Aula 07/04 - Lista duplamente encadeada

Listas duplamente encadeadas são um tipo de estrutura de dados onde um elemento dentro da lista contém uma referência ao elemento sucessor e anterior.

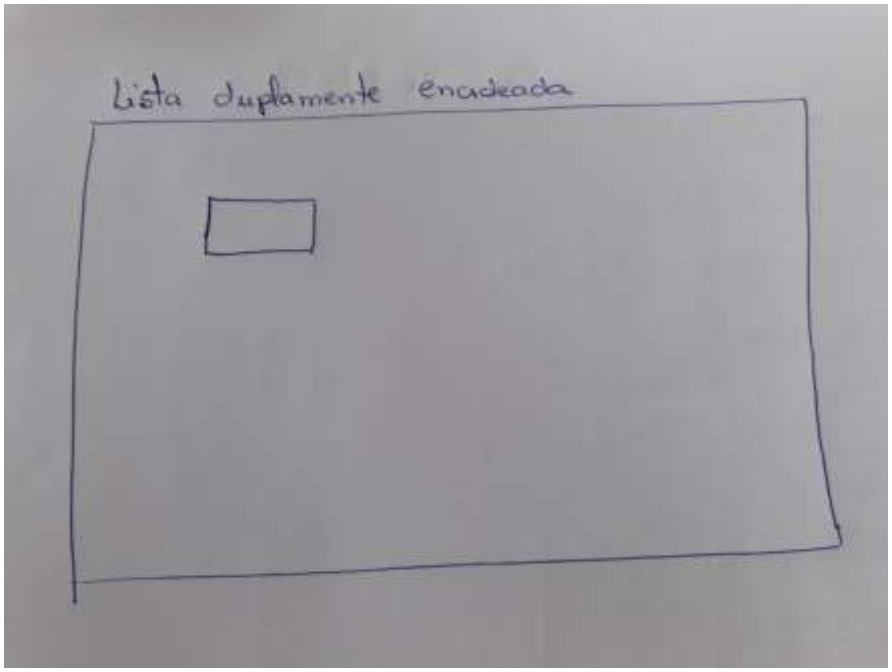
Representação de uma lista dinâmica duplamente encadeada

Numa lista cada elemento, ou nó, é composto normalmente por uma variável que guarda a informação e ponteiros (referências a endereços de memória) que permitem a ligação entre os vários nós desta lista.

Esses ponteiros em uma lista simplesmente encadeada apontam para o próximo nó da lista, até que o último aponte para uma região de memória nula. Assim, é possível saber onde a lista termina.

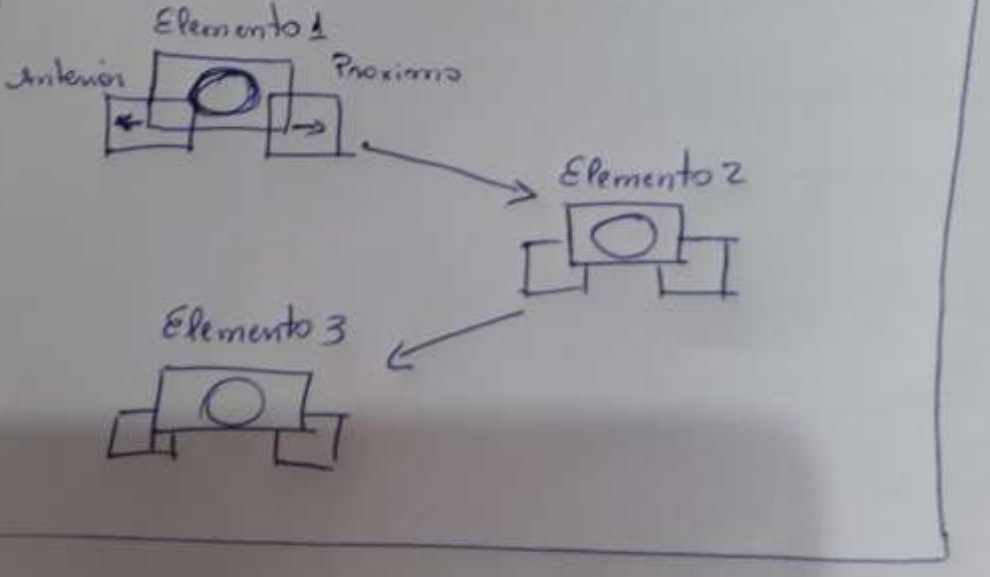
Já a lista duplamente ligada além de saber o próximo nó, cada elemento também conhece o nó anterior a ele na lista. Isso facilita e melhora o desempenho de algumas operações executadas sobre a lista como por exemplo remover um elemento, mostrar os elementos na ordem inversa (do último para o primeiro elemento), etc.

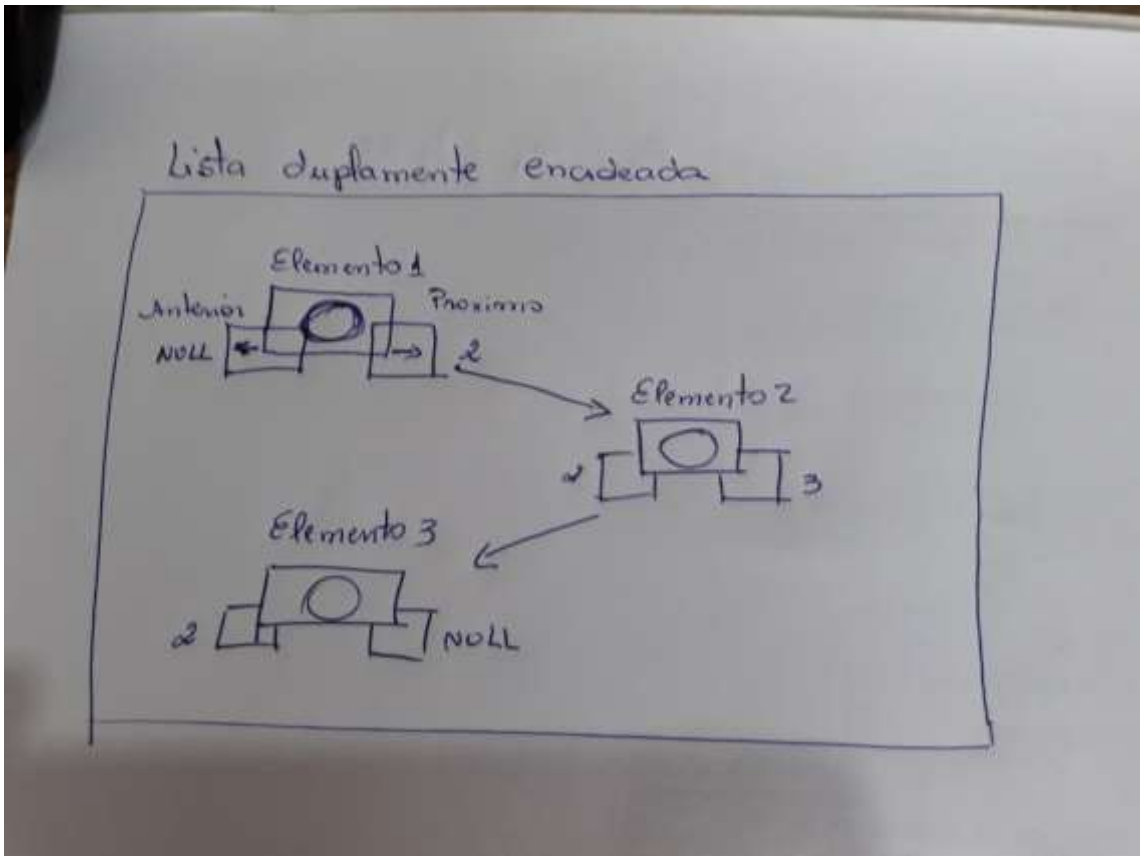
Isso também facilita na alocação dinâmica de objetos, no exemplo usarei o elemento como inteiro só para a aula não ficar extensa, mas você pode substituir o inteiro por um objeto, mas ao substitui-lo, não esqueça de instancia-lo.

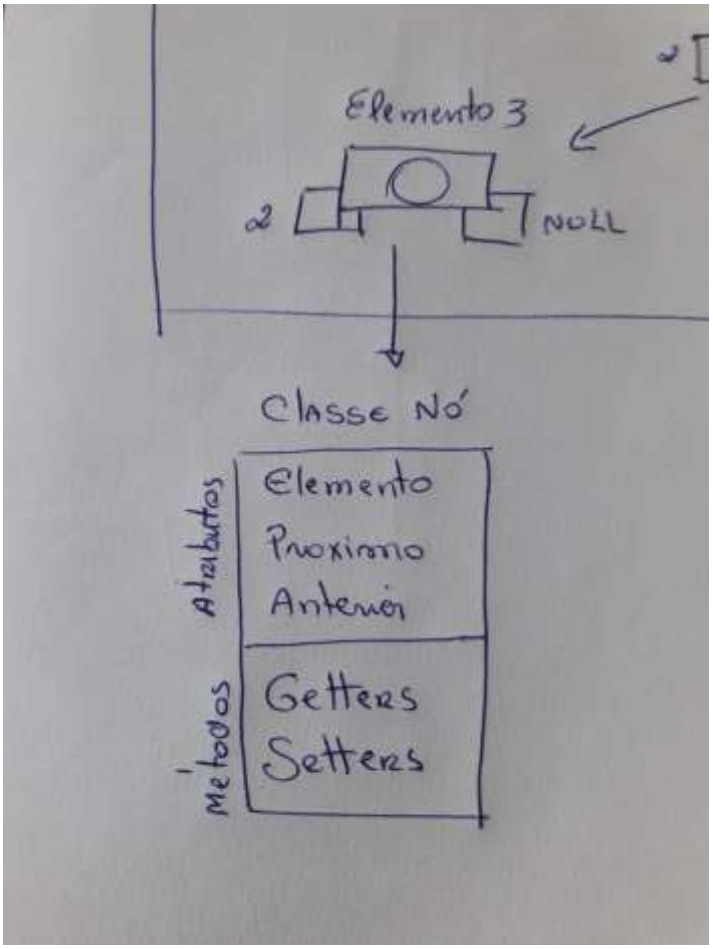


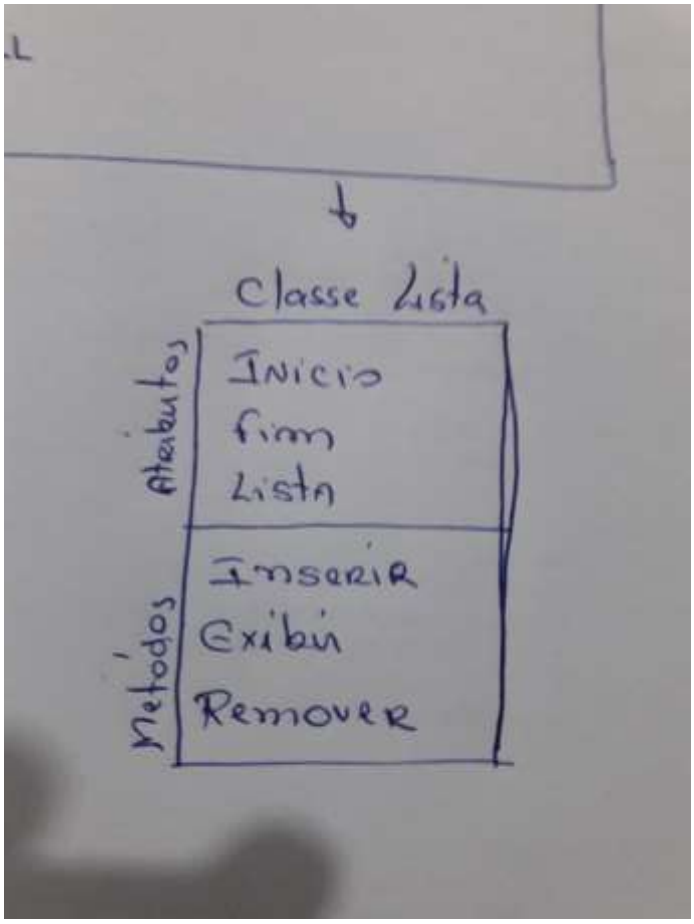
Lista duplamente encadeada

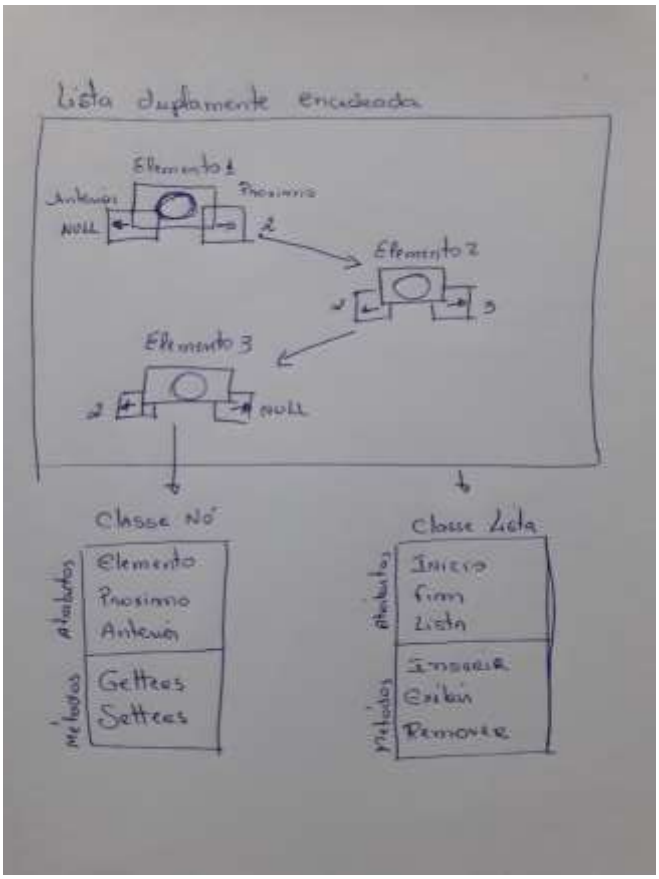
Lista duplamente encadeada




Lista duplamente encadeada
Elemento 1
Anterior
Proximo
NULL
2
Elemento 2
2
3
Elemento 3
2
NULL


2
Elemento 3
2
NULL
Classe Nó
Atributos
Elemento
Proximo
Anterior
Metodos
Getters
Setters


Classe Lista
Atributos
Inicio
fim
Lista
Métodos
Inserir
Exibir
Remover


Lista duplamente encadeada
Elemento 1
Anterior
NULL
Proximo
2
Elemento 2
Elemento 3
NULL
Classe Nó
Atributos
Elemento
Proximo
Anterior
Métodos
Getters
Setters
Classe Lista
Atributos
Inicio
fim
Lista
Métodos
Inserir
Exibir
Remover

Vamos ao código:

Primeiro implementaremos a clase nó:

```
package listduplenc;

public class No
{
/* as variaveis ant e prox são do tipo No pois
irão receber o endereço dos outros objetos
do tipo No */
private No ant, prox;
private int info;
//construtor
public No(No ant,No prox,int info)
{
this.ant=p_ant;
this.prox=p_prox;
this.info=info;
}
//getter e setter
public No getAnt() { return ant; }
public No getProx() { return prox; }
public void setAnt(No p_ant) { ant=p_ant; }
public void setProx(No p_prox) { prox=p_prox; }
public int getInfo() { return info; }
public void setInfo(int elemento) { info=elemento; }
}
```

E esta criada nossa classe nó… mas ai você vem e me pergunta… sóóóóóó issoooooo… a grande jogada da classe nó não é a quantidade de codigo mas sim as variaveis ant e prox que guardam o endereço de outros objetos do tipo nó

Agora vamos a classe que irá utiliza os nós:

```
package listduplenc;

public class Lista {
/* Aqui criamos 2 variaveis do tipo No
para não perdermos a nossa lista
*imagine a lista como um varal… e
os nós como roupas… o ant e prox
do nó será nossos pregadores e os
bambus que seguram o varal são as
```

```
variaveis inicio e fim… se essas
variaveis perderem seu valor é como
se soltasse o fio do varal…
continuaremos a ter 2 bambus mas não
teremos mais o varal… */
private No inicio;
private No fim;

//construtor
public Lista()
{
inicializa();
//chama inicializa para economizar codigo
}

public void inicializa()
{
inicio=null;
fim=null;
//inicializa faz a lista ficar vazia
}

/* ai vem a famoza pergunta…se eu tenho uma
lista cheia e pego meu inicio e fim e seto
eles para que a lista esteja vazia.. e meus
nós que ja instanciei??? Bom quando um objeto
no java (no caso o nó) perde sua referencia
(no caso o inicio e fim) o próprio java retira
da memória o que está sobrando quem faz isso é
o Garbage Collector… ou como preferir…
coletor de lixo */

//aqui fazemos uma lista para inserir somente no final
public void insereNoFim(int info){
//declaramos e instanciamos a variavel caixa
//do tipo nó. seu anterior vai ser o fim
//pois estamos inserindo depois do fim
No caixa = new No(inicio,null,info);
if (inicio==null)//se lista estiver vazia
inicio = fim = caixa;
else{
//seta prox do No do fim para receber caixa
fim.setProx(caixa);
fim = caixa;
}
}
```

```
//aqui fazemos uma lista para inserir somente no comeco
public void insereNoComeco(int info){
//declaramos e instanciamos a variavel caixa
//do tipo nó. seu proximo vai ser o incio
//pois estamos inserindo antes do inicio
No caixa = new No(null,inicio,info);
if (inicio==null)//se lista estiver vazia
inicio = fim = caixa;
else{
//seta ant do No do inicio para receber caixa
inicio.setAnt(caixa);
inicio = caixa;
}
}

public void exibeLista()
{
No aux;
aux=inicio;
while (aux!=null)
{
System.out.println(aux.getInfo());
aux=aux.getProx();
}
}

public No Busca_Exaustiva(int elemento)
{
No p=inicio;
while ((p!=null) && (p.getInfo()!=elemento))
{
p=p.getProx();
}
if ((p!=null) && (p.getInfo()==elemento))
return p;
else return null;
}

//no remove temos 5 casos a considerar
public void removeLista(int elemento)
{
No pos;
pos=Busca_Exaustiva(elemento);
if (pos!=null)//1- se existe o No a ser deletado
{
if (inicio!=fim)//2- se só existe um Nó na lista
```

```
{
if (pos==inicio)//3- se o Nó esta no começo
{
inicio=pos.getProx();
pos.getProx().setAnt(null);
}
else
if (pos==fim)//4- se o Nó esta no fim
{
fim=pos.getAnt();
pos.getAnt().setProx(null);
}
else//5- se o no esta no meio
{
pos.getAnt().setProx(pos.getProx());
pos.getProx().setAnt(pos.getAnt());
}
pos.setAnt(null);
pos.setProx(null);
}
else
{
inicio=null;
fim=null;
}
}
else
System.out.println('Elemento nao encontrado');
}
}
```

Agora vamos fazer uma classe principal para nossa aplicação:

```
package listduplenc;

public class Aplicacao {
Lista lista;
public Aplicacao()
{
lista=new Lista();
}

public void executa()
{
for (int i=1 ; i&lt;=5 ; i++)
lista.insereLista_noFinal(i);
```

```
System.out.println(‘Lista inserindo no final’);
lista.exibeLista();
System.out.println(‘\n\n’);

System.out.println(‘Lista inserindo no inicio (Tipo Pilha)’);
lista.insereLista_tipoPilha(50);
lista.insereLista_tipoPilha(40);
lista.insereLista_tipoPilha(30);
lista.exibeLista();

//Removendo alguns elementos (30,1,5)
lista.removeLista(30);
lista.removeLista(1);
lista.removeLista(5);
//Lista apÃ³s remover alguns dos elementos (30,1,5)
System.out.println(‘\n\nDepois de Remover os elementos (30,1,5)’);
lista.exibeLista();
}

public static void main(String args[])
{
Aplicacao a = new Aplicacao();
a.executa();
}
}”
```