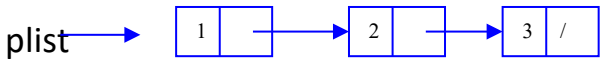
FAESA CENTRO UNIVERSITÁRIO
Unidade de Computação e Sistemas
Disciplina Estrutura de Dados
Profª Me. Renata Cristina Laranja Leite

Atividade 02 - Lista Linear Simplesmente Encadeada

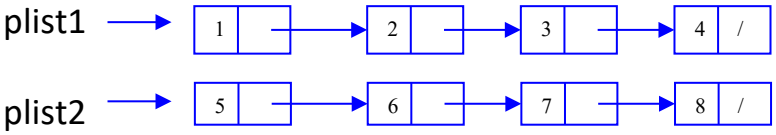
1) Suponha que temos uma lista encadeada conforme é exibido a seguir. Qual problema o uso da seguinte instrução pode causar?
Dado: plist = plist.getProx();



Resposta: remove o primeiro nó

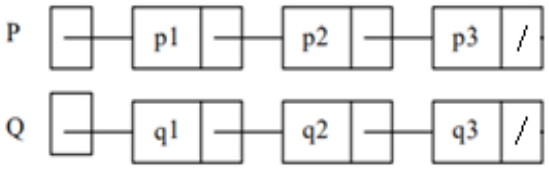
2) Suponha que exista uma lista encadeada conforme é mostrado a seguir. O que acontece se executarmos a seguinte instrução para estas duas listas?

plist1 = plist2;



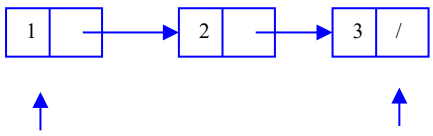
Resposta: perde todos os nós da plist1

3) Partindo da situação mostrada no desenho abaixo, e sendo P e Q duas listas encadeadas dinâmicas, explique (com desenhos) o que acontece nas atribuições seguintes:
(obs: cada item parte da situação inicial mostrada abaixo)



a) P.setProx(Q.getProx());
b) P = Q;
c) P = NULL;
d) P = P.getProx();
e) P.setInfo(P.getProx().getInfo());
Resposta: feito em sala de aula/laboratório

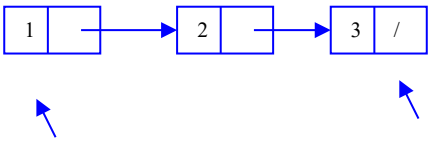
4) Escreva o enunciado das seguintes operações:
a) __inserir após o ultimo_____



primeiro ultimo

```
No aux = new No (elem);
ultimo.setProx(aux);
ultimo = aux;
```

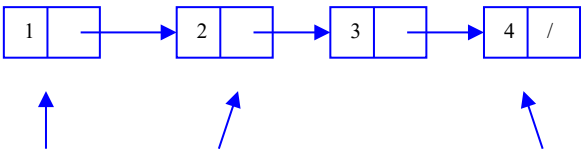
b) ____inserir após o primeiro_____



primeiro ultimo

```
No aux = new No (elem);
aux.setProx(primeiro.getProx());
primeiro.setProx(aux);
```

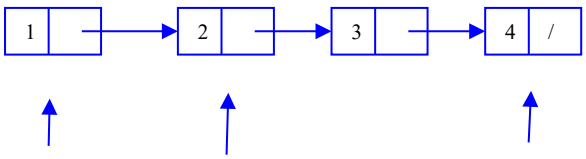
c) _____remover o segundo nó_____



primeiro atual ultimo

```
atual = atual.getProx();
primeiro.setProx(atual);
```

d) ____remover o ultimo nó_____



primeiro atual ultimo

```
ultimo = atual.getProx();
ultimo.setProx(null);
```

5) Assuma que uma **Lista Linear Simplesmente Encadeada** é construída de elementos com dois campos, *info* e *prox*, sendo que o campo *prox* de cada elemento é usado para apontar para o próximo na lista. Qual das seguintes é uma implementação correta em Java da operação *insira p depois de q*, onde *q* aponta para um elemento da lista e *p* para um elemento a ser inserido?

(A)
```
q.setProx(p.getProx());
p.setProx(q);
```
(B)
```
q.setProx(p.getProx());
p.setProx(q.getProx());
```
(C)
```
p.setProx(q.getProx());
q.setProx( p);
```
(D)
```
p.setProx(q);
q.setProx(p.getProx());
```
(E)
```
q.setProx(p);
p.setProx(q.getProx());
```

6) (FCC - 2009 - TJ-PI - Analista Judiciário - Análise de Sistemas - Desenvolvimento) Uma lista ligada é uma estrutura que corresponde a uma sequência lógica de entradas ou nós. Cada nó armazena a localização do próximo elemento na sequência, ou seja, de seu nó sucessor. Nessa estrutura,
(A) para estabelecer a ligação entre um nó já pertencente a uma lista e um novo nó, basta fazer com que o novo nó referencie no, campo *next*, o nó que anteriormente era referenciado pelo nó original, desde que esse campo não tenha o valor nulo.
(B) a existência de um ponteiro apontando para o 1º elemento e outro para o fim da lista permite que a inserção ou deleção de dados de um nó que esteja no meio da lista seja rapidamente executada.
(C) enquanto a entrada que determina o topo da lista é mantida em um nó descritor dessa lista, a entrada que marca o fim da lista é mantida fora do descritor.
(D) o armazenamento de uma lista requer uma área contígua de memória para permitir a otimização no processamento de criação e remoção de nós da lista.
(E) o armazenamento de uma lista não requer uma área contígua de memória. Como listas são estruturas dinâmicas, normalmente são definidos procedimentos que permitem criar e remover nós na memória.

7) Dada duas **Listas Simplesmente Encadeadas** (L1 e L2), escreva um método para concatenar as duas listas. A L2 deverá ficar vazia.

```
public void concatenarListas(ListaSimples lista2) {
          if (this.eVazia() && !lista2.eVazia()) {
```

```
                this.prim = lista2.prim;
                this.ult = lista2.ult;
                this.quantNos = lista2.quantNos;
          }else {
                if (!lista2.eVazia()) {
                      this.ult.setProx(lista2.prim);
                      this.ult = lista2.ult;
                      this.quantNos += lista2.quantNos;
                }
          }
          lista2.prim = null;
          lista2.ult = null;
          lista2.quantNos = 0;
     }
```

8) Dada duas **Listas Simplesmente Encadeadas** (L1 com vários nós e L2 está vazia), escreva um método para partir aproximadamente a L1 ao meio e colocar a metade dos seus dados na lista L2.

```
public void partirLista(ListaSimples lista2) {
          if (!this.eVazia()) {
                No atual = this.prim;
                for (int i=1; i<this.quantNos/2;i++) {
                      atual = atual.getProx();
                }
                lista2.prim = atual.getProx();
                lista2.ult = this.ult;
                this.ult = atual;
                this.ult.setProx(null);
                lista2.quantNos = this.quantNos - this.quantNos/2;
                this.quantNos = this.quantNos/2;
          }
     }
```

9) Seja a **Lista Simplesmente Encadeada** L, escreva um método para calcular a média dos valores da lista.

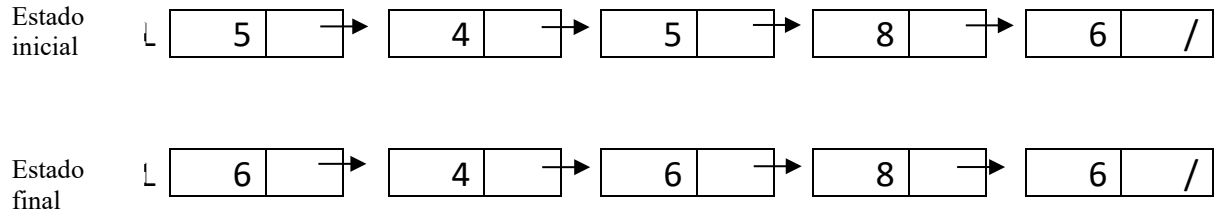
```
public double calcularMedia() {
          double soma=0;
          No atual = this.prim;
          for (int i=1 ; i<= this.quantNos; i++){
                soma += atual.getInfo().getChave();
                atual = atual.getProx();
          }
          return soma/this.quantNos;
     }
```

10) Implemente uma função que utiliza duas **Listas Lineares Simplesmente Encadeadas** de inteiros, L1 e L2. Esta função deverá retornar VERDADEIRO se as listas forem idênticas (os mesmos elementos na mesma ordem), ou FALSO se pelo menos um elemento for diferente da ordem. Admita que as listas não estão vazias.

```
public boolean identificarIguais(ListaSimples lista2) {
          if (this.quantNos != lista2.quantNos) {
               return false;
          }else {
               No atualL1 = this.prim;
               No atualL2 = lista2.prim;
               for (int i=1; i <= this.quantNos;i++) {
                         if (atualL1.getInfo().getChave()!=
                         atualL2.getInfo().getChave()) {
                              return false;
                         }
                         atualL1 = atualL1.getProx();
                         atualL2 = atualL2.getProx();
               }
               return true;
          }
}
```

11) Dada uma **Lista Linear Simplesmente Encadeada**, implemente um método que receba como parâmetro dois valores (inteiros), e retorne o número de trocas efetuadas, resultante da troca de todas as ocorrências do primeiro valor(n1) pelo segundo(n2) dentro da lista. A lista não está vazia. Não precisa mudar os ponteiros.

Exemplo:
Parâmetros: n1=5 e n2=6



Retornar o número de trocas = 2

```
//n1 é o número para ser procurado
//n2 é o novo valor
public int efetuarTrocas(int n1, int n2){
     No atual = this.prim;
     int soma = 0;
     while (atual!=null){
          if (atual.getInfo().getChave()==n1){
               atual.getInfo().setChave(n2);
               soma++;
          }
          atual = atual.getProx();
     }
     return soma;
}
```

12) Escreva um método que recebe como parâmetro um valor inteiro. Procure este valor na lista e se o valor não existir, ele deve ser inserido no final da lista; caso contrário, retorne a quantidade de nós que o valor aparece na lista. A lista é **Linear Simplesmente Encadeada com descritor**, e o nó da lista possui um campo do tipo inteiro. Não se sabe o estado da lista.

```
//atividade 02 - questão 12
public int procurarInserirNo(int valor){
	if (this.eVazia()){
		//se a lista está vazia, inserir o valor na lista
		this.inserirUltimo(new Item(valor));
		return 1;
	}else{
		No atual = this.prim;
		int cont=0;
		//procurar quantas vezes o valor foi encontrado na lista
		while (atual!= null){
			if (atual.getInfo().getChave()==valor){
				cont++;
			}
			atual = atual.getProx();
		}
		//se não encontrar, inserir no final da lista
		if (cont==0){
			this.inserirUltimo(new Item(valor));
			return 1;
		}else{
			return cont;   //total de vezes encontrado
		}
	}
}
```

13) Implemente uma função que utiliza duas **Listas Lineares Simplesmente Encadeadas** de inteiros, L1(objeto do método) e L2(passada por parâmetro). Esta função deverá retornar TRUE se a lista L1 está contida na lista L2 (independente da ordem), ou FALSE se pelo menos um elemento da lista L1 não for encontrado na L2. Você não sabe como estão as listas (vazias ou não vazias, se uma tem mais ou menos elementos que a outra).

```
public boolean estarContida(ListaSimples lista2) {
	if (this.eVazia() || lista2.eVazia()) {
		return false;
	}else {
		if (this.quantNos > lista2.quantNos) {
			return false;
		}else {
			No atualL1 = this.prim;
			No atualL2;
			boolean achou;
			while (atualL1 != null) {
```

```
                //ou for(int i=1;i<=this.quantNos;i++){
                    atualL2 = lista2.prim;
                    achou = false;
                    //percorre várias vezes a lista2 procurando
                    //cada valor da lista1
                    while (atualL2 != null && !achou) {
                        if (atualL1.getInfo().getChave()==
                        atualL2.getInfo().getChave()) {
                            achou = true;
                        }
                        atualL2 = atualL2.getProx();
                    }
                    if (!achou) { //se não encontrou o valor
                        return false;
                    }
                    //caminha para o próximo na lista1
                    atualL1 = atualL1.getProx();
                }
                return true;
            }
        }
}
```