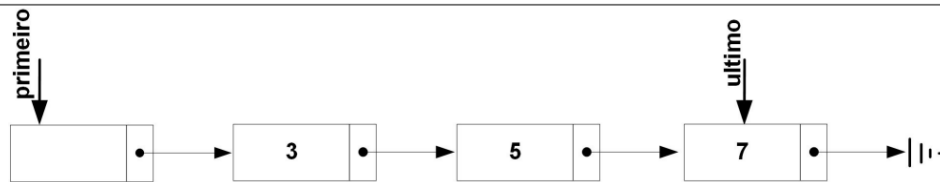


Lista Simples

- Listas simplesmente encadeadas/ Listas ligadas;
- Ela sempre tem a primeira posição nula, usada pra indicar a próxima célula;



-Possui:

Inserir Início, Inserir Fim, Inserir qualquer posição

Remover Início, Remover Fim, Remover qualquer posição

Classes

-----Célula-----

```
1 class Celula {
2
3     public Jogador elemento; // Elemento inserido na celula.
4     public Celula prox; // Aponta a celula prox.
5
6     public Celula() {
7         this(null);
8     }
9
10    public Celula(Jogador elemento) {
11        this.elemento = elemento;
12        this.prox = null;
13    }
14 }
```

-----Lista Simples-----

```
1 //-----CLASSE LISTA FLEXIVEL-----//
2 class ListaFlexivel {
3
4     private Celula primeiro;
5     private Celula ultimo;
6
7     // construtores
8
9     public ListaFlexivel() {
10        primeiro = new Celula();
11        ultimo = primeiro;
12    }
```

Funções

-----Inserir Início-----

```
1 public void inserirInicio(Jogador jogador) {
2
3     Celula tmp = new Celula(jogador);
4
5     tmp.prox = primeiro.prox;
6     primeiro.prox = tmp;
7
8     if (primeiro == ultimo) { // se a lista estiver vazia
9         ultimo = tmp;
10    }
11
12    tmp = null;
13
14 }
```

3. Cria uma variável de Célula **tmp**, com os valores do elemento a ser inserido;

(Sendo, **tmp.elemento=jogador**, e **tmp.prox=Null**)

5. Faz a célula **tmp.prox** apontar para **primeiro.prox**;

(Agora, **tmp.prox=primeiro.prox**)

6. Faz a célula **primeiro.prox** apontar para **tmp**;

(Agora, **primeiro.prox= tmp**)

Antes era:

```
/* primeiro-->primeiro.prox-->primeiro.prox.prox */
```

Criamos e apontamos:

```
/* temp.prox-->primeiro.prox */
```

Inserimos:

```
/* primeiro-->temp-->primeiro.prox-->primeiro.prox.prox */
```

Resultado:

```
/*primeiro-->primeiro.prox(tmp)->primeiro.prox.prox(primeiro.prox)*/
```

-----Inserir Fim-----

```
1 public void inserirFim(Jogador jogador) {  
2  
3     ultimo.prox = new Celula(jogador);  
4     ultimo = ultimo.prox;  
5  
6 }
```

3. Faz a célula `ultimo.prox` ter o valor da célula com o elemento a inserir;
(Iniciando com `ultimo.prox = Null` e agora `ultimo.prox = jogador`)

4. Faz a célula `último` se tornar o `ultimo.prox`;
(Agora, `último = ultimo.prox`)

Antes era:

```
/* ultimo--> ultimo.prox=null */
```

Demos valor:

```
/* ultimo--> ultimo.prox= Jogador*/
```

Realocamos último:

```
/* último==ultimo.prox*/
```

Resultado:

```
/* último(ultimo.prox)--> ultimo.prox==null*/
```

-----Inserir Em Qualquer Posição-----

```
1 public void inserir(Jogador jogador, int pos)
2 {
3     int tamanho = tamanho();
4
5     if(pos<0 || pos>tamanho){System.err.println("Erro ao Inserir na posição");}
6
7     else if(pos == 0){inserirInicio(jogador);}
8     else if(pos == tamanho){inserirFim(jogador);}
9
10    else{
11        Celula i = primeiro;
12        for(int j = 0; j < pos; j++, i = i.prox);
13
14        Celula tmp = new Celula(jogador);
15        tmp.prox = i.prox;
16        i.prox = tmp;
17
18        tmp=i=null;
19    }
20 }
```

7/8. Facilita nosso trabalho puxando a função inserir início e fim;

(Se **pos=0** vai inserir no início e se **pos=tamanho** vai inserir no fim)

11/12. Cria uma célula **i** com valor de **primeiro**, movendo-a até a célula antes da célula que a posição queremos inserir;

14. Cria célula tmp, com o valor que queremos inserir;

(Sendo, **tmp.elemento=jogador**, e **tmp.prox=Null**)

15. Faz a célula **tmp.prox** apontar para **i.prox**;

(Agora, **tmp.prox = i.prox**)

6.Faz a célula **i.prox** apontar para **tmp**;

Antes era:

```
/* i-->i.prox-->i.prox.prox */
```

Criamos e apontamos:

```
/* tmp.prox-->i.prox */
```

Inserimos:

```
/* i-->tmp-->i.prox-->i.prox.prox */
```

Resultado:

```
/* i-->i.prox(tmp)-->i.prox.prox(i.prox) */
```

-----Remover Início-----



4. Cria uma variável de Célula **tmp**, com os valores da célula **primeiro.prox**;
(Sendo **tmp.elemento==primeiro.prox.elemento** e **tmp.prox== primeiro.prox.prox**)
(Para mover as células, salvando os apontamentos do **primeiro.prox** original)

5. Faz a célula **primeiro.prox** ter o valor da célula **primeiro.prox.prox**;
(Agora, **primeiro.prox =tmp.prox**)

7. Salva o valor do elemento de primeiro para retornar ele depois;
(**removido= tmp.elemento**)

9/10. Tornamos todos os valores de tmp nulos
(Tornamos o **primeiro.prox**, no **primeiro.prox.prox** e jogamos o **primeiro.prox** fora)

Antes era:

```
/* primeiro-->primeiro.prox-->primeiro.prox.prox */
```

Criamos e alocamos:

```
/* temp=primeiro.prox--> primeiro.prox.prox */
```

```
/* primeiro.prox=primeiro.prox.prox-->primeiro.prox.prox.prox */
```

Resultado:

```
/*primeiro-->primeiro.prox(primeiro.prox)*/
```

```
/*primeiro.prox(primeiro.prox)-->primeiro.prox.prox(tmp.prox.prox)*/
```

-----Remover Fim-----



3/5. Criamos uma célula **i** com valor de **primeiro**, movendo-a até a célula antes da célula **último**;

7. Salva o valor do elemento de primeiro para retornar ele depois;

(`removido=ultimo.elemento`)

8. Faz a célula `ultimo` ter o valor da penúltima célula;

(Agora, `ultimo=i`)

11/12. Tornamos `ultimo.prox` em nulo e `i` também;

(Tornamos o `ultimo`, na penúltima célula, e `ultimo.prox=null`)

Antes era:

```
/* i-->ultimo->null*/
```

Mudamos:

```
/* ultimo=i-->i.prox-->null */
```

Resultado:

```
/* ultimo(i)-->null(ultimo original) */
```

-----Remover Em Qualquer Posição-----



5-9. Facilita nosso trabalho puxando a função remover início e fim;

(Se `pos=0` vai remover no início e se `pos=tamanho` vai remover no fim)

11/12. Cria célula `i=primeiro`, movendo-a até a anterior da célula que `removeremos`;

15. Cria célula `tmp`, com a célula que queremos remover;

(Sendo `tmp.= i.prox`, e `tmp.prox= i.prox.prox`)

15. Faz a célula `i.prox` virar `i.prox.prox` e desconecta o `tmp`;

(Agora `i.prox=tmp.prox`, e `tmp.prox = i = null`)

Antes era:

```
/* i-->i.prox-->i.prox.prox */
```

Removemos:

```
/* i.prox=tmp.prox(i.prox.prox)*/
```

Resultado:

```
/* i-->i.prox(i.prox.prox)-->i.prox.prox(i.prox.prox.prox) */
```

Pilha Simples

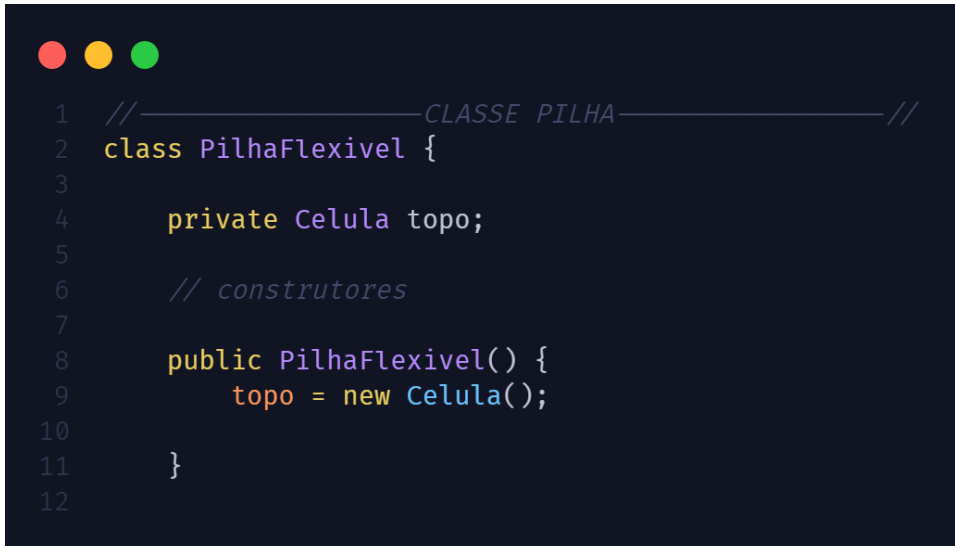
- As pilhas podem inserir e remover elementos só no fim (basicamente uma lista reduzida);
- Possui: Inserir Fim e Remover Fim

Classes

-----Célula-----

```
1 class Celula {
2
3     public Jogador elemento; // Elemento inserido na celula.
4     public Celula prox; // Aponta a celula prox.
5
6     public Celula() {
7         this(null);
8     }
9
10    public Celula(Jogador elemento) {
11        this.elemento = elemento;
12        this.prox = null;
13    }
14 }
```

-----Pilha Simples-----



```

1  //-----CLASSE PILHA-----//
2  class PilhaFlexivel {
3
4      private Celula topo;
5
6      // construtores
7
8      public PilhaFlexivel() {
9          topo = new Celula();
10
11      }
12

```

Funções

Inserir (Igual Inserir Fim da Lista Simples)

Remover (Igual Remover Fim da Lista Simples)

Fila Circular

- As filas podem só inserir no fim e remover no início (uma pilha com funções oposta);
- Possui: **Inserir Fim** e **Remover Início**, e contando com um limitador de tamanho;
- Diferente das outras pode ser feita em C;

Classes

-----Célula-----


```

1  typedef struct Celula
2  {
3      Jogador jogador;    // Elemento inserido na celula.
4      struct Celula *prox; // Aponta a celula prox.
5  } Celula;
6
7  Celula *novaCelula(Jogador elemento)
8  {
9      Celula *nova = (Celula *)malloc(sizeof(Celula));
10     nova->jogador = elemento;
11     nova->prox = NULL;
12
13     return nova;
14 }

```

-----Pilha Simples-----

```

1  //-----CLASSE FILA-----//
2  typedef struct Fila
3  {
4      Celula *primeiro, *ultimo;
5      int size, maxSize;
6
7  } Fila;

```

Funções

Inserir (Igual Inserir Fim da Lista Simples, com repetidor)

Remover (Igual Remover Início da Lista Simples)