Código 4 - Estruturas de dados genéricas personalizadas e pilha encadeada

Resumo: O código apresentado exemplifica o uso de conceitos fundamentais de Java, como generics, encapsulamento, orientação a objetos e manipulação de pilhas. A classe No é utilizada para criar uma estrutura de dados genérica que armazena dados e referências para o próximo nó na pilha. A classe Pilha implementa uma pilha genérica, com operações básicas como empilhar (push), desempilhar (pop) e visualizar o topo da pilha (peek), todas manipulando elementos através da estrutura de nós encadeados. A classe Principal demonstra o uso da pilha, incluindo a inserção, remoção e visualização de elementos. As principais referências para a elaboração deste material incluem o livro "Java: Como Programar" (Deitel) para os conceitos de orientação a objetos, encapsulamento e generics, além da documentação oficial da linguagem Java para detalhes técnicos específicos.

Será detalhado abaixo cada parte das classes ***No, Pilha*** e ***Principal***, explicando os conceitos usados nos códigos.

Classe “No”

```
class No<T>{
    private T dado;
    private No<T> nextNo;
```

Essa classe define um nó **genérico No<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, e **nextNo** é uma referência ao próximo nó na lista.

```
public No(T dado){
    this(dado, null);
}
```

Esse construtor cria um novo nó inicializando o **dado** com o valor fornecido e define **nextNo** como **null**, indicando que o nó não está vinculado a outro nó.

```
public No(T dado, No<T> no){
    this.dado = dado;
    this.nextNo = no;
}
```

Esse construtor permite criar um nó e ao mesmo tempo especificar a referência para o próximo nó (**nextNo**). Se **no** for **null**, o nó será o último da lista.

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

public T getDado(){
    return this.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 void setNextNo(No<T> nextNo){
    this.nextNo = nextNo;
}

public No<T> getNextNode(){
    return this.nextNo;
}
```

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

```
@Override
public String toString(){
    return "{ " + getDado() + " }";
}
```

O método **toString** foi sobrescrito para fornecer uma representação em string do objeto **No**. Ele retorna uma **string** que exibe o valor armazenado em **dado**.

Classe “Pilha”

```
public class Pilha<T>{
    private No<T> topo;
    private String nomePilha;
```

A classe **Pilha** é uma implementação genérica de uma pilha encadeada, onde **T** representa o tipo de dados que a pilha armazenará. Estas variáveis de instância armazenam o nó no **topo** da pilha (**topo**) e o nome da pilha (**nomePilha**). A variável topo é do tipo **No<T>**, o que permite que a pilha seja genérica e armazene qualquer tipo de dado.

```
public Pilha(){
    this("");
}
```

Este é o **construtor** padrão da classe **Pilha**, que chama o outro construtor passando uma string vazia como argumento. Isso inicializa a pilha com um nome (**nomePilha**) vazio e o **topo** como **null**.

```
public Pilha(String nomePilha){
    this.nomePilha = nomePilha;
    this.topo = null;
}
```

Este **construtor** permite a criação de uma pilha com um nome específico. Ele define o nome da pilha (**nomePilha**) e inicializa o **topo** como **null**, indicando que a pilha está inicialmente vazia.

```
public void push(T dado){
    No<T> novoNo = new No<T>(dado);
    novoNo.setNextNo(topo);
    topo = novoNo;
}
```

O método **push** insere um novo elemento no **topo** da pilha. Ele cria um novo nó (**novoNo**) contendo o dado fornecido e ajusta a referência ao próximo nó (**nextNo**) para apontar para o nó atualmente no topo. Depois, o topo é atualizado para o novo nó.

```
public T peek(){
    if(topo == null){
        System.out.println("Pilha Vazia");
        return null;
    }
    return topo.getDado();
}
```

O método **peek** permite visualizar o dado no **topo** da pilha sem removê-lo. Se a pilha estiver vazia (ou seja, **topo** é **null**), ele exibe uma mensagem e retorna null. Caso contrário, ele retorna o dado armazenado no **topo**.

```
public T pop(){
    if(topo == null){
        System.out.println("Pilha Vazia");
        return null;
    }
    T dadoTemp = topo.getDado();
    topo = topo.getNextNo();
    return dadoTemp;
}
```

O método **pop** remove e retorna o dado no **topo** da pilha. Se a pilha estiver vazia, ele exibe uma mensagem e retorna **null**. Caso contrário, ele salva o dado do **topo**, atualiza o **topo** para o próximo nó na pilha e retorna o dado removido.

```
public boolean isEmpty() {
    return topo == null;
}
```

O método **isEmpty** verifica se a pilha está vazia. Ele retorna **true** se **topo** for **null**, indicando que não há elementos na pilha, e **false** caso contrário.

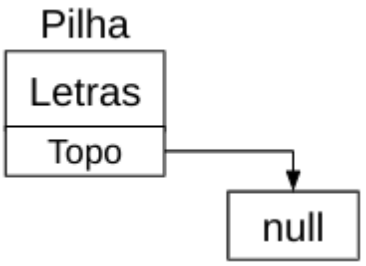
```
public void imprimePilha(){
    if(topo == null){
        System.out.println("Pilha Vazia");
    }else{
        System.out.printf("Dados da pilha %s:\n",nomePilha);
        No<T> aux = topo;
        while (aux != null) {
            System.out.printf("{ %s }\n", aux.toString());
            aux = aux.getNextNo();
        }
    }
}
```

O método **imprimePilha** exibe os elementos da pilha, começando do **topo** até o final (onde **nextNo** é null). Se a pilha estiver vazia, ele exibe "Pilha Vazia". Caso contrário, percorre os nós da pilha, exibindo os dados armazenados em cada nó.

Classe “Principal”

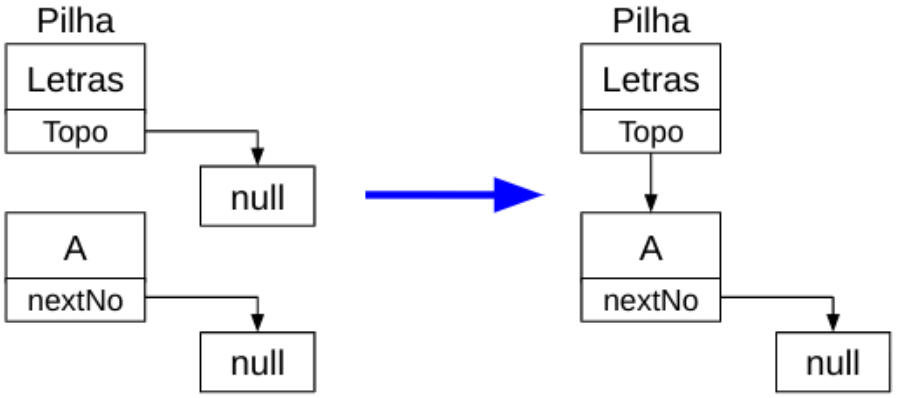
```
public class Principal{
    public static void main(String[] args){
        //Testando Pilha
        System.out.println("=== Testando a Pilha ===");
        Pilha<String> pilha = new Pilha<String>("Letras");
```

O método **main** é o ponto de entrada do programa. Ele cria uma nova pilha de **String** chamada "**Letras**", como ilustrado na Figura abaixo, e imprime uma mensagem indicando que a pilha está sendo testada.



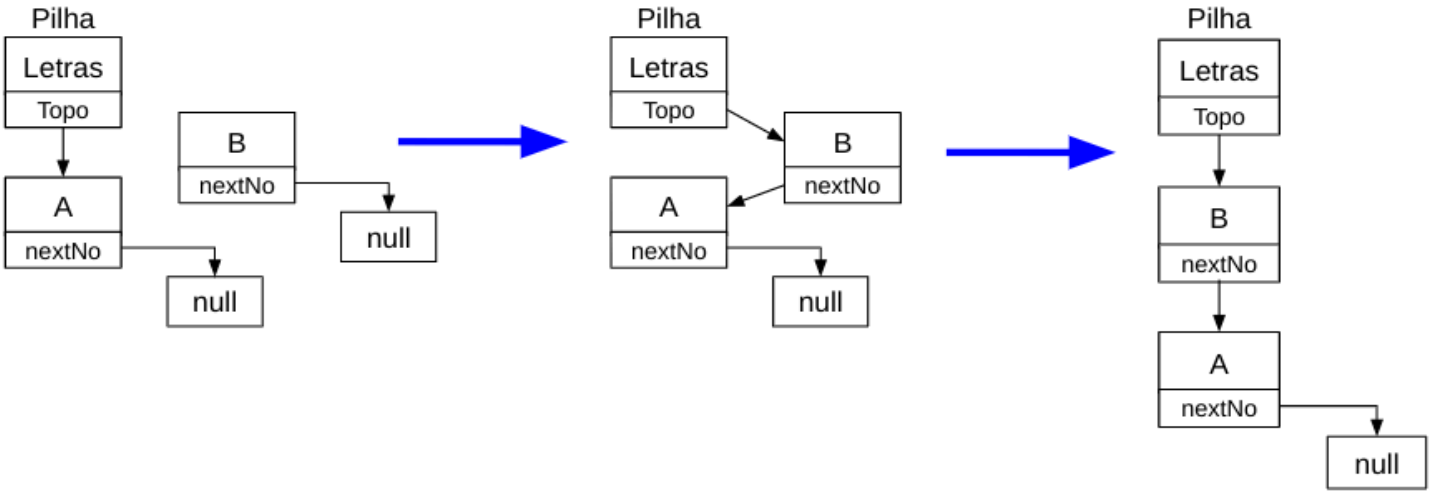

```
pilha.push("A");
```

O método push é chamado com o argumento "A". Esse método cria um novo nó contendo o dado "A" e o insere no **topo** da pilha. Se a pilha estava vazia, esse nó se torna o primeiro nó da pilha, como ilustrado na Figura abaixo.



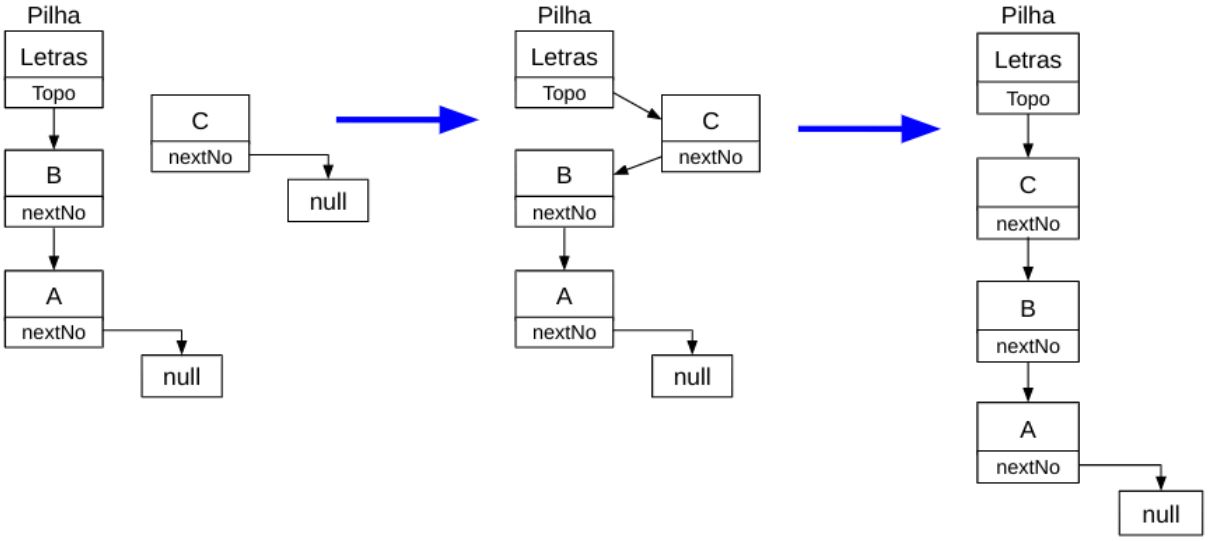

```
pilha.push("B");
```

Essa linha adiciona o dado "B" no topo da pilha, assim como foi feito com "A". Agora, "B" se torna o topo (primeiro) da pilha, e o nó que contém "A" se torna o último.



```
pilha.push("C");
```

Essa linha adiciona o dado "C" no topo da pilha, assim como foi feito com "B". Agora, "C" se torna o topo (primeiro) da pilha.




```
pilha.imprimePilha();
```

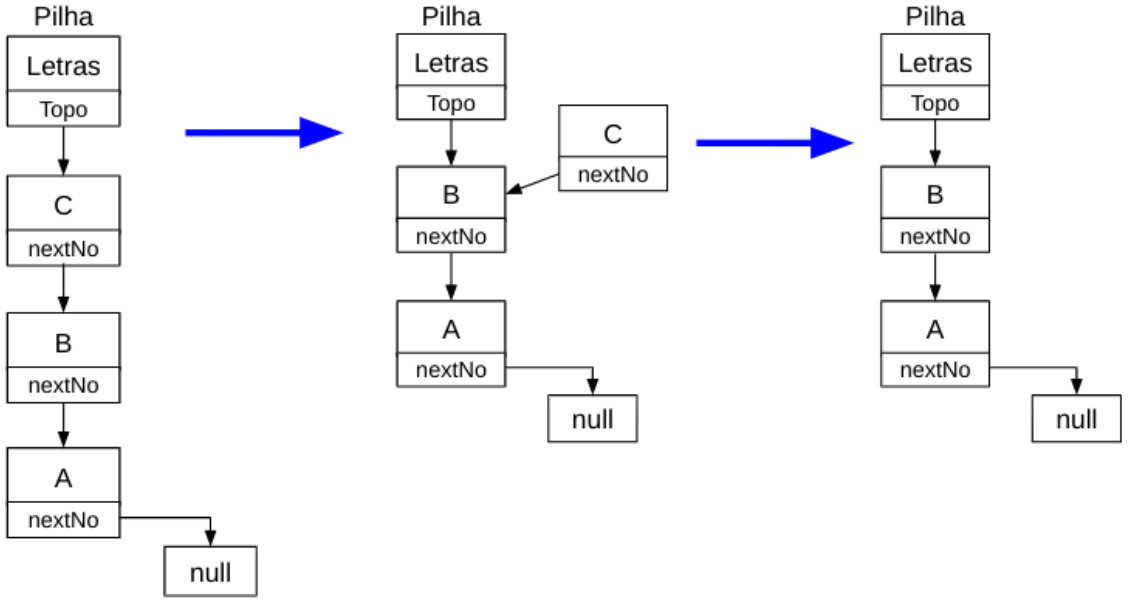
Após adicionar "**C**", a pilha contém três elementos. Ao chamar **imprimePilha**, o método imprime os dados dos nós em ordem: { C } { B } { A }

```
System.out.println("Topo da pilha: " + pilha.peek());
```

Esta linha chama o método **peek** para visualizar o dado no topo da pilha, que será "**C**", e imprime este valor.

```
System.out.println("Elemento removido: " + pilha.pop());
```

Nessa linha, o método **pop** é chamado para remover o nó **topo** da pilha. Se a pilha estiver vazia (o que não é o caso), uma mensagem indicando que a pilha está vazia seria exibida. Como a pilha contém elementos, o método pop remove o nó que contém "C" e o próximo nó (que contém "B") se torna o novo **topo** da pilha.




```
pilha.imprimePilha();
```

Após remover "**C**", a pilha contém dois elementos. Ao chamar **imprimePilha**, o método imprime os dados dos nós em ordem: { B } { A }