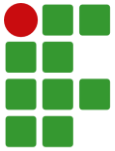


INSTITUTO FEDERAL
Sertão Pernambucano
Campus Salgueiro

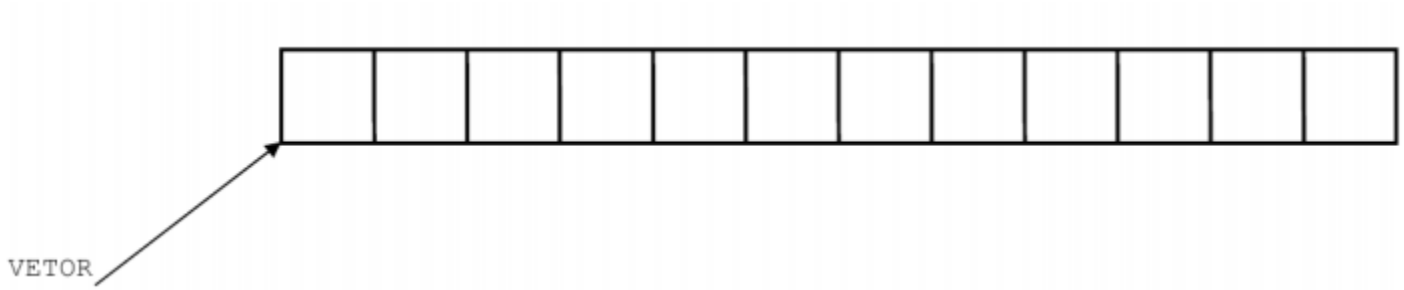
Estrutura de Dados e Algoritmos com Java

Prof. Heraldo Gonçalves Lima Junior
heraldo.junior@ifsertao-pe.edu.br

1. Estruturas Dinâmicas

1.1. Motivação

- Vetor:
  - Ocupa um espaço contíguo de memória ;
  - Deve ser dimensionado com um número máximo de elementos;



1.1. Motivação

- **Estruturas de dados dinâmicas:** crescem ou decrescem à medida que elementos são inseridos ou removidos.
- **Exemplo:** Listas Encadeadas (ligadas)

1. Listas Encadeadas

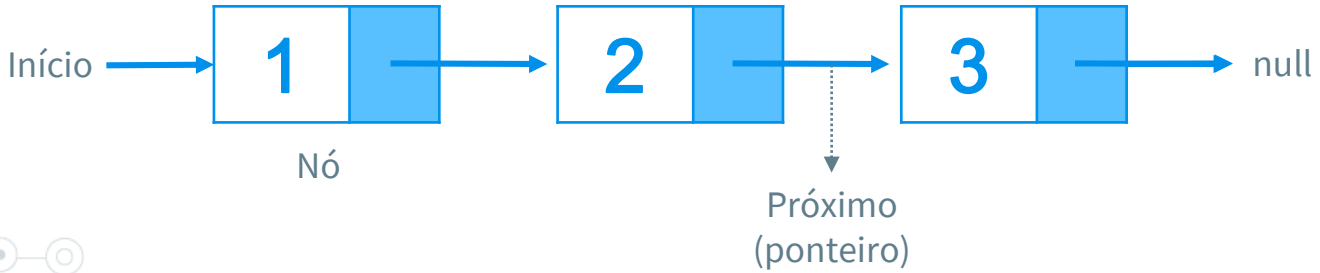
2.1. Definição

- Uma lista encadeada é uma estrutura de dados ligada que consiste numa cadeia única de nós, cada um está conectado ao seguinte por uma ligação.

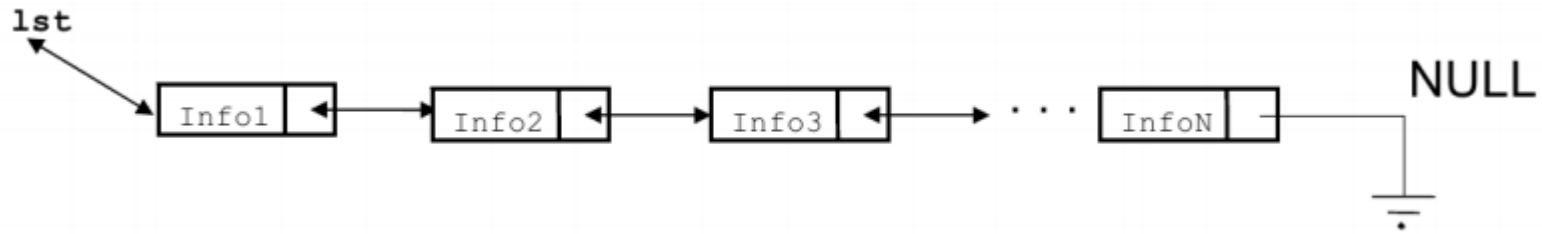
- Este é o tipo mais simples de estrutura de dados encadeado, mas é, no entanto, amplamente utilizada.

2.1. Definição

- Nó da lista é representado por 2 campos:
  - A informação armazenada
  - O ponteiro para o próximo elemento da lista

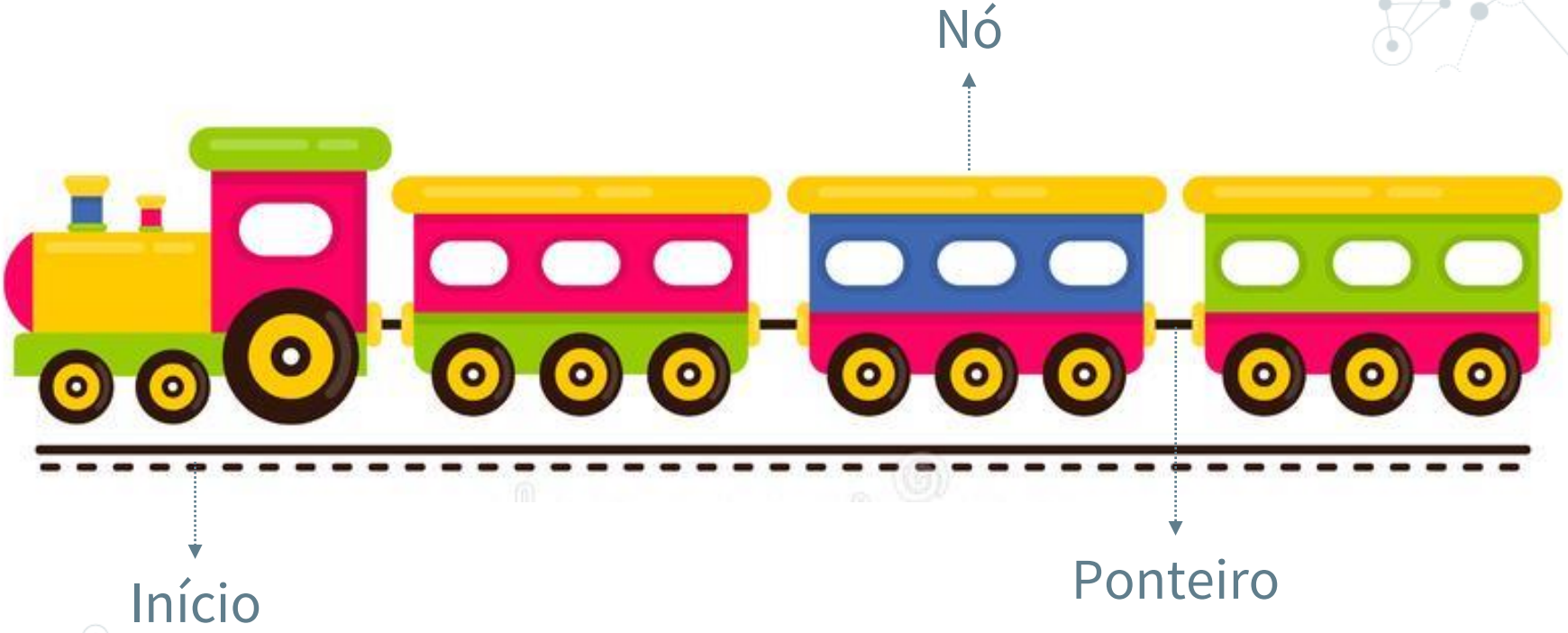


2.1. Definição



- A lista é representada por uma referência ao primeiro elemento;
- O ponteiro do último elemento aponta para null.

2.1. Definição



2.2. Criando a lista

- Vamos então isolar a manipulação de nossa estrutura de dados, para isso teremos uma classe para representar uma célula(nó).

- Ela deve possuir uma referência para o elemento a qual ela se refere, e uma referência para o próximo nó, que pode ser null caso essa seja o última da Lista.

2.2. Criando a lista

```
public class No {

    private No proxima;
    private Object elemento;

    //defino também os métodos Getters,
    //Setters e Constructor
}
```

2.2. Criando a lista

```
public class ListaEncadeada {
    private No inicio;
    private No fim;
    private int totalDeElementos;
}
```

3. Operações sobre Listas Encadeadas

3.1. Adicionando no COMEÇO da Lista

- Inserir no começo da Lista é bastante trivial, basta criarmos uma nova célula, e esta nova célula terá a referência **proxima** apontando para a atual **primeira** da lista.

- Depois atualizamos o atributo primeira para se referenciar a esta nova célula recém criada.

3.1. Adicionando no COMEÇO da Lista

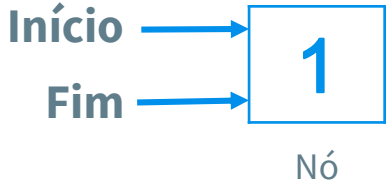
- Exemplo de inserção em lista **vazia**:

ANTES DA INSERÇÃO

Início = Null

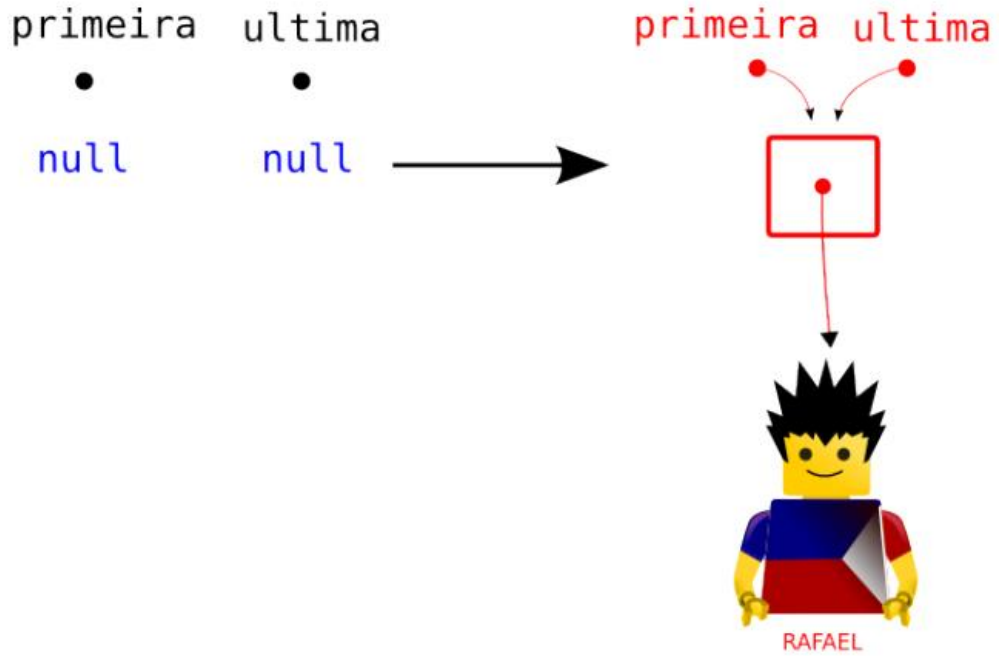
Fim = Null

DEPOIS DA INSERÇÃO



3.1. Adicionando no COMEÇO da Lista

- Exemplo de inserção em lista **vazia**:

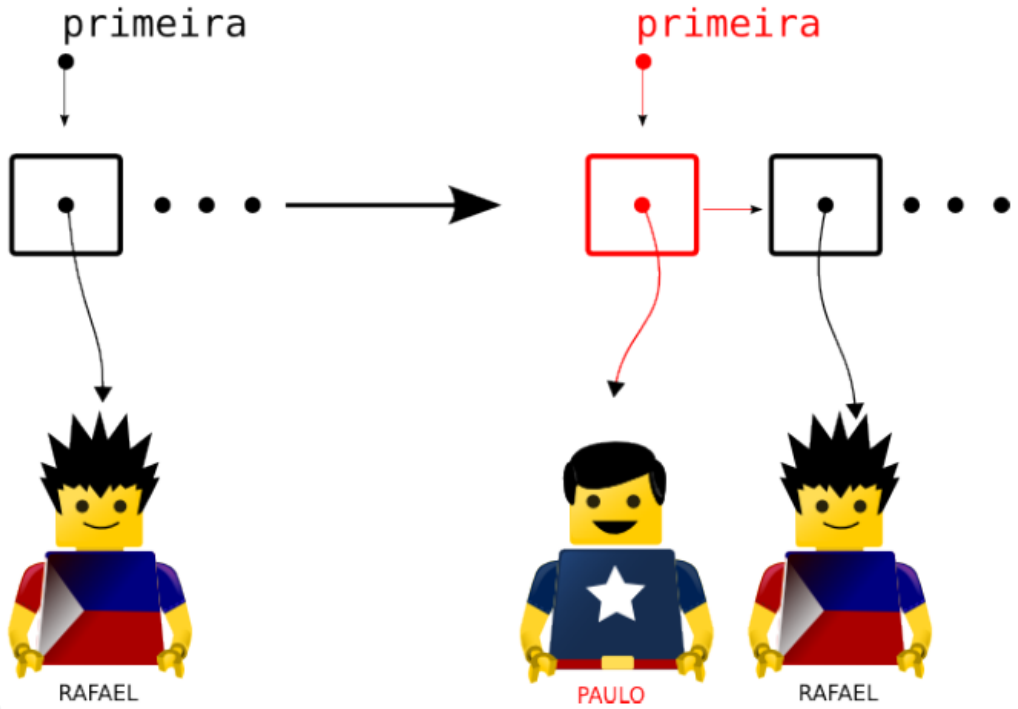


3.1. Adicionando no COMEÇO da Lista

- Exemplo de inserção em lista **não vazia**:

3.1. Adicionando no COMEÇO da Lista

- Exemplo de inserção em lista **não vazia**:



3.1. Adicionando no COMEÇO da Lista

```
public void adicionaNoInicio(Object elemento) {
    No novo = new No(elemento, this.inicio);
    this.inicio = novo;
    if(this.totalDeElementos==0) {
        this.fim = novo;
    }
    this.totalDeElementos++;
}
```

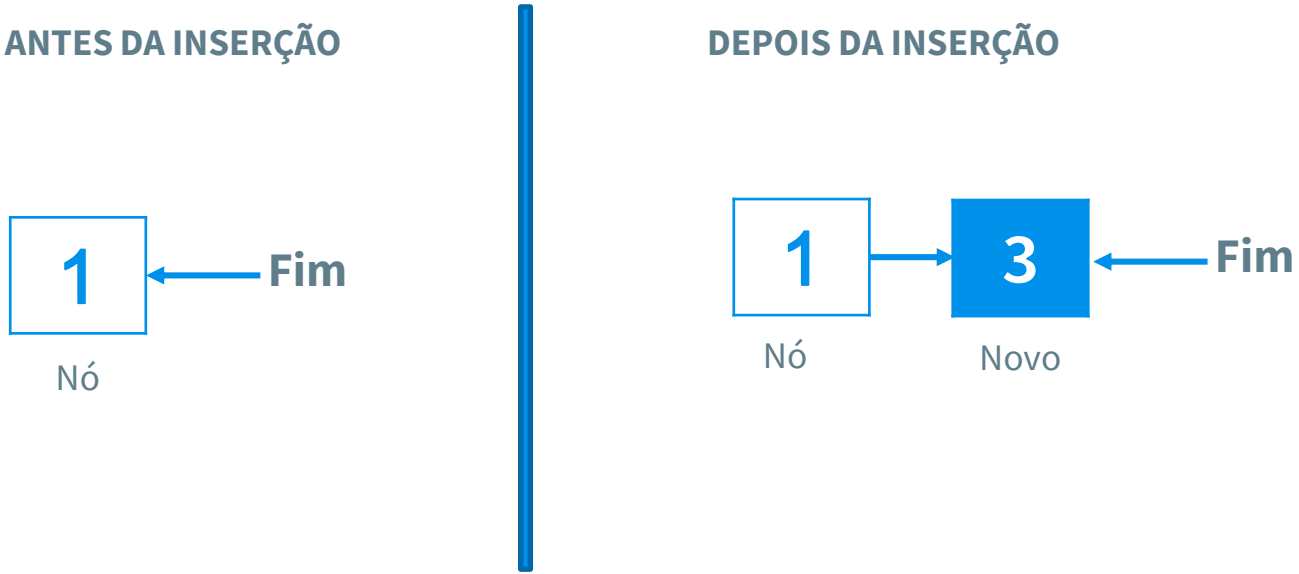
3.2. Adicionando no FIM da Lista

- Se não tivéssemos guardado a referência para a última célula precisaríamos percorrer célula a célula até o fim da Lista para alterar a referência **proximo** da última célula!

- Com um grande número de elementos isso ficaria lento, pois leva tempo linear.

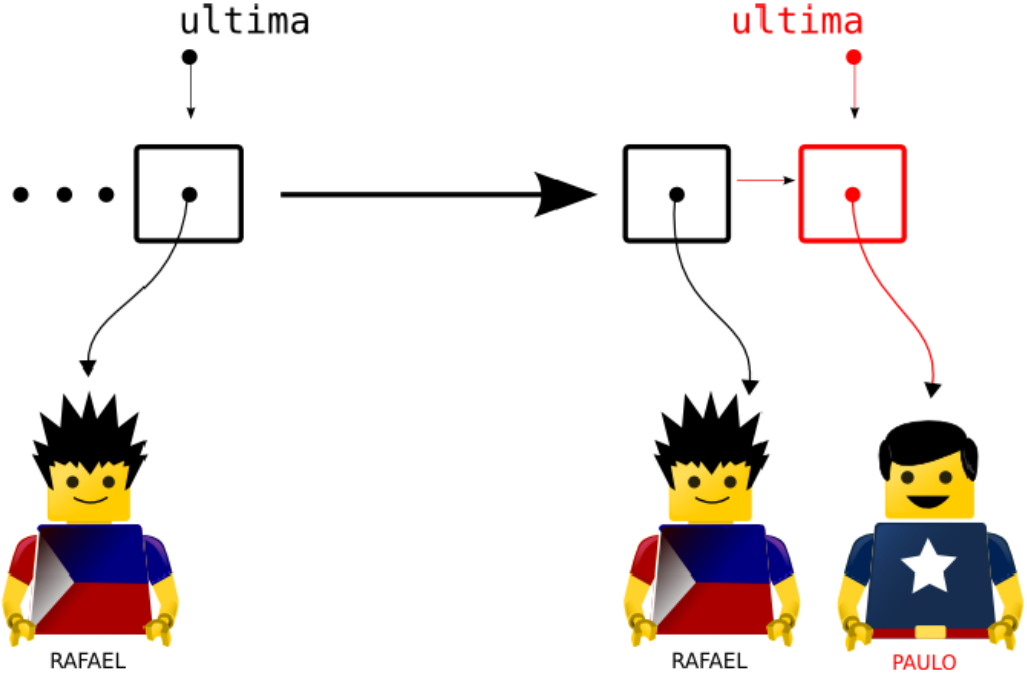
3.2. Adicionando no FINAL da Lista

- Exemplo de inserção em lista **não vazia**:



3.2. Adicionando no FINAL da Lista

- Exemplo de inserção em lista **não vazia**:



3.2. Adicionando no FIM da Lista

- Como este será o novo último elemento, ele não apontará para nenhum próximo. Assim, crio outro método construtor na classe **No** que recebe como parâmetro apenas o elemento.

```
public No(Object elemento, No proximo) {
    this.proximo = proximo;
    this.elemento = elemento;
}

public No(Object elemento) {
    this.elemento = elemento;
}
```

3.2. Adicionando no FIM da Lista

- No caso especial da Lista estar vazia, adicionar no começo ou no fim dessa lista dá o mesmo efeito. Então, se a Lista estiver vazia, chamaremos o método **adicionaNoComeco(Object)**.

3.2. Adicionando no FINAL da Lista

```
public void adicionaNoFinal(Object elemento) {
    if(this.totalDeElementos==0) {
        this.adicionaNoInicio(elemento);
    }else {
        No novo = new No(elemento);
        this.fim.setProximo(novo);
        this.fim = novo;
        this.totalDeElementos++;
    }
}
```

3.2. Adicionando no FIM da Lista

- Para testarmos os métodos **adicionaNoFinal()** e **adicionaNoComeco()**, por enquanto, podemos gerar um método **toString()** de forma automática pelo eclipse na classe **ListaEncadeada**.

- No próximo passo veremos como percorrer a lista inteira.

3.3. Percorrendo a lista

- Para poder rodar alguns testes, precisamos imprimir o conteúdo da nossa Lista.

- Como ainda não temos como acessar um elemento pela sua posição, vamos reescrever o método **toString()** da classe **ListaEncadeada** para que ele concatene todos os elementos de dentro da Lista Ligada em uma única String.

3.3. Percorrendo a lista

- O nosso método utiliza-se de uma referência temporária para um nó chamado **atual**, que vai apontando para o próximo a cada iteração, concatenando o elemento desse nó.

- Iremos utilizar a classe **StringBuilder** que é muito útil para criar Strings potencialmente grandes, em vez de concatenar Strings pelo operador **+”texto da string”**.

3.3. Percorrendo a lista

```
@Override
public String toString() {
    if(this.totalDeElementos==0) {
        return "[]";
    }
    StringBuilder builder = new StringBuilder("[");
    No atual = this.inicio;
    for(int i=0; i<this.totalDeElementos-1; i++) {
        builder.append(atual.getElemento());
        builder.append(", ");
        atual = atual.getProximo();
    }
    builder.append(atual.getElemento());
    builder.append("]");
    return builder.toString();
}
```

3.4. Adicionando em qualquer posição da lista

- Para inserir um elemento em qualquer posição precisamos pegar o **nó anterior ao da posição desejada**, porque precisamos mexer na sua referência **proxima**.

- Para isso vamos criar um método auxiliar que **pega determinado nó**. Este método deve tomar cuidado com o caso da posição não existir. A verificação se a posição existe ou não é feita pelo método **posicaoValida(int).**

3.4. Adicionando em qualquer posição da lista

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

private No pegaNo(int posicao) {
    if(!this.posicaoValida(posicao)) {
        throw new IllegalArgumentException("Posição inválida!");
    }
    No atual = this.inicio;
    for(int i=0; i < posicao; i++) {
        atual = atual.getProximo();
    }
    return atual;
}
```

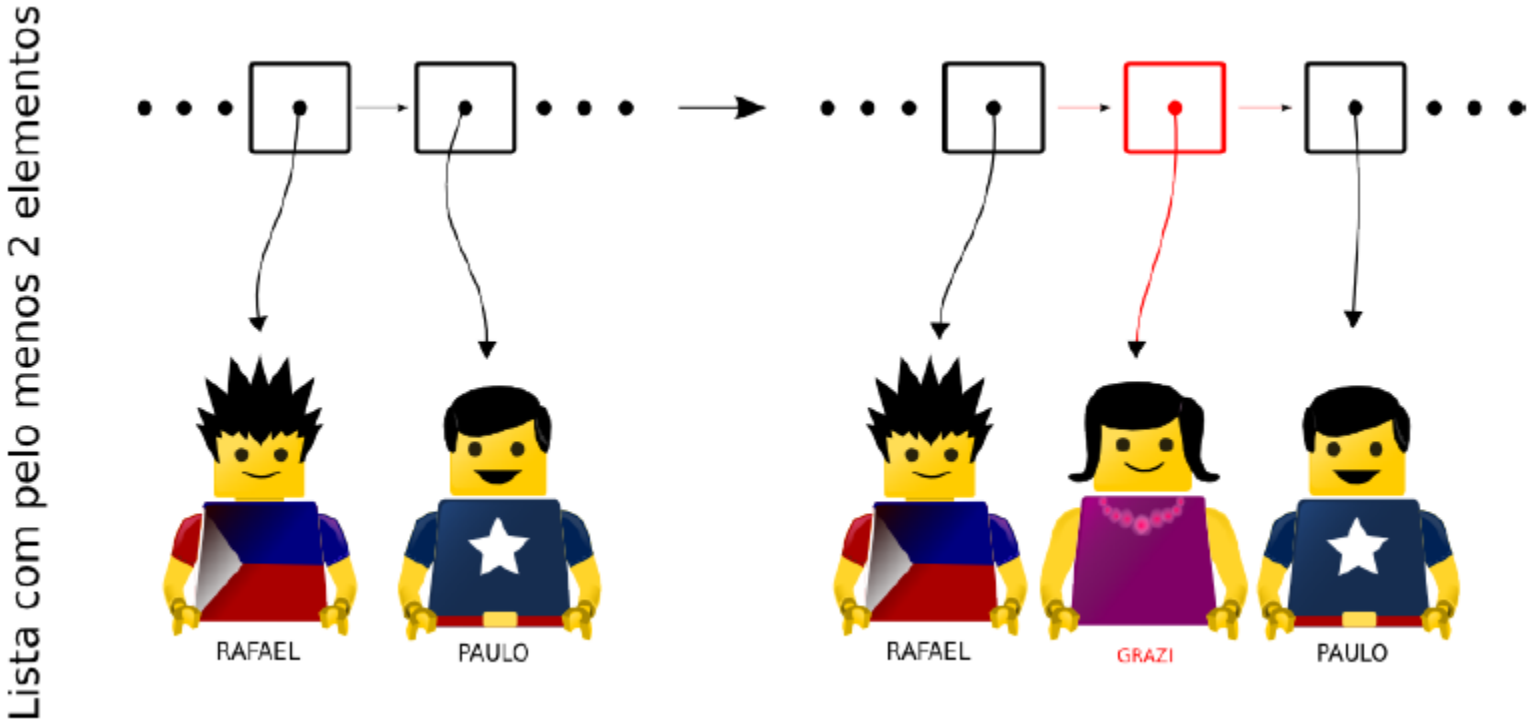
3.4. Adicionando em qualquer posição da lista

- Agora o método **adiciona(int, Object)** fica simples de ser feito.

- Basta pegar o nó anterior à posição onde a inserção será feita e atualizar as referências.

- O anterior deve apontar para um novo nó e o novo nó deve apontar para o antigo próximo do anterior.

3.4. Adicionando em qualquer posição da lista



3.4. Adicionando em qualquer posição da lista

```
public void adiciona(int posicao, Object elemento) {
    if(posicao==0) {
        this.adicionaNoInicio(elemento);
    }else if(posicao==this.totalDeElementos-1) {
        this.adicionaNoFinal(elemento);
    }else {
        No anterior = this.pegaNo(posicao-1);
        No novo = new No(elemento, anterior.getProximo());
        anterior.setProximo(novo);
        this.totalDeElementos++;
    }
}
```

CALMA, RESPIRA!

Obrigado!

Perguntas?

 heraldo.junior@ifsertao-pe.edu.br

 heraldolimajr.com.br