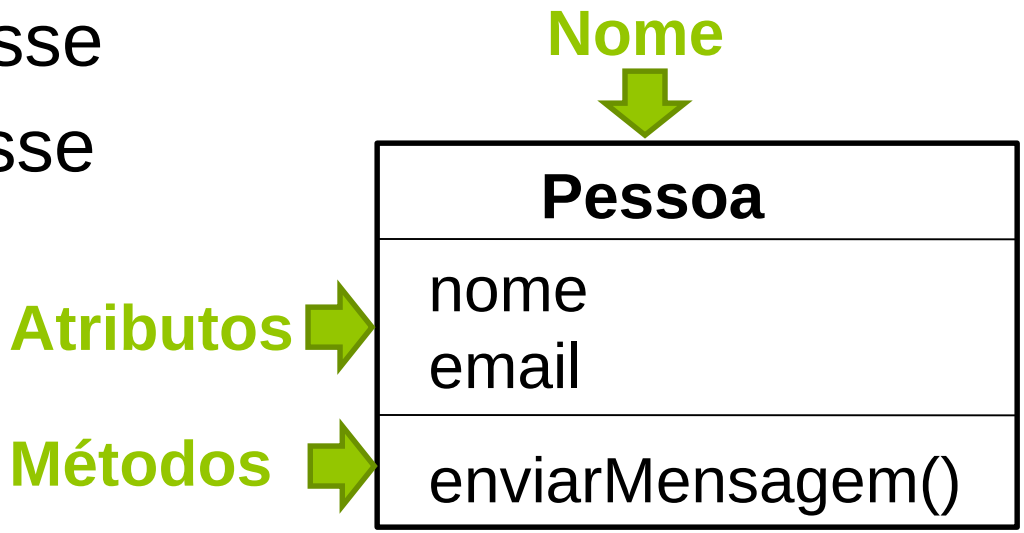
Programação Orientada a Objetos

Aula 06 – Associação entre Classes e ArrayList

Prof. Luiz Mário Lustosa Pascoal

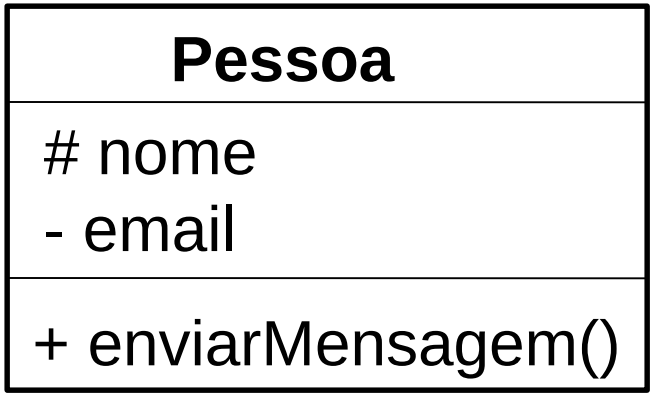
Representação de uma Classe

- Uma classe é representada por um retângulo com três divisões:
  - Nome da Classe
  - Atributos da Classe
  - Métodos da Classe



Tipos de visibilidade

- Pública (+)
  - O atributo ou método pode ser utilizado por qualquer classe
- Protegida (#)
  - Somente a classe ou sub-classes terão acesso
- Privada (-)
  - Somente a classe terá acesso



Associação entre classes

- Objetos diferentes podem se conhecer, podem conversar uns com os outros

- **Mas como objetos podem conversar?**
  - Imagine 2 pessoas. Para elas conversarem, eles devem se conhecer antes. Logo, para dois objetos conversarem, eles devem se *conhecer*
  - Vamos imaginar, então, que a Floribela quer jogar o dado
  - Dois objetos diferentes, de duas classes diferentes
  - A classe Pessoa possui um método que chama-se *jogarDado()*

Associação entre classes

- **Mas como a Floribela pode acessar este método?**
- Somente quando ela *conhecer* o objeto dado, ou seja, quando ela tiver uma **instância da classe Dado**
- Assim, vamos poder dizer que uma pessoa tem um dado

Associação entre classes

- Mais do que isso, uma Pessoa (classe) tem um atributo do tipo Dado (classe), e este atributo (um objeto) deve ser declarado e inicializado
- Logo, na classe Pessoa teremos:

```
public class Pessoa
{
    private String nome;
    ...
    private Dado meuDado;
    ...
}
```

Declaração de uma instância da classe Dado, associando a classe Dado à classe Pessoa

Associação entre classes

- Para que este atributo se torne utilizável, devemos inicializá-lo, informando ao Java que trata-se de um novo objeto
- Assim, devemos utilizar o new, conforme dito anteriormente
- No construtor de Pessoa, teremos então:

```
public Pessoa()
{
    ...
    meuDado = new Dado(6);
    ...
}
```

Inicialização do objeto

Construtor da classe Dado pede o número de lados!!

Associação entre classes

- **Uma pessoa não precisa ter necessariamente 1 dado**

```
public class Pessoa
{
    private Dado meuDado1;
    private Dado meuDado2;
    private Dado meuDado3;
    ...

    public Pessoa(...)
    {
        meuDado1 = new Dado(6);    //cria um dado de 6 lados
        meuDado2 = new Dado(13);  //cria um dado de 13 lados
        meuDado3 = new Dado(6);    //cria outro dado de 6 lados
        ...
    }
}
```

Associação entre classes

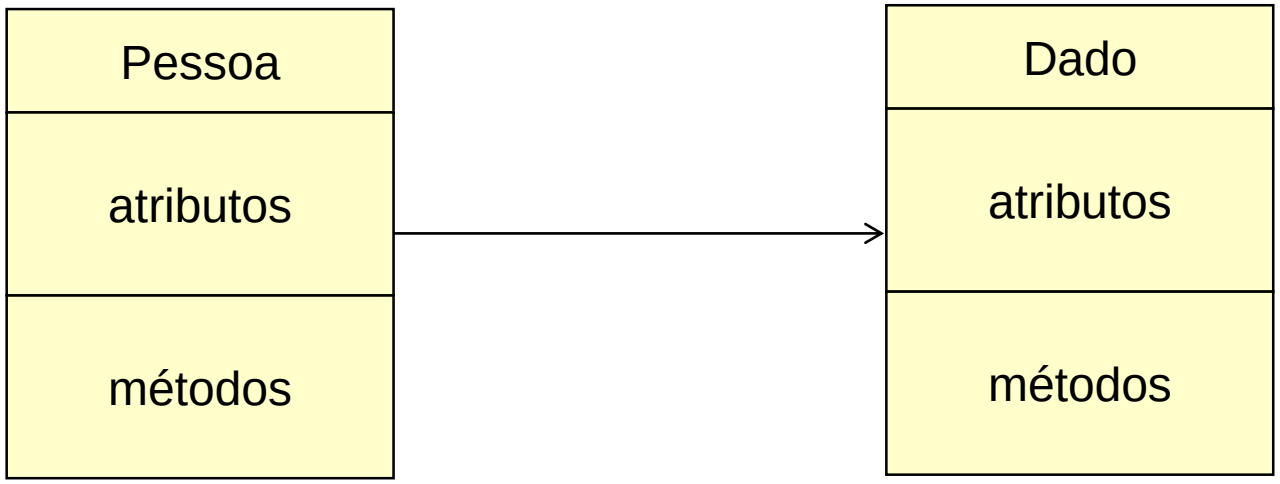
- **A pessoa então, pode jogar o dado de acordo usando o método jogarDados() que devolve um valor aleatório (usando a classe Random) entre 1 e o número de Lados do Dado.**

```
public Pessoa(...)
{
    meuDado = new Dado(6);    //cria um dado de 6 lados
.
}

public int jogarDados()
{
    Random rd = new Random();
    return rd.nextInt(meuDado.getnLados()) + 1 ;
}
```

Associação entre classes

- **O diagrama UML ilustra o relacionamento entre as classes da seguinte maneira:**

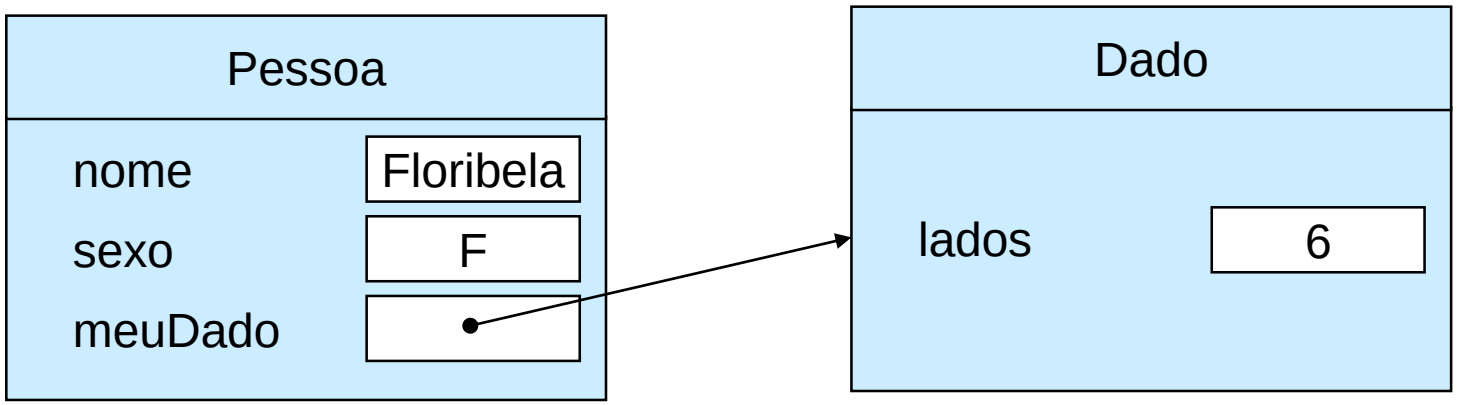


O objeto da classe Pessoa conhece o objeto da classe Dado,
mas o contrário não é verdade

Associação dentre classes

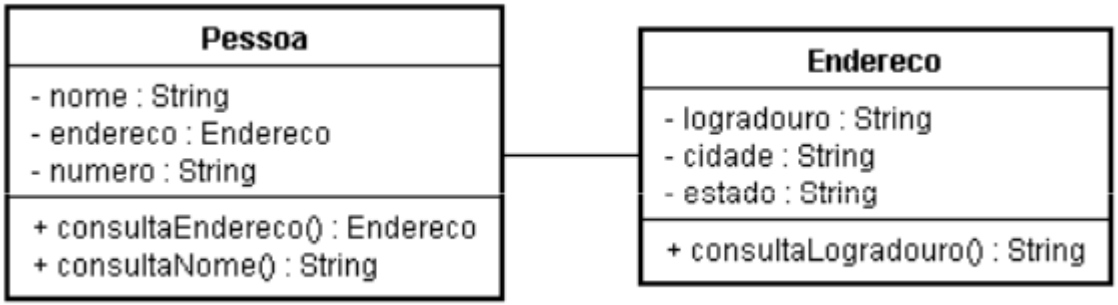
- Uma associação é um relacionamento estrutural entre duas classes, significando que os objetos de uma classe estão ligados aos objetos de outra

- O objeto pessoa está ligado ao objeto dado



Exercício para praticar

- Desenvolver o seguinte relacionamento, onde pessoa terá um objeto do tipo Endereço.



- Defina a classe Principal instanciando pelo menos duas Pessoas e imprimindo todas as suas informações.

Relacionamento

- Classes possuem relacionamentos entre elas
  - Compartilham informações
  - Colaboram umas com as outras
- Principais tipos de relacionamentos
  - Associação
  - Agregação / Composição
  - Herança
  - Dependência

Comunicação entre Objetos (I)

- Conceitualmente, objetos se comunicam através da troca de mensagens.
- Mensagens definem:
  - O nome do serviço requisitado
  - A informação necessária para a execução do serviço
  - O nome do requisitante.

Comunicação entre Objetos (II)

- Na prática, mensagens são implementadas como chamadas de métodos
  - Nome = o nome do método
  - Informação = a lista de parâmetros
  - Requisitante = o método que realizou a chamada

Associações

- Descreve um vínculo entre duas classes
  - Chamado **Associação Binária**

- Determina que as instâncias de uma classe estão de alguma forma ligadas às instâncias da outra classe

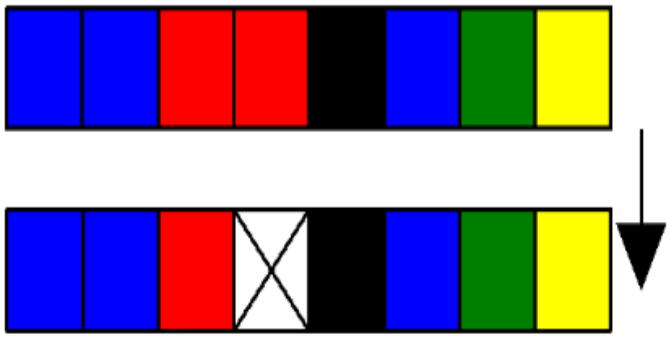
Multiplicidade

0..1	No máximo um. Indica que os Objetos da classe associada não precisam obrigatoriamente estar relacionados.
1..1	Um e somente um. Indica que apenas um objeto da classe se relaciona com os objetos da outra classe.
0..*	Muitos. Indica que podem haver muitos objetos da classe envolvidos no relacionamento
1..*	Um ou muitos. Indica que há pelo menos um objeto envolvido no relacionamento.
3..5	Valores específicos.

ArrayList

Relembrando...

- Conforme vimos anteriormente trabalhar com Arrays é cansativo.
  - Impossibilidade de redimensionar o array.
  - Impossibilidade de buscar diretamente um elemento sem conhecimento do índice.
  - Impossibilidade de saber quantas posições do array estão ocupadas.



Retire a quarta Conta

conta[3] = **null**;

Introdução

- O que é uma coleção?
  - Um objeto que agrupa múltiplos elementos em uma única unidade
- Usadas para armazenar, recuperar e manipular elementos que formam um grupo natural.
- Ex.: Vector, Hashtable, array (JDK 1.1)

Collections Framework

- Conjunto de interfaces, implementações e algoritmos
- Reside no pacote java.util desde o Java2 1.2.
- Vantagens:
  - Reduz esforço de programação
  - Aumenta velocidade e qualidade na programação
  - Permite interoperabilidade entre API´s
  - Reduz esforço para aprender uma nova API
  - Reduz esforço para criar uma nova API
  - Promove reuso

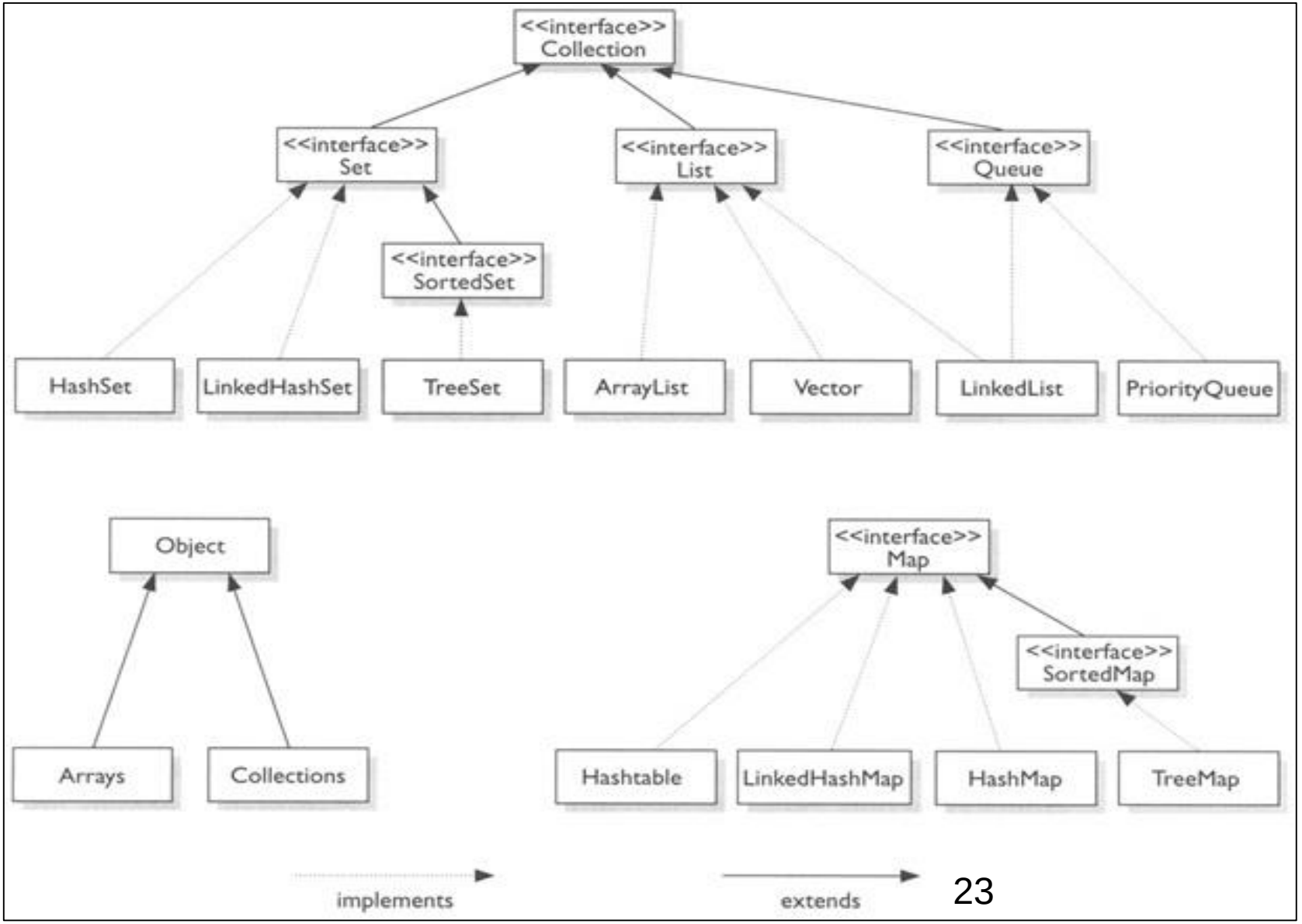
Collections

- A **API** do **Collections** é robusta e possui diversas classes que representam estruturas de dados avançadas.

- É a interface absoluta na hierarquia de coleções. Dela descendem as interfaces **Set** (conjunto), **Queue** (filas) e **List** (listas) que formam a base das coleções genéricas da linguagem Java.

  - **Set** – define uma coleção que não contém valores duplicados.
  - **Queue** – define uma coleção que representa uma fila, ou seja, implementa o modelo FIFO (First-In, First-Out)
  - **List** - define uma coleção ordenada que pode conter elementos duplicados.

Classes e Interfaces presentes na Collection



Interface List
Listas: java.util.List

- Um primeiro recurso que a API de Collections traz são listas. Uma lista é uma coleção que permite elementos duplicados e mantém uma ordenação específica entre os elementos.

- A implementação mais utilizada da interface **List** é a **ArrayList**, que trabalha com um array interno para gerar uma lista. Portanto, ela é mais rápida na pesquisa do que sua concorrente, a **LinkedList**, que é mais rápida na inserção e remoção de itens nas pontas.

- É comum confundirem uma **ArrayList** com um **array**, porém ela **NÃO É** um array. O que ocorre é que, internamente, ela usa um array como estrutura para armazenar os dados, porém este atributo está propriamente encapsulado e não é possível acessá-lo. Repare, também, que não se pode usar [] com uma ArrayList, nem acessar atributo *length*, pois não há relação!

Declarando um Arraylist

Também é possível abstrair a lista a partir da Interface List:

```
List <String> nomes = new ArrayList <String>();
```

Ou simplesmente:

```
ArrayList <String> nomes = new ArrayList <String> ();
```

Principais Métodos presentes no Arraylist

- Adicionando elementos **(add)**
  - nomes.add(“Joao”);
  - nomes.add(1, “Maria”);

- Removendo elementos **(remove)**
  - nomes.remove(5);
  - nomes.remove(15); //se for passado um elemento do tipo Object elepor posição

- Verificando se está vazia **(isEmpty)**
  - retorna **true** caso esteja vazia ou **false** se houver algum elemento na lista
  - if (nomes.isEmpty()) syso(“A lista está vazia!!”);

- Verifica se contém um elemento específico **(contains)**
  - nomes.contains(“Joao”);

Principais Métodos presentes no Arraylist

- Recupera um elemento **(get)**
  - nomes.get(2);
- Retorna o tamanho da lista **(size)**
  - nomes.size();
- Retorna a posição de um elemento **(indexOf)**
  - nomes.indexOf(“Joao”);
- Troca o elemento em uma determinada posição **(set)**
  - nomes.set(1, “Marcos”);

Percorrendo uma Collection

- **For-each**
  - O For-each é um ciclo for, mas que é adaptado para utilização em Collections e outras listas. Ele serve para percorrer todos os elementos de qualquer Collection contida na API Collections.
  - Sintaxe: **for (tipo elemento : collection)**

Ex.:

```
List<Integer> minhaLista = new ArrayList<Integer>();
minhaLista.add(1);
minhaLista.add(2);
for (Integer listaElementos : minhaLista) {
    System.out.println(listaElementos);
}
```

Percorrendo uma Collection

- **Iterator**
  - O iterator é uma interface disponível no pacote java.util que permite percorrer coleções da API Collection, desde que tenham implementado a Collection, fornecendo métodos como o next(), hasnext() e remove().

```
List<Integer> minhaLista = new ArrayList<Integer>();

minhaLista.add(1);
minhaLista.add(2);

Iterator iMinhaLista = minhaLista.iterator();

for(Integer listaElementos: minhaLista){
    System.out.println(iMinhaLista.next());
}
```

Tipos Abstratos de Dados com Collections

- Toda lista (na verdade, toda Collection) trabalha do modo mais genérico possível. Isto é, não há uma ArrayList específica para Strings, outra para Números, outra para Datas etc. Todos os métodos trabalham com Object.

- Desta forma, é possível criar um ArrayList de uma classe específica, como por exemplo, Pessoa.

```
Pessoa p = new Pessoa (“Joao”);
ArrayList <Pessoa> pessoas = new ArrayList <Pessoa> ();
pessoas.add(p);
```

Links Interessantes para Estudo

- **Documentação Java sobre Collections:**
  - http://docs.oracle.com/javase/1.4.2/docs/api/java/util/Collections.html
- **Apostila Caelum:**
  - http://www.caelum.com.br/apostila-java-orientacao-objetos/collections-framework/#16-2-listas-java-util-list
- **DevMedia:**
  - http://www.devmedia.com.br/visao-geral-da-interface-collection-em-java/25822
  - http://www.devmedia.com.br/api-collections-em-java-fundamentos-e-implementacao-basica/28445