



Programação II

Collections

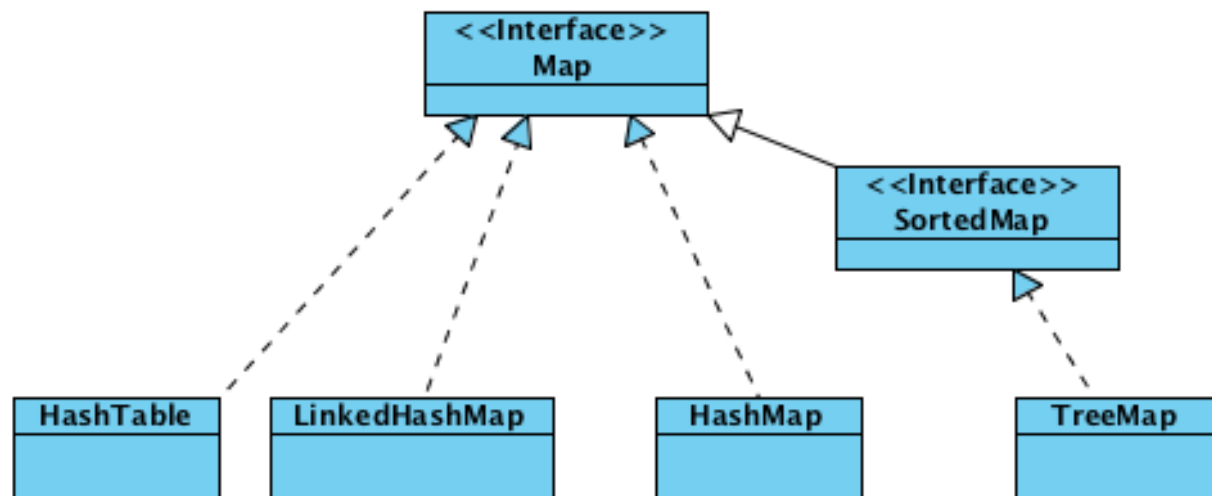
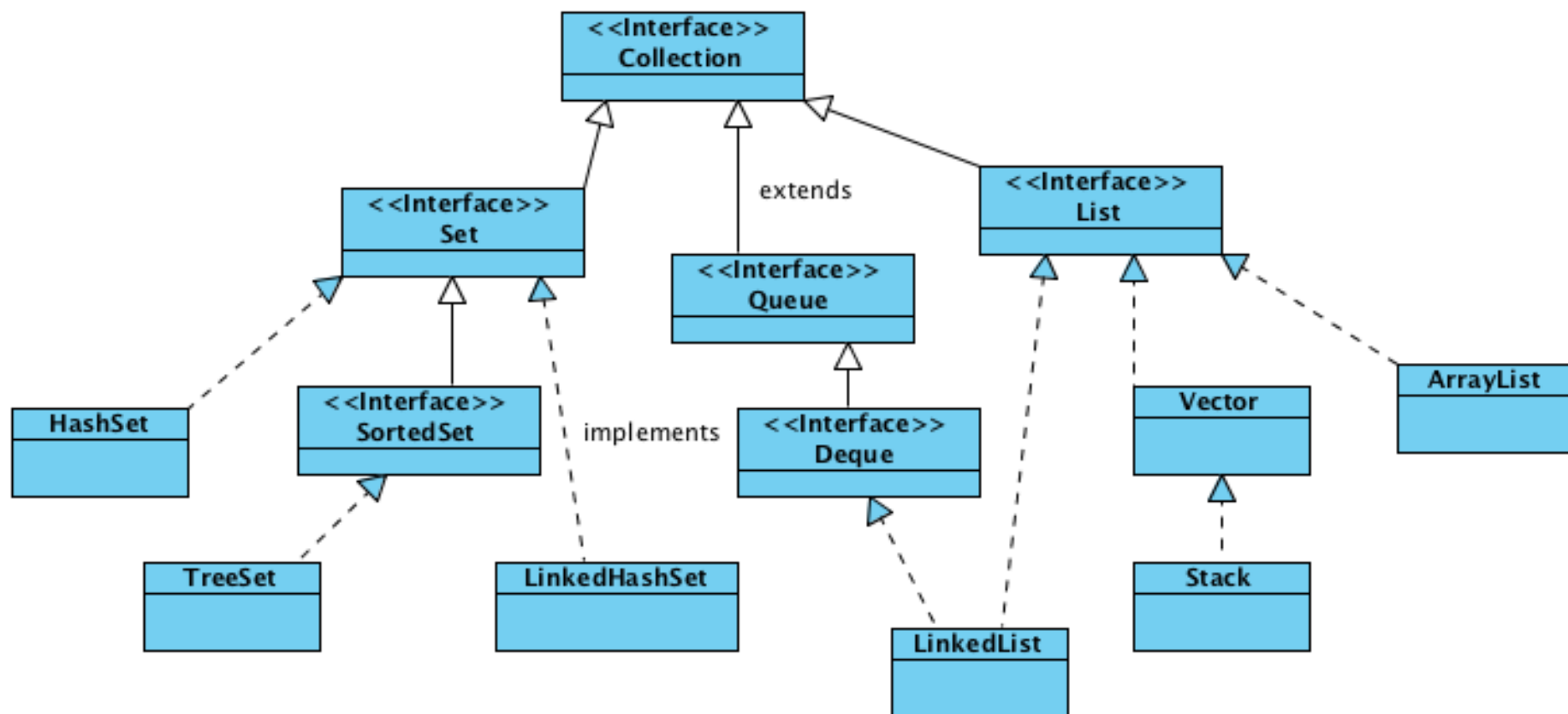
<http://dl.dropbox.com/u/3025380/prog2/aula3.pdf>

flavio.cec@unisul.br

Introdução

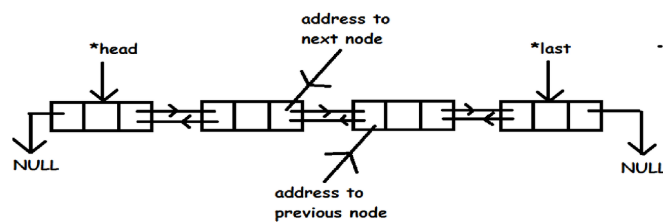
- As classes que implementam as estruturas de dados foram adicionadas ao Java a partir da versão 1.2. (Vector e HashTable);
- O Framework Java Collections foi inserido ao Java na versão do Java 5;
- Mais detalhes sobre a parte teórica podem ser encontrados no link abaixo:

<http://dl.dropbox.com/u/3025380/ED/aula17.pdf>





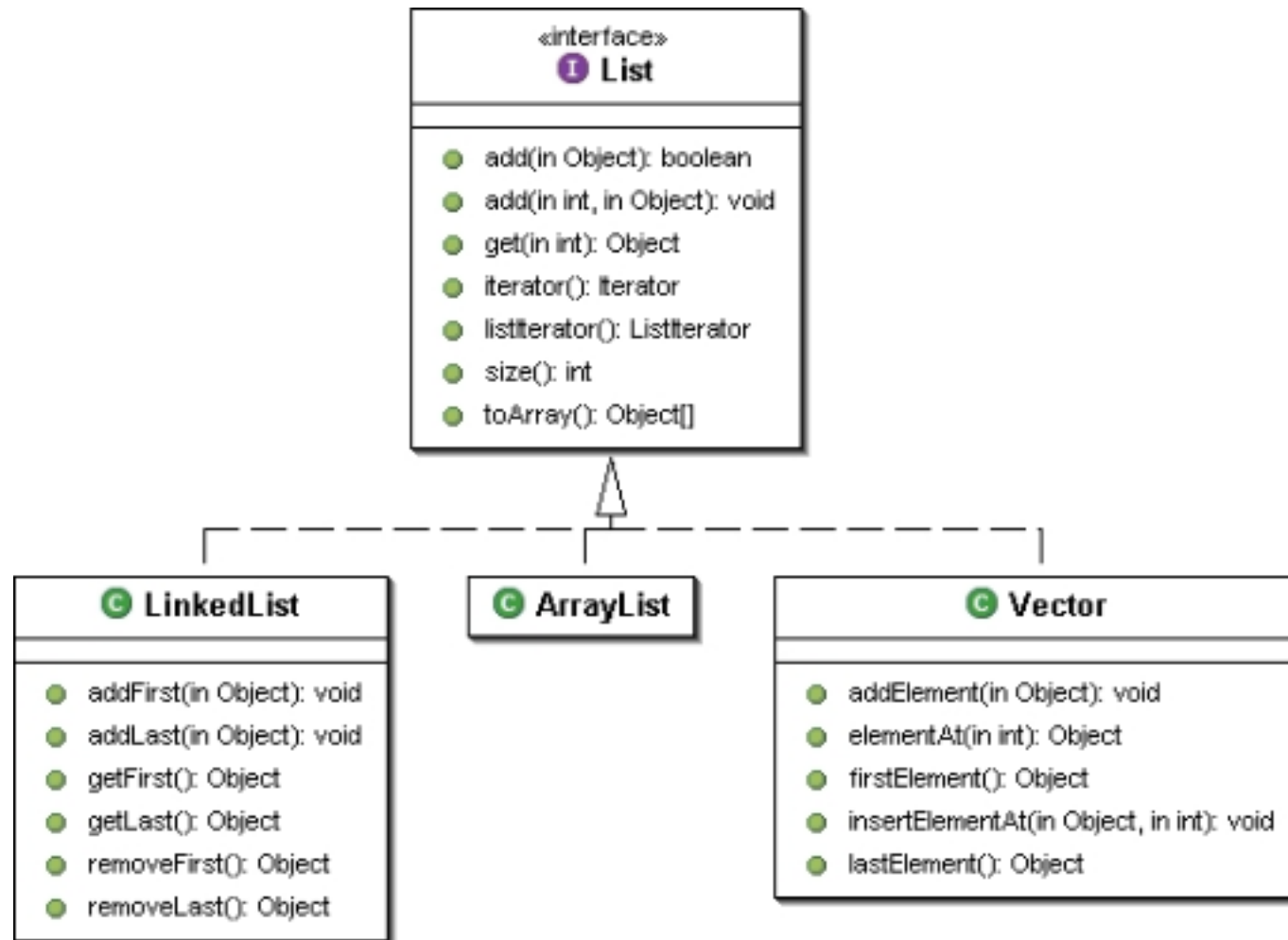
Listas



Listas (List)

- São estruturas lineares de armazenamento.
- Características:
 - Possuem um inicio e um fim;
 - Permitem operações em qualquer posição;
 - Não são permitidas lacunas entre os elementos;
 - Podem possuir disciplinas de acesso;
- Em estrutura de dados:
 - Listas sequenciais (arrays);
 - ArrayList
 - Listas Encadeadas;
 - LinkedList

Interface *List*



Fonte: <http://bit.ly/14lq0k4>

ArrayList

ArrayList

- Características:
 - Possui um **Array** encapsulado como estrutura de dados.
 - Permite acesso indexado;
 - Possui deslocamento em operações:
 - Início; e
 - Posições específicas;
 - Possui tamanho inicial definido (“fixo”);
 - Inicia com 10 posições e a medida que a quantidade se aproxima do valor total, um novo array é criado com tamanho maior;
 - Todos os elementos do array original são copiados para o novo array.

ArrayList

- A classe ArrayList implementa todos os métodos da interface List;
- Não possui características adicionais à interface.

```
List<String> dezValores = new ArrayList<String>(10);
```

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



Declarando como **List**



Instanciando como **ArrayList**

List

```
+add(element : Object) : boolean  
+add(index : int, element : Object) : void  
+addAll(collection : Collection) : boolean  
+addAll(index : int, collection : Collection) : boolean  
+clear() : void  
+contains(element : Object) : boolean  
+containsAll(collection : Collection) : boolean  
+equals(object : Object) : boolean  
+get(index : int) : Object  
+hashCode() : int  
+indexOf(element : Object) : int  
+iterator() : Iterator  
+lastIndexOf(element : Object) : int  
+listIterator() : ListIterator  
+listIterator(startIndex : int) : ListIterator  
+remove(element : Object) : boolean  
+remove(index : int) : Object  
+removeAll(collection : Collection) : boolean  
+retainAll(collection : Collection) : boolean  
+set(index : int, element : Object) : Object  
+size() : int  
+subList(fromIndex : int, toIndex : int) : List  
+toArray() : Object[]  
+toArray(array : Object[]) : Object[]
```

Adicionando valores

```
//Declarando e instanciando  
List<String> lista = new ArrayList<String>();
```

```
//Adicionando valores na lista  
lista.add("Aluno 1");  
lista.add("Aluno 2");  
lista.add("Aluno 3");
```

```
//Percorrendo os elementos da lista  
for(String valor: lista) {  
    System.out.println(valor);  
}
```

Pode-se utilizar as duas estratégias para varrer o array, na primeira utiliza-se o foreach (para-cada) e no segundo exemplo um for normal.

OU

```
for(int i = 0; i < lista.size(); i++) {  
    String valor = lista.get(i);  
    System.out.println(valor);  
}
```

Quantidade de elementos da lista

CONSOLE:

Properties Problems Console

<terminated> ArrayListTest [Java Application] /Sy

Aluno 1
Aluno 2
Aluno 3

Adicionando valores

- Posição específica:

```
//Declarando e instanciando  
List<String> lista = new ArrayList<String>();
```

```
//Adicionando valores na lista  
lista.add("Aluno 1");  
lista.add("Aluno 2");  
lista.add("Aluno 3");
```

add desta forma sempre adiciona no fim da lista.

```
//Adiciona o valor no índice 1.  
lista.add(1, "Novo aluno");
```

Posição do índice da lista

```
for(int i = 0; i < lista.size(); i++) {  
    //Recupera o valor do índice i  
    String valor = lista.get(i);  
    System.out.println(valor);  
}
```

CONSOLE

```
Properties Problems Console  
<terminated> ArrayListTest [Java Application] /  
Aluno 1  
Novo aluno  
Aluno 2  
Aluno 3
```

Recuperando valores

```
//Declarando e instanciando
List<String> lista = new ArrayList<String>();

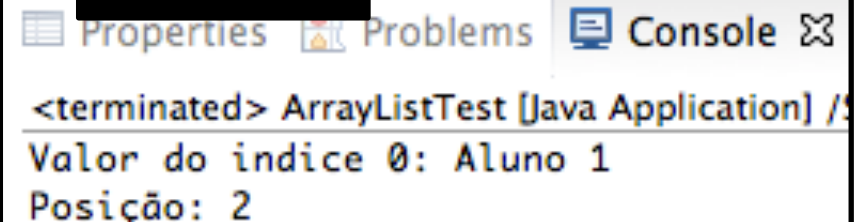
//Adicionando valores na lista
lista.add("Aluno 1");
lista.add("Aluno 2");
lista.add("Aluno 3");

//Recuperando o valor do índice 0 da lista
String valor = lista.get(0);
System.out.println("Valor do índice 0: "+valor);

//Verificando a posição em que o valor existe
int pos = lista.lastIndexOf("Aluno 3");
System.out.println("Posição: "+pos);
```

Caso não exista é
retornado -1.

CONSOLE:



The screenshot shows an IDE window with tabs for Properties, Problems, and Console. The Console tab is active, displaying the output of the Java application. The text in the console is as follows:

```
<terminated> ArrayListTest [Java Application] /s
Valor do índice 0: Aluno 1
Posição: 2
```

Removendo valores

```
//Declarando e instanciando
List<String> lista = new ArrayList<String>();

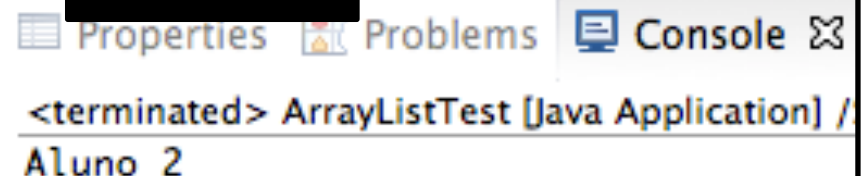
//Adicionando valores na lista
lista.add("Aluno 1");
lista.add("Aluno 2");
lista.add("Aluno 3");

//Remove o elemento da posição 0
lista.remove(0);

//Remove o elemento pelo seu valor.
lista.remove("Aluno 3");

//Percorrendo os elementos da lista
for(String valor: lista) {
    System.out.println(valor);
}
```

CONSOLE:

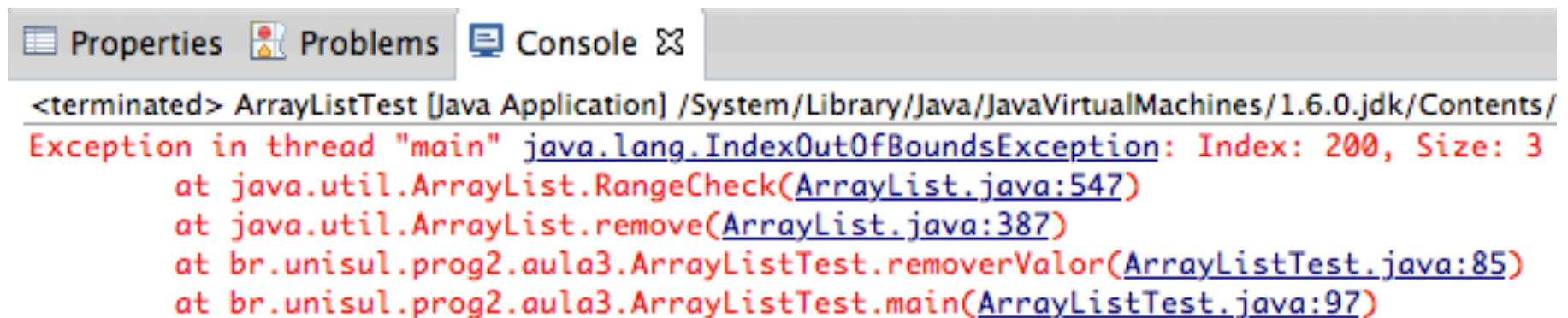


The screenshot shows a standard IDE console window with tabs for Properties, Problems, and Console. The Console tab is active, displaying the output of the Java application. The text in the console is as follows:

```
<terminated> ArrayListTest [Java Application] /
Aluno 2
```

Removendo valores

- Para remover no final é necessário recuperar o tamanho da lista e diminuir por 1 para chegar no último índice preenchido.
- Caso seja solicitado para remoção um índice não preenchido, uma ***exception*** é lançada:



The screenshot shows an IDE interface with tabs for Properties, Problems, and Console. The Console tab is active, displaying the following text:

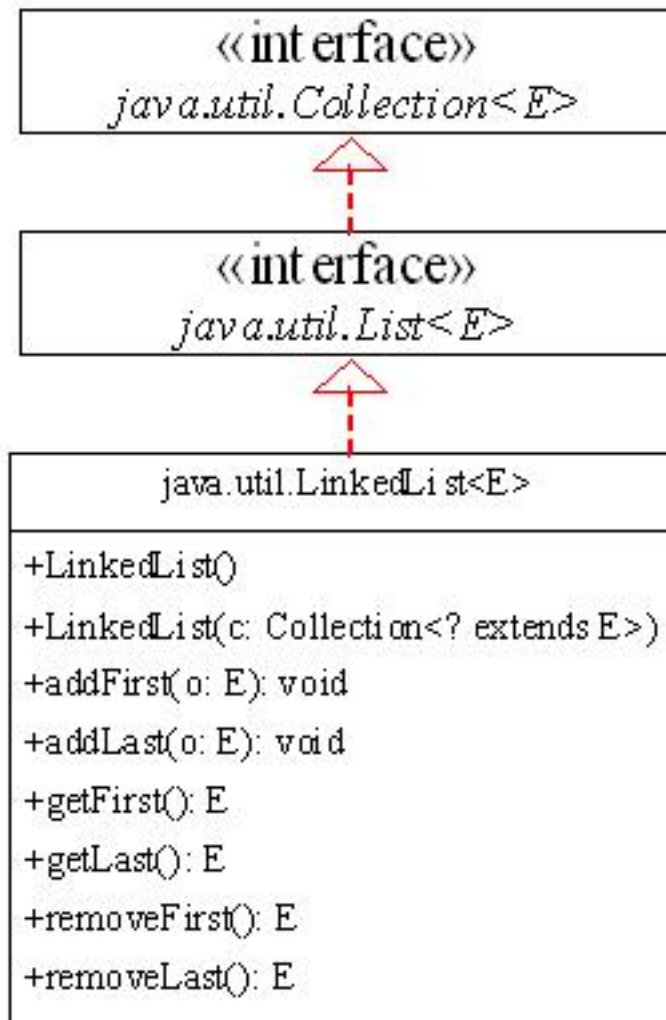
```
<terminated> ArrayListTest [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/  
Exception in thread "main" java.lang.IndexOutOfBoundsException: Index: 200, Size: 3  
    at java.util.ArrayList.RangeCheck(ArrayList.java:547)  
    at java.util.ArrayList.remove(ArrayList.java:387)  
    at br.unisul.prog2.aula3.ArrayListTest.removerValor(ArrayListTest.java:85)  
    at br.unisul.prog2.aula3.ArrayListTest.main(ArrayListTest.java:97)
```

LinkedList

LinkedList

- Características:
 - Encapsula uma lista duplamente encadeada com descritor;
 - Não possui acesso indexado;
 - Para acessar uma posição específica é necessário varrer a lista a partir da referência início.
 - Para operações na lista são feitas apenas trocas de apontamento das referências;
 - Não possui tamanho inicial (“fixo”).

LinkedList



```
//Declarando e instanciando
List<String> lista = new LinkedList<String>();
```

Creates a default empty linked list.

Creates a linked list from an existing collection.

Adds the object to the head of this list.

Adds the object to the tail of this list.

Returns the first element from this list.

Returns the last element from this list.

Returns and removes the first element from this list.

Returns and removes the last element from this list.

Adicionando valores

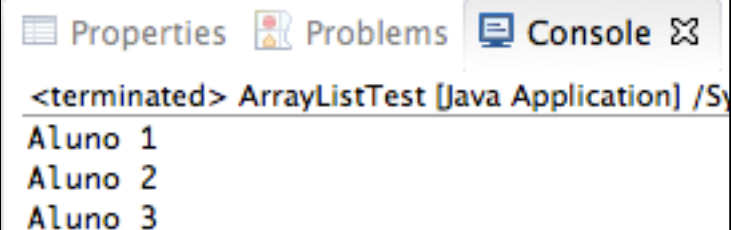
- Todos os métodos que o ArrayList implementa, também são implementados pelo LinkedList, ou seja, podem ser utilizados na mesma forma:

```
//Declarando e instanciando  
List<String> lista = new LinkedList<String>();
```

```
//Adicionando valores na lista  
lista.add("Aluno 1");  
lista.add("Aluno 2");  
lista.add("Aluno 3");
```

```
for(int i = 0; i < lista.size(); i++) {  
    String valor = lista.get(i);  
    System.out.println(valor);  
}
```

CONSOLE:



```
<terminated> ArrayListTest [Java Application] /Sy  
Aluno 1  
Aluno 2  
Aluno 3
```

Adicionando valores

```
//Declarando e instanciando
LinkedList<String> lista = new LinkedList<String>();

//Adicionando valores na lista
lista.add("Aluno 1");
lista.add("Aluno 2");
lista.add("Aluno 3");

//Adiciona no inicio
lista.addFirst("Aluno inicio");

//Adiciona no fim
lista.addLast("Aluno fim");

for(String valor : lista) {
    System.out.println(valor);
}
```

Para se acessar os métodos addFirst e addLast é necessário que se declare a lista como LinkedList.

CONSOLE

Properties Problems Console

<terminated> LinkedListTest [Java Application]

Aluno inicio
Aluno 1
Aluno 2
Aluno 3
Aluno fim

Adicionando valores

```
public List<String> funcaoAddLinkedList() {  
    //Declarando e instanciando  
    LinkedList<String> lista = new LinkedList<String>();  
  
    //Adicionando valores na lista  
    lista.add("Aluno 1");  
    lista.add("Aluno 2");  
    lista.add("Aluno 3");  
  
    //Adiciona no inicio  
    lista.addFirst("Aluno inicio");  
  
    //Adiciona no fim  
    lista.addLast("Aluno fim");  
  
    for(String valor : lista) {  
        System.out.println(valor);  
    }  
  
    return lista;  
}
```

No caso de uma função deve-se sempre retornar a interface:

Removendo valores

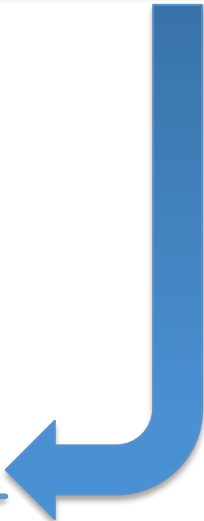
```
//Declarando e instanciando
LinkedList<String> lista = new LinkedList<String>();

//Adicionando valores na lista
lista.add("Aluno 1");
lista.add("Aluno 2");
lista.add("Aluno 3");
lista.addFirst("Aluno inicio");
lista.addLast("Aluno fim");

//Remove no inicio (Aluno inicio)
lista.removeFirst();

//Remove no fim (Aluno fim)
lista.removeLast();
//Removendo pos. especifica (Aluno 2)
lista.remove(1);

for(String valor : lista) {
    System.out.println(valor);
}
```



CONSOLE:

Properties Problems Console

<terminated> LinkedListTest [Java Application]

Aluno 1
Aluno 3

Recuperando valores

```
//Declarando e instanciando  
LinkedList<String> lista = new LinkedList<String>();
```

```
//Adicionando valores na lista  
lista.add("Aluno 1");  
lista.add("Aluno 2");  
lista.add("Aluno 3");
```

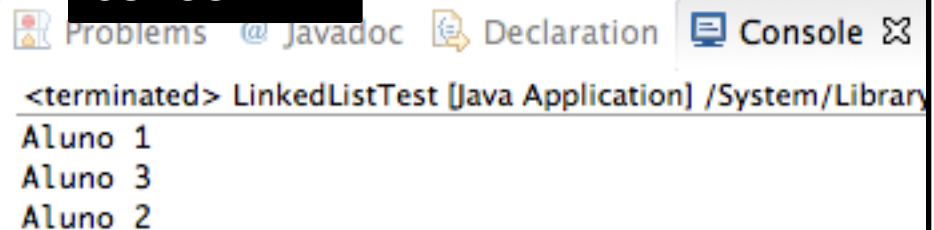
```
String valor = "";
```

```
//Recupera o valor do inicio;  
valor = lista.getFirst();  
System.out.println(valor);
```

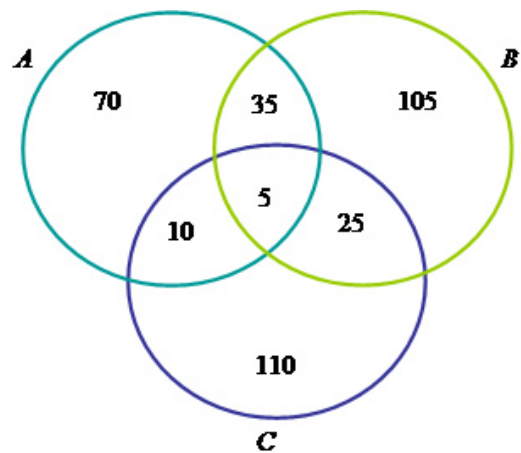
```
//Recupera o valor do fim;  
valor = lista.getLast();  
System.out.println(valor);
```

```
//Recupera valor da pos esp.  
valor = lista.get(1);  
System.out.println(valor);
```

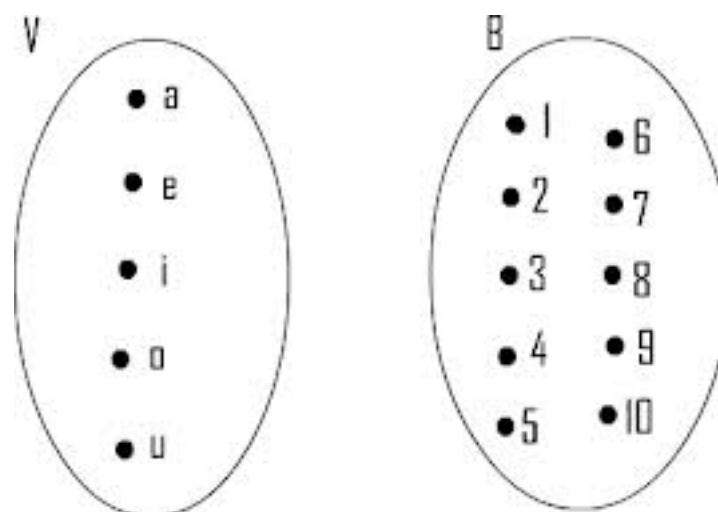
CONSOLE:



```
<terminated> LinkedListTest [Java Application] /System/Library  
Aluno 1  
Aluno 3  
Aluno 2
```

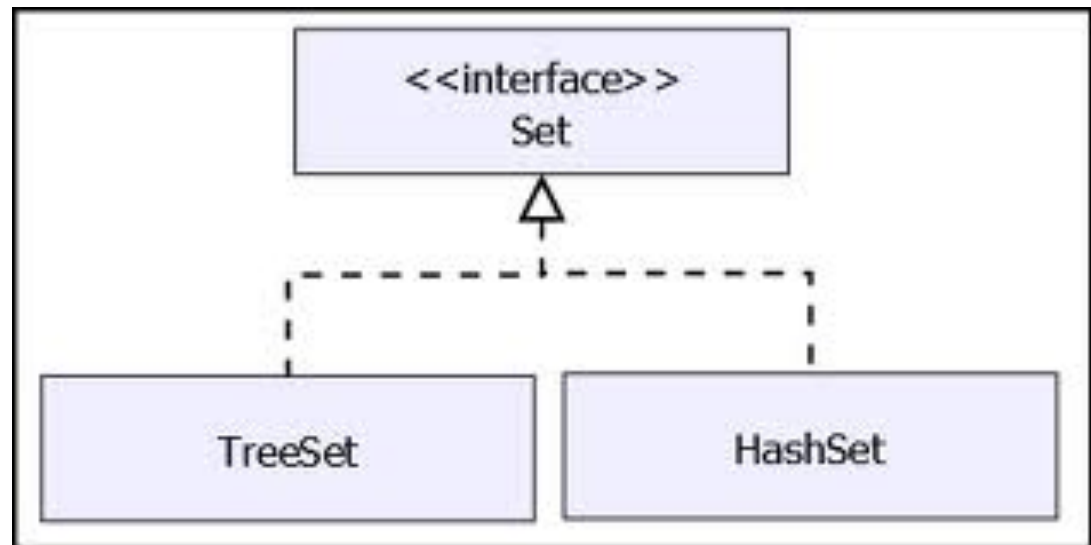


Conjuntos

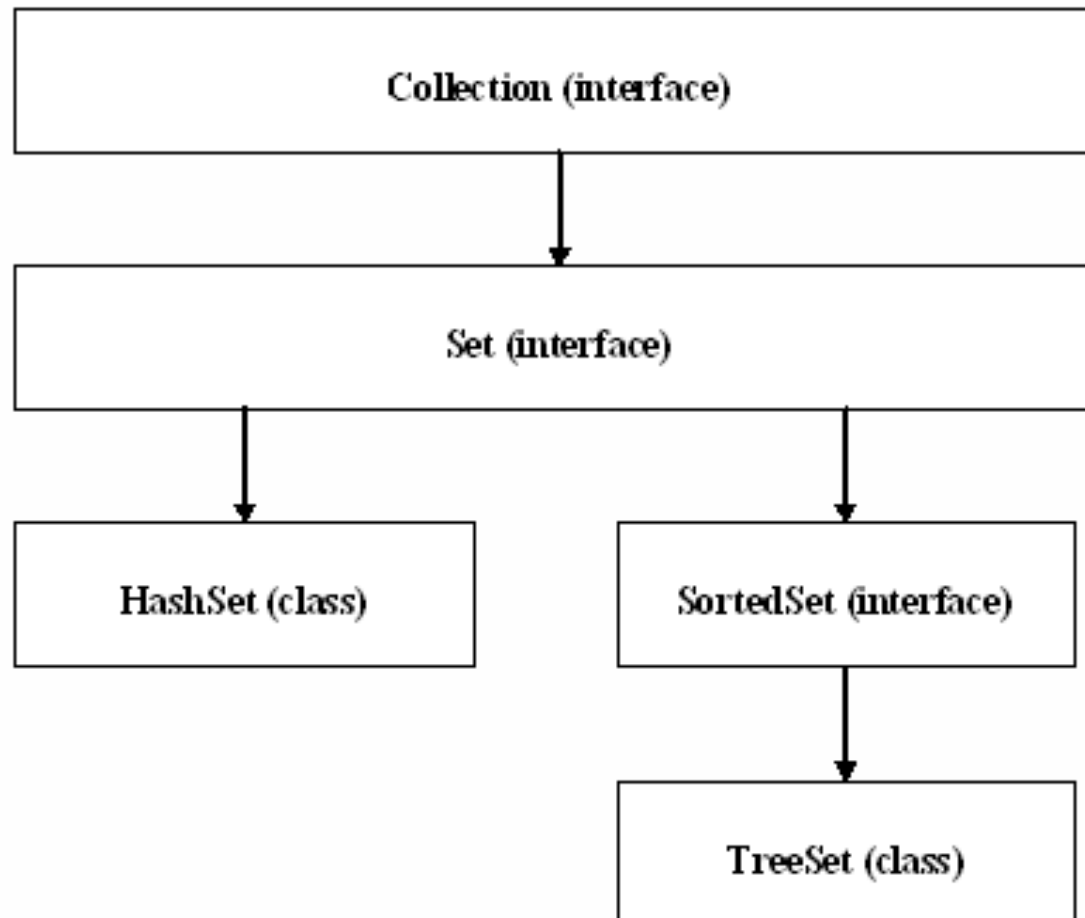


Conjuntos (Set)

- Os conjuntos são estruturas que não permitem elementos repetidos.
- Eles **não** possuem características lineares.
- Podem ser implementados a partir de:
 - Tabelas Hash; ou
 - Árvores.



Interface *Set*



Set and HashSet Hierarchy

Por que somente a implementação na forma de árvore pode ser ordenada?

Set

- Relembrando arvores:

- **Pré-Ordem:**

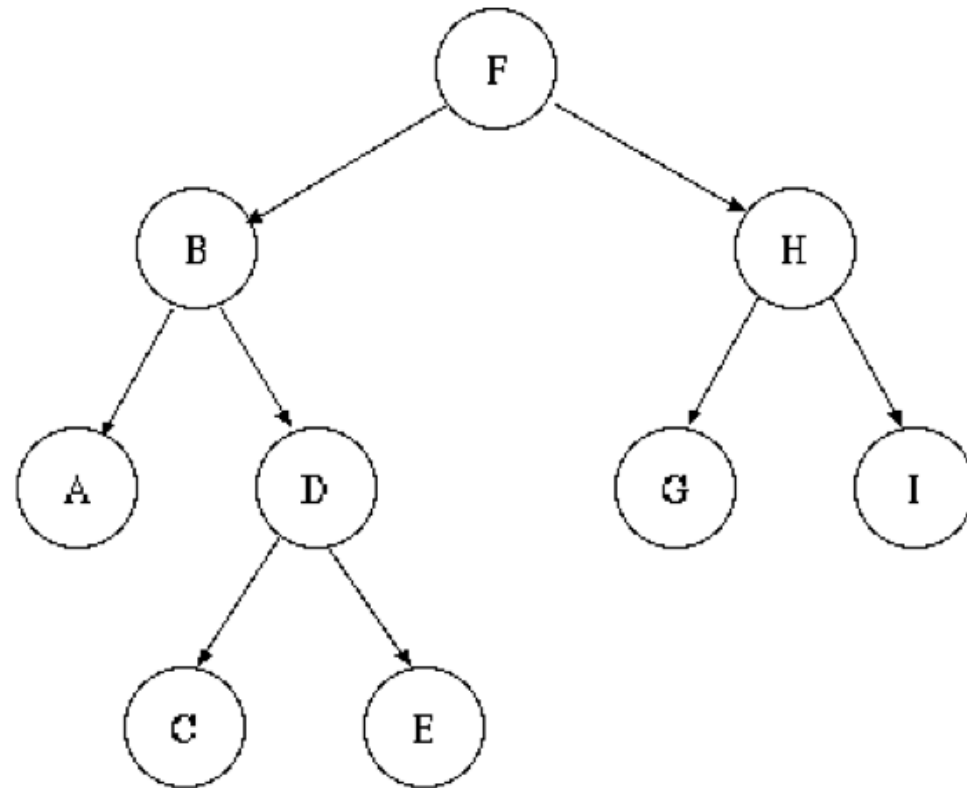
– F B A D C E H G I

- **Central:**

– A B C D E F G H I

- **Pós-Ordem:**

– A C E D B G I H F



Set

- Relembrando Hash:
Valores para teste: 72, 2, 29, 25, 82, 100, 85, 42, 39, 45

The screenshot shows a Java IDE console window titled "Entrada" with a menu bar (File, Edit, Navigate, Search, Project, Run, Window, Help) and a toolbar. The console output displays a hash table structure for "PrincipalHash [Java Application] /usr/lib/jvm/java-7-oracle/bin/java (28/05/2013 11:27:22)".

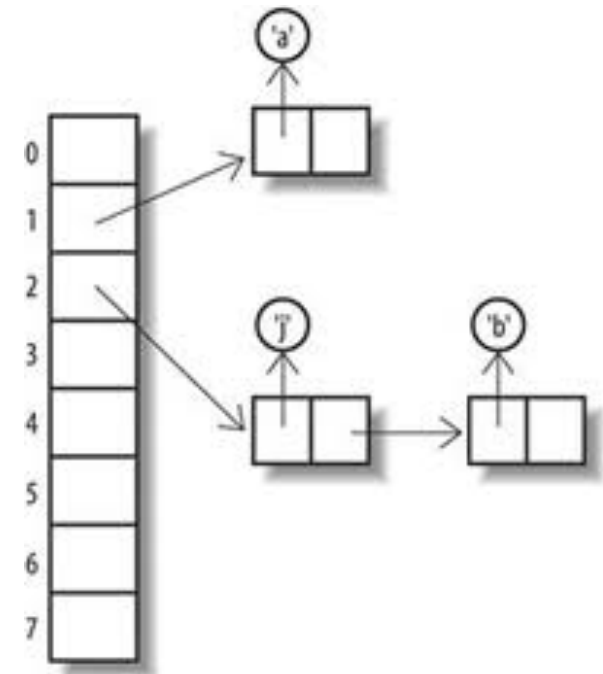
Indice hash:	Valores:
[0]	{75->25->100->nulo}
[1]	{nulo}
[2]	{2->nulo}
[3]	{nulo}
[4]	{29->nulo}
[5]	{nulo}
[6]	{nulo}
[7]	{82->nulo}
[8]	{nulo}
[9]	{nulo}
[10]	{85->nulo}
[11]	{nulo}
[12]	{nulo}
[13]	{nulo}
[14]	{39->nulo}
[15]	{nulo}
[16]	{nulo}
[17]	{42->nulo}
[18]	{nulo}
[19]	{nulo}
[20]	{45->nulo}
[21]	{nulo}
[22]	{nulo}
[23]	{nulo}
[24]	{nulo}

Overlaid on the console is a small dialog box titled "Entrada" with a question mark icon. It contains the text "Escolha a opção desejada:" followed by a list of options: "1 - Para inserir um valor na Hash", "2 - Para remover um valor da Hash", "3 - Para localizar um valor na Hash", and "0 - Para sair". Below the list is a text input field and two buttons labeled "OK" and "Cancelar".

HashSet

HashSet

- Conjunto implementado como **Hash**:
 - Utiliza uma lista sequencial como tabela de endereços;
 - Utiliza uma função para fazer o “espalhamento” dos valores;
 - Utiliza uma lista encadeada para tratar as colisões;
 - Não mantem a ordem de inserção dos valores;



HashSet

- Características:
 - Possui um excelente tempo de inserção dos valores;
 - Indicado para acesso a dados pelo seu valor;
 - Para operações de “localizar”;
 - Estrutura muito utilizada pelos Sistemas gerenciadores de banco de dados (SGBD);

Adicionando valores

```
//Declarando e intanciando
Set<String> conjunto = new HashSet<String>();

//Adicionando os valores ao conjunto
conjunto.add("Aluno A");
conjunto.add("Aluno B");
conjunto.add("Aluno C");

//Listando os valores via foreach
for(String valor : conjunto) {
    System.out.println(valor);
}
```

Pode-se informar na construção da classe **HashSet** a quantidade De endereços disponíveis.

LEMBRE-SE: O uso do HashSet não garante a ordem de inserção dos elementos.

CONSOLE:

Problems @ Javadoc Console

<terminated> HashSetTest [Java Application]

Aluno B
Aluno A
Aluno C

Localizando um valor

```
//Declarando e intanciando
Set<String> conjunto = new HashSet<String>();

//Adicionando os valores ao conjunto
conjunto.add("Aluno A");
conjunto.add("Aluno B");
conjunto.add("Aluno C");

//Verificando se o valor existe no conjunto
boolean possui = conjunto.contains("Aluno B");

if(possui) {
    System.out.println("Elemento encontrado");
} else {
    System.out.println("Elemento não encontrado");
}
```

CONSOLE:



Recuperando valores

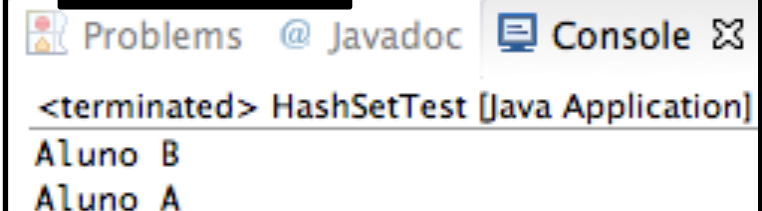
- Não é possível acessar um elemento via método get.
- É permitido apenas iterar sobre todo o conjunto.
 - Utilizando o foreach.

```
//Listando os valores via foreach  
for(String valor : conjunto) {  
    System.out.println(valor);  
}
```

Removendo valores

```
public void removendoValores() {  
  
    //Declarando e intanciando  
    Set<String> conjunto = new HashSet<String>();  
  
    //Adicionando os valores ao conjunto  
    conjunto.add("Aluno A");  
    conjunto.add("Aluno B");  
    conjunto.add("Aluno C");  
  
    //Removendo o elemento pelo seu valor  
    conjunto.remove("Aluno C");  
  
    //Listando os valores via foreach  
    for(String valor : conjunto) {  
        System.out.println(valor);  
    }  
}
```

CONSOLE:



The screenshot shows an IDE console window with three tabs: 'Problems', '@ Javadoc', and 'Console'. The 'Console' tab is active, displaying the output of a Java application. The output shows the text '<terminated> HashSetTest [Java Application]' followed by two lines: 'Aluno B' and 'Aluno A'. This indicates that the element 'Aluno C' was successfully removed from the set before the final iteration of the loop.

```
<terminated> HashSetTest [Java Application]  
Aluno B  
Aluno A
```

TreeSet

TreeSet

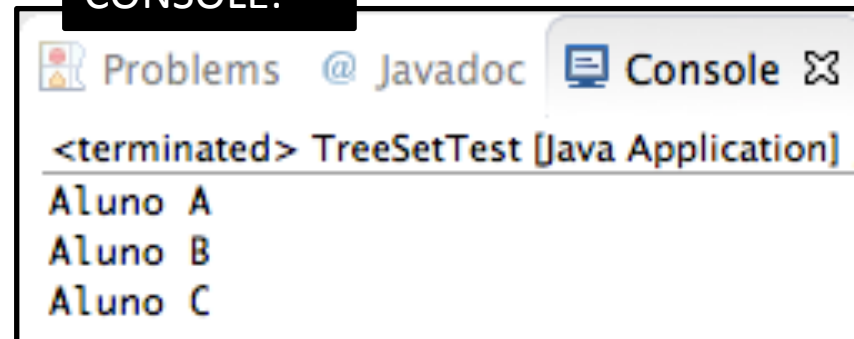
- O uso dos métodos apresentados no HashSet são ou mesmos para o TreeSet:

```
//Declarando e intanciando  
Set<String> conjunto = new TreeSet<String>();
```

```
//Adicionando os valores ao conjunto  
conjunto.add("Aluno A");  
conjunto.add("Aluno B");  
conjunto.add("Aluno C");
```

```
//Listando os valores via foreach  
for(String valor : conjunto) {  
    System.out.println(valor);  
}
```

CONSOLE:



The screenshot shows an IDE console window with three tabs: 'Problems', '@ Javadoc', and 'Console'. The 'Console' tab is active, displaying the output of a Java application named '<terminated> TreeSetTest [Java Application]'. The output consists of three lines: 'Aluno A', 'Aluno B', and 'Aluno C', which are the elements added to the TreeSet in the code above.

```
<terminated> TreeSetTest [Java Application]  
Aluno A  
Aluno B  
Aluno C
```

TreeSet implements *SortedSet*

- Adicionando valores:

```
//Declarando e intanciando
SortedSet<String> conjunto = new TreeSet<String>();

//Adicionando os valores ao conjunto
conjunto.add("Aluno C");
conjunto.add("Aluno A");
conjunto.add("Aluno B");

//Listando os valores via foreach
for(String valor : conjunto) {
    System.out.println(valor);
}
```

Uso da interface SortedSet, já garante a ordenação na forma crescente

CONSOLE:

```
<terminated> TreeSetTest [Java Application]
Aluno A
Aluno B
Aluno C
```

TreeSet implements *SortedSet*

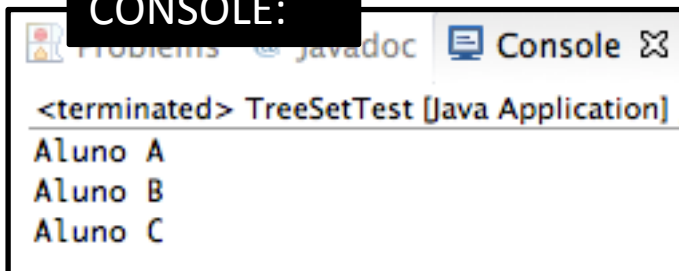
- Exemplo do uso do Iterator

```
//Declarando e intanciando  
SortedSet<String> conjunto = new TreeSet<String>();
```

```
//Adicionando os valores ao conjunto  
conjunto.add("Aluno C");  
conjunto.add("Aluno A");  
conjunto.add("Aluno B");
```

```
//Listando os valores via Iterator  
Iterator<String> it = conjunto.iterator();  
while (it.hasNext()) {  
    String elemento = it.next();  
    System.out.println(elemento);  
}
```

CONSOLE:



```
<terminated> TreeSetTest [Java Application]  
Aluno A  
Aluno B  
Aluno C
```

Iterator pode ser aplicado
para qualquer estrutura
filha de Collections



Mapas

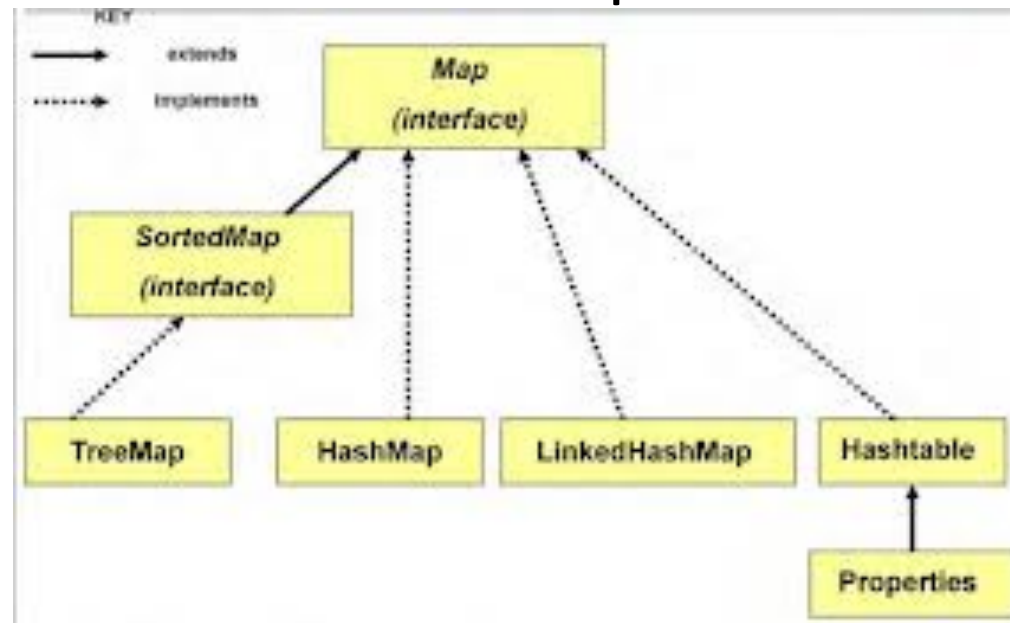


Mapas

- Estruturas de dados baseado na combinação:
 - Chave -> valor;
- Onde:
 - Para cada valor deve-se ter uma chave diferente;
 - Elementos com a mesma chave são sobrescritos;
- Podem ser implementas utilizando:
 - Hash; e
 - Arvores.

Mapas (Map)

- Pode-se utilizar como chave e valor, qualquer tipo de Objeto.
- Para tipos abstratos é necessário:
 - Implementar o método equals e hashCode.



Mapas (Maps)

- Indicados para:
 - Recuperação de elementos por uma chave;
 - Acesso muito rápido à elementos;
- É possível:
 - Pode-se recuperar as chaves (na forma de um conjunto);
 - Pode-se recuperar apenas os valores (na forma de uma lista);

HashMap

- É uma mapa utilizando a estrutura de dados Hashing;
- Recebe todas as características de uma implementação de Hash com as ações esperadas para um mapa.
- Exemplo de inserção de valores:

```
//Declarando e instanciando o Mapa.
```

```
Map<String, String> mapa = new HashMap<String, String>();
```

```
//Adicionando as chaves e os valores
```

```
mapa.put("Chave1", "Valor da chave 1");
```

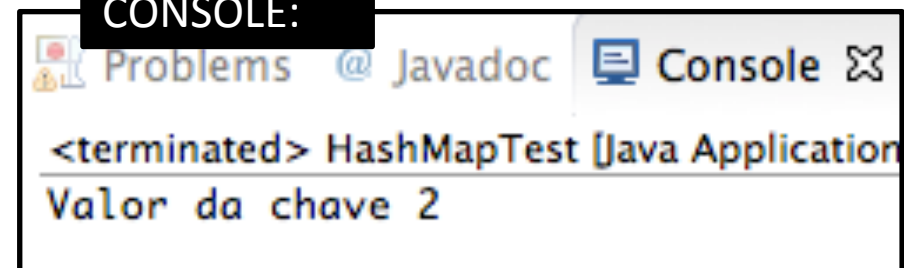
```
mapa.put("Chave2", "Valor da chave 2");
```

```
mapa.put("Chave3", "Valor da chave 3");
```

Localização de um elemento

```
//Declarando e instanciando o Mapa.  
Map<String, String> mapa = new HashMap<String, String>();  
  
//Adicionando as chaves e os valores  
mapa.put("Chave1", "Valor da chave 1");  
mapa.put("Chave2", "Valor da chave 2");  
mapa.put("Chave3", "Valor da chave 3");  
  
//Recuperando o elemento que possui a Chave2  
String valor = mapa.get("Chave2");  
  
System.out.println(valor);
```

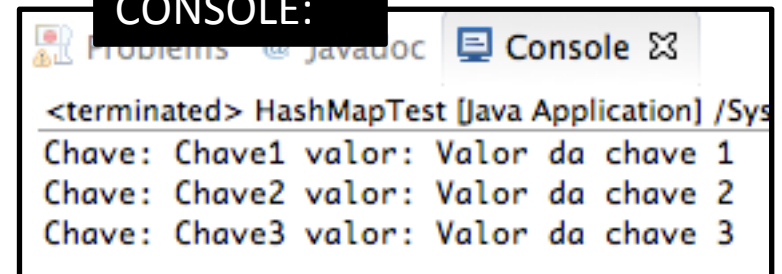
CONSOLE:



Listando todos os elementos

```
//Declarando e instanciando o Mapa.  
Map<String, String> mapa = new HashMap<String, String>();  
  
//Adicionando as chaves e os valores  
mapa.put("Chave1", "Valor da chave 1");  
mapa.put("Chave2", "Valor da chave 2");  
mapa.put("Chave3", "Valor da chave 3");  
  
//Recupera todas as chaves (conjunto)  
Set<String> chaves = mapa.keySet();  
  
//Varre todas as chaves do conjunto  
for(String chave : chaves) {  
  
    //Recupera o valor do mapa  
    String valor = mapa.get(chave);  
    System.out.println("Chave: "+chave+" valor: "+valor);  
}
```

CONSOLE:



```
<terminated> HashMapTest [Java Application] /Sys  
Chave: Chave1 valor: Valor da chave 1  
Chave: Chave2 valor: Valor da chave 2  
Chave: Chave3 valor: Valor da chave 3
```

TreeMap

- Recebe todas as características de uma árvore;
- Permite todas as operações de um mapa;
- Sua implementação e utilização para ordenação é muito similar a do TreeSet.
- Indicado:
 - Para mapas em que precisa-se de ordenação das chaves.

Exercício

Exercício



Um Pub solicitou o desenvolvimento de um sistema para auxiliar no sorteio de chopp para os clientes que estejam no estabelecimento.



Exercício

- Como funciona a promoção:
 - Todo cliente que ingressar no Pub, deve fornecer o seu número de celular (somente os números) para a atendente.
 - De tempos em tempos a atendente vai solicitar o sistema para que o mesmo sorteie um número de celular.
 - Após sorteado o número deve ser retirado da lista, garantindo que sempre serão sorteadas pessoas diferentes.

Exercício

- Requisitos do sistema:
 - Todo novo cliente que ingressar no Pub deve ser adicionado ao final da lista;
 - O sistema deve permitir a baixa do cliente quando o mesmo pagar a conta;
 - Deve-se localizar se o mesmo está na lista e remover o número do celular;
 - Caso o número do celular não exista na lista, deve-se informar que ele não foi encontrado.

Exercício

- Requisitos do sistema:
 - O sistema deve sortear os números disponíveis na lista, para isso deve-se utilizar a classe Random do Java a partir do índice da lista.
 - Ao sortear um número de celular o mesmo deve ser removido da lista;
 - Caso o sistema não possua nenhum número de celular cadastrado e seja solicitado um novo sorteio ou a baixa de um cliente, deve-se apresentar um mensagem informando que a lista está vazia.

Exercício

- Requisitos do sistema:
 - Deve-se escolher uma das Collections do Java para implementar estas operações;
 - Procure a estrutura mais adequada para esta operação.
- Exemplo do uso da classe Random:

```
Random randomico = new Random();  
int aux = randomico.nextInt(6);  
  
System.out.println(aux);
```