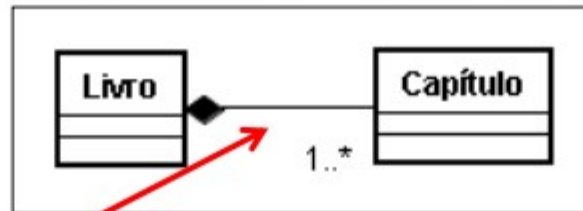


Relacionamento entre objetos através de Listas

Recapitulando....



Os atributos são derivados dos relacionamentos. Não existem no diagrama

```
public class Livro {  
    private Capítulo[] capitulos;  
  
    public Livro(int qtdCapitulo){  
        capitulos = new Capítulo[qtdCapitulo];  
    }  
}
```


```
public class Capítulo {  
    private Livro livro;  
  
    /* Definição da classe Capítulo */  
}
```

Referência pode ou não ser bidirecional. Capítulo não precisa ter o atributo livro

Fonte: Prof. Bruno Gomes

Relacionamento com Listas

- **Uma olhadinha em arrays**

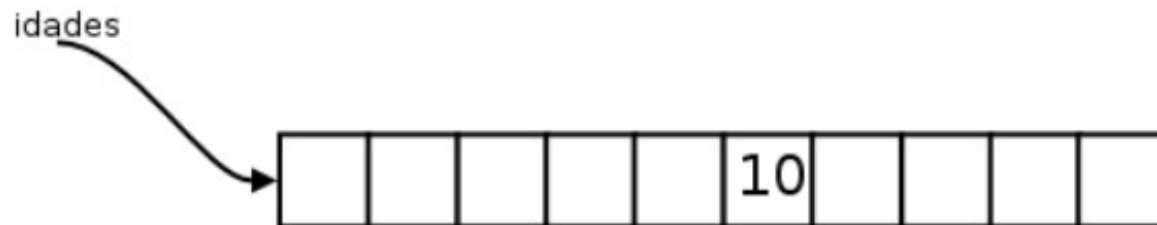
- `int idade1;`
 - `int idade2;`
 - `int idade3;`
 - `int idade4;`
-  `int [] idades;`

- O `int[]` é um tipo
- Um array é um objeto, portanto, a variável `idades` é uma referência
- É necessário criar um objeto para poder usar o array

Relacionamento com Listas

- **Uma olhadinha em arrays**

- `idades = new int[10];`
- array de inteiros com 10 posições, e o endereço no qual foi criado
- `idades[5] = 10`



Fonte: [//www.caelum.com.br/apostilas/](http://www.caelum.com.br/apostilas/)

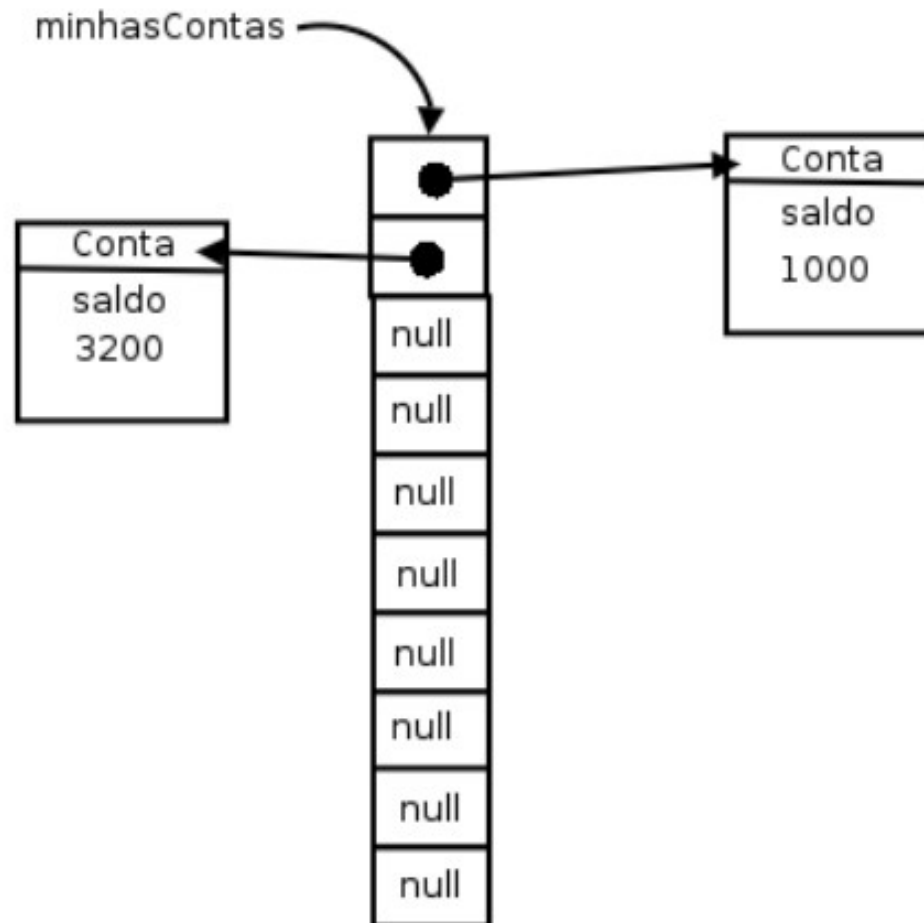
Uma olhadinha em arrays

- **Arrays de referência**

- É comum utilizar o termo **array de objetos**
- Porém um array para classe possui referências
- O objeto está na memória principal e, na sua array, só ficam guardadas as referências (endereços)

```
Conta[] minhasContas;  
minhasContas = new Conta[10];
```

- Quantas contas foram criadas aqui? Na verdade, nenhuma. Foram criados 10 espaços para guardar uma referência a uma Conta
- Por enquanto, eles se referenciam para lugar nenhum (null)



Uma array de tipos primitivos guarda valores, uma array de objetos guarda referências.

Fonte: [//www.caelum.com.br/apostilas/](http://www.caelum.com.br/apostilas/)

Uma olhadinha em arrays

- **Percorrendo arrays**

```
int[] idades = new int[10];  
for (int i = 0; i < 10; i++) {  
    idades[i] = i * 10;  
}  
for (int i = 0; i < 10; i++) {  
    System.out.println(idades[i]);  
}  
}
```

Uma olhadinha em arrays

- **Percorrendo arrays**

```
void imprimeArray(int[] array) {  
    for (int i = 0; i < ???; i++)  
        System.out.println(array[i]);  
}
```

```
void imprimeArray(int[] array) {  
    for (int i = 0; i < array.length; i++)  
        System.out.println(array[i]);  
}
```

A partir do momento que uma array foi criada, não pode mudar de tamanho. Se houver necessidade de mais espaço, será necessário criar uma nova array e, antes de se referir ela, copiar os elementos da array velha

Uma olhadinha em arrays

- **Percorrendo arrays**

- No caso de não haver necessidade de manter uma variável com o índice que indica a posição do elemento no array (que é uma grande parte dos casos), é possível usar o enhanced-for (foreach)

```
int[] idades = new int[10];  
    for (int i = 0; i < 10; i++)  
        idades[i] = i * 10;
```

```
void imprimeArray(int[] array) {  
    for (int x : array)  
        System.out.println(x);  
}
```

```
// imprimindo o array  
for (int x : idades)  
    System.out.println(x);
```

Problemas com arrays????

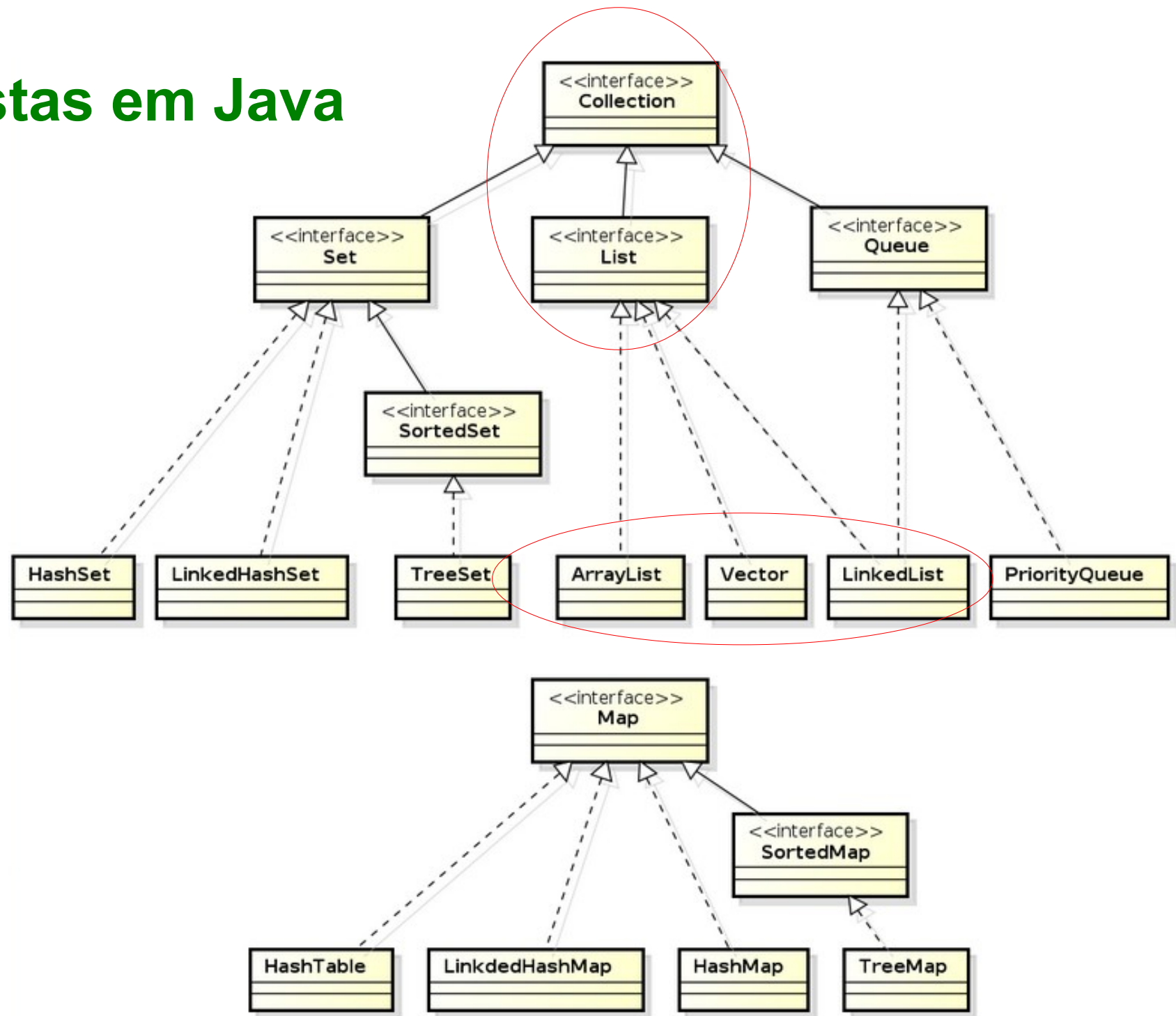
Uma olhadinha em arrays

- **Arrays – Problemas????**

- não é possível redimensionar um *array* em Java
- não é possível buscar diretamente por um determinado elemento cujo índice não se sabe
- não é possível saber quantas posições do *array* já foram populadas sem criar, para isso, métodos auxiliares

Para evitar esses problemas e trabalhar de uma maneira mais robusta com estruturas de dados básicas é disponibilizado um conjunto de classes e interfaces conhecido como **Collections Framework**, que reside no pacote `java.util`

Listas em Java



Listas em Java

- **Collections**

- Collection<E> - Coleções

- Esta é a interface raiz do *framework*, embora não seja fornecida implementações diretas a esta interface ela serve de denominador comum para as implementações de coleções(exceto Map), fornecendo o máximo de generalização possível já que as coleções podem conter vários tipos de restrições, como por exemplo não aceitarem objetos repetidos, ou manterem os objetos em uma determinada ordem.

Listas em Java

- **Collections**

- Operações

- Add(E e): Adiciona o objeto passado por parâmetro a coleção
- addAll(Collection<? extends E> c): Adiciona todos os elementos da coleção passada por parâmetro a na coleção que chamou o método
- remove(Object o): Remove o objeto passado por parâmetro da coleção
- removeAll(Collection<?> c): Remove todos os elemento da coleção passada por parâmetro da coleção que chamou o método
- contains(Object o): Retorna true se o objeto passado por parâmetro estiver dentro da coleção, senão retorna false

Lista em Javas

- **Collections**

- Operações

- isEmpty(): Retorna true se a coleção não tiver nenhum objeto dentro dela.
 - size(): Retorna o número de objetos da coleção.
 - clear(): Remove todos os objetos de dentro da coleção.
 - toArray(): Retorna um array contendo todos os objetos da coleção.
 - toArray(T[] a): Retorna um array contendo todos os objetos da coleção. Se os objetos da coleção couberem no array passado por parâmetro este será retornado, senão um novo array será criado e retornado.
 - iterator(): Retorna um objeto Iterator utilizado para percorrer os objetos da coleção. Este método é especificado na interface Iterable a qual Collection estende

Listas em Java

- **Lists**

- List<E> – Listas

Em um conjunto do tipo List, os objetos são organizados um após o outro em sequência, temos o 1º objeto, o 2º, 3º, etc. Como em um *array* há uma ordem bem definida no armazenamento (sendo uma coleção ordenada), mas com a vantagem de não ter um tamanho fixo. Como em um *array* esta coleção dá muita importância ao índice ou seja a posição que ele ocupa na coleção, fornecendo vários métodos relacionados a ele

Listas em Java

- **Lists**

- Operações

- `get(int index)`: Retorna o objeto armazenado na posição informada pelo parâmetro passado.
 - `indexOf(E e)`: Retorna um `int` para a posição da primeira vez que o objeto passado por parâmetro aparece na coleção. Ele retornará `-1` se o objeto não for encontrado.
 - `add(E e)`: Insere o objeto passado por parâmetro no final da lista
 - `add(int index , E e)`: Insere um o objeto passado e na posição `index` da lista
 - `remove(int index)`: Remove o objeto que está na posição `index`.

Listas em Java

- **ArrayList e LinkedList**

- Basicamente a diferença entre as classes ArrayList e LinkedList é a forma que as duas armazenam seus itens

- **ArrayList**

- Um ArrayList armazena seus itens com um array só que de forma dinâmica

Listas em Java

- **ArrayList – Exemplo**

```
import java.util.ArrayList;
```

```
import java.util.Iterator;
```

```
public static void main(String[] args) {
```

```
    ArrayList<String> c = new ArrayList<>(); //criando um ArrayList só de Strings
```

```
    c.add("Rodrigo"); //adicionando ao final da lista, posição 0
```

```
    c.add("Maria"); //adicionando ao final da lista, posição 1
```

```
    c.add("Thiago"); //adicionando ao final da lista, posição 2
```

```
    c.add("João"); //adicionando ao final da lista, posição 3
```

Listas em Java

- **ArrayList – Exemplo**

//pegando o a quantidade de objetos - saída: Tamanho: 4

```
System.out.println("Tamanho: " + c.size() );
```

//pegando um objeto especifico - saída: A string na posição 2: Thiago

```
System.out.println("A string na posição 2: " + c.get(2) );
```

//pegando a posição de um determinado objeto - saída: A posição da String 'Maria': 1

```
System.out.println("A posição da String 'Maria': " + c.indexOf("Maria") );
```

//removendo a String na posição 1, ou seja "Maria"

```
c.remove( 1 );
```

Listas em Java

- **ArrayList – Exemplo**

//percorrendo através de um iterator(objeto responsável por percorrer uma coleção)

```
Iterator<String> i = c.iterator(); // pegando o iterator da coleção
```

```
while( i.hasNext() ){ //enquanto tiver um próximo objeto na coleção
```

```
//retorna o próximo elemento da coleção e incrementa o contador interno do iterator
```

```
String nome = i.next();
```

```
System.out.println( nome );
```

```
}
```

Listas em Java

- **ArrayList – Exemplo**

```
// adicionando na posição 1 a String "Mario", sendo que a o objeto que está  
// na posição 1 passa para a posição 2, o objeto na posição 2 para 3 e assim  
// por diante
```

```
c.add(1, "Mario");
```

```
//percorrendo a lista através de um "for aprimorado"
```

```
//para cada objeto presente na lista "c" o objeto será colocado em "nome" e  
executará o bloco
```

```
for(String nome : c){
```

```
System.out.println( nome );
```

```
}
```

Listas em Java

- **ArrayList – Exemplo**

//criando um array com o mesmo tamanho dos objetos da coleção

```
String[] nomes = new String[ c.size() ];
```

//colocando os objetos da coleção dentro do array passado por parâmetro

```
c.toArray( nomes );
```

//percorrendo o array

```
for(int j = 0 ; j < nomes.length ; j++){
```

```
    System.out.println(nomes[j]);
```

```
}
```

Listas em Java

- **ArrayList – Exemplo**

//criando um array com o mesmo tamanho dos objetos da coleção

```
String[] nomes = new String[ c.size() ];
```

//colocando os objetos da coleção dentro do array passado por parâmetro

```
c.toArray( nomes );
```

//percorrendo o array

```
for(int j = 0 ; j < nomes.length ; j++){
```

```
    System.out.println(nomes[j]);
```

```
}
```

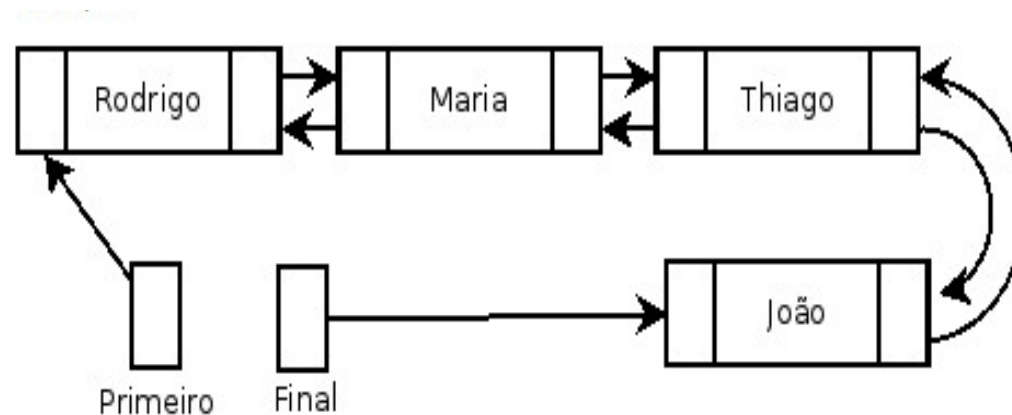
Listas em Java

- **LinkedList**

- A LinkedList utiliza uma abordagem diferente para armazenar os seus objetos
- Utiliza uma lista duplamente ligada onde cada um dos elementos adicionado é colocado dentro de outro objeto(Nó), que além de conter o valor armazenado tem uma referência ao próximo Nó e ao anterior
- Com isso cada Nó pode ser colocado em qualquer lugar da memória e não somente um após ao outro como na implementação de ArrayList

Listas em Java

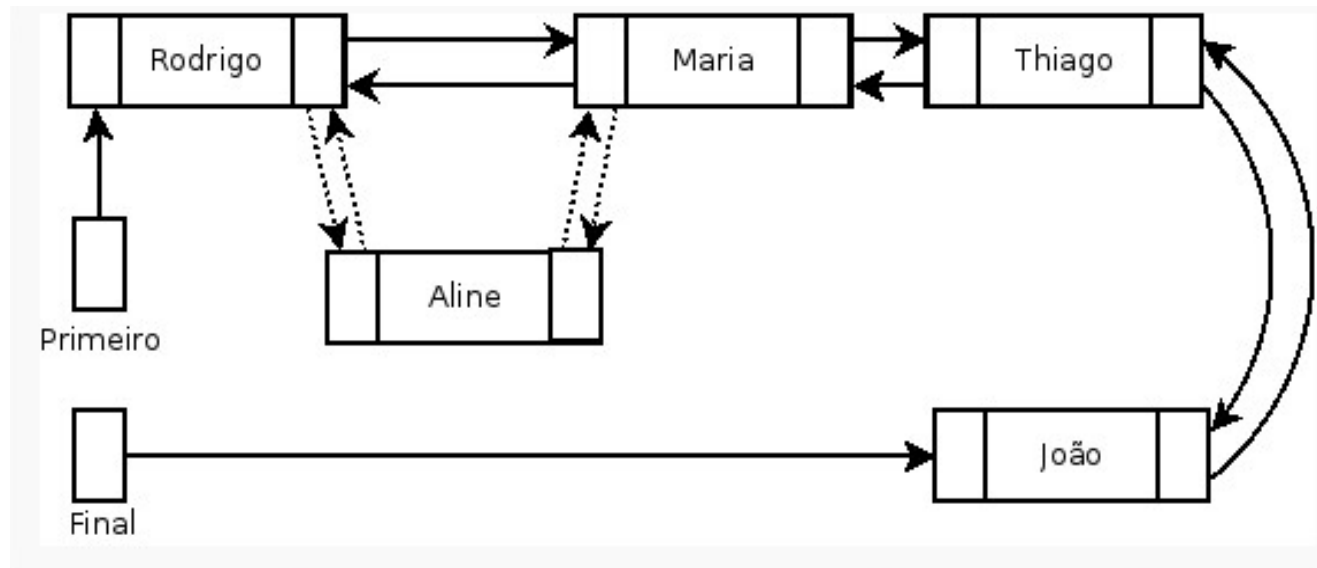
- **LinkedList**



- Em uma LinkedList para inserir no inicio da lista ou no meio não é necessário mover todos os objetos da memória uma posição para frente, basta alterar as referências ao próximo Nó e ao anterior, tornando as operações de inserção e remoção mais eficientes principalmente no inicio e no final

Listas em Java

- LinkedList



Listas em Java

- **LinkedList**

- Operações

- `addFirst(E e)`: Insere o objeto `e` no início da lista
 - `addLast(E e)`: Insere o objeto `e` no final da lista
 - `getFirst()`: Retorna o primeiro objeto da lista
 - `getLast()`: Retorna o último objeto da lista
 - `removeFirst()`: Remove e retorna o primeiro objeto da lista
 - `removeLast()`: Remove e retorna o último objeto da lista

Listas em Java

- **LinkedList**

- Exemplo

```
// Cria uma LinkedList de String
```

```
LinkedList<String> lista = new LinkedList<>()
```

```
// adiciona três elementos na lista
```

```
lista.add("Cuiabá");
```

```
lista.add("Goiânia");
```

```
lista.add("Belo Horizonte");
```

```
// obtém um ListIterator para percorrer toda a lista, começando no  
//primeiro elemento
```

```
ListIterator<String> iterador = lista.listIterator(0);
```

```
while(iterador.hasNext()){
```

```
    String cidade = iterador.next();
```

```
    System.out.println(cidade);
```

```
}
```

Listas em Java

- **Para pensar...**
- Para criar um ArrayList , basta chamar o construtor:

```
ArrayList lista = new ArrayList();
```
- É possível abstrair a lista a partir da interface List?
 - ```
List lista = new ArrayList();
```