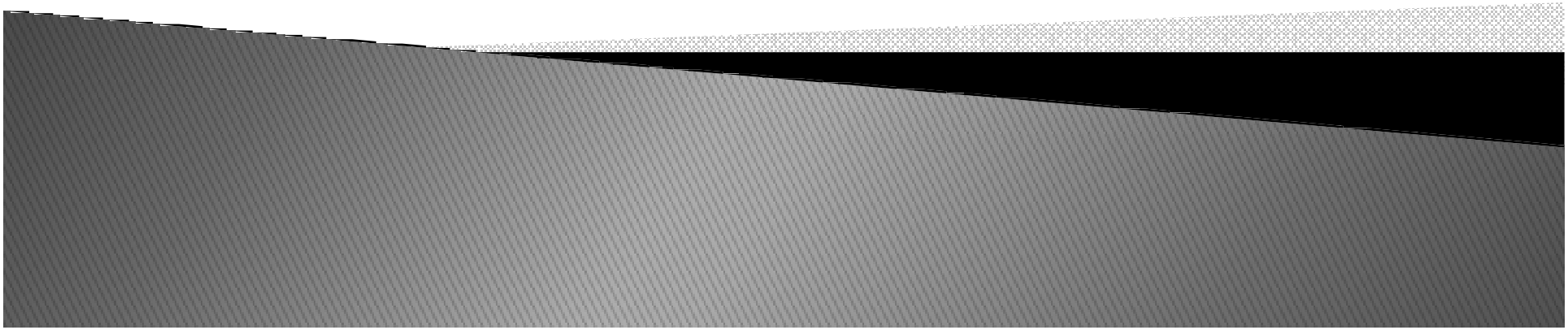
Java

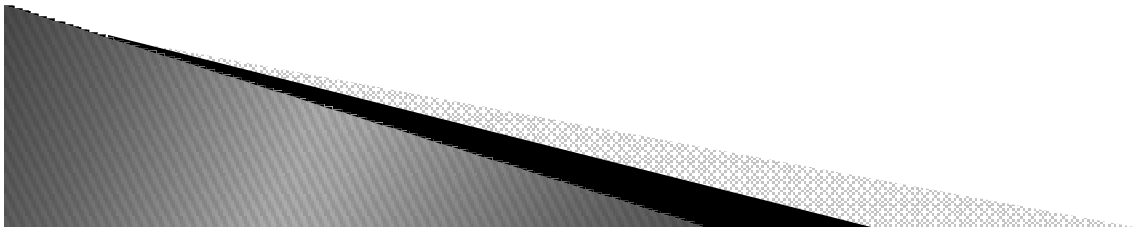
Aula08

BSI - UFRPE
Prof. Gustavo Callou
gcallou@gmail.com



Tópicos

- Coleções
  - List
  - Set
  - Map



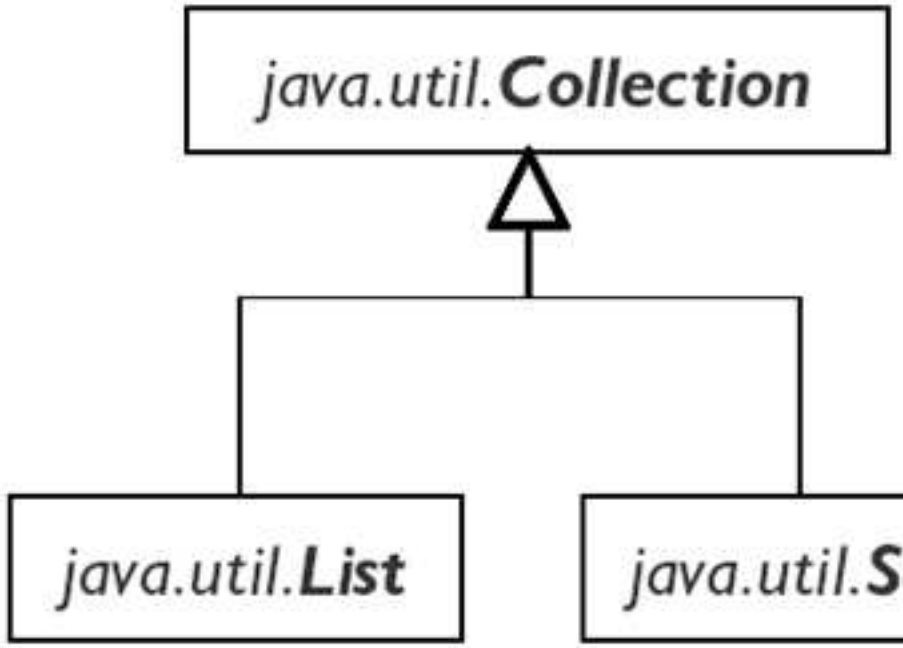
java.util – Coleções

- Classes e interfaces do pacote **`java.util`** que representam listas, conjuntos e mapas.
- Solução flexível para armazenar objetos.
- Quantidade armazenada de objetos não é fixa, como ocorre com arrays.
- Poucas interfaces (duas servem de base) permitem maior reuso e um vocabulário menor de métodos.
  - **`add()`**, **`remove()`**, **`contains()`** - principais métodos de ***`Collection`***
  - **`put()`**, **`get()`** - principais métodos de interface ***`Map`***
- Desvantagens:
  - Menos eficientes que arrays.
  - Não aceitam tipos primitivos (só empacotados).

java.util – Coleções

Coleções de elementos individuais

Coleções de pares de elementos



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

- *seqüência definida*
- *elementos indexados*

- *seqüência arbitrária*
- *elementos não repetem*

java.util.*Collection*

- Uma coleção representa um grupo de objetos.
- Algumas coleções permitem valores repetidos, outras não.
- Algumas coleções são ordenadas, outras não.
- Principais subinterfaces:
  - **`List`**
  - **`Set`**
- Principais métodos:
  - **`boolean add(Object o)`**: adiciona objeto na coleção
  - **`boolean contains(Object o)`**
  - **`boolean isEmpty()`**
  - **`Iterator iterator()`**: retorna iterator
  - **`boolean remove(Object o)`**
  - **`int size()`**: retorna o número de elementos
  - **`Object[] toArray(Object[])`**: converte coleção em Array

java.util.*List*

- Estende a interface ***Collection***.
- Principais subclasses:
  - **`Vector`** – ideal para acesso randômico. Sincronizado.
  - **`ArrayList`** – ideal para acesso randômico. Não sincronizada.
  - **`LinkedList`** – ideal para acesso sequencial. Não sincronizada.
- Principais métodos adicionais
  - **`void add(int index, Object o)`**: adiciona objeto na posição indicada (empurra elementos existentes para a frente)
  - **`Object get(int index)`**: recupera objeto pelo índice
  - **`int indexOf(Object o)`**: procura objeto e retorna índice da primeira ocorrência
  - **`Object set(int index, Object o)`**: grava objeto na posição indicada (apaga qualquer outro que ocupava a posição).
  - **`Object remove(int index)`**
  - ***`ListIterator`*** **`listIterator()`**: retorna uma ***`ListIterator`***

java.util.Vector

- Implementa um array redimensionável bastante eficiente para leitura.
- Possui um construtor na forma: **`Vector(int capacidadeInicial, int incremento)`**
  - **capacidadeInicial** – o tamanho inicial do vector (se não especificado no construtor, assume o tamanho de 10 elementos).
  - **incremento** – especifica em quantos elementos o vector deverá crescer quando sua capacidade for extrapolada (Se não especificado no construtor, o vector terá seu tamanho duplicado).
- Implementado internamente com arrays.
- Otimiza o espaço alocado pelo array que encapsula.
- Ideal para acesso aleatório.
- É sincronizado, permitindo que somente uma *thread* por vez acesse um método sincronizado do objeto.
- Métodos comumente utilizados:

```
add(Object o)                          contains(Object element)
elementAt(int index)                   indexOf(Object element)
isEmpty()                              lastIndexOf(Object element)
remove(int index)                              remove(Object o)
size()                                 setElementAt(Object obj, int index)
toArray()
```

java.util.LinkedList

- Muito mais eficiente que **`ArrayList`** e **`Vector`** para remoção e inserção no meio da lista.
- Pode ser usada para implementar pilhas, filas unidirecionais (queue) e bidirecionais (deque – double -ended queue). Possui métodos para manipular essas estruturas.
- Ideal para acesso seqüencial.
- Não sincronizado.
- Principais métodos:
  - **`add(int index, Object element)`**
  - **`remove(int index)`**
  - **`get(int index)`**
  - **`indexOf(Object o)`**
  - **`lastIndexOf(Object o)`**

Listas

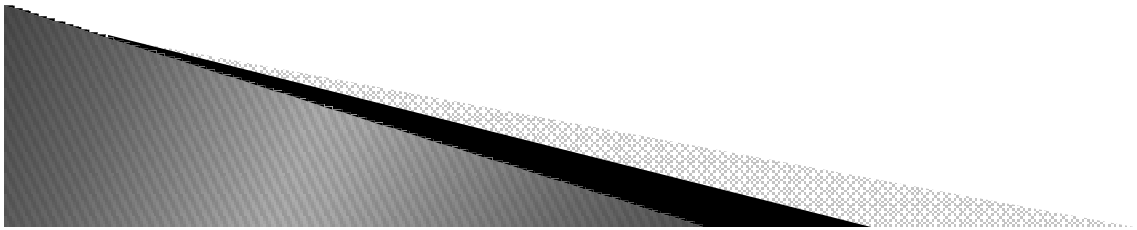
- A implementação mais utilizada da interface List é ArrayList.
- ArrayList é ideal para pesquisa
- LinkedList é ideal para inserção e remoção de itens nas pontas.
- A partir do Java 5 podemos usar o recurso de Generics para restringir as listas a um determinado tipo de objetos (e não qualquer Object):

```
List<ContaCorrente> contas = new
ArrayList<ContaCorrente>();
contas.add(c1);
contas.add(c3);
contas.add(c2);
```

Listas

- O uso de Generics também elimina a necessidade de casting, já que seguramente todos os objetos inseridos na lista serão do tipo ContaCorrente:

```
for(int i = 0; i < contas.size(); i++) {
      ContaCorrente cc = contas.get(i); // sem
casting!
      System.out.println(cc.getSaldo());
}
```

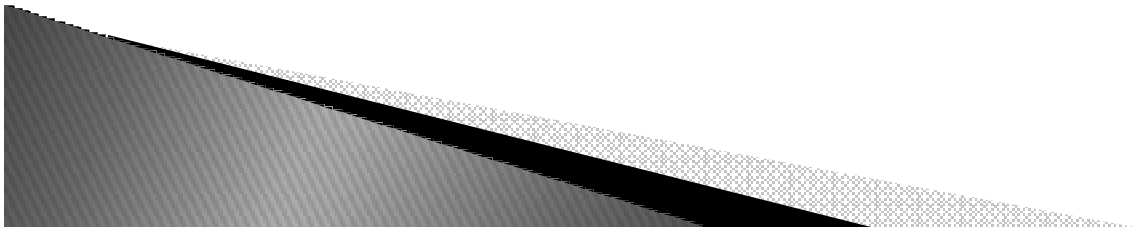


java.util.Collections

- Esta classe contém métodos estáticos que retornam ou operam sobre coleções.
- Principais métodos:
  - **`binarySearch(List list, Object key)`** – Busca um determinado elemento de uma lista em ordem crescente usando o algoritmo de busca binária.
  - **`copy(List dest, List src)`** – Copia todos os elementos de uma lista para outra.
  - **`fill(List list, Object obj)`** – Substitui todos os elementos da lista pelo objeto especificado.
  - **`sort(List list)`** – Orderna uma lista usando uma modificação do algoritmo *Mergesort*.
  - **`shuffle(List list)`** – Randomicamente permuta os elementos de uma lista.
  - **`reverse(List list)`** – inverte a ordem dos elementos da lista.
  - **`max(Collection coll)`** – retorna o maior elemento da coleção.
  - **`min(Collection coll)`** - retorna o menor elemento da coleção.

Ordenação

```
List lista = new ArrayList();
lista.add("Sérgio");
lista.add("Paulo");
lista.add("Guilherme");
System.out.println(lista);
Collections.sort(lista);
System.out.println(lista);
```



java.util.*Set*

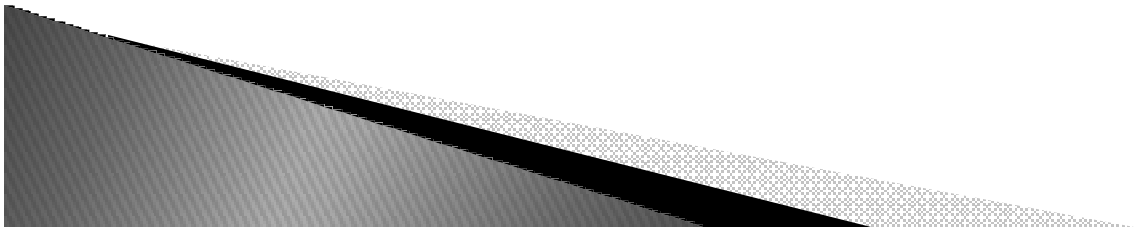
- Set representa um conjunto matemático.
- Não possui valores repetidos.
- Pode possuir um único elemento **`null`**.
- Principais subclasses:
  - **`HashSet`** (implements ***`Set`***)
  - **`TreeSet`** (implements ***`SortedSet`***)
- Principais métodos:
  - **`add(Object o)`**
  - **`contains(Object o)`**
  - **`equals(Object o)`**
  - **`isEmpty()`**
  - **`iterator()`**
  - **`remove(Object o)`**
  - **`size()`**
  - **`toArray()`**

java.util.*Map*

- Objetos Map são semelhantes a arrays mas, em vez de índices numéricos, usam objetos como chaves.
- Chaves são únicas.
- Valores podem ser duplicados.
- Principais subclasses:
  - **`Hashtable`** – Sincronizada. Não aceita **`null`** como chave. Acesso rápido devido ao uso do método **`hashCode()`**.
  - **`HashMap`** – Não sincronizada. Aceita **`null`** como chave. Acesso rápido devido ao uso do método **`hashCode()`**.
  - **`TreeMap`** – Não sincronizada. Mapa ordenado. Contém métodos para manipular elementos ordenados.
- Métodos principais:
  - **`void put(Object key, Object value)`**: acrescenta um objeto
  - **`Object get(Object key)`**: recupera um objeto
  - **`Set keySet()`**: retorna um **`Set`** de chaves
  - **`Collection values()`**: retorna uma **`Collection`** de valores.

Exercício

- Defina uma classe Conta, cujos objetos representam contas bancarias, contendo os seguintes atributos privados: número da conta, nome e CPF do correntista, senha e saldo. Defina um construtor de objetos dessa classe, que receba como parâmetros essas informações (exceto saldo) e crie uma Conta com saldo igual a zero. Defina também dois métodos que possibilitem o saque (deve receber como parâmetros à senha e o valor) e deposito em conta (recebe como parâmetro o valor do depósito). Esses métodos devem retornar o saldo corrente em conta após efetuada a operação. O método correspondente ao saque deve lançar uma exceção caso o saque não puder ser efetuado (saldo insuficiente ou senha inválida).



Exercício

- Defina uma classe ChequeEspecial, como subclasse de Conta, que contenha como valor adicional o limite de crédito da conta. Redefina os métodos: construtor e saque para esta classe. O método construtor deve receber como parâmetro além das informações anteriores o limite de credito, e deve chamar o construtor da superclasse. O método saque deve permitir o saque enquanto o limite de crédito não for extrapolado.

- Defina uma classe Banco, como uma coleção de Contas. A classe deve fornecer métodos para cadastrar uma nova conta, imprimir um relatório para todas as contas existentes (contento número da conta, nome do correntista e saldo), efetuar saques e depósitos em uma conta especifica (usar polimorfismo).

- Implemente um menu que permita cadastrar novas contas, imprimir o relatório, efetuar deposito/saque em uma das contas. As contas devem ser identificadas pelo número.

Referência

- http://www.slideshare.net/regispires/java-12-colecoes-presentation