

POO

Programação Orientada a Objetos

Material 009

Professor Maromo





Agenda

- 
- 
- Métodos equals() e hashCode()
 - Conjuntos
 - List
 - ArrayList
 - LinkedList
 - Principais métodos das classes que implementam coleções
 - Genéricos
 - Interface Comparable

Métodos equals() e hashCode()

- Método equals()
 - Serve para determinar se dois objetos são significativamente equivalentes.
- Método hashCode()
 - Sua sobreposição é muito importante quando se trata de conjuntos.
 - Veja a analogia no próximo slide:

Analogia: Elementos guardados em caixa

- Supondo que os elementos sejam nomes e o comprimento do nome determina a caixa em que o nome será guardado.

JUCA

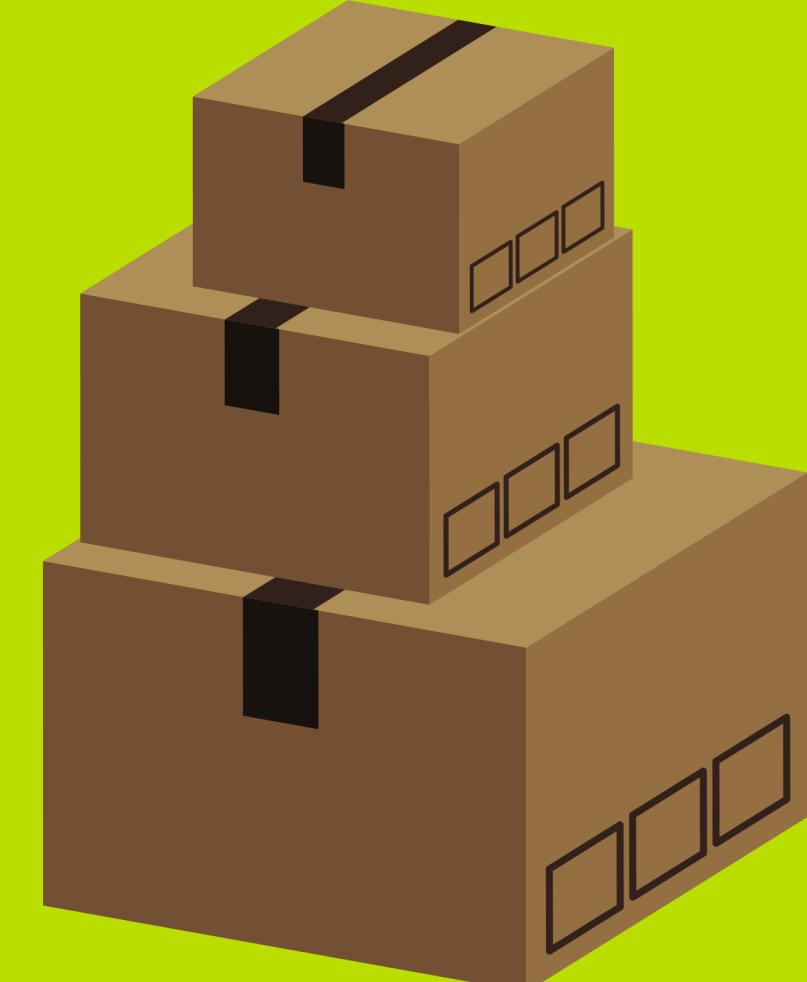
4 letras

PEDRO
OSCAR

5 letras

ANTONIO

7 letras



- Utilizar a quantidade de letras para determinar em que caixa um objeto deve ser armazenado trata-se de um método de hashing.

Analogia: Elementos guardados em caixa

- Quanto mais eficiente ele for, maior quantidade de caixas haverá. Quando se realiza uma busca, ele será tão eficiente quanto for o método de hashing.
- Quando uma busca é realizada em um conjunto que usa hashing, o método hashCode() é aplicado e, a partir do seu retorno, o método equals() é aplicado e, nesse momento, pode-se determinar a igualdade de fato.

JUCA

4 letras

PEDRO
OSCAR

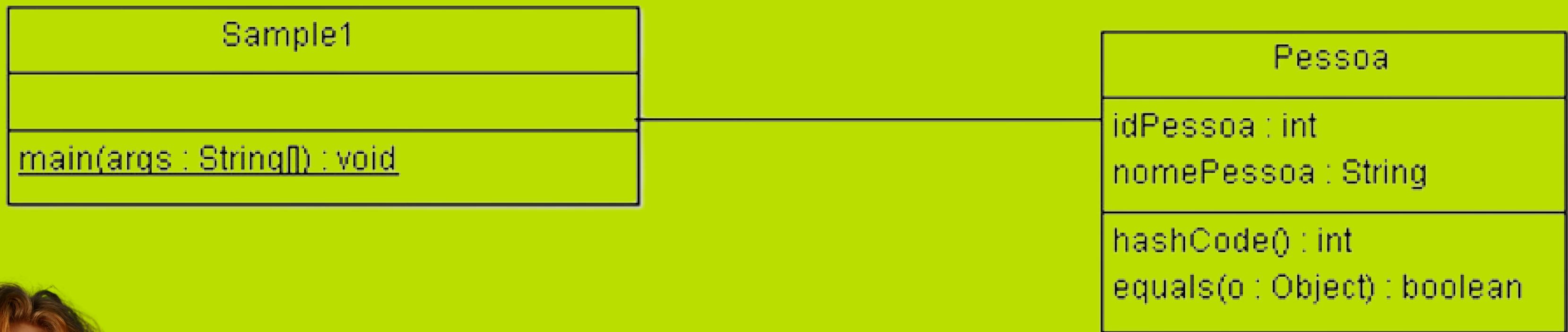
5 letras

ANTONIO

7 letras



Exemplo: prjSample1



Exemplo: prjSample1

```
public class Pessoa {  
    public int idPessoa;  
    public String nomePessoa;  
  
    @Override  
    public int hashCode() {  
        return nomePessoa.length();  
    }  
    @Override  
    public boolean equals(Object o) {  
        if ((o instanceof Pessoa) && ((Pessoa) o).idPessoa == this.idPessoa) {  
            return true;  
        } else {  
            return false;  
        }  
    }  
}
```

Exemplo: prjSample1

```
public class Sample1 {  
    public static void main(String[] args) {  
        Pessoa p1 = new Pessoa();  
        p1.idPessoa =1;  
        p1.nomePessoa = "Oscar";  
        Pessoa p2 = new Pessoa();  
        p2.idPessoa = 1;  
        p2.nomePessoa = "Oscar";  
        Pessoa p3 = new Pessoa();  
        p3.idPessoa = 3;  
        p3.nomePessoa = "Mariana";  
        System.out.println("hashCode de p1: " + p1.hashCode());  
        System.out.println("hashCode de p2: " + p2.hashCode());  
        System.out.println("hashCode de p3: " + p3.hashCode());  
        System.out.println(p1.equals(p2));  
        System.out.println(p2.equals(p3));  
    }  
}
```

Resultado Comentado

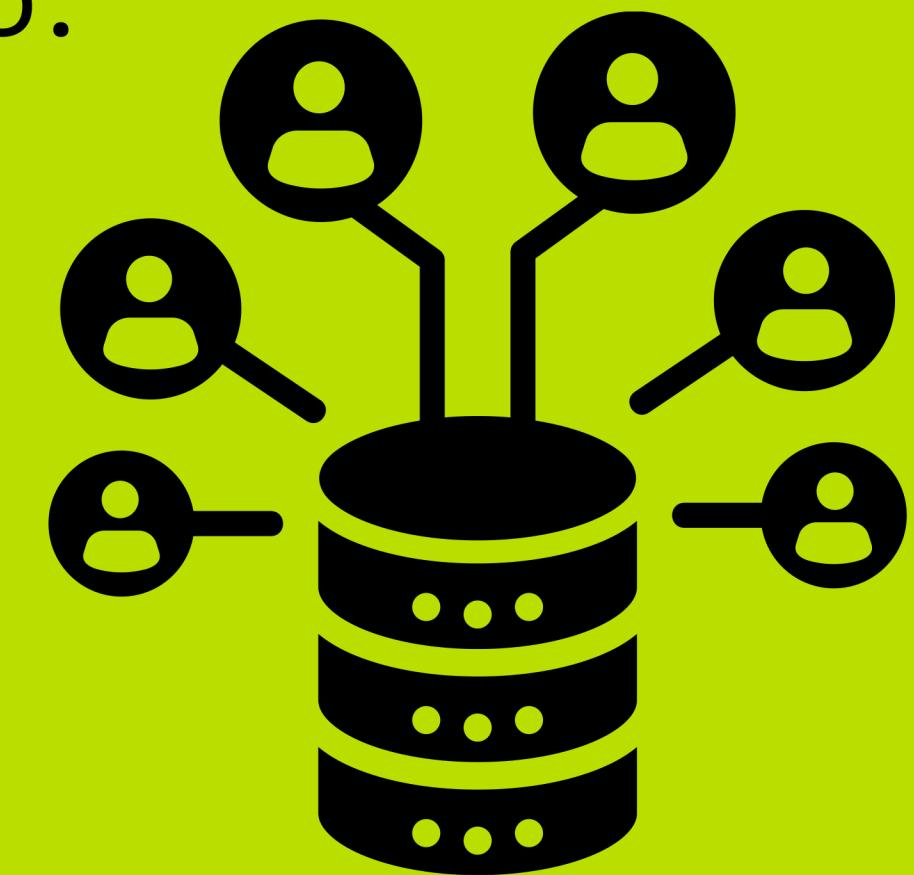
- Sobre o método hashCode() deve ser público e retornar um inteiro. Neste exemplo usamos como retorno o tamanho (quantidade de letras) do nome da pessoa.
- O método equals() retorna verdadeiro quando os objetos são significativamente equivalentes.

hashCode de p1: 5
hashCode de p2: 5
hashCode de p3: 7
true
false

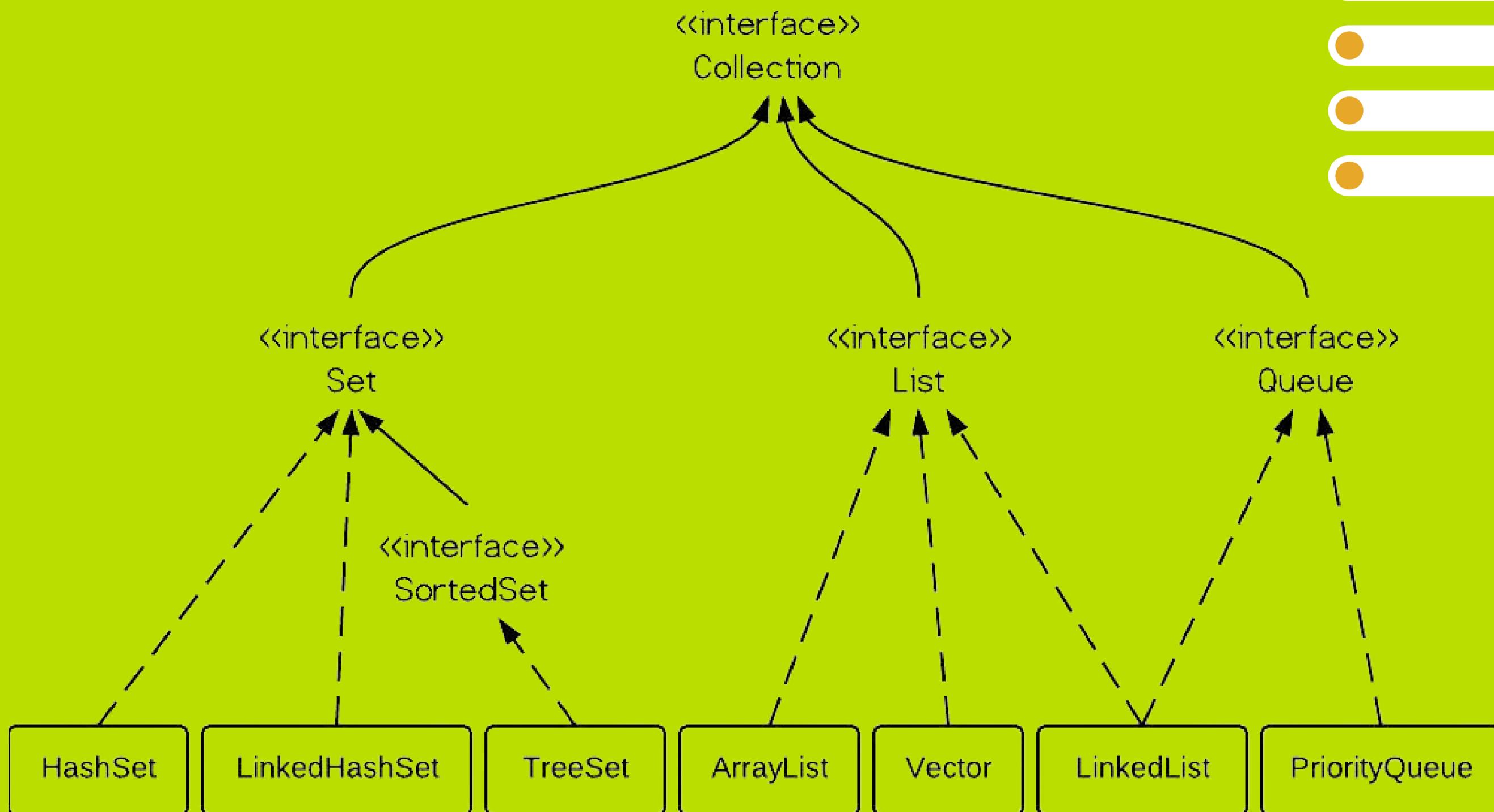


Conjuntos

- Fazem parte de nosso cotidiano.
 - Ex: um carrinho de compras, um conjunto de peças.
- Conjuntos permitem operações como:
 - **Adição,**
 - **Remoção,**
 - **Busca,**
 - **Pesquisa,**
 - **Recuperação e**
 - **Iteração de objetos.**
- Existem diversos tipos de conjuntos, cada um com um propósito específico.



Collection (Conjuntos)



Fonte: <http://www.programcreek.com/wp-content/uploads/2009/02/java-collection-hierarchy.jpeg>

List <<interface>>

- Classes que implementam a interface List relevam o índice; com isso podemos inserir, por exemplo, um item no meio da lista.
- Características:
 - Ordenadas por meio de um índice;
 - Uma espécie de sequencia de armazenamento de objetos.
 - Tipos mais comuns de implantação: ArrayList e LinkedList. Tipo em desuso – Vector.

ArrayList

- Trata-se de uma estrutura de dados que tem como base um array. No entanto um tipo de array que pode ser alterado.
- Características:
 - Acesso sequencial / aleatório rápido.
 - Em função do índice, o acesso a um elemento no meio da lista é uma operação rápida para a recuperação de um item.
 - Inserção também é rápida.

Sample2

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class Sample2 {
    public static void main(String[] args) {
        long inicio, fim;
        int n = 2600000;
        inicio = System.currentTimeMillis();
        List array = new ArrayList();
        for(int i=0; i<n;i++){
            array.add(new Integer(i));
        }
        fim = System.currentTimeMillis();
        System.out.println("Tempo para inserir: " + (fim-inicio)/1000.0 + " segundo");
        inicio = System.currentTimeMillis();
        Iterator o = array.iterator();
        while(o.hasNext()){
            Integer x = (Integer)o.next();
        }
        fim = System.currentTimeMillis();
        System.out.println("Tempo para iterar: " + (fim-inicio)/1000.0 + " segundo");
    }
}
```



LinkedList

- Adequado para inserção de elementos no final ou no início, ou seja filas e pilhas.
- Características:
 - Mais lento na iteração do que o ArrayList;
 - Mas será uma boa opção quando se deseja inserir ou remover rapidamente um item na coleção.
 - Lista ordenada, podemos iterar em uma ordem específica, seja ela pela ordem de inserção ou pela ordem do índice.

Sample2

```
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
public class Sample2 {
    public static void main(String[] args) {
        long inicio, fim;
        int n = 2600000;
        inicio = System.currentTimeMillis();
        List array = new LinkedList();
        for(int i=0; i<n;i++){
            array.add(new Integer(i));
        }
        fim = System.currentTimeMillis();
        System.out.println("Tempo para inserir: " + (fim-inicio)/1000.0 + " segundo");
        inicio = System.currentTimeMillis();
        Iterator o = array.iterator();
        while(o.hasNext()){
            Integer x = (Integer)o.next();
        }
        fim = System.currentTimeMillis();
        System.out.println("Tempo para iterar: " + (fim-inicio)/1000.0 + " segundo");
    }
}
```



Observe

- Observe o tempo de inserção e varredura em um ArrayList X LinkedList



Principais métodos das classes que implementam coleções

- **add(Object objeto)** - Adiciona a coleção a um determinado objeto
- **addAll(Collection outraCollection)** – Adiciona todos elementos de outra coleção.
- **clear()** – Limpa todos os elementos de uma coleção.
- **contains(Object objeto)** – Retorna true se o objeto já fizer parte da coleção.
- **containsAll(Collection outraCollection)** – Retorna true caso todos os elementos de outra coleção estiverem presentes em determinada coleção.
- **hashcode()** - Retorna o hashcode do objeto.
- **iterator()** - Retorna o objeto de iteração com os elementos desta coleção.
- **remove(Object objeto)** – Remove o objeto da coleção
- **removeAll(Collection outraCollection)** – Remove todos os elementos que pertençam à coleção corrente e á outra coleção determinada.
- **retainAll(Collection outraCollection)** – Remove todos os elementos que não façam parte da coleção corrente e da outra coleção.
- **Size()** - Retorna a quantidade elementos existentes na coleção.
- **toArray()** - Retorna uma matriz de objetos(Object[]) dos elementos que estão contidos na coleção.
- **toArray([]matriz)** – Retorna uma matriz do fornecido e, se a matriz contiver a quantidade de elementos suficiente, passa a ser utilizada para armazenamento.

Novo Exemplo: Sample3

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

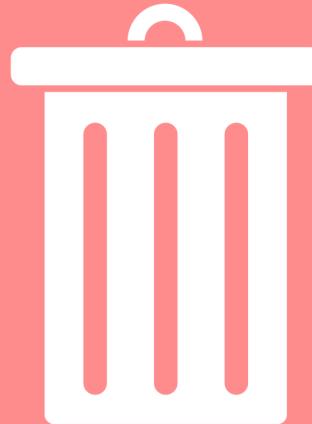
public class Sample3 {
    public static void main(String[] args) {
        List lista = new ArrayList();
        lista.add(10);
        lista.add("Maromo");
        lista.add(System.currentTimeMillis());
        lista.add(26);
        lista.add(26.3);
        Iterator i = lista.iterator();
        while(i.hasNext()){
            System.out.println(i.next());
        }
    }
}
```

Método *add* serve para adicionar um elemento na lista.



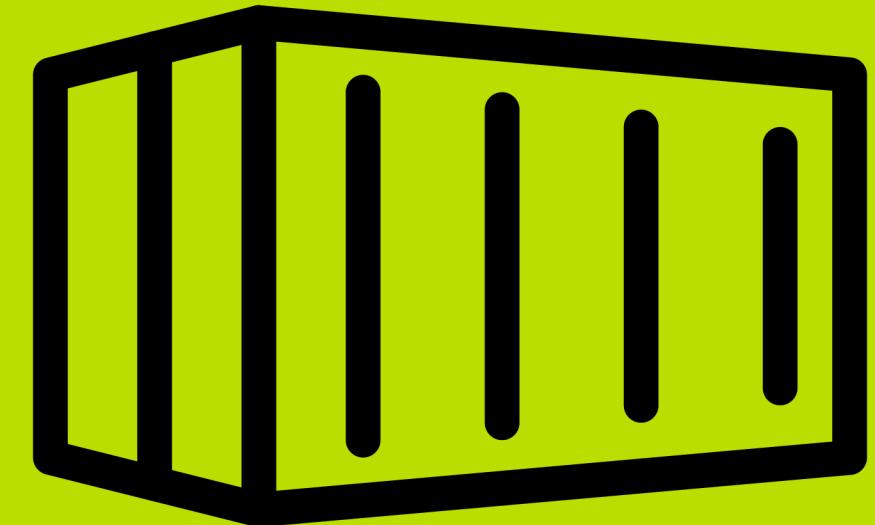
Removendo: Método remove

```
lista.add(26.3);
//remover o primeiro elemento da lista
lista.remove(0);
Iterator i = lista.iterator();
```



Usando o índice para a remoção, neste caso remove-se o primeiro item da lista.

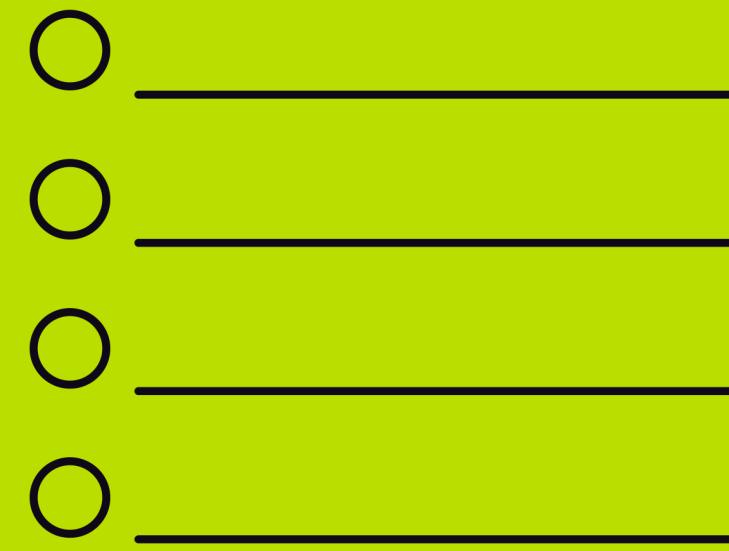
Procurando: Método contains()



Retorna **true** se o item for encontrado na lista.

```
//remover o primeiro elemento da lista
lista.remove(0);
//procurando "Maromo" na lista
Boolean t = lista.contains("Maromo");
if(t)
    System.out.println("Encontrei Maromo na lista");
else
    System.out.println("Não encontrei Maromo");
```

Tamanho: Método size()



Retorna a
quantidade de
itens na lista

```
// Mostrando o tamanho da lista
int tam = lista.size();
System.out.println("Tamanho da lista: " + tam);
```

Limpar a Lista

```
//limpa a lista  
lista.clear();
```



Genéricos

- Até agora declaramos um **LinkedList** e um **ArrayList** e nessas estruturas de dados não definimos que tipos de elementos seriam capazes de armazenar.
- Genéricos entre outras coisas garantem o tipo de referência que uma coleção será capaz de armazenar.
- O que caracteriza o uso de genéricos é o uso dos caracteres “<” e “>” envolvendo o nome de alguma classe.

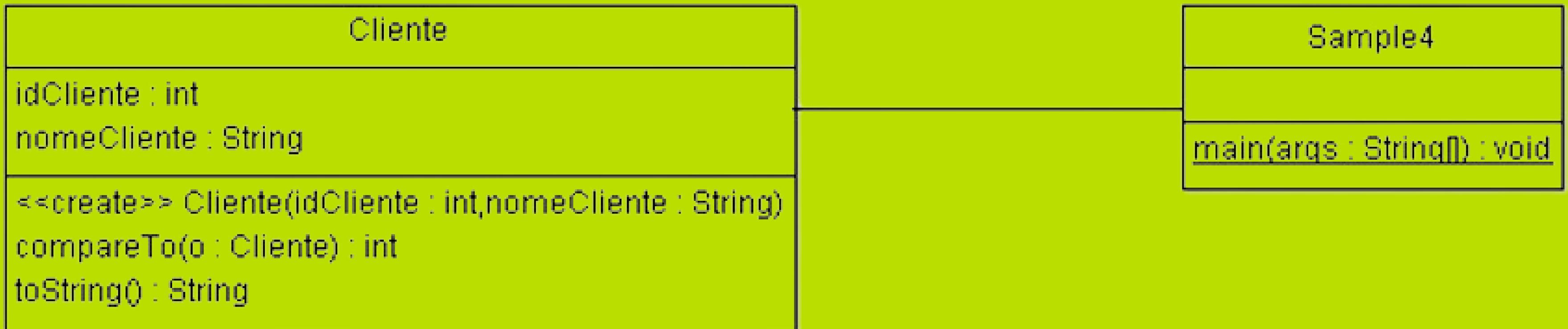
Genéricos - vantagens

- Evita problemas com exceções de conversão.
- Apresenta erro de compilação, caso seja adicionado algum elemento não pertencente ao tipo identificado.
- Não é necessário o uso do cast no uso do método get.
 - Use:
 - `String s = lista.get(0);`
 - Ao invés de:
 - `String s = (String)lista.get(0);`

Exemplo: Sample4

- A ideia neste exemplo é criar uma coleção de clientes e em seguida ordená-la pelo código do cliente (idCliente).
- Considerando isso, a classe Cliente deve implementar a interface [java.lang.Comparable](#) que define o que será nossa “ordem natural”.
- A interface possui apenas um método **compareTo()**.

Exemplo: prjSample4



```
public class Cliente implements Comparable<Cliente> {  
    public int idCliente;  
    public String nomeCliente;  
  
    Cliente(int idCliente, String nomeCliente) {  
        this.idCliente = idCliente;  
        this.nomeCliente = nomeCliente;  
    }  
  
    @Override  
    public int compareTo(Cliente o) {  
        if (this.idCliente < o.idCliente) {  
            return -1;  
        }  
        if (this.idCliente > o.idCliente) {  
            return 1;  
        }  
        return 0;  
    }  
}
```

```
@Override  
public String toString() {  
    StringBuilder sb = new StringBuilder();  
    sb.append("Identificação do Cliente \n");  
    sb.append(this.idCliente);  
    sb.append("\nNome do Cliente \n");  
    sb.append(this.nomeCliente);  
    sb.append("\n\n");  
    return sb.toString();  
}  
}
```



Classe: Cliente

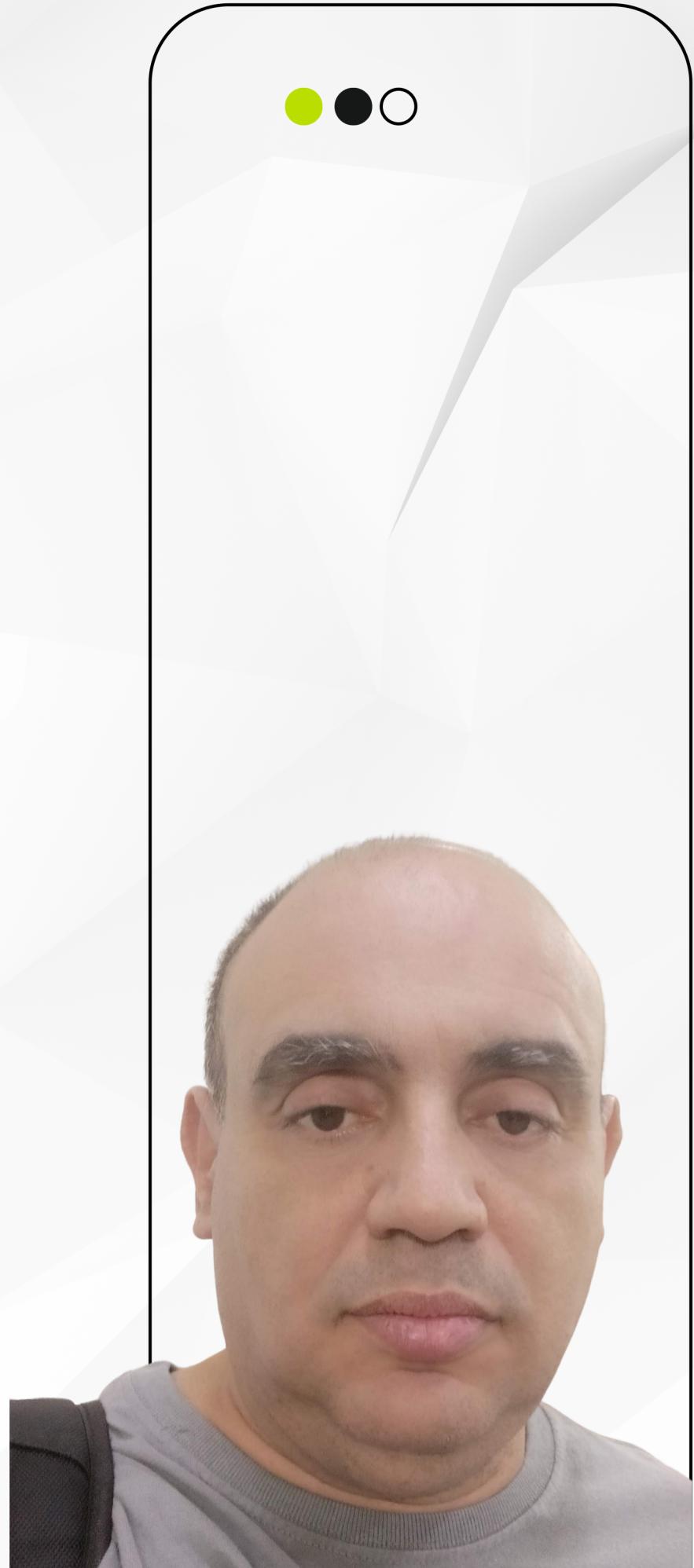
Classe: Sample4



```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;

public class Sample4{

    public static void main(String[] args) {
        List lista = new ArrayList<Cliente>();
        Cliente c1, c2, c3;
        c1 = new Cliente(1, "Oscar");
        c2 = new Cliente(7, "Maria");
        c3 = new Cliente(2, "Laércio");
        lista.add(c1);
        lista.add(c2);
        lista.add(c3);
        Iterator it = lista.iterator();
        while(it.hasNext()){
            Cliente x = (Cliente) it.next();
            System.out.println(x.toString());
        }
        Collections.sort(lista);
        it = lista.iterator();
        while(it.hasNext()){
            Cliente x = (Cliente) it.next();
            System.out.println(x.toString());
        }
    }
}
```



Obrigado

fim



Até a próxima aula



Referências Bibliográficas



- Mendes; **Java com Ênfase em Orientação a Objetos**, Novatec.
- Deitel; **Java, como programar** – 10º edição. Java SE 7 e 8
- Arnold, Gosling, Holmes; **A linguagem de programação Java** – 4º edição.
- Apostilas da Caelum.