

Coleções

Orientação a Objetos - DCC025

Prof. Edmar Welington Oliveira
oliveira.edmar@ufjf.edu.br

Universidade Federal de Juiz de Fora - UFJF
Departamento de Ciência da Computação - DCC

Coleção

■ Definição

- Uma coleção é uma estrutura de dados que permite armazenar vários objetos
- A coleção é, em si, um objeto (de agregação)
- Incorpora-se a funcionalidade de
 - Criar
 - Destruir
 - Localizar e
 - Organizar (ordenar, filtrar) as instâncias agregadas à coleção
- São naturais para lógica de negócios que envolve várias instâncias de uma mesma classe ou de tipos compatíveis Ex: aplicação dos rendimentos mensais em cadernetas de poupança

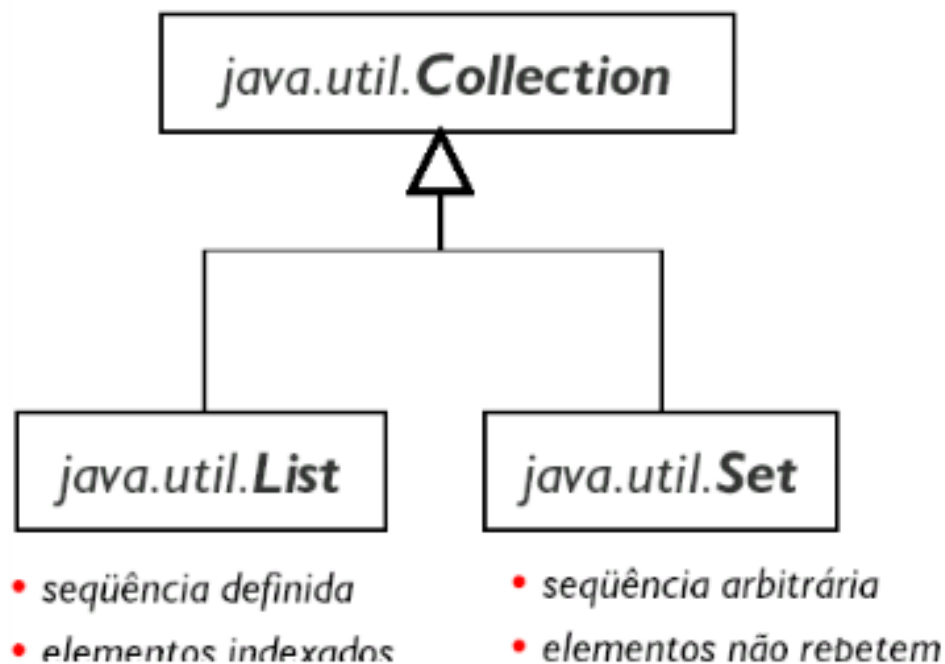
Tipos de Coleções

■ Tipos

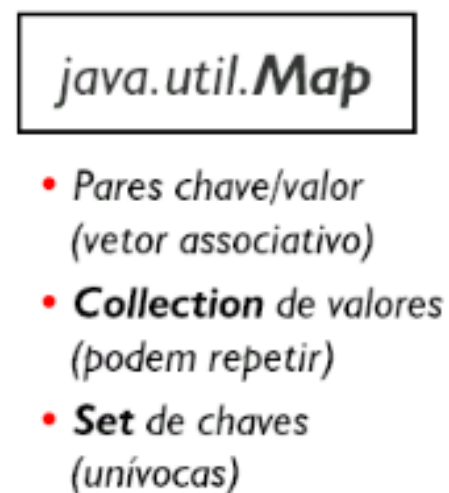
- Dependendo da forma de fazer 4 operações básicas (adição, remoção, acesso e pesquisa), vários tipos de coleções são definidas. Certas operações poderão ter um **desempenho** melhor ou pior ou poderão ter **restrições** ou **funcionalidade especial** dependendo do tipo de coleção
- Os 3 grandes tipos de coleções
 - Listas
 - Conjunto
 - Mapa

Framework Collections

*Coleções de
elementos individuais*



*Coleções de
pares de elementos*



Collections

■ Interface Collections

- Interface base para todos os tipos de coleção. Ela define as operações mais básicas para coleções de objetos Adição (**add**) e remoção (**remove**) abstratos (sem informações quanto à ordenação dos elementos), esvaziamento (**clear**), tamanho (**size**), conversão para array (**toArray**), objeto de iteração (**iterator**), e verificações de existência (**contains** e **isEmpty**).

Framework Collections

Define operações comuns
a todas as subclasses

*Coleções de
elementos individuais*

*java.util.**Collection***



*java.util.**List***

- sequência definida
- elementos indexados

*java.util.**Set***

- sequência arbitrária
- elementos não repetem

Pode conter elementos repetidos

Não pode conter elementos repetidos

Idéia de chave/valor
Associa pares de objetos


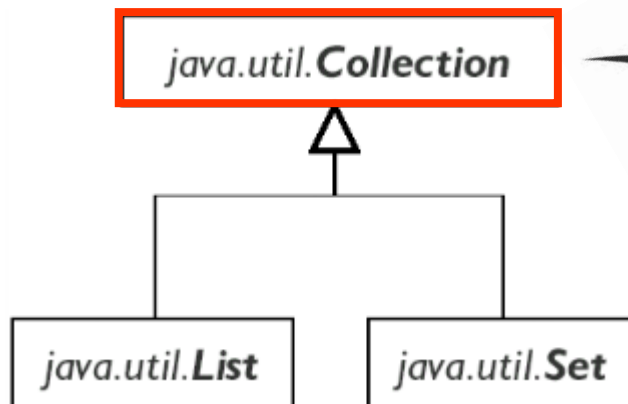
*Coleções de
pares de elementos*

*java.util.**Map***

- Pares chave/valor
(vetor associativo)
- **Collection** de valores
(podem repetir)
- **Set** de chaves
(unívocas)

Framework Collections

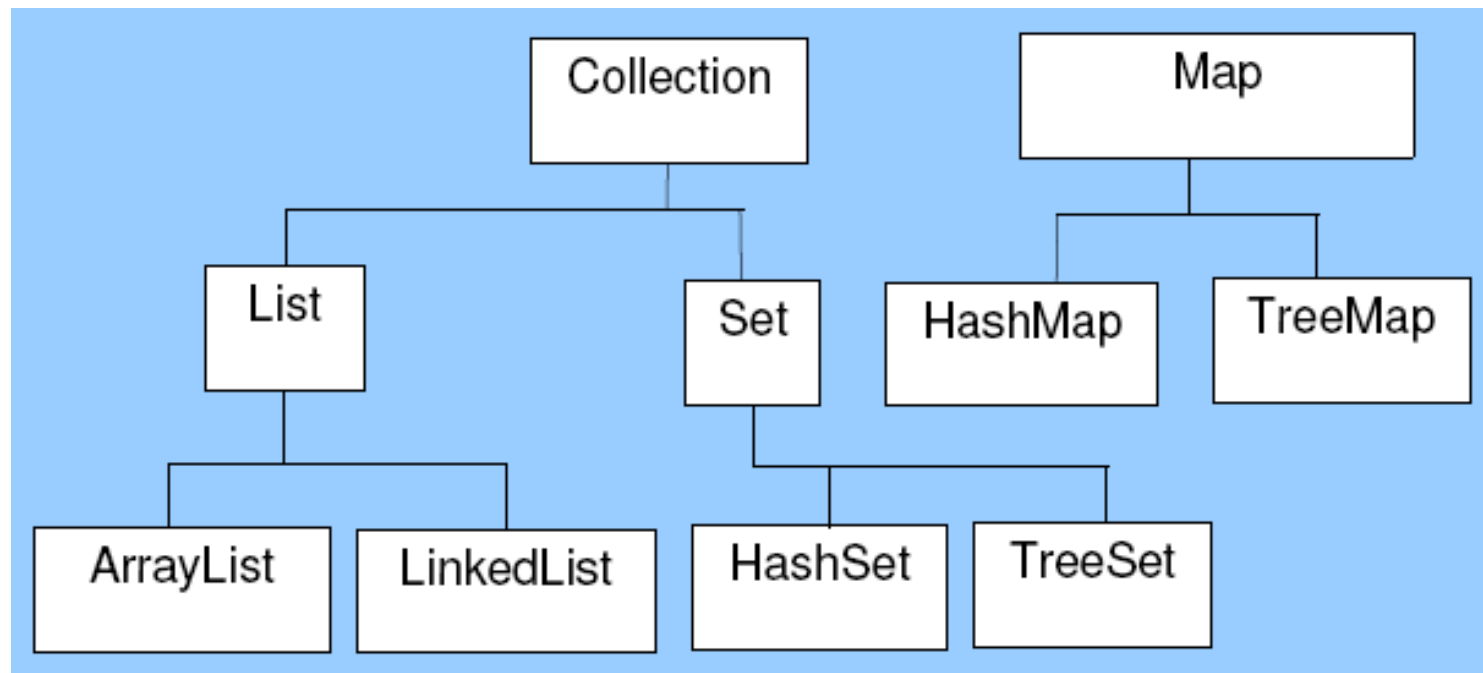
Define operações comuns
a todas as subclasses



```
size()
isEmpty()
clear()
add(Object)
contains(Object)
remove(Object)
containsAll(Collection)
removeAll(Collection)
retainAll(Collection)
toArray()
```

A list of common methods defined in the `java.util.Collection` interface, including `size()`, `isEmpty()`, `clear()`, `add(Object)`, `contains(Object)`, `remove(Object)`, `containsAll(Collection)`, `removeAll(Collection)`, `retainAll(Collection)`, and `toArray()`.

Coleções



Listas

Listas

- Definição

- Uma lista é uma coleção de elementos arrumados numa ordem linear, isto é, onde cada elemento tem um antecessor (exceto o primeiro) e um sucessor (exceto o último)

- Características

- Normalmente implementada como "Array" ou "Lista Encadeada"
- A Lista pode ser mantida ordenada ou não

Interface List

- List
 - Pode possuir valores repetidos
 - Principais subclasses
 - Vector
 - ArrayList
 - LinkedList

Interface List

■ ArrayList

- Implementação de List que utiliza internamente um array de objetos. Em uma inserção onde o tamanho do array interno não é suficiente, um novo array é alocado (de tamanho igual a 1.5 vezes o array original), e todo o conteúdo é copiado para o novo array. Em uma inserção no meio da lista (índice < tamanho), o conteúdo posterior ao índice é deslocado em uma posição. Esta implementação é a recomendada quando o tamanho da lista é previsível (evitando realocações) e as operações de inserção e remoção são feitas, em sua maioria, no fim da lista (evitando deslocamentos), ou quando a lista é mais lida do que modificada (otimizado para leitura aleatória).

Interface List

■ LinkedList

- Implementação de List que utiliza internamente uma **lista encadeada**. A localização de um elemento na n-ésima posição é feita percorrendo-se a lista da ponta mais próxima até o índice desejado. A inserção é feita pela adição de novos nós, entre os nós adjacentes, sendo que antes é necessária a localização desta posição. Esta implementação é recomendada quando as modificações são feitas em sua maioria tanto no início quanto no final da lista, e o percorrimento é feito de forma sequencial (via Iterator) ou nas extremidades, e não aleatória (por índices). Um exemplo de uso é como um fila (FIFO - First-In-First-Out), onde os elementos são retirados da lista na mesma sequência em que são adicionados.

Interface List

- Vector
 - Implementação de List com o mesmo comportamento da ArrayList, tem performance inferior. Contudo, pode ser utilizado em um ambiente multitarefa (acessado por várias threads) sem perigo de perda da consistência de sua estrutura interna.

Interface List

- List
 - Interface que estende Collection, e que define coleções ordenadas (sequências), onde se tem o controle total sobre a posição de cada elemento, identificado por um índice numérico.

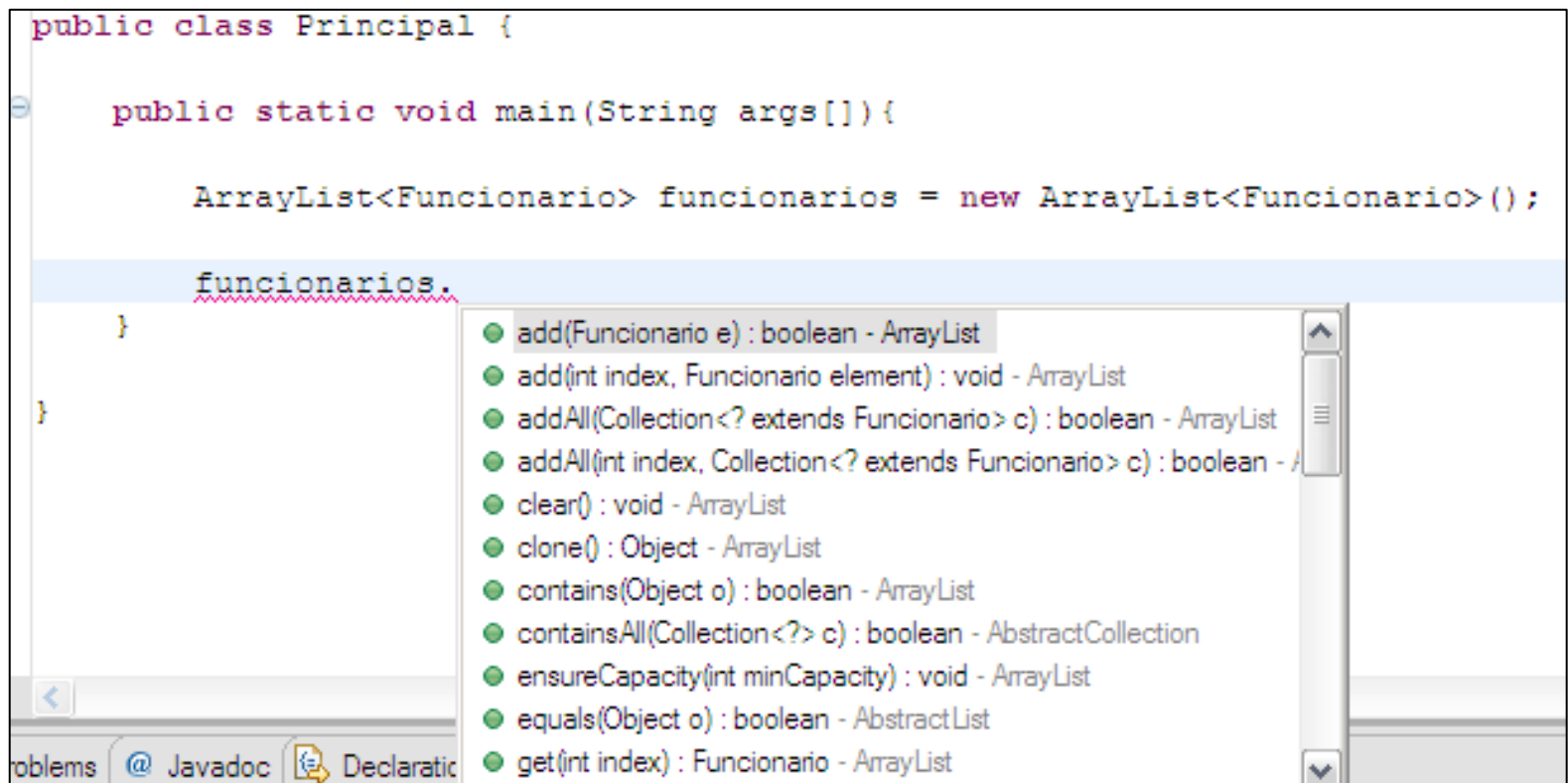
Interface List

```
public class Principal {  
  
    public static void main(String args[]){  
  
        ArrayList<Funcionario> funcionarios1 = new ArrayList<Funcionario>();  
        LinkedList<Funcionario> funcionarios2 = new LinkedList<Funcionario>();  
        Vector<Funcionario> funcionarios3 = new Vector<Funcionario>();  
  
    }  
  
}
```


Exemplo List

```
public static void main(String[] args) {  
    ArrayList<String> nomes = new ArrayList<String>();  
    nomes.add("João");  
    nomes.add("José");  
    nomes.add("Maria");  
    nomes.add("Bianca");  
    System.out.println("Qtd elementos: "+nomes.size());  
    String string = nomes.get(3);  
    System.out.println(string);  
}
```

Interface List



Interface Vector



download.oracle.com/javase/1.4.2/docs/api/java/util/Vector.html



Method Summary

void	add (int index, Object element) Inserts the specified element at the specified position in this Vector.
boolean	add (Object o) Appends the specified element to the end of this Vector.
boolean	addAll (Collection c) Appends all of the elements in the specified Collection to the end of this Vector, in the order that they appear in the Collection.
boolean	addAll (int index, Collection c) Inserts all of the elements in the specified Collection into this Vector at the specified position.
void	addElement (Object obj) Adds the specified component to the end of this vector, increasing its size by one.
int	capacity () Returns the current capacity of this vector.

Conjunto

Conjuntos

- Definição
 - Um conjunto é uma coleção que não possui elementos duplicados
- Características
 - Não existe a noção de "ordem dos elementos"
 - O Conjunto **pode** ser mantido ordenado ou não
 - Normalmente implementada como "Tabela Hash" ou "Árvore"

Interface Set

■ Set

- Interface que define uma coleção, ou conjunto, que não contém duplicatas de objetos. Isto é, são ignoradas as adições caso o objeto ou um objeto equivalente já exista na coleção. Por objetos equivalentes, entenda-se objetos que tenham o mesmo código hash (retornado pelo método `hashCode()`) e que retornem verdadeiro na comparação feita pelo método `equals()`. Não é garantida a ordenação dos objetos, isto é, a ordem de iteração dos objetos não necessariamente tem qualquer relação com a ordem de inserção dos objetos. Por isso, não é possível indexar os elementos por índices numéricos, como em uma List.

Interface Set

- Set
 - Não possui valores repetidos
 - Pode possuir um único valor “null”
 - Principais subclasses
 - HashSet
 - TreeSet
 - LinkedHashSet

Interface Set

- HashSet
 - Implementação de Set que utiliza uma **tabela hash** para guardar seus elementos. Não garante a ordem de iteração, nem que a ordem permanecerá constante com o tempo (uma modificação da coleção pode alterar a ordenação geral dos elementos). Por utilizar o algoritmo de tabela hash, o acesso é rápido, tanto para leitura quanto para modificação.

Interface Set

- **LinkedHashSet**

- Implementação de Set que estende HashSet, mas adiciona previsibilidade à ordem de iteração sobre os elementos, isto é, uma iteração sobre seus elementos (utilizando o Iterator) mantém a ordem de inserção (a inserção de elementos duplicados não altera a ordem anterior). Internamente, é mantida uma lista duplamente encadeada que mantém esta ordem. Por ter que manter uma lista paralelamente à tabela hash, a modificação deste tipo de coleção acarreta em uma leve queda na performance em relação à HashSet, mas ainda é mais rápida que uma TreeSet, que utiliza comparações para determinar a ordem dos elementos.

Interface Set

- SortedSet

- Interface que estende Set, adicionando a semântica de ordenação natural dos elementos. A posição dos elementos no percorrimento da coleção é determinado pelo retorno do método `compareTo(o)`, caso os elementos implementem a interface `Comparable`, ou do método `compare(o1, o2)` de um objeto auxiliar que implemente a interface `Comparator`.

Interface Set

- TreeSet

- Implementação de SortedSet que utiliza internamente uma TreeMap, que por sua vez utiliza o algoritmo Red-Black para a ordenação da árvore de elementos. Isto garante a ordenação ascendente da coleção, de acordo com a ordem natural dos elementos. Use esta classe quando precisar de um conjunto (de elementos únicos) que deve estar sempre ordenado, mesmo sofrendo modificações. Para casos onde a escrita é feita de uma só vez, antes da leitura dos elementos, talvez seja mais vantajoso fazer a ordenação em uma List, seguida de uma cópia para uma LinkedHashSet (dependendo do tamanho da coleção e do número de repetições de elementos).

Exemplo Set

```
public static void main(String[] args) {  
    Set<String> nomes = new HashSet<String>();  
    nomes.add("Joao");  
    nomes.add("Jose");  
    nomes.add("Maria");  
    nomes.add("Bianca");  
    System.out.println("Qtd elementos: " + nomes.size());  
    if (nomes.contains("Maria"))  
        System.out.println("Contém Maria");  
}
```

Mapas

Mapas

■ Definição

- Um mapa armazena pares (chave, valor) chamados itens
- Chaves e valores podem ser de qualquer tipo
- A chave é utilizada para achar um elemento rapidamente
- Mapas são estruturas especiais usadas para que a pesquisa seja rápida
- Diz-se, portanto, que um mapa "mapeia chaves para valores"

■ Características

- O Mapa pode ser mantido ordenado ou não (com respeito às chaves)
- Normalmente implementada como "Tabela Hash" ou "Árvore"

Interface Map

- Map

- Interface que define um array associativo, isto é, ao invés de números, objetos são usados como chaves para se recuperar os elementos. As chaves não podem se repetir (seguindo o mesmo princípio da interface Set), mas os valores podem ser repetidos para chaves diferentes. Um Map também não possui necessariamente uma ordem definida para o percorrimento.

Interface Map

■ Map

- Trabalham com o conceito de chave/valor
- Chaves são únicas
- Principais subclasses
 - LinkedHashMap
 - HashMap
 - TreeMap
- Principais métodos
 - `put(chave, valor)`
Coloca par chave/elemento no mapa
 - `get(chave)`
Retorna o valor correspondente a chave passada como parâmetro

Interface Map

- **HashMap**
 - Implementação de Map que utiliza uma tabela hash para armazenar seus elementos. O tempo de acesso aos elementos (leitura e modificação) é constante (muito bom) se a função de hash for bem distribuída, isto é, a chance de dois objetos diferentes retornarem o mesmo valor pelo método `hashCode()` é pequena.

Interface Map

- **LinkedHashMap**

- Implementação de Map que estende HashMap, mas adiciona previsibilidade à ordem de iteração sobre os elementos, isto é, uma iteração sobre seus elementos (utilizando o Iterator) mantém a ordem de inserção (a inserção de elementos duplicados não altera a ordem anterior). Internamente, é mantida uma lista duplamente encadeada que mantém esta ordem. Por ter que manter uma lista paralelamente à tabela hash, a modificação deste tipo de coleção acarreta em uma leve queda na performance em relação à HashMap, mas ainda é mais rápida que uma TreeMap, que utiliza comparações para determinar a ordem dos elementos.

Interface Map

- SortedMap

- Interface que estende Map, adicionando a semântica de ordenação natural dos elementos, análogo à SortedSet. Também adiciona operações de partição da coleção, com os métodos headMap(k) - que retorna um SortedMap com os elementos de chaves anteriores a k -, subMap(k1,k2) - que retorna um SortedMap com os elementos de chaves compreendidas entre k1 e k2 - e tailMap(k) - que retorna um SortedMap com os elementos de chaves posteriores a k.

Interface Map

- TreeMap

- Implementação de SortedMap que utiliza o algoritmo Red-Black para a ordenação da árvore de elementos. Isto garante a ordenação ascendente da coleção, de acordo com a ordem natural dos elementos, definida pela implementação da interface Comparable ou Comparator. Use esta classe quando precisar de um conjunto (de elementos únicos) que deve estar sempre ordenado, mesmo sofrendo modificações. Análogo ao TreeSet, para casos onde a escrita é feita de uma só vez, antes da leitura dos elementos, talvez seja mais vantajoso fazer a ordenação em uma List, seguida de uma cópia para uma LinkedHashSet (dependendo do tamanho da coleção e do número de repetições de chaves).

Exemplo Map

```
public static void main(String[] args) {  
    HashMap<String,String> nomes = new HashMap<String,String>();  
    nomes.put("joao", "Joao Da Silva");  
    nomes.put("jose", "Jose Matos");  
    nomes.put("maria", "Maria das Dores");  
    nomes.put("bianca", "Bianca Patricia");  
    System.out.println("Qtd elementos: "+nomes.size());  
    String nome = nomes.get("maria");  
    System.out.println(nome);  
}
```

Resumo

Interfaces	Implementations				
	Hash table	Resizable array	Tree	Linked list	Hash table + Linked list
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Queue					
Map	HashMap		TreeMap		LinkedHashMap

Resumo

Interfaces	Implementations				
	Hash table	Resizable array	Tree	Linked list	Hash table + Linked list
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Queue					
Map	HashMap		TreeMap		LinkedHashMap

Normalmente utilizamos esta interface quando queremos uma coleção do tipo FIFO (First-In-First-Out), também conhecida como fila.

A classe **LinkedList** implementa, além da interface **List**, a interface **Queue**, portanto, podendo ser utilizada para construir filas.

Percorrer Coleções

Percorrendo Coleções

- Duas formas de percorrer Coleções
 - Iterator
 - For-each

Iterator

- Iterator
 - Interface que define as operações básicas para o percorrimto dos elementos da coleção. Utiliza o pattern de mesmo nome (Iterator, GoF), desacoplando o código que utiliza as coleções de suas estruturas internas.

Percorrendo Coleções

■ Interface Iterator

- Objeto que permite que todos os elementos da coleção sejam acessados
- Define métodos para percorrer Collections:

■ iterator()

- Método que deve retornar um objeto Iterator para a coleção.
- A coleção deve implementar esse método
- Uma vez implementado, os seguintes métodos podem ser utilizados

▪ hasNext()

Devolve o valor true se ainda existir objeto a ser percorrido na coleção.
Indica se chegamos ou não ao fim da coleção

▪ next()

Devolve o próximo objeto da coleção.

Percorrendo Coleções

- Interface Iterator
 - Percorre a coleção sequenciamento, do início ao fim

```
3 public class Pessoa {
4
5     private String nome;
6
7     public Pessoa(String nome){
8         this.nome = nome;
9     }
10    public String getNome() {
11        return nome;
12    }
13
14    public static void main(String args[]){
15
16        Pessoa p1 = new Pessoa("Pessoa1");
17        Pessoa p2 = new Pessoa("Pessoa2");
18        Pessoa p3 = new Pessoa("Pessoa3");
19
20        ArrayList<Pessoa> conjuntoPessoas = new ArrayList<Pessoa>();
21
22        conjuntoPessoas.add(p1);
23        conjuntoPessoas.add(p2);
24        conjuntoPessoas.add(p3);
25
26        Iterator<Pessoa> it = conjuntoPessoas.iterator();
27
28        while(it.hasNext()){
29            System.out.println(it.next().getNome());
30        }
31    }
32 }
```

```
4 public class Pessoa {
5
6     private String nome;
7
8     public Pessoa(String nome){
9         this.nome = nome;
10    }
11    public String getNome() {
12        return nome;
13    }
14
15    public static void main(String args[]){
16
17        Pessoa p1 = new Pessoa("Pessoa1");
18        Pessoa p2 = new Pessoa("Pessoa2");
19        Pessoa p3 = new Pessoa("Pessoa3");
20
21        ArrayList<Pessoa> conjuntoPessoas = new ArrayList<Pessoa>();
22
23        conjuntoPessoas.add(p1);
24        conjuntoPessoas.add(p2);
25        conjuntoPessoas.add(p3);
26
27        for(Pessoa p: conjuntoPessoas){
28            System.out.println(p.getNome());
29        }
30    }
31 }
```

Exemplos

1

```
5 public class Executa {
6
7     public static void executaImpressao(Set<Pessoa> conjuntoPessoas) {
8
9         Iterator<Pessoa> it = conjuntoPessoas.iterator();
10
11         while(it.hasNext()){
12             System.out.println(it.next().getNome());
13         }
14     }
15
16     public static void main(String args[]){
17
18         Pessoa p1 = new Pessoa("A");
19         Pessoa p2 = new Pessoa("B");
20         Pessoa p3 = new Pessoa("C");
21         Pessoa p4 = new Pessoa("D");
22
23         Set<Pessoa> conjuntoPessoas = new HashSet<Pessoa>();
24
25         conjuntoPessoas.add(p1);
26         conjuntoPessoas.add(p2);
27         conjuntoPessoas.add(p3);
28         conjuntoPessoas.add(p4);
29
30         executaImpressao(conjuntoPessoas);
31     }
```


1

```
3 import java.util.Set;
4
5 public class Executa {
6
7     public static void executaImpressao(Set<Pessoa> conjuntoPessoas) {
8
9         Iterator<Pessoa> it = conjuntoPessoas.iterator();
10
11         while(it.hasNext()) {
12             System.out.println(it.next().getNome());
13         }
14     }
15
16     public static void main(String args[]) {
17
18         Pessoa p1 = new Pessoa("A");
19         Pessoa p2 = new Pessoa("B");
20         Pessoa p3 = new Pessoa("C");
21         Pessoa p4 = new Pessoa("D");
22     }
```

Problems @ Javadoc Declaration Console Progress
<terminated> Executa [Java Application] C:\Program Files\Java\jdk1.6.0_21\jre\bin\javaw.exe (31/05/2011 09:42:17)

B
C
A
D

- Não é garantido que a ordem da iteração será a mesma da inserção
- Inserimos ACBD e o resultado impresso foi BCAD
- Não foi feita nenhuma ordenação nos elementos inseridos;

1a

```
10     Iterator<Pessoa> it = conjuntoPessoas.iterator();
11
12     while(it.hasNext()){
13         System.out.println(it.next().getNome());
14     }
15 }
16
17 public static void main(String args[]){
18
19     Pessoa p1 = new Pessoa("A");
20     Pessoa p2 = new Pessoa("B");
21     Pessoa p3 = new Pessoa("C");
22     Pessoa p4 = new Pessoa("D");
23
24     Set<Pessoa> conjuntoPessoas = new LinkedHashSet<Pessoa>();
25
26     conjuntoPessoas.add(p1);
27     conjuntoPessoas.add(p2);
28     conjuntoPessoas.add(p3);
29     conjuntoPessoas.add(p4);
```

Problems Javadoc Declaration Console Progress
terminated> Executa [Java Application] C:\Program Files\Java\jdk1.6.0_21\jre\bin\javaw.exe (31/05/2011 09:52:44)

A
B
C
D

Coleções

Orientação a Objetos - DCC025

Prof. Edmar Welington Oliveira
oliveira.edmar@ufjf.edu.br

Universidade Federal de Juiz de Fora - UFJF
Departamento de Ciência da Computação - DCC