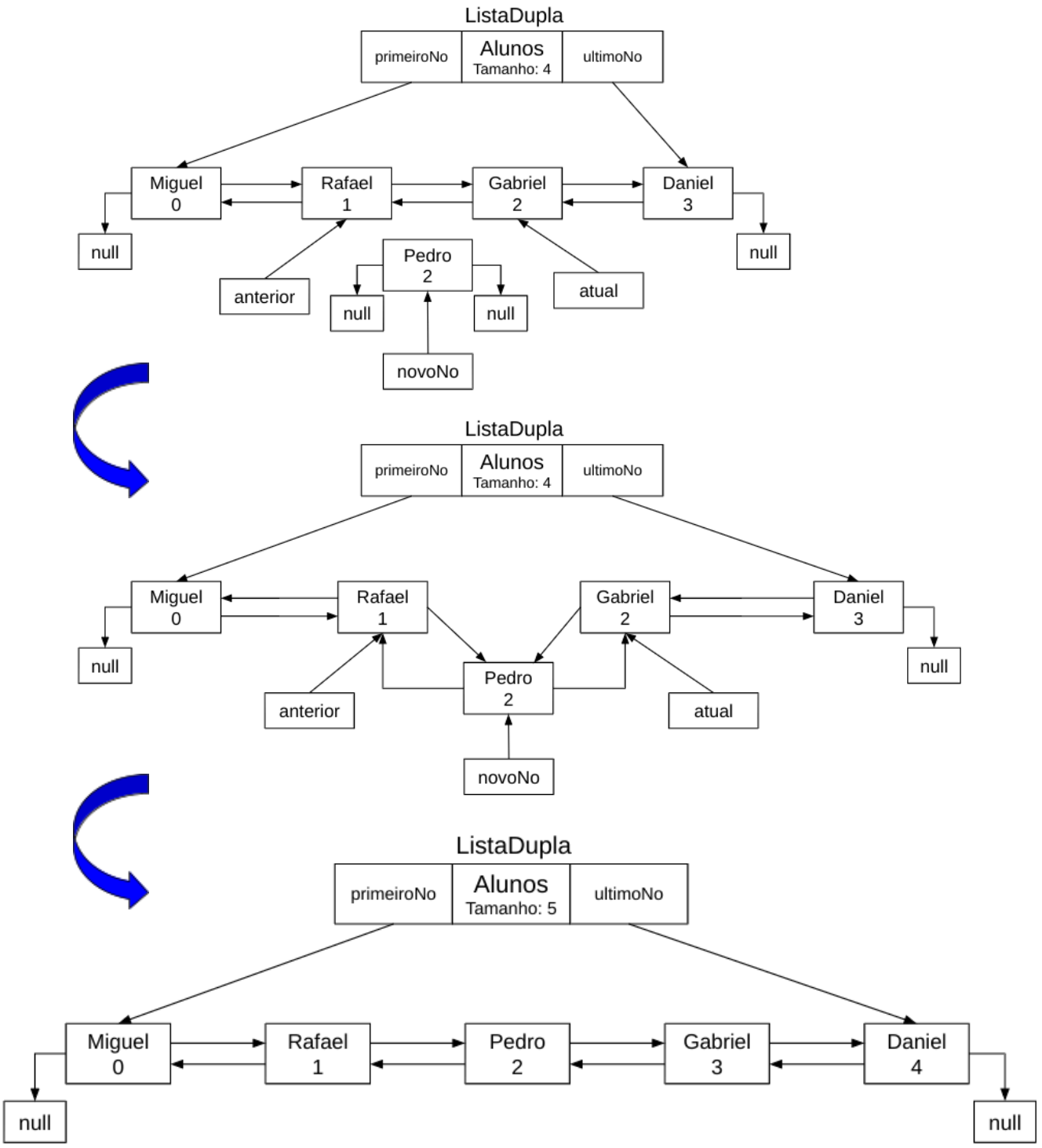



```
lista.imprimeLista();
```

Com quatro elementos na lista, a chamada ao método **imprimeLista** imprime os dados dos nós em ordem: "**Miguel**", "**Rafael**", "**Gabriel**" e "**Daniel**".

```
lista.addMeio("Pedro", 2);
```

Nessa linha, o método **addMeio()** é chamado para inserir o elemento "**Pedro**" na posição de índice **2** da lista. Esse método percorre a lista até encontrar a posição correta onde o novo nó deve ser inserido. Isso faz com que o nó que continha "Gabriel" e todos os nós subsequentes sejam deslocados uma posição para frente, como ilustrado nas figuras a seguir.

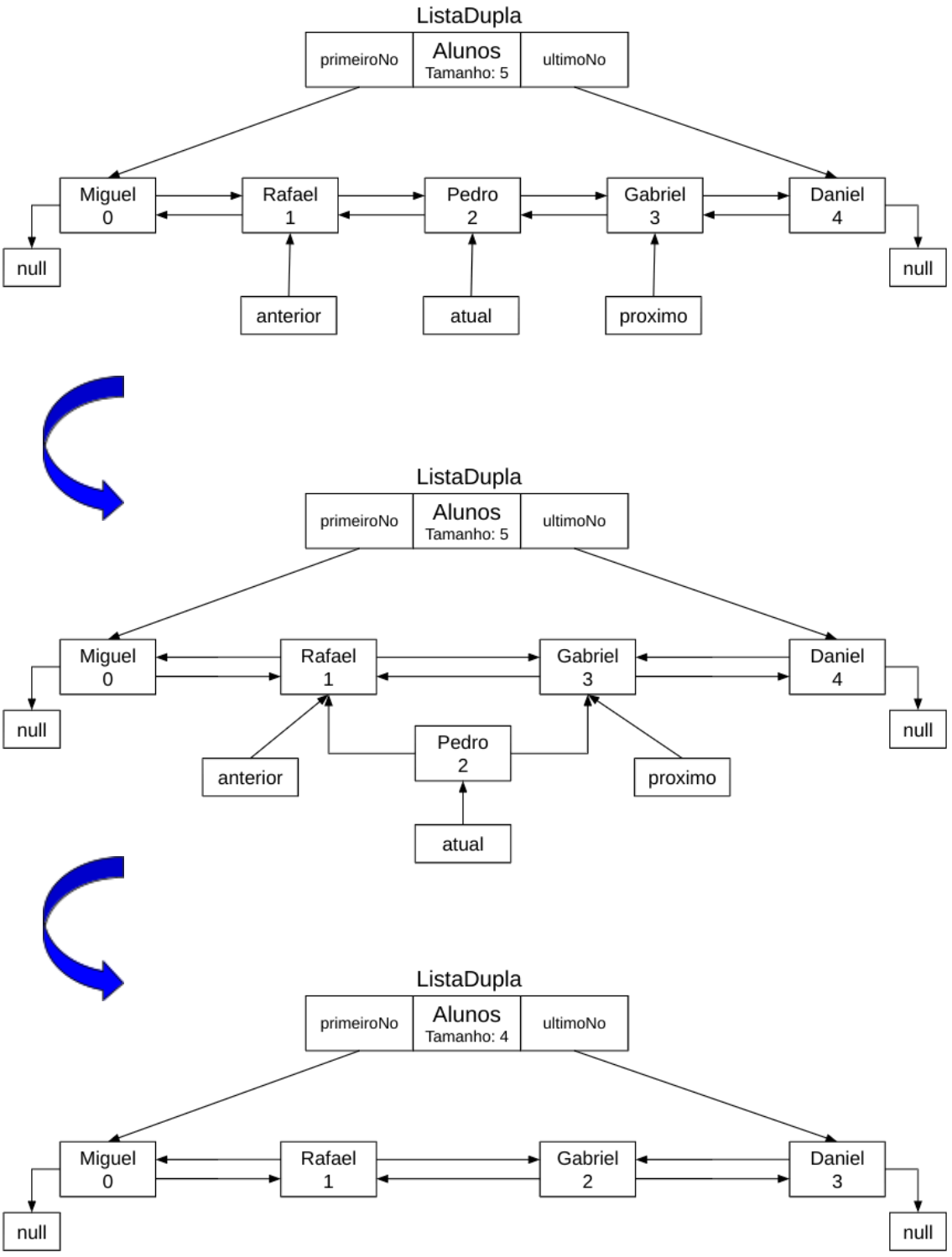



```
lista.imprimeLista();
```

Com cinco elementos na lista, a chamada ao método **imprimeLista** imprime os dados dos nós em ordem: "**Miguel**", "**Rafael**", "**Pedro**", "**Gabriel**" e "**Daniel**".

```
lista.removeMeio(2);
```

Nessa linha, o método **removeMeio()** é chamado para remover o nó na posição de índice 2 da lista. O método localiza o nó que está na posição de índice 2, que contém "**Pedro**", e o remove. O nó seguinte, que contém "**Gabriel**", ocupa o lugar de "**Pedro**", e todos os nós subsequentes são deslocados uma posição para trás. Essas operações demonstram como a lista pode ser manipulada para adicionar ou remover elementos em uma posição específica, com os índices sendo ajustados automaticamente após cada operação, como ilustrado nas figuras a seguir.

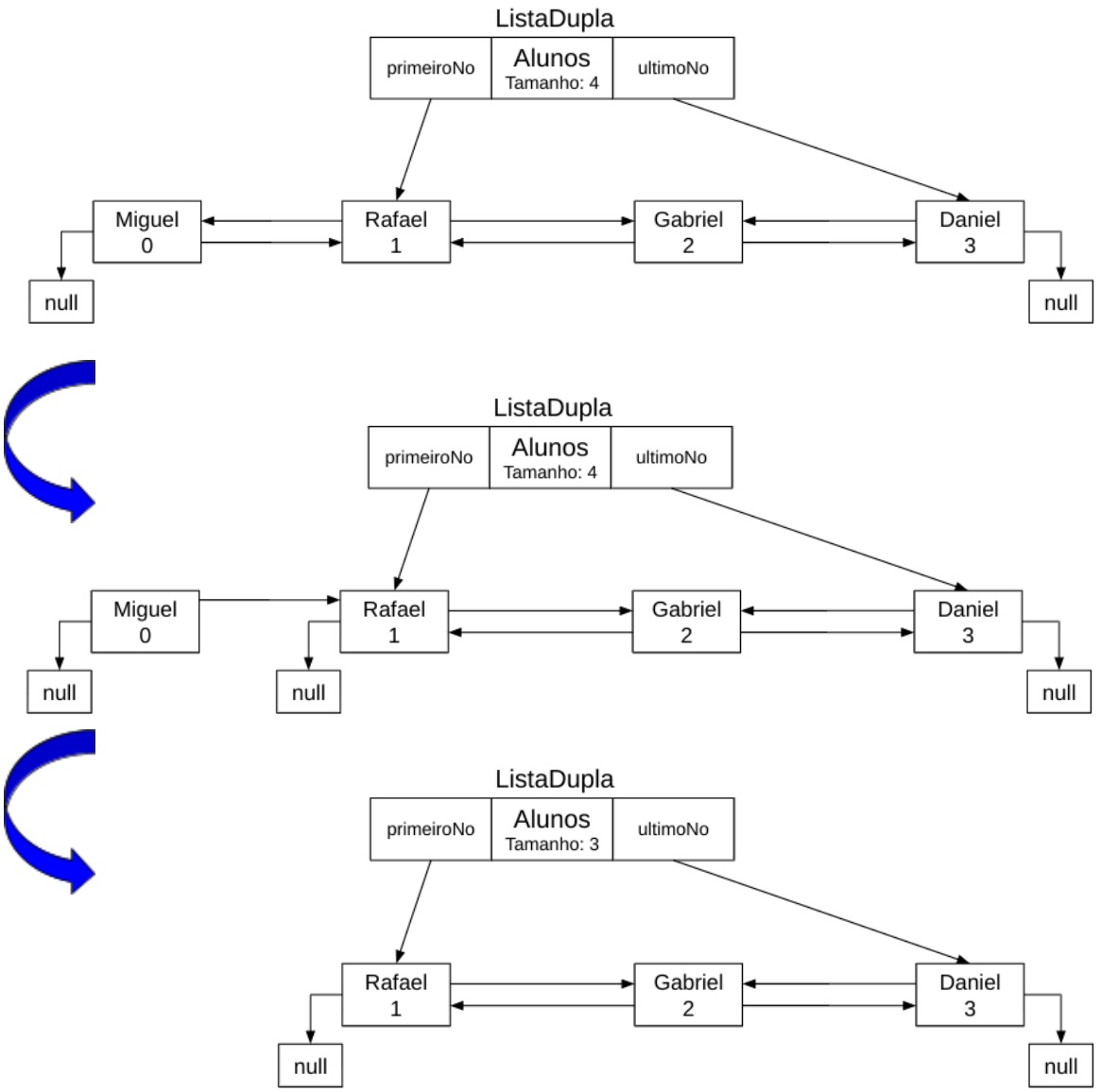



```
lista.imprimeLista();
```

Agora, com quatro elementos na lista, a chamada ao método **imprimeLista()** imprime os dados dos nós em ordem: "**Miguel**", "**Rafael**", "**Gabriel**" e "**Daniel**".

```
lista.removeInicio();
```

Nessa linha, o método **removeInicio()** é chamado para remover o primeiro nó da lista. Se a lista estiver vazia (o que não é o caso), uma mensagem indicando que a lista está vazia seria exibida. Como a lista contém elementos, o método remove o nó que contém "**Miguel**" e o próximo nó (que contém "**Rafael**") se torna o novo primeiro nó, como ilustrado nas figuras a seguir.

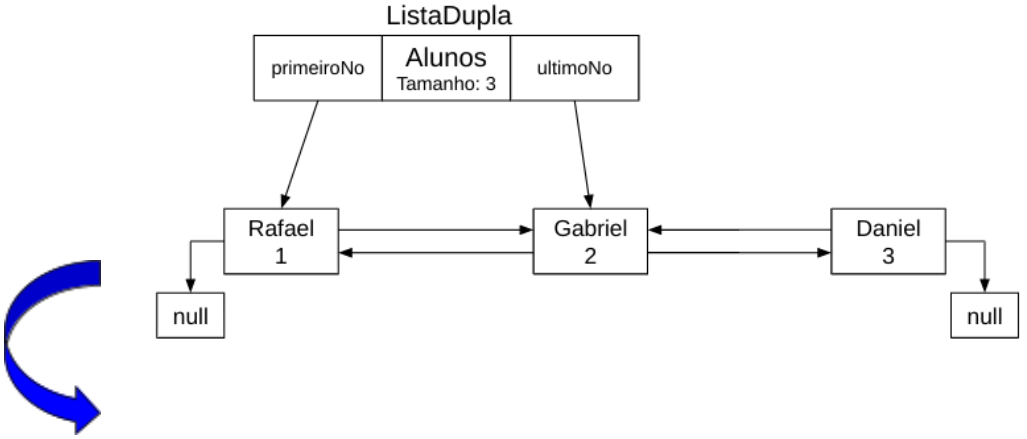


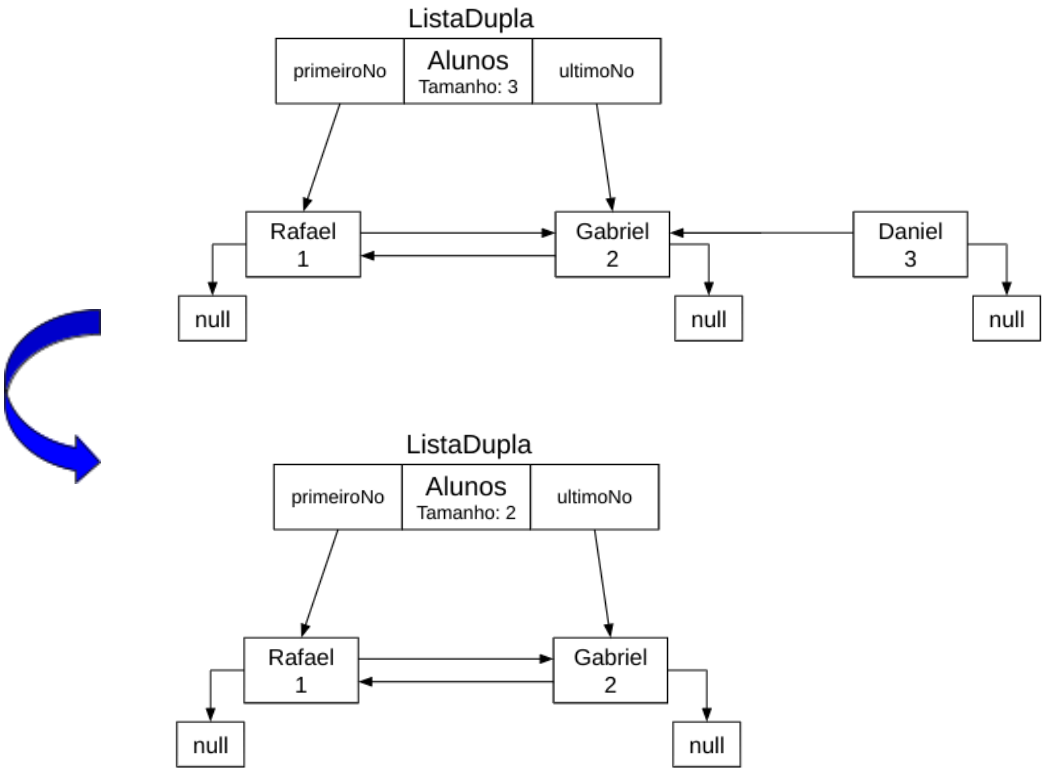

```
lista.imprimeLista();
```

Após a remoção de "**Miguel**", a lista agora contém três elementos: "**Rafael**", "**Gabriel**", e "**Daniel**". O método **imprimeLista** imprime os dados desses três nós.

```
lista.removeFinal();
```

Essa linha chama o método **removeFinal**, que remove o último nó da lista. Se a lista estiver vazia, uma mensagem de "Lista Vazia" é exibida. Como a lista tem elementos, o método remove o nó que contém "**Daniel**", e o penúltimo nó (que contém "**Gabriel**") se torna o novo último nó, como ilustrado nas figuras a seguir.






```
lista.imprimeLista();
```

Após a remoção de "**Daniel**", a lista contém dois elementos: "Rafael" e "Gabriel". O método **imprimeLista** imprime os dados desses dois nós.