1. Atividade de Estruturas de Dados: Listas, Listas Duplamente Encadeadas, Filas e Pilhas

Objetivos:

- Compreender a implementação de listas simples e listas duplamente encadeadas.
- Implementar estruturas de dados de fila e pilha utilizando listas duplamente encadeadas.
- Praticar habilidades de programação em Java.

Parte 1: Implementação de uma Lista Simples

1. **Criação de uma Lista Simples:** Implemente uma classe chamada `Lista` que terá as seguintes funcionalidades:
    - Criação do Objeto elemento.
    - Criação dos atributos próximo e anterior.
    - A implementação dos Getters e Setters

```
public class Lista {
        private Object elemento;
    private Lista proximo;
    private Lista anterior; //Inserção do Anterior

    public Lista(Lista proximo, Object elemento) {
        this.proximo = proximo;
        this.elemento = elemento;       }

    public void setProximo(Lista proximo) {
                this.proximo = proximo; }

    public Lista getProximo() {
                return proximo; }

    public Object getElemento() {
                return elemento;}

    public Lista(Object elemento) {
        this(null, elemento);   }

    public Lista getAnterior() {
                return anterior;}

        public void setAnterior(Lista anterior) {
                this.anterior = anterior;       }
}
```

Parte 2: Implementação de uma Lista Duplamente Encadeada

2. **Criação de uma Lista Duplamente Encadeada:** Implemente uma classe chamada `ListaDuplamenteEncadeada` com as seguintes funcionalidades:
    - Adicionar um elemento no final da lista.
    - Remover um elemento da lista.
    - Buscar um elemento na lista.
    - Exibir todos os elementos da lista.

```
public class ListaDuplamenteEncadeada {
```

```
    private Lista primeira;
    private int totalDeElementos;
    private Lista ultima;

    public void adicionaNoComeco(Object elemento) {
          if(this.totalDeElementos == 0) {
                Lista nova = new Lista(elemento);
                this.primeira = nova;
                this.ultima = nova;}
           else {
                Lista nova = new Lista(this.primeira, elemento);
                this.primeira.setAnterior(nova);
                this.primeira = nova;  }
            this.totalDeElementos++;}

   public void adiciona(Object elemento) {
        if(this.totalDeElementos == 0) {
            adicionaNoComeco(elemento);
        } else {
            Lista nova = new Lista(elemento);
            this.ultima.setProximo(nova);
            nova.setAnterior(this.ultima);
            this.ultima = nova;
            this.totalDeElementos++;}    }

    public void adiciona(int posicao, Object elemento) {
        if(posicao == 0) {
            adicionaNoComeco(elemento);
        } else if (posicao == this.totalDeElementos) {
            this.adiciona(elemento);
        } else {
            Lista anterior = busca(posicao - 1);
            Lista proxima = anterior.getProximo();
            Lista nova = new Lista(anterior.getProximo(), elemento);
            nova.setAnterior(anterior);
            anterior.setProximo(nova);
            proxima.setAnterior(nova);
            this.totalDeElementos++;}    }


    private boolean posicaoOcupada(int posicao) {
        return posicao >= 0 && posicao < this.totalDeElementos;
    }

    private Lista busca(int posicao) {

        if(!posicaoOcupada(posicao)) {
            throw new IllegalArgumentException("posicao inexistente");
        }

        Lista atual = primeira;

        for(int i = 0; i < posicao; i++) {
            atual = atual.getProximo();
        }
        return atual;}

    public void removeDoComeco() {
        if(this.totalDeElementos == 0) {
            throw new IllegalArgumentException("lista vazia"); }

        this.primeira = this.primeira.getProximo();
        this.totalDeElementos--;
        if(this.totalDeElementos == 0) {
            this.ultima = null; }  }

    public void removeDoFim() {
        if(this.totalDeElementos == 1) {
            this.removeDoComeco();
        } else {
            Lista penultima = this.ultima.getAnterior();
            penultima.setProximo(null);
            this.ultima = penultima;
```

```
            this.totalDeElementos--; }  }

    public void remove(int posicao) {
        if(posicao == 0) {
            this.removeDoComeco();
        } else if (posicao == this.totalDeElementos - 1) {
            this.removeDoFim();
        } else {
            Lista anterior = this.busca(posicao - 1);
            Lista atual = anterior.getProximo();
            Lista proxima = atual.getProximo();
            anterior.setProximo(proxima);
            proxima.setAnterior(anterior);
            this.totalDeElementos--;  }  }

    public int tamanho() {
        return this.totalDeElementos;
    }


    public boolean Localiza(Object elemento) {
        Lista atual = this.primeira;

        while(atual != null) {
            if(atual.getElemento().equals(elemento)) {
                return true;
            }
            atual = atual.getProximo();
        }
        return false;
    }

    @Override

    public String toString () {

        if(this.totalDeElementos == 0) {
            return "()"; }

        Lista atual = primeira;
        StringBuilder builder = new StringBuilder("(");
        for(int i = 0; i < totalDeElementos; i++) {
            builder.append(atual.getElemento());
            builder.append(",");
            atual = atual.getProximo();
        }
        builder.append(")");
        return builder.toString(); }}
```

Parte 3: Implementação de Fila e Pilha Utilizando Lista Duplamente Encadeada

3. **Implementação de uma Fila:** Utilizando a classe `ListaDuplamenteEncadeada`, implemente uma fila (`Queue`) com as seguintes operações:
    - Enfileirar (inserir no final).
    - Desenfileirar (remover do início).
    - Verificar se a fila está vazia.
    - Obter o tamanho da fila.

```
public class Fila {


private ListaDuplamenteEncadeada elemento;

        public Fila() {
                this.elemento=new ListaDuplamenteEncadeada();
```

```
        }

        public boolean vazio() {
                return this.elemento.tamanho()==0 ;
                        }

        public void enfileirar (Object elemento) {
                this.elemento.adiciona(elemento);
        }

        public Object desemfeleirar() {
                if (this.elemento.tamanho()==0) {
                        return null;
                }
                Object resultado= this.elemento.Localiza(this.elemento.tamanho()-1);
                this.elemento.removeDoComeco();
                return resultado;

        }

        @Override
        public String toString() {
                return "Fila[elemento=" + elemento + "]";
        }}
```

3.1 **Implementação** classe Principal - Fila

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

        Fila f = new Fila();

        System.out.println(f);
        f.enfileirar("ANDERSON");
        System.out.println(f);
        f.enfileirar("LUAN");
        System.out.println(f);

        f.enfileirar("LORENA");
        System.out.println(f);

        f.desemfeleirar();
        System.out.println(f);

}
```

4. **Implementação de uma Pilha:** Utilizando a classe ListaDuplamenteEncadeada, implemente uma pilha (Stack) com as seguintes operações:

- Empilhar (inserir no final).
- Desempilhar (remover do final).
- Verificar se a pilha está vazia.
- Obter o tamanho da pilha.

```
public class Pilha {

        private ListaDuplamenteEncadeada elemento;


        public Pilha() {
                this.elemento=new ListaDuplamenteEncadeada();
        }

        public boolean vazio() {
                return this.elemento.tamanho()==0 ;
                        }
```

```
	public void empilhar (Object elemento) {
		this.elemento.adiciona(elemento);
	}

	public Object desempilhar() {
		if (this.elemento.tamanho()==0) {
			return null;
		}
		Object resultado= this.elemento.Localiza(this.elemento.tamanho()-1);
		this.elemento.removeDoFim();
		return resultado;

	}

	@Override
	public String toString() {
		return "Pilha [elemento=" + elemento + "]";
	}
```

4.1 **Implementação** classe Principal - Pilha

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

		Pilha p = new Pilha();

	System.out.println(p);
	p.empilhar("ANDERSON");
	System.out.println(p);
	p.empilhar("LUAN");
	System.out.println(p);

	p.empilhar("LORENA");
	System.out.println(p);

	p.desempilhar();
	System.out.println(p);



	}}
```

Atividade Prática:

Implemente as classes acima e crie um programa principal para testar cada uma das operações das listas, fila e pilha. Verifique se todos os métodos estão funcionando conforme o esperado, realizando operações de adição, remoção, busca e exibição de elementos.