Aula 21 – Listas Ligadas

Norton Trevisan Roman

13 de junho de 2018

Listas Ligadas

- Considere nosso condomínio

```
class Projeto {
  ...
  public static void main(String[] args) {
    CasaRet cr = new CasaRet(10,5,1320);
    CasaQuad cq = new CasaQuad(10,1523);


    Residencia r1 = new Residencia(cr, null);
    Residencia r2 = new Residencia(cq, null);

    Projeto p = new Projeto(2);
    p.adicionaRes(r1);
    p.adicionaRes(r2);









  }
}
```

Listas Ligadas

- Considere nosso condomínio
- E se, durante a execução, precisássemos alocar mais uma casa (além dos limites do condomínio)?

```
class Projeto {
  ...
  public static void main(String[] args) {
    CasaRet cr = new CasaRet(10,5,1320);
    CasaQuad cq = new CasaQuad(10,1523);


    Residencia r1 = new Residencia(cr, null);
    Residencia r2 = new Residencia(cq, null);

    Projeto p = new Projeto(2);
    p.adicionaRes(r1);
    p.adicionaRes(r2);









  }
}
```

Listas Ligadas

- Considere nosso condomínio
- E se, durante a execução, precisássemos alocar mais uma casa (além dos limites do condomínio)?

```
class Projeto {
  ...
  public static void main(String[] args) {
    CasaRet cr = new CasaRet(10,5,1320);
    CasaQuad cq = new CasaQuad(10,1523);
    CasaQuad cq2 = new CasaQuad(8,1523);

    Residencia r1 = new Residencia(cr, null);
    Residencia r2 = new Residencia(cq, null);

    Projeto p = new Projeto(2);
    p.adicionaRes(r1);
    p.adicionaRes(r2);
    Residencia r3 = new Residencia(cq2, null);

  }
}
```

Listas Ligadas

- Considere nosso condomínio
- E se, durante a execução, precisássemos alocar mais uma casa (além dos limites do condomínio)?
  - Não haverá espaço alocado para ela no arranjo

```
class Projeto {
  ...
  public static void main(String[] args) {
    CasaRet cr = new CasaRet(10,5,1320);
    CasaQuad cq = new CasaQuad(10,1523);
    CasaQuad cq2 = new CasaQuad(8,1523);

    Residencia r1 = new Residencia(cr, null);
    Residencia r2 = new Residencia(cq, null);

    Projeto p = new Projeto(2);
    p.adicionaRes(r1);
    p.adicionaRes(r2);
    Residencia r3 = new Residencia(cq2, null);

  }
}
```

Listas Ligadas

- Que fazer?

```
class Projeto {
  ...
  public static void main(String[] args) {
    CasaRet cr = new CasaRet(10,5,1320);
    CasaQuad cq = new CasaQuad(10,1523);
    CasaQuad cq2 = new CasaQuad(8,1523);

    Residencia r1 = new Residencia(cr, null);
    Residencia r2 = new Residencia(cq, null);

    Projeto p = new Projeto(2);
    p.adicionaRes(r1);
    p.adicionaRes(r2);
    Residencia r3 = new Residencia(cq2, null);

  }
}
```

Listas Ligadas

- Que fazer?
  - Alocar um novo arranjo maior

```
class Projeto {
  ...
  public static void main(String[] args) {
    CasaRet cr = new CasaRet(10,5,1320);
    CasaQuad cq = new CasaQuad(10,1523);
    CasaQuad cq2 = new CasaQuad(8,1523);

    Residencia r1 = new Residencia(cr, null);
    Residencia r2 = new Residencia(cq, null);

    Projeto p = new Projeto(2);
    p.adicionaRes(r1);
    p.adicionaRes(r2);
    Residencia r3 = new Residencia(cq2, null);









  }
}
```

Listas Ligadas

- Que fazer?
  - Alocar um novo arranjo maior

```
class Projeto {
  ...
  public static void main(String[] args) {
    CasaRet cr = new CasaRet(10,5,1320);
    CasaQuad cq = new CasaQuad(10,1523);
    CasaQuad cq2 = new CasaQuad(8,1523);

    Residencia r1 = new Residencia(cr, null);
    Residencia r2 = new Residencia(cq, null);

    Projeto p = new Projeto(2);
    p.adicionaRes(r1);
    p.adicionaRes(r2);
    Residencia r3 = new Residencia(cq2, null);

    Residencia[] maior = new Residencia[3];









  }
}
```

Listas Ligadas

- Que fazer?
  - Alocar um novo arranjo maior
  - Copiar o conteúdo do velho nesse

```
class Projeto {
  ...
  public static void main(String[] args) {
    CasaRet cr = new CasaRet(10,5,1320);
    CasaQuad cq = new CasaQuad(10,1523);
    CasaQuad cq2 = new CasaQuad(8,1523);

    Residencia r1 = new Residencia(cr, null);
    Residencia r2 = new Residencia(cq, null);

    Projeto p = new Projeto(2);
    p.adicionaRes(r1);
    p.adicionaRes(r2);
    Residencia r3 = new Residencia(cq2, null);

    Residencia[] maior = new Residencia[3];

  }
}
```

Listas Ligadas

- Que fazer?
  - Alocar um novo arranjo maior
  - Copiar o conteúdo do velho nesse

```
class Projeto {
  ...
  public static void main(String[] args) {
    CasaRet cr = new CasaRet(10,5,1320);
    CasaQuad cq = new CasaQuad(10,1523);
    CasaQuad cq2 = new CasaQuad(8,1523);

    Residencia r1 = new Residencia(cr, null);
    Residencia r2 = new Residencia(cq, null);

    Projeto p = new Projeto(2);
    p.adicionaRes(r1);
    p.adicionaRes(r2);
    Residencia r3 = new Residencia(cq2, null);

    Residencia[] maior = new Residencia[3];
    maior[0] = p.condominio[0];
    maior[1] = p.condominio[1];



  }
}
```

Listas Ligadas

- Que fazer?
  - Alocar um novo arranjo maior
  - Copiar o conteúdo do velho nesse
  - Incluir a nova casa

```
class Projeto {
  ...
  public static void main(String[] args) {
    CasaRet cr = new CasaRet(10,5,1320);
    CasaQuad cq = new CasaQuad(10,1523);
    CasaQuad cq2 = new CasaQuad(8,1523);

    Residencia r1 = new Residencia(cr, null);
    Residencia r2 = new Residencia(cq, null);

    Projeto p = new Projeto(2);
    p.adicionaRes(r1);
    p.adicionaRes(r2);
    Residencia r3 = new Residencia(cq2, null);

    Residencia[] maior = new Residencia[3];
    maior[0] = p.condominio[0];
    maior[1] = p.condominio[1];



  }
}
```

Listas Ligadas

- Que fazer?
  - Alocar um novo arranjo maior
  - Copiar o conteúdo do velho nesse
  - Incluir a nova casa

```
class Projeto {
  ...
  public static void main(String[] args) {
    CasaRet cr = new CasaRet(10,5,1320);
    CasaQuad cq = new CasaQuad(10,1523);
    CasaQuad cq2 = new CasaQuad(8,1523);

    Residencia r1 = new Residencia(cr, null);
    Residencia r2 = new Residencia(cq, null);

    Projeto p = new Projeto(2);
    p.adicionaRes(r1);
    p.adicionaRes(r2);
    Residencia r3 = new Residencia(cq2, null);

    Residencia[] maior = new Residencia[3];
    maior[0] = p.condominio[0];
    maior[1] = p.condominio[1];
    maior[2] = r3;


  }
}
```

Listas Ligadas

- Que fazer?
  - Alocar um novo arranjo maior
  - Copiar o conteúdo do velho nesse
  - Incluir a nova casa
  - Substituir o arranjo antigo pelo maior

```
class Projeto {
  ...
  public static void main(String[] args) {
    CasaRet cr = new CasaRet(10,5,1320);
    CasaQuad cq = new CasaQuad(10,1523);
    CasaQuad cq2 = new CasaQuad(8,1523);

    Residencia r1 = new Residencia(cr, null);
    Residencia r2 = new Residencia(cq, null);

    Projeto p = new Projeto(2);
    p.adicionaRes(r1);
    p.adicionaRes(r2);
    Residencia r3 = new Residencia(cq2, null);

    Residencia[] maior = new Residencia[3];
    maior[0] = p.condominio[0];
    maior[1] = p.condominio[1];
    maior[2] = r3;


  }
}
```

Listas Ligadas

- Que fazer?
  - Alocar um novo arranjo maior
  - Copiar o conteúdo do velho nesse
  - Incluir a nova casa
  - Substituir o arranjo antigo pelo maior

```
class Projeto {
  ...
  public static void main(String[] args) {
    CasaRet cr = new CasaRet(10,5,1320);
    CasaQuad cq = new CasaQuad(10,1523);
    CasaQuad cq2 = new CasaQuad(8,1523);

    Residencia r1 = new Residencia(cr, null);
    Residencia r2 = new Residencia(cq, null);

    Projeto p = new Projeto(2);
    p.adicionaRes(r1);
    p.adicionaRes(r2);
    Residencia r3 = new Residencia(cq2, null);

    Residencia[] maior = new Residencia[3];
    maior[0] = p.condominio[0];
    maior[1] = p.condominio[1];
    maior[2] = r3;
    p.condominio = maior;
    p.ultimo++;
  }
}
```

Listas Ligadas

- Que fazer?
  - Alocar um novo arranjo maior
  - Copiar o conteúdo do velho nesse
  - Incluir a nova casa
  - Substituir o arranjo antigo pelo maior
- Incrivelmente ineficiente

```
class Projeto {
  ...
  public static void main(String[] args) {
    CasaRet cr = new CasaRet(10,5,1320);
    CasaQuad cq = new CasaQuad(10,1523);
    CasaQuad cq2 = new CasaQuad(8,1523);

    Residencia r1 = new Residencia(cr, null);
    Residencia r2 = new Residencia(cq, null);

    Projeto p = new Projeto(2);
    p.adicionaRes(r1);
    p.adicionaRes(r2);
    Residencia r3 = new Residencia(cq2, null);

    Residencia[] maior = new Residencia[3];
    maior[0] = p.condominio[0];
    maior[1] = p.condominio[1];
    maior[2] = r3;
    p.condominio = maior;
    p.ultimo++;
  }
}
```

Listas Ligadas

- Deveria haver um modo de simplesmente aumentarmos o arranjo

Listas Ligadas

- Deveria haver um modo de simplesmente aumentarmos o arranjo
  - Com arranjos... não

Listas Ligadas

- Deveria haver um modo de simplesmente aumentarmos o arranjo
  - Com arranjos... não
- Alternativa: Lista Ligada

Listas Ligadas

- Deveria haver um modo de simplesmente aumentarmos o arranjo
  - Com arranjos... não
- Alternativa: Lista Ligada

> Uma lista ligada é uma lista onde cada elemento – chamado de nó – contém um valor e uma referência para o elemento seguinte na lista

Listas Ligadas

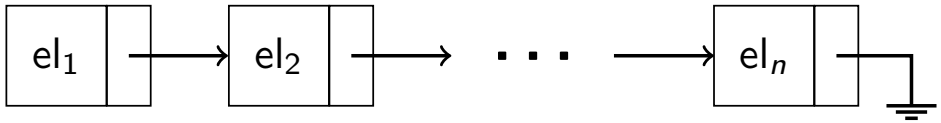
- Deveria haver um modo de simplesmente aumentarmos o arranjo
  - Com arranjos... não
- Alternativa: Lista Ligada

> Uma lista ligada é uma lista onde cada elemento – chamado de nó – contém um valor e uma referência para o elemento seguinte na lista

  - Assim, sabendo onde está o primeiro elemento da lista, podemos chegar a qualquer outro elemento

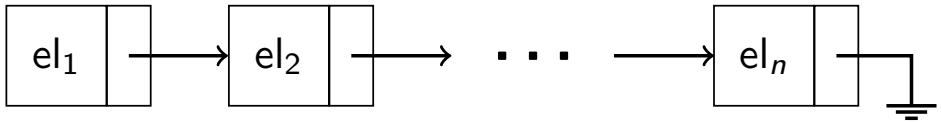
Listas Ligadas

- Simples:

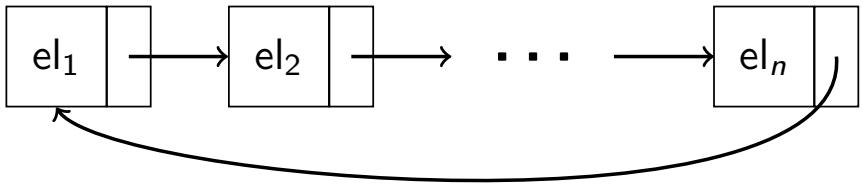


Listas Ligadas

- Simples:

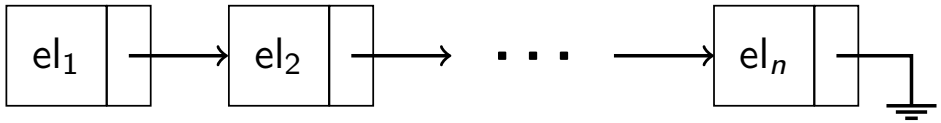


- Circular:

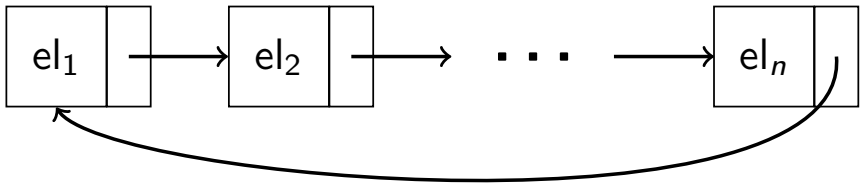


Listas Ligadas

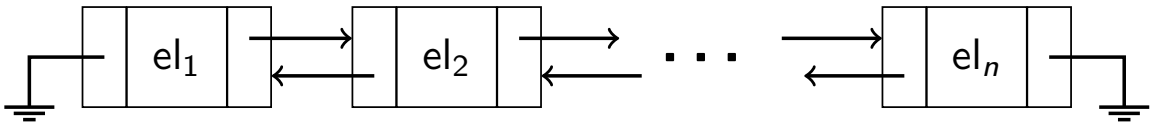
- Simples:



- Circular:

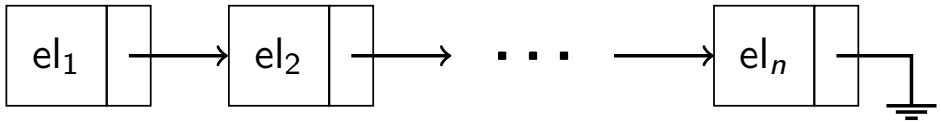


- Duplamente ligada:

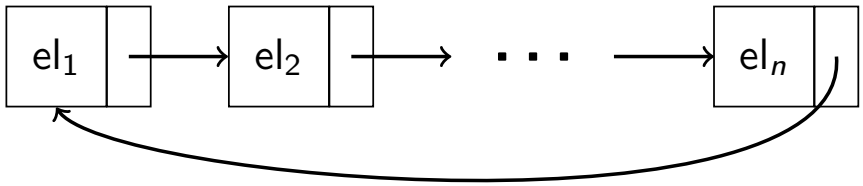


Listas Ligadas

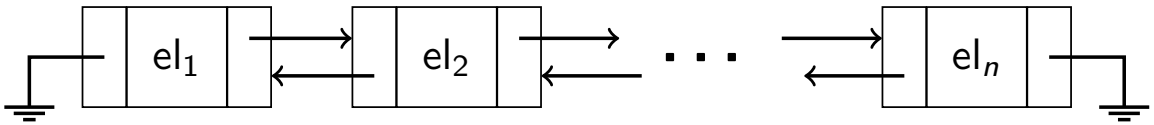
- Simples:



- Circular:



- Duplamente ligada:



- Etc

Listas Ligadas

- Dado armazenado:

Listas Ligadas

- Dado armazenado:
  - (Endereço de) objeto `Residencia`

Listas Ligadas

- Dado armazenado:
  - (Endereço de) objeto `Residencia`
- Criando o nó:

```
public class No {
  Residencia r;
  No prox = null;

  public No(Residencia r) {
    this.r = r;
  }
}
```

Listas Ligadas

- Dado armazenado:
  - (Endereço de) objeto `Residencia`
- Criando o nó:
  - Temos o dado armazenado

```
public class No {
  Residencia r;
  No prox = null;

  public No(Residencia r) {
    this.r = r;
  }
}
```

Listas Ligadas

- Dado armazenado:
  - (Endereço de) objeto `Residencia`
- Criando o nó:
  - Temos o dado armazenado
  - E uma referência para o próximo elemento

```
public class No {
  Residencia r;
  No prox = null;

  public No(Residencia r) {
    this.r = r;
  }
}
```

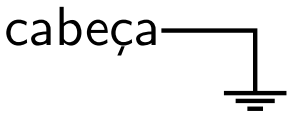
Listas Ligadas

- Criando uma lista simples:

```
public class No {
  Residencia r;
  No prox = null;

  public No(Residencia r) {
    this.r = r;
  }
}

public class ListaSimples {
  No cabeca;

  public ListaSimples() {
    this.cabeca = null;
  }

  public void insere(Residencia el) {
    No aux = new No(el);
    aux.prox = this.cabeca;
    this.cabeca = aux;
  }
}
```

Listas Ligadas

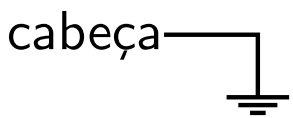
- Criando uma lista simples:




```
public class No {
  Residencia r;
  No prox = null;

  public No(Residencia r) {
    this.r = r;
  }
}

public class ListaSimples {
  No cabeca;

  public ListaSimples() {
    this.cabeca = null;
  }

  public void insere(Residencia el) {
    No aux = new No(el);
    aux.prox = this.cabeca;
    this.cabeca = aux;
  }
}
```

Listas Ligadas

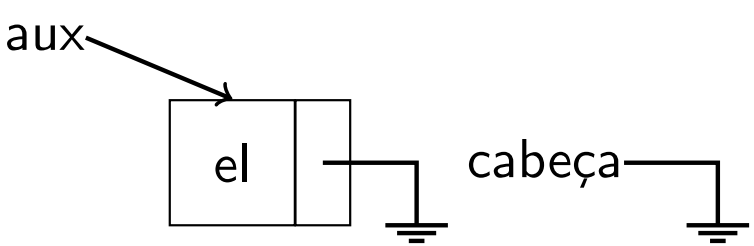
- Inserindo elemento no início:




```
public class No {
  Residencia r;
  No prox = null;

  public No(Residencia r) {
    this.r = r;
  }
}

public class ListaSimples {
  No cabeca;

  public ListaSimples() {
    this.cabeca = null;
  }

  public void insere(Residencia el) {
    No aux = new No(el);
    aux.prox = this.cabeca;
    this.cabeca = aux;
  }
}
```

Listas Ligadas

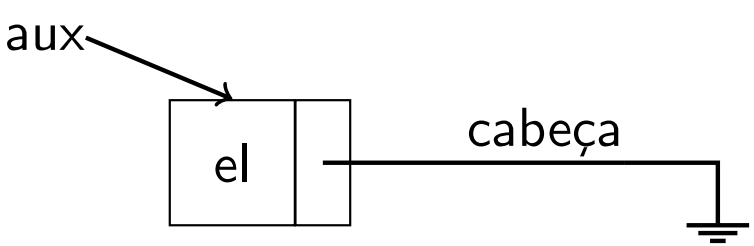
- Inserindo elemento no início:
  - Crie o novo elemento a ser inserido




```
public class No {
  Residencia r;
  No prox = null;

  public No(Residencia r) {
    this.r = r;
  }
}

public class ListaSimples {
  No cabeca;

  public ListaSimples() {
    this.cabeca = null;
  }

  public void insere(Residencia el) {
    No aux = new No(el);
    aux.prox = this.cabeca;
    this.cabeca = aux;
  }
}
```

Listas Ligadas

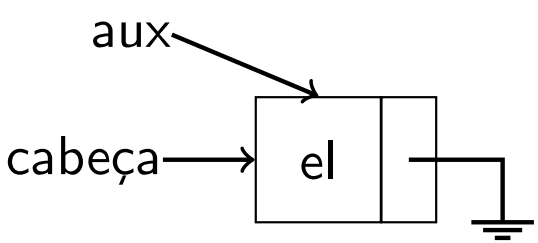
- Inserindo elemento no início:
  - Crie o novo elemento a ser inserido
  - Faça o elemento do início da lista ser seu próximo




```
public class No {
  Residencia r;
  No prox = null;

  public No(Residencia r) {
    this.r = r;
  }
}

public class ListaSimples {
  No cabeca;

  public ListaSimples() {
    this.cabeca = null;
  }

  public void insere(Residencia el) {
    No aux = new No(el);
    aux.prox = this.cabeca;
    this.cabeca = aux;
  }
}
```

Listas Ligadas

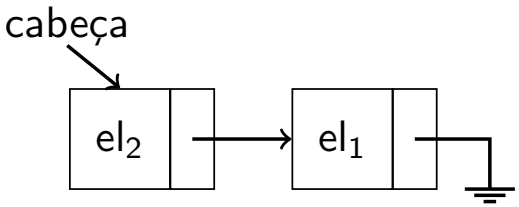
- Inserindo elemento no início:
  - Crie o novo elemento a ser inserido
  - Faça o elemento do início da lista ser seu próximo
  - Torne esse novo elemento o novo início da lista




```
public class No {
  Residencia r;
  No prox = null;

  public No(Residencia r) {
    this.r = r;
  }
}

public class ListaSimples {
  No cabeca;

  public ListaSimples() {
    this.cabeca = null;
  }

  public void insere(Residencia el) {
    No aux = new No(el);
    aux.prox = this.cabeca;
    this.cabeca = aux;
  }
}
```

Listas Ligadas

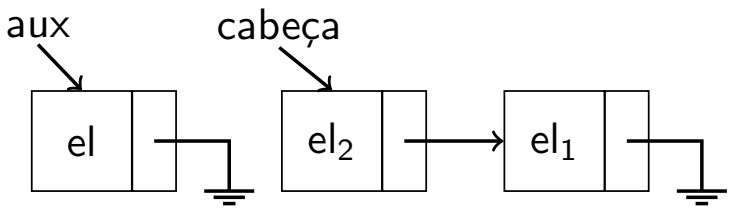
- E se a lista já tiver algo?




```
public class No {
  Residencia r;
  No prox = null;

  public No(Residencia r) {
    this.r = r;
  }
}

public class ListaSimples {
  No cabeca;

  public ListaSimples() {
    this.cabeca = null;
  }

  public void insere(Residencia el) {
    No aux = new No(el);
    aux.prox = this.cabeca;
    this.cabeca = aux;
  }
}
```

Listas Ligadas

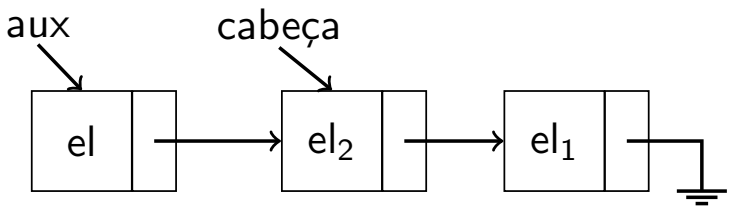
- E se a lista já tiver algo?




```
public class No {
  Residencia r;
  No prox = null;

  public No(Residencia r) {
    this.r = r;
  }
}

public class ListaSimples {
  No cabeca;

  public ListaSimples() {
    this.cabeca = null;
  }

  public void insere(Residencia el) {
    No aux = new No(el);
    aux.prox = this.cabeca;
    this.cabeca = aux;
  }
}
```

Listas Ligadas

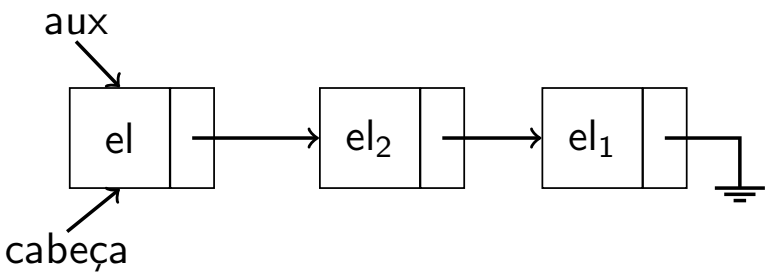
- E se a lista já tiver algo?




```
public class No {
  Residencia r;
  No prox = null;

  public No(Residencia r) {
    this.r = r;
  }
}

public class ListaSimples {
  No cabeca;

  public ListaSimples() {
    this.cabeca = null;
  }

  public void insere(Residencia el) {
    No aux = new No(el);
    aux.prox = this.cabeca;
    this.cabeca = aux;
  }
}
```

Listas Ligadas

- E se a lista já tiver algo?




```
public class No {
  Residencia r;
  No prox = null;

  public No(Residencia r) {
    this.r = r;
  }
}

public class ListaSimples {
  No cabeca;

  public ListaSimples() {
    this.cabeca = null;
  }

  public void insere(Residencia el) {
    No aux = new No(el);
    aux.prox = this.cabeca;
    this.cabeca = aux;
  }
}
```

Listas Ligadas

- E como podemos usar isso?

Listas Ligadas

- E como podemos usar isso?

```
public static void main(String[] args) {
  CasaRet cr = new CasaRet(10,5,1320);
  CasaQuad cq = new CasaQuad(10,1523);
  CasaQuad cq2 = new CasaQuad(8,1523);

  Residencia r1 = new Residencia(cr,null);
  Residencia r2 = new Residencia(cq,null);
  Residencia r3 = new Residencia(cq2,null);

  ListaSimples l = new ListaSimples();

  l.insere(r1);
  l.insere(r2);
  l.insere(r3);

  No n = l.cabeca;
  while (n != null) {
    System.out.println(n.r.casa.area());
    n = n.prox;
  }
}
```

Listas Ligadas

- E como podemos usar isso?
- Repare!

```
public static void main(String[] args) {
  CasaRet cr = new CasaRet(10,5,1320);
  CasaQuad cq = new CasaQuad(10,1523);
  CasaQuad cq2 = new CasaQuad(8,1523);

  Residencia r1 = new Residencia(cr,null);
  Residencia r2 = new Residencia(cq,null);
  Residencia r3 = new Residencia(cq2,null);

  ListaSimples l = new ListaSimples();

  l.insere(r1);
  l.insere(r2);
  l.insere(r3);

  No n = l.cabeca;
  while (n != null) {
    System.out.println(n.r.casa.area());
    n = n.prox;
  }
}
```

Listas Ligadas

- E como podemos usar isso?
- Repare!
  - Não precisamos de variável auxiliar para guardar o resultado de um método

```
public static void main(String[] args) {
  CasaRet cr = new CasaRet(10,5,1320);
  CasaQuad cq = new CasaQuad(10,1523);
  CasaQuad cq2 = new CasaQuad(8,1523);

  Residencia r1 = new Residencia(cr,null);
  Residencia r2 = new Residencia(cq,null);
  Residencia r3 = new Residencia(cq2,null);

  ListaSimples l = new ListaSimples();

  l.insere(r1);
  l.insere(r2);
  l.insere(r3);

  No n = l.cabeca;
  while (n != null) {
    System.out.println(n.r.casa.area());
    n = n.prox;
  }
}
```

Listas Ligadas

- E como podemos usar isso?
- Repare!
  - Não precisamos de variável auxiliar para guardar o resultado de um método
  - O computador irá buscar o atributo `r`, de `n`

```
public static void main(String[] args) {
  CasaRet cr = new CasaRet(10,5,1320);
  CasaQuad cq = new CasaQuad(10,1523);
  CasaQuad cq2 = new CasaQuad(8,1523);

  Residencia r1 = new Residencia(cr,null);
  Residencia r2 = new Residencia(cq,null);
  Residencia r3 = new Residencia(cq2,null);

  ListaSimples l = new ListaSimples();

  l.insere(r1);
  l.insere(r2);
  l.insere(r3);

  No n = l.cabeca;
  while (n != null) {
    System.out.println(n.r.casa.area());
    n = n.prox;
  }
}
```

Listas Ligadas

- E como podemos usar isso?
- Repare!
  - Não precisamos de variável auxiliar para guardar o resultado de um método
  - O computador irá buscar o atributo `r`, de `n`
  - Então olhará o atributo `casa`, de `r`

```
public static void main(String[] args) {
  CasaRet cr = new CasaRet(10,5,1320);
  CasaQuad cq = new CasaQuad(10,1523);
  CasaQuad cq2 = new CasaQuad(8,1523);

  Residencia r1 = new Residencia(cr,null);
  Residencia r2 = new Residencia(cq,null);
  Residencia r3 = new Residencia(cq2,null);

  ListaSimples l = new ListaSimples();

  l.insere(r1);
  l.insere(r2);
  l.insere(r3);

  No n = l.cabeca;
  while (n != null) {
    System.out.println(n.r.casa.area());
    n = n.prox;
  }
}
```

Listas Ligadas

- Repare!
  - Finalmente, chamará o método `area()` desse atributo.

```
public static void main(String[] args) {
  CasaRet cr = new CasaRet(10,5,1320);
  CasaQuad cq = new CasaQuad(10,1523);
  CasaQuad cq2 = new CasaQuad(8,1523);

  Residencia r1 = new Residencia(cr,null);
  Residencia r2 = new Residencia(cq,null);
  Residencia r3 = new Residencia(cq2,null);

  ListaSimples l = new ListaSimples();

  l.insere(r1);
  l.insere(r2);
  l.insere(r3);

  No n = l.cabeca;
  while (n != null) {
    System.out.println(n.r.casa.area());
    n = n.prox;
  }
}
```

Listas Ligadas

- Repare!
  - Finalmente, chamará o método area() desse atributo.

Saída

```
$ java Projeto
64.0
100.0
150.0
```

```
public static void main(String[] args) {
  CasaRet cr = new CasaRet(10,5,1320);
  CasaQuad cq = new CasaQuad(10,1523);
  CasaQuad cq2 = new CasaQuad(8,1523);

  Residencia r1 = new Residencia(cr,null);
  Residencia r2 = new Residencia(cq,null);
  Residencia r3 = new Residencia(cq2,null);

  ListaSimples l = new ListaSimples();

  l.insere(r1);
  l.insere(r2);
  l.insere(r3);

  No n = l.cabeca;
  while (n != null) {
    System.out.println(n.r.casa.area());
    n = n.prox;
  }
}
```

Listas Ligadas

- Repare!
  - Finalmente, chamará o método `area()` desse atributo.

Saída

```
$ java Projeto
64.0
100.0
150.0
```

(Note que a ordem está inversa à de inserção)

```
public static void main(String[] args) {
  CasaRet cr = new CasaRet(10,5,1320);
  CasaQuad cq = new CasaQuad(10,1523);
  CasaQuad cq2 = new CasaQuad(8,1523);

  Residencia r1 = new Residencia(cr,null);
  Residencia r2 = new Residencia(cq,null);
  Residencia r3 = new Residencia(cq2,null);

  ListaSimples l = new ListaSimples();

  l.insere(r1);
  l.insere(r2);
  l.insere(r3);

  No n = l.cabeca;
  while (n != null) {
    System.out.println(n.r.casa.area());
    n = n.prox;
  }
}
```

Listas Ligadas – Mudando Projeto

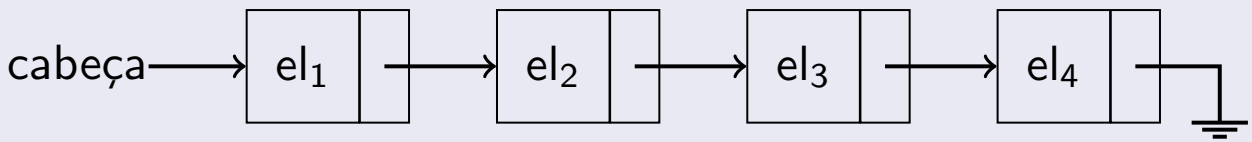
```
class Projeto {
    Residencia[] condominio;
    int ultimo = -1;
    boolean adicionaRes(Residencia r) {
        if (this.ultimo < this.condominio.length-1) {
            ultimo++;
            this.condominio[ultimo] = r;
            return(true);
        }
        return(false);
    }
    Projeto(int tam) {
        this.condominio = new Residencia[tam];
    }
    public static void main(String[] args) {
        CasaRet cr = new CasaRet(10,5,1320);
        CasaQuad cq = new CasaQuad(10,1523);
        CasaQuad cq2 = new CasaQuad(8,1523);

        Residencia r1 = new Residencia(cr, null);
        Residencia r2 = new Residencia(cq, null);
        Residencia r3 = new Residencia(cq2, null);

        Projeto p = new Projeto(3);

        p.adicionaRes(r1);
        p.adicionaRes(r2);
        p.adicionaRes(r3);
    }
}
```

Listas Ligadas – Mudando Projeto

```
class Projeto {
    Residencia[] condominio;
    int ultimo = -1;
    boolean adicionaRes(Residencia r) {
        if (this.ultimo < this.condominio.length-1) {
            ultimo++;
            this.condominio[ultimo] = r;
            return(true);
        }
        return(false);
    }
    Projeto(int tam) {
        this.condominio = new Residencia[tam];
    }
    public static void main(String[] args) {
        CasaRet cr = new CasaRet(10,5,1320);
        CasaQuad cq = new CasaQuad(10,1523);
        CasaQuad cq2 = new CasaQuad(8,1523);

        Residencia r1 = new Residencia(cr, null);
        Residencia r2 = new Residencia(cq, null);
        Residencia r3 = new Residencia(cq2, null);

        Projeto p = new Projeto(3);

        p.adicionaRes(r1);
        p.adicionaRes(r2);
        p.adicionaRes(r3);
    }
}
```

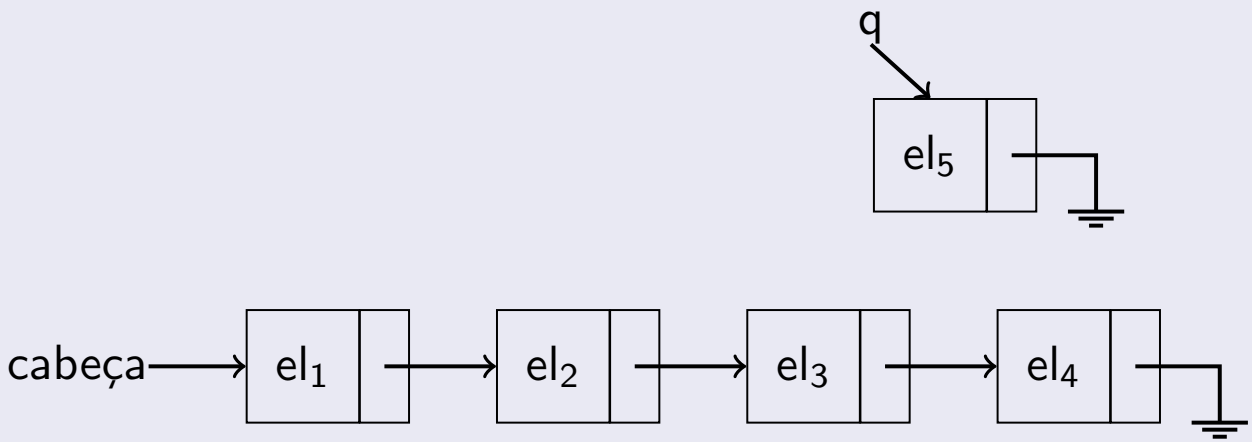
```
class Projeto {
    ListaSimples condominio;

    void adicionaRes(Residencia r) {
        this.condominio.insere(r);
    }

    Projeto() {
        this.condominio = new ListaSimples();
    }
    public static void main(String[] args) {
        CasaRet cr = new CasaRet(10,5,1320);
        CasaQuad cq = new CasaQuad(10,1523);
        CasaQuad cq2 = new CasaQuad(8,1523);

        Residencia r1 = new Residencia(cr, null);
        Residencia r2 = new Residencia(cq, null);
        Residencia r3 = new Residencia(cq2, null);

        Projeto p = new Projeto();

        p.adicionaRes(r1);
        p.adicionaRes(r2);
        p.adicionaRes(r3);
    }
}
```

Incluindo Elemento em uma Posição (4^a)



Incluindo Elemento em uma Posição (4^a)

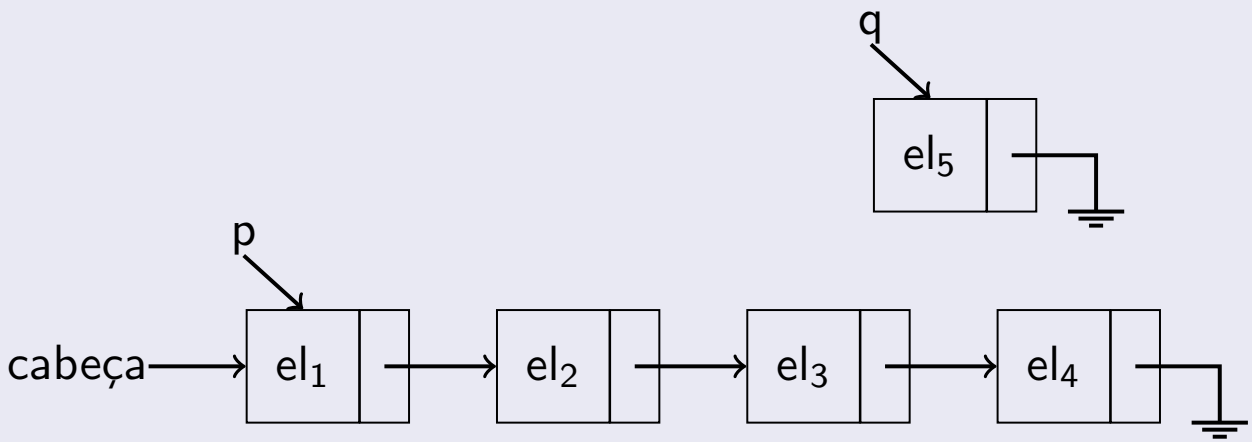
- Alocamos espaço para o novo elemento



Listas Ligadas

Incluindo Elemento em uma Posição (4^a)

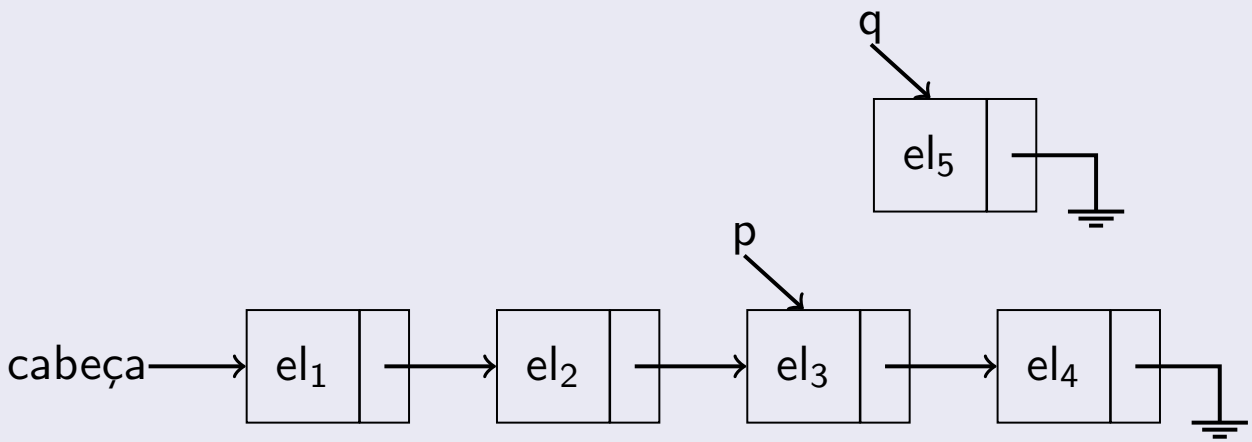
- Alocamos espaço para o novo elemento
- Marcamos o início da lista



Listas Ligadas

Incluindo Elemento em uma Posição (4^a)

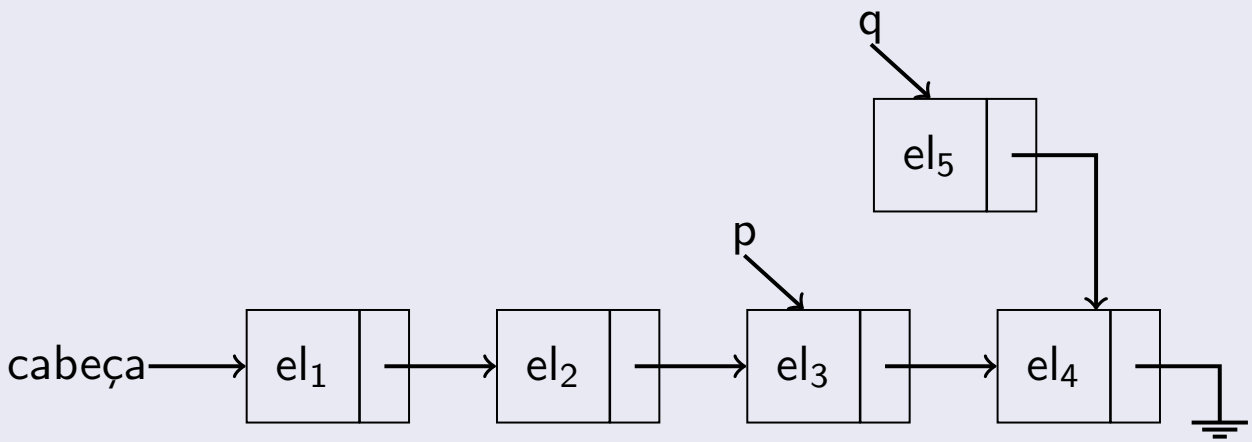
- Alocamos espaço para o novo elemento
- Marcamos o início da lista
- Andamos até a (n-1)^a posição:



Listas Ligadas

Incluindo Elemento em uma Posição (4^a)

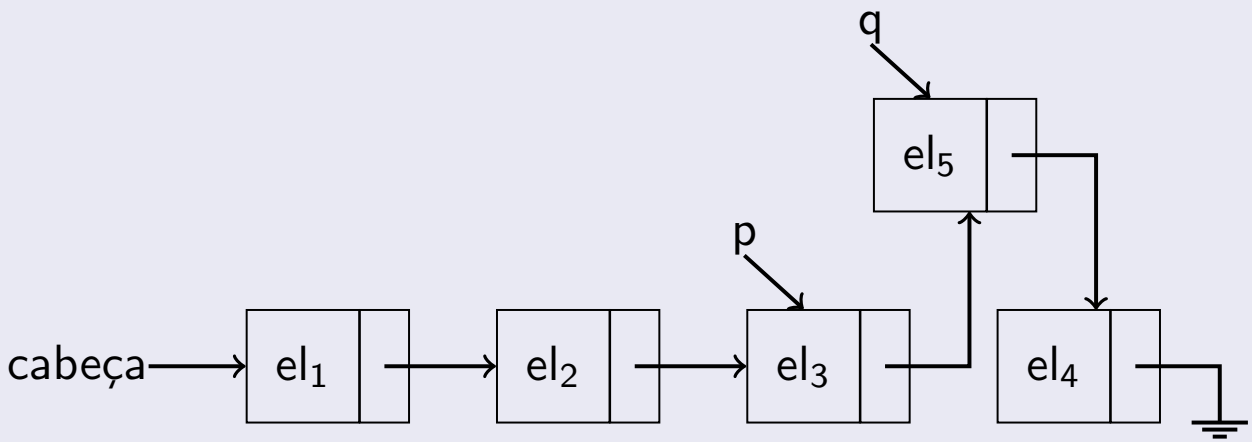
- Fazemos o campo prox do novo elemento apontar para o nº elemento da lista



Listas Ligadas

Incluindo Elemento em uma Posição (4^a)

- Fazemos o campo prox do novo elemento apontar para o nº elemento da lista
- Fazemos o elemento seguinte ao (n-1)º ser o novo elemento



Listas Ligadas

- Isso contudo funciona para posições > 0

Listas Ligadas

- Isso contudo funciona para posições > 0
- E se for na posição 0?

Listas Ligadas

- Isso contudo funciona para posições > 0
- E se for na posição 0?
  - Já vimos inserção na primeira posição

Listas Ligadas

- Isso contudo funciona para posições > 0
- E se for na posição 0?
  - Já vimos inserção na primeira posição
- Então...

```
public void insere(Residencia el, int pos) {
  No q = new No(el);
  if (pos == 0) {
    q.prox = this.cabeca;
    this.cabeca = q;
  }
  else {
    No p = this.cabeca;
    for (int i=0; i<pos-1; i++) p = p.prox;
    q.prox = p.prox;
    p.prox = q;
  }
}
```

Listas Ligadas

- Naturalmente, falta ainda verificar se não tentamos colocar um elemento após o último.

```
public void insere(Residencia el, int pos) {
  No q = new No(el);
  if (pos == 0) {
    q.prox = this.cabeca;
    this.cabeca = q;
  }
  else {
    No p = this.cabeca;
    for (int i=0; i<pos-1; i++) p = p.prox;
    q.prox = p.prox;
    p.prox = q;
  }
}
```

Listas Ligadas

- Naturalmente, falta ainda verificar se não tentamos colocar um elemento após o último.
  - Para isso, podemos criar um método que retorne o número de elementos da lista:

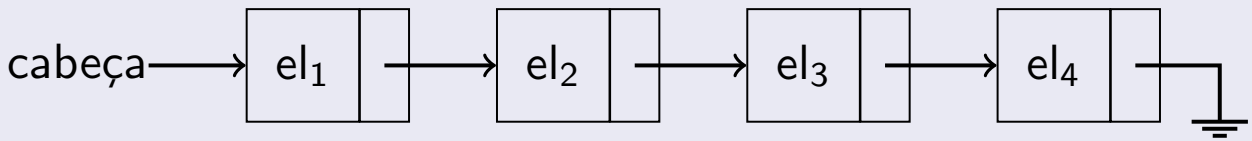
```
public void insere(Residencia el, int pos) {
  No q = new No(el);
  if (pos == 0) {
    q.prox = this.cabeca;
    this.cabeca = q;
  }
  else {
    No p = this.cabeca;
    for (int i=0; i<pos-1; i++) p = p.prox;
    q.prox = p.prox;
    p.prox = q;
  }
}
```

Listas Ligadas

- Naturalmente, falta ainda verificar se não tentamos colocar um elemento após o último.
  - Para isso, podemos criar um método que retorne o número de elementos da lista:

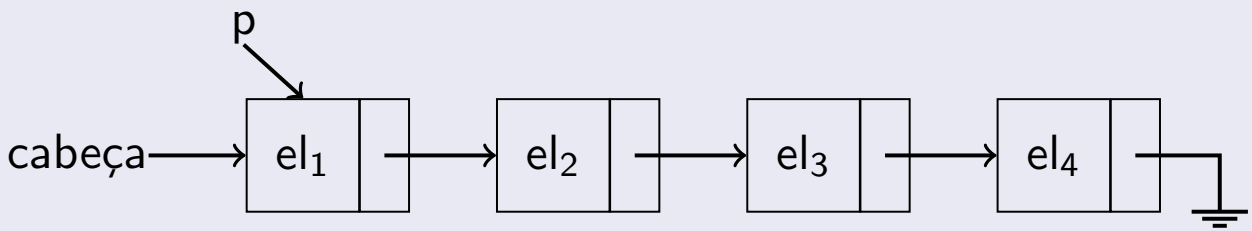
```
public void insere(Residencia el, int pos) {
  No q = new No(el);
  if (pos == 0) {
    q.prox = this.cabeca;
    this.cabeca = q;
  }
  else {
    No p = this.cabeca;
    for (int i=0; i<pos-1; i++) p = p.prox;
    q.prox = p.prox;
    p.prox = q;
  }
}

public int elementos() {
  int cont = 0;
  No p = this.cabeca;
  while (p != null) {
    p = p.prox;
    cont++;
  }
  return(cont);
}
```

Excluindo Elemento em uma Posição (3ª)



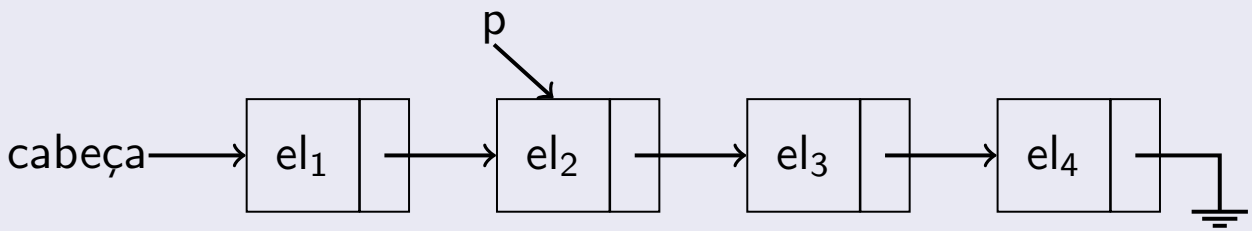
Excluindo Elemento em uma Posição (3^a)

- Marcamos o início da lista



Excluindo Elemento em uma Posição (3ª)

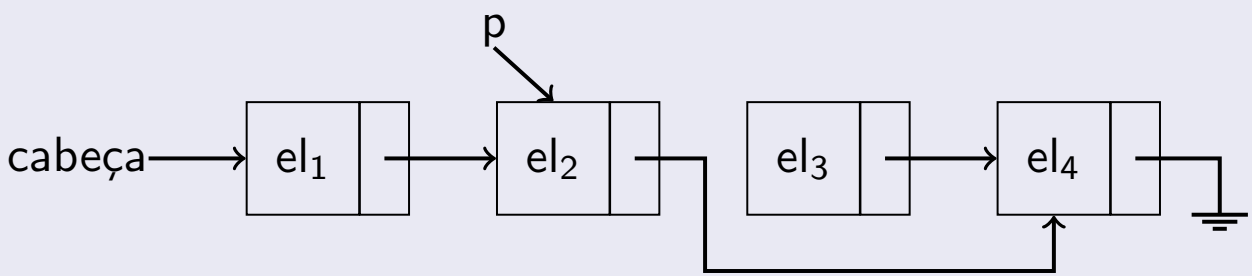
- Marcamos o início da lista
- Movemos até o elemento anterior ao nº



Listas Ligadas

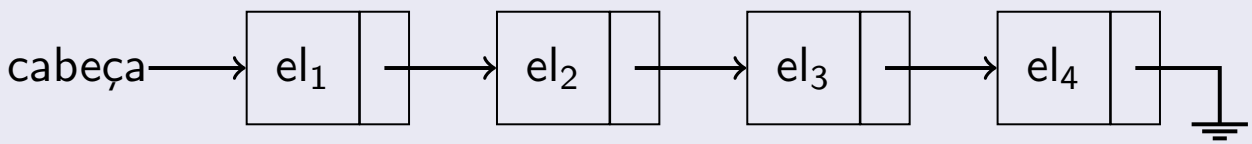
Excluindo Elemento em uma Posição (3ª)

- Marcamos o início da lista
- Movemos até o elemento anterior ao nº
- Fazemos o próximo elemento de p ser o elemento que está após seu próximo



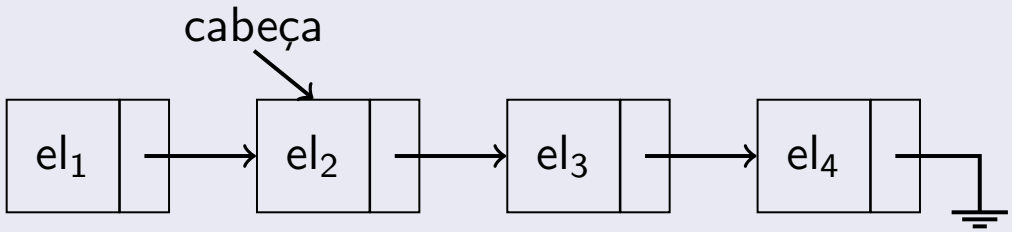
Excluindo Elemento em uma Posição (3^a)

- E se a posição pretendida for a primeira (0)?



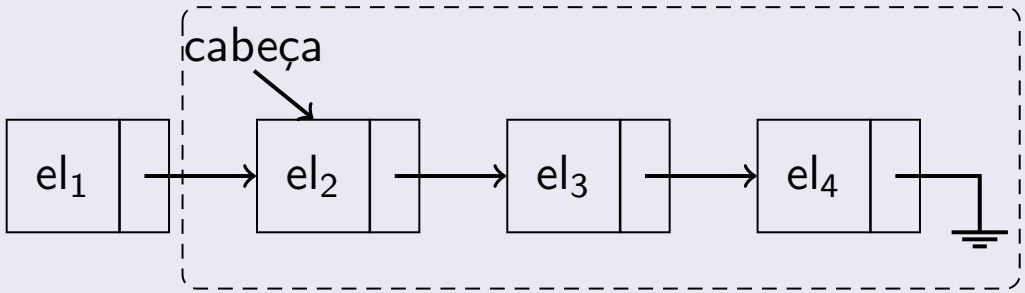
Excluindo Elemento em uma Posição (3ª)

- E se a posição pretendida for a primeira (0)?
- Movemos a cabeça



Excluindo Elemento em uma Posição (3ª)

- E se a posição pretendida for a primeira (0)?
- Movemos a cabeça
- A lista fica



Listas Ligadas

- Então...

```
public boolean exclui(int i) {
  if (i < 0 || i >= this.elementos() ||
          this.cabeca == null) return(false);
  if (i == 0) this.cabeca = this.cabeca.prox;
  else {
    No p = this.cabeca;
    for (int j=0; j<(i-1); j++) {
      p = p.prox;
    }
    p.prox = p.prox.prox;
  }
  return(true);
}
```

Listas Ligadas

- Então...
- Incluímos testes dos parâmetros

```
public boolean exclui(int i) {
  if (i < 0 || i >= this.elementos() ||
          this.cabeca == null) return(false);
  if (i == 0) this.cabeca = this.cabeca.prox;
  else {
    No p = this.cabeca;
    for (int j=0; j<(i-1); j++) {
      p = p.prox;
    }
    p.prox = p.prox.prox;
  }
  return(true);
}
```

Listas Ligadas

- Modificamos `insere` também, incluindo os testes dos parâmetros

```
public boolean insere(Residencia el, int pos)
{
  if (pos < 0 || pos > this.elementos())
    return(false);
  No q = new No(el);
  if (pos == 0) {
    q.prox = this.cabeca;
    this.cabeca = q;
  }
  else {
    No p = this.cabeca;
    for (int i=0; i<pos-1; i++) p = p.prox;
    q.prox = p.prox;
    p.prox = q;
  }
  return(true);
}
```

Listas Ligadas

- Alternativamente à chamada constante de `elementos()`, poderíamos ter um atributo `elementos` na classe `ListaSimples`, que guardasse o número de elementos da lista

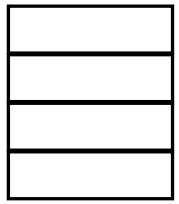
```
public boolean insere(Residencia el, int pos)
{
  if (pos < 0 || pos > this.elementos())
    return(false);
  No q = new No(el);
  if (pos == 0) {
    q.prox = this.cabeca;
    this.cabeca = q;
  }
  else {
    No p = this.cabeca;
    for (int i=0; i<pos-1; i++) p = p.prox;
    q.prox = p.prox;
    p.prox = q;
  }
  return(true);
}
```

Pilhas

- Estruturas muito usadas em computação

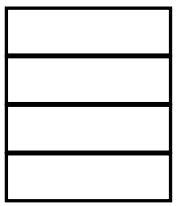
Pilhas

- Estruturas muito usadas em computação
- Pense numa pilha de livros



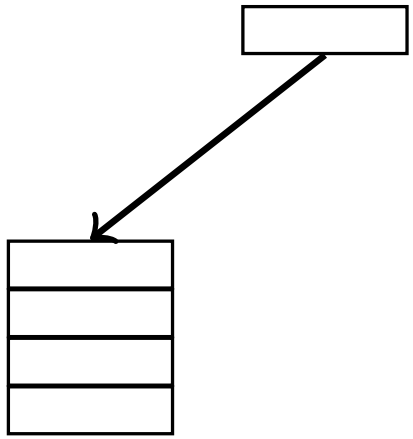
Pilhas

- Estruturas muito usadas em computação
- Pense numa pilha de livros
  - Se queremos por mais um livro na pilha, onde colocamos?



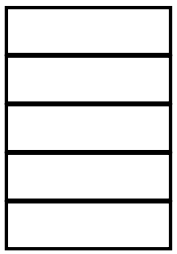
Pilhas

- Estruturas muito usadas em computação
- Pense numa pilha de livros
  - Se queremos por mais um livro na pilha, onde colocamos?



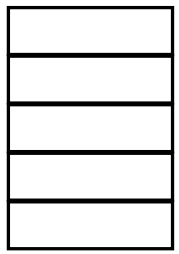
Pilhas

- Estruturas muito usadas em computação
- Pense numa pilha de livros
  - Se queremos por mais um livro na pilha, onde colocamos?



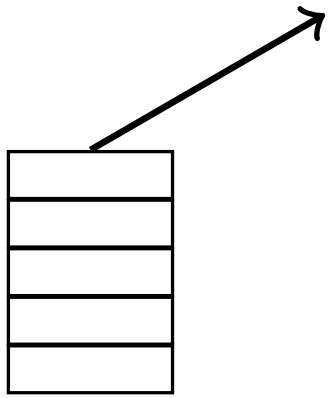
Pilhas

- Estruturas muito usadas em computação
- Pense numa pilha de livros
  - Se queremos por mais um livro na pilha, onde colocamos?
  - Se queremos tirar um livro, de onde tiramos?



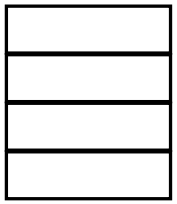
Pilhas

- Estruturas muito usadas em computação
- Pense numa pilha de livros
  - Se queremos por mais um livro na pilha, onde colocamos?
  - Se queremos tirar um livro, de onde tiramos?



Pilhas

- Estruturas muito usadas em computação
- Pense numa pilha de livros
  - Se queremos por mais um livro na pilha, onde colocamos?
  - Se queremos tirar um livro, de onde tiramos?



Pilhas

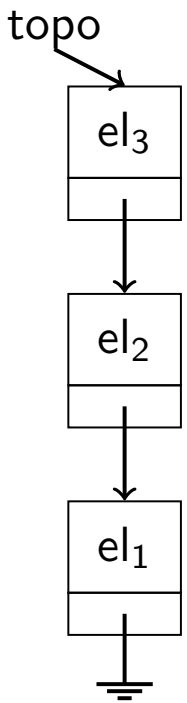
- E como representamos isso?

Pilhas

- E como representamos isso?
  - Dentre outras coisas, com uma lista ligada:

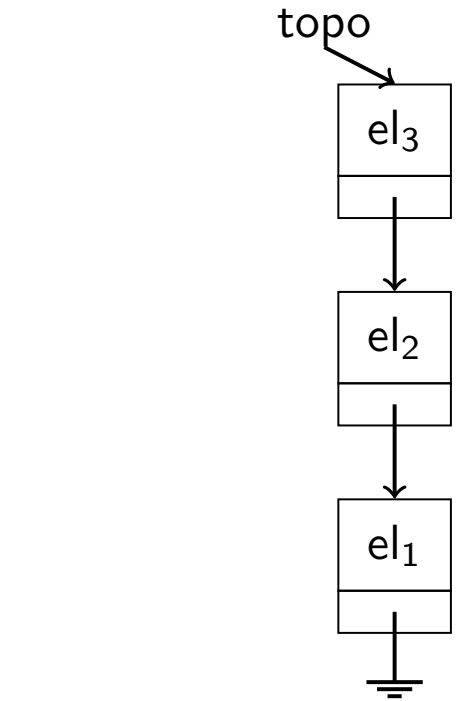
Pilhas

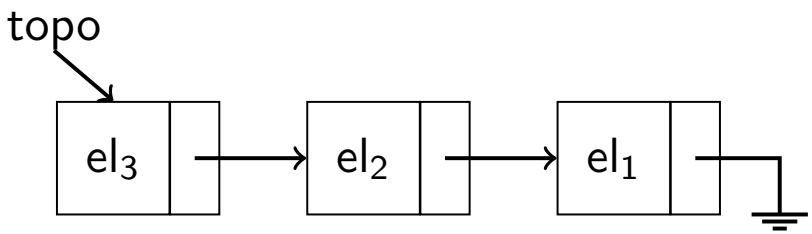
- E como representamos isso?
  - Dentre outras coisas, com uma lista ligada:



Pilhas

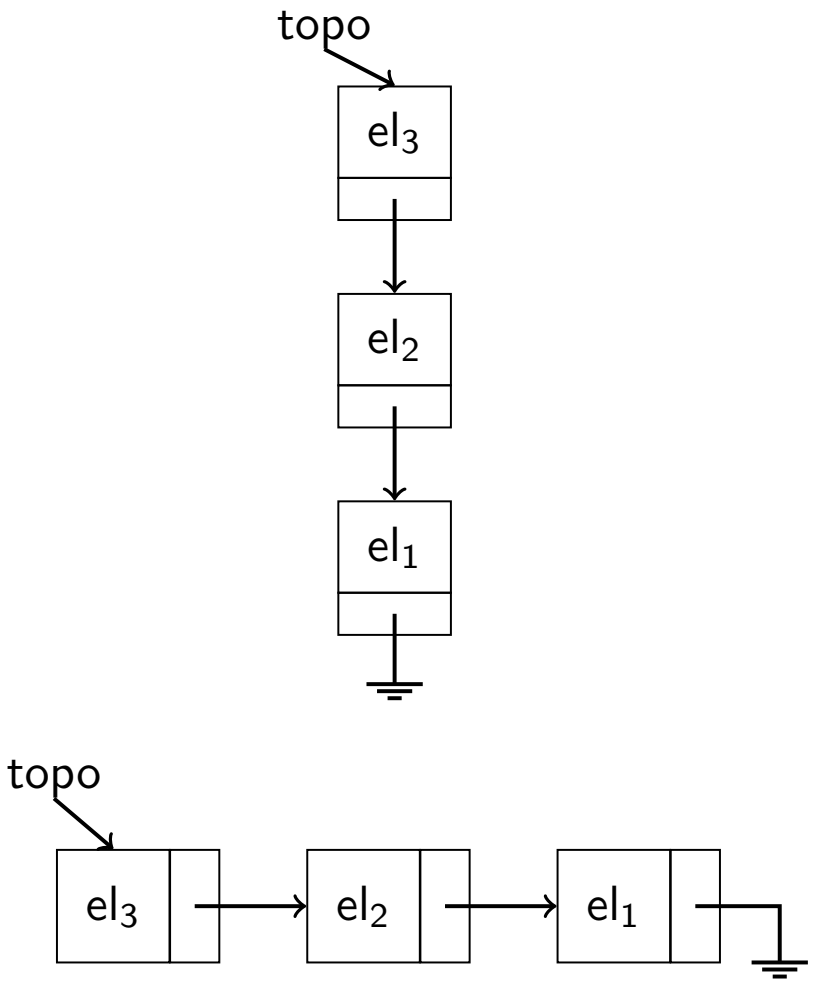
- E como representamos isso?
  - Dentre outras coisas, com uma lista ligada:





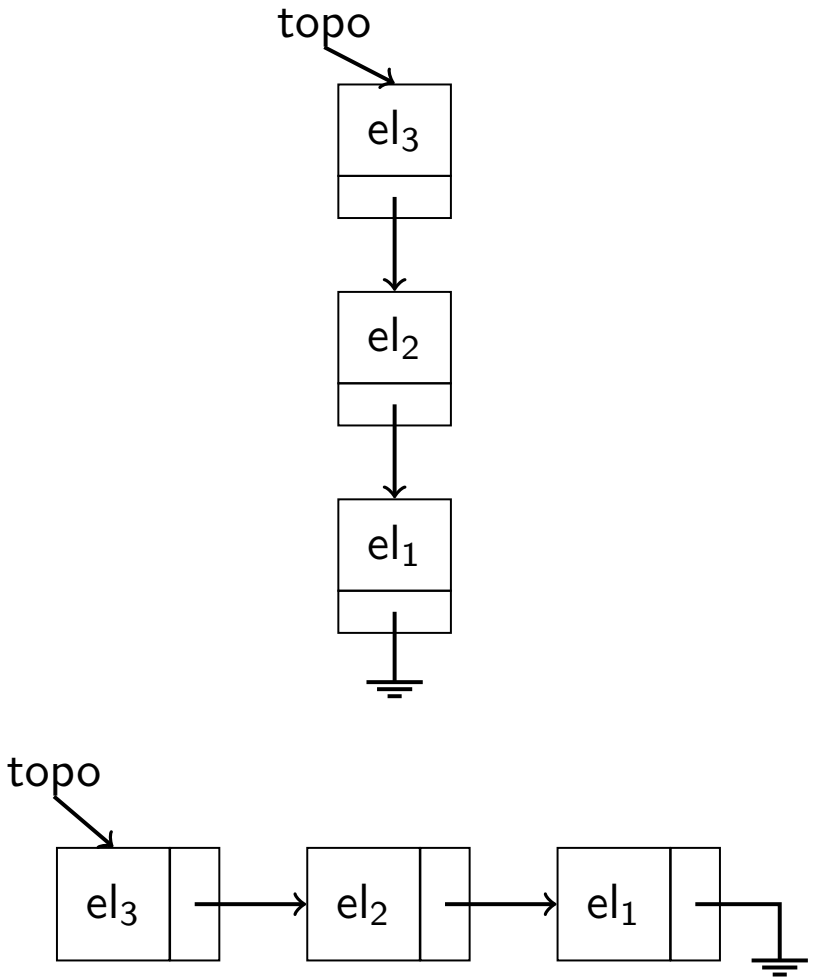
Pilhas

- E como representamos isso?
  - Dentre outras coisas, com uma lista ligada:
- Empilhamentos e Desempilhamentos são feitos sempre na posição 0



Pilhas

- E como representamos isso?
  - Dentre outras coisas, com uma lista ligada:
- Empilhamentos e Desempilhamentos são feitos sempre na posição 0
  - O topo da pilha



Filas

- Pilhas são do tipo LIFO (Last In First Out)

Filas

- Pilhas são do tipo LIFO (Last In First Out)
  - O último a entrar é o primeiro a sair

Filas

- Pilhas são do tipo LIFO (Last In First Out)
  - O último a entrar é o primeiro a sair
- Uma estrutura alternativa à pilha é a Fila

Filas

- Pilhas são do tipo LIFO (Last In First Out)
  - O último a entrar é o primeiro a sair
- Uma estrutura alternativa à pilha é a Fila
  - FIFO (First In First Out)

Filas

- Pilhas são do tipo LIFO (Last In First Out)
  - O último a entrar é o primeiro a sair
- Uma estrutura alternativa à pilha é a Fila
  - FIFO (First In First Out)
  - O primeiro item a entrar é o primeiro a sair

Filas

- Pense numa fila de banco

Filas

- Pense numa fila de banco
  - Se uma nova pessoa aparece, onde entra?

Filas

- Pense numa fila de banco
  - Se uma nova pessoa aparece, onde entra?
  - Quem será o primeiro a ser atendido?

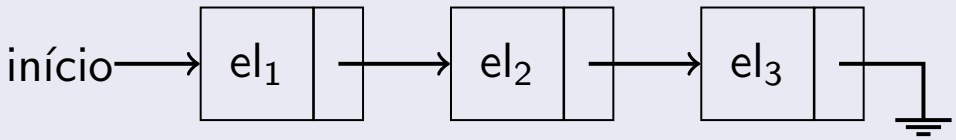
Filas

- Pense numa fila de banco
  - Se uma nova pessoa aparece, onde entra?
  - Quem será o primeiro a ser atendido?
- Se queremos retirar um elemento, retiramos da frente; se queremos incluir, incluimos no final da fila

Filas

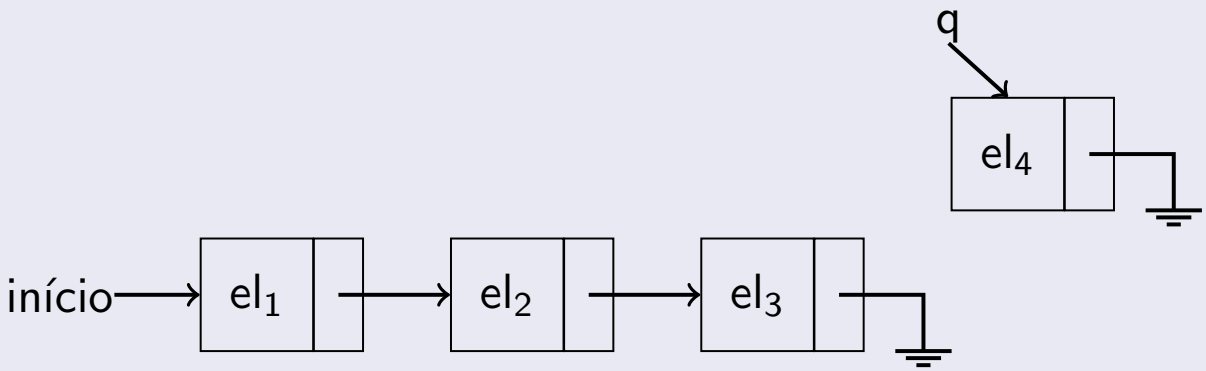
- Pense numa fila de banco
  - Se uma nova pessoa aparece, onde entra?
  - Quem será o primeiro a ser atendido?
- Se queremos retirar um elemento, retiramos da frente; se queremos incluir, incluimos no final da fila
- Da mesma forma que em uma pilha, em uma fila não podemos retirar um elemento do meio da fila, ou lá colocar um

Incluindo um Elemento



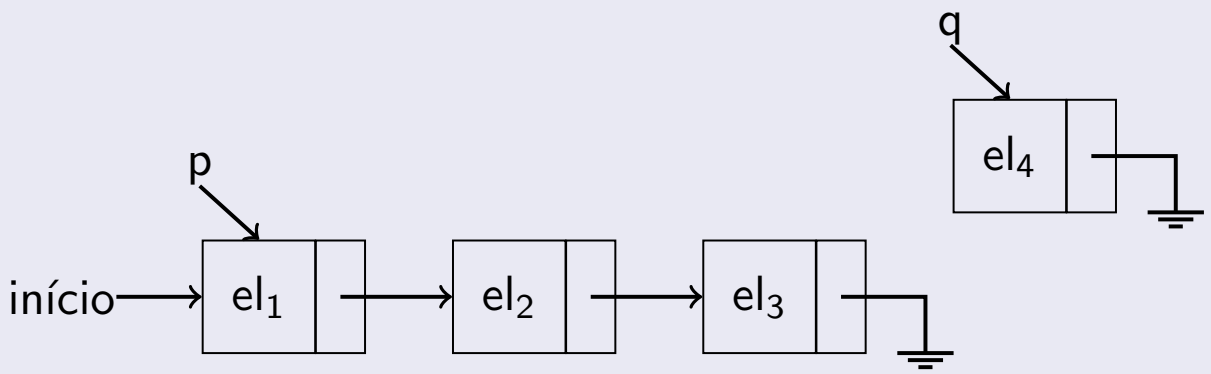
Incluindo um Elemento

- Criamos o elemento novo



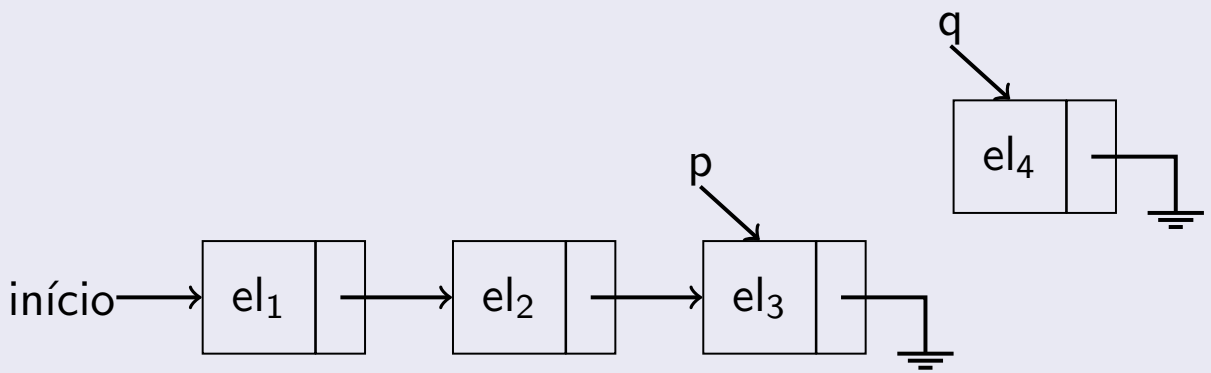
Incluindo um Elemento

- Criamos o elemento novo
- Marcamos o início da fila



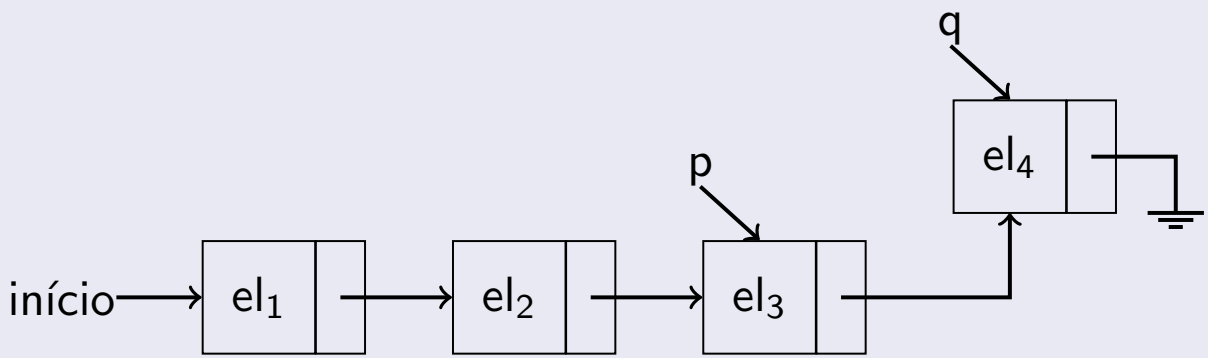
Incluindo um Elemento

- Corremos o ponteiro até o final da fila



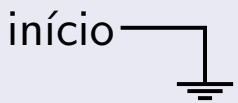
Incluindo um Elemento

- Corremos o ponteiro até o final da fila
- Fazemos o elemento seguinte ao final da fila ser o novo elemento



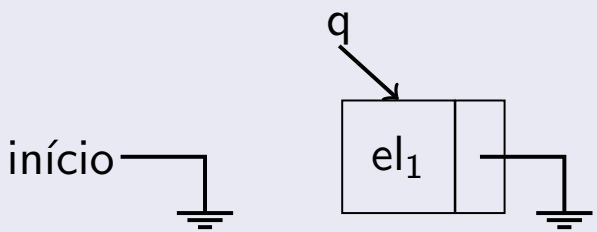
Incluindo um Elemento

- E se a fila estiver inicialmente vazia?



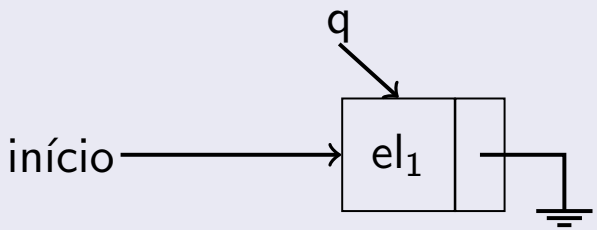
Incluindo um Elemento

- E se a fila estiver inicialmente vazia?
- Criamos o novo elemento



Incluindo um Elemento

- E se a fila estiver inicialmente vazia?
- Criamos o novo elemento
- E fazemos ele ser o início da fila



Excluindo um Elemento

- E como excluímos de uma fila?

Filas

Excluindo um Elemento

- E como excluímos de uma fila?
- Do mesmo modo que em uma pilha

Listas Ligadas – Considerações Finais

- Trabalhamos até agora com acesso direto aos atributos

```
public class No {
  Residencia r;
  No prox = null;

  public No(Residencia r) {
    this.r = r;
  }
}
```

Listas Ligadas – Considerações Finais

- Trabalhamos até agora com acesso direto aos atributos
- Podemos restringir esse acesso, via getters e setters

```
public class No {
  Residencia r;
  No prox = null;

  public No(Residencia r) {
    this.r = r;
  }
}
```

Listas Ligadas – Considerações Finais

- Trabalhamos até agora com acesso direto aos atributos
- Podemos restringir esse acesso, via getters e setters
- Tornamos os atributos private, com métodos de acesso public

```
public class No {
  private Residencia r;
  private No prox = null;

  public No(Residencia r) {
    this.r = r;
  }

  public Residencia getRes() {
    return(this.r);
  }

  public No getProx() {
    return(this.prox);
  }

  public void setRes(Residencia r) {
    this.r = r;
  }

  public void setProx(No prox) {
    this.prox = prox;
  }
}
```

Listas Ligadas – Considerações Finais

- Boa prática de programação

```
public class No {
  private Residencia r;
  private No prox = null;

  public No(Residencia r) {
    this.r = r;
  }

  public Residencia getRes() {
    return(this.r);
  }

  public No getProx() {
    return(this.prox);
  }

  public void setRes(Residencia r) {
    this.r = r;
  }

  public void setProx(No prox) {
    this.prox = prox;
  }
}
```

Listas Ligadas – Considerações Finais

- Boa prática de programação
- Nos permite dar acesso apenas para leitura, por exemplo

```
public class No {
  private Residencia r;
  private No prox = null;

  public No(Residencia r) {
    this.r = r;
  }

  public Residencia getRes() {
    return(this.r);
  }

  public No getProx() {
    return(this.prox);
  }

  public void setRes(Residencia r) {
    this.r = r;
  }

  public void setProx(No prox) {
    this.prox = prox;
  }
}
```

Listas Ligadas – Considerações Finais

- Boa prática de programação
- Nos permite dar acesso apenas para leitura, por exemplo
  - Removendo `setRes` e `setProx`

```
public class No {
  private Residencia r;
  private No prox = null;

  public No(Residencia r) {
    this.r = r;
  }

  public Residencia getRes() {
    return(this.r);
  }

  public No getProx() {
    return(this.prox);
  }
}
```

QUE SCANANAGEM!!!

Java já faz tudo pra você!

- LinkedList (duplamente ligada): `https://docs.oracle.com/javase/9/docs/api/java/util/LinkedList.html`
- Vector (arranjo de tamanho variável): `https://docs.oracle.com/javase/9/docs/api/java/util/Vector.html`
- Stack (de Vector): `https://docs.oracle.com/javase/9/docs/api/java/util/Stack.html`
- Queue (interface – mais adiante veremos): `https://docs.oracle.com/javase/9/docs/api/java/util/Queue.html`

Videoaula

Não há.