Programação Orientada a Objetos I

CÁSSIO CAPUCHO PEÇANHA – 09

Coleções

- Uma coleção é uma estrutura de dados que permite armazenar vários objetos.
- Em Java, a coleção também é um objeto.
- As operações que podem ser feitas em coleções variam mas normalmente incluem:
  - Adição de elementos;
  - Remoção de elementos;
  - Acesso aos elementos;
  - Pesquisa de elementos;

Coleções

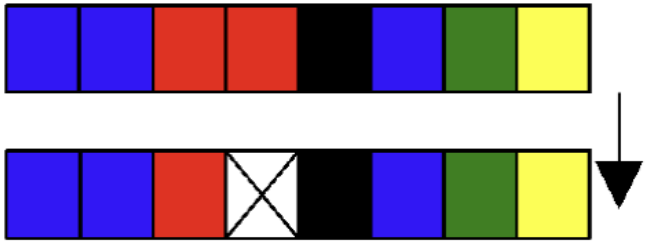
- Dependendo da forma de fazer as 4 operações básicas (adição, remoção, acesso e pesquisa), teremos vários tipos de coleções
- Os três grandes tipos de coleções são:
  - Lista (também chamado de “sequência“);
  - Conjunto;
  - Mapa(também chamado de “dicionário“).

Coleções

- Manipular arrays é bastante trabalhoso. Essa dificuldade aparece em diversos momentos:
  - Não podemos redimensionar um array em Java;
  - É impossível buscar diretamente por um determinado elemento cujo índice não se sabe;
  - Não conseguimos saber quantas posições do array já foram “populadas”

Coleções

- Supondo que os dados armazenados representem contas, o que acontece quando precisarmos inserir uma nova conta no banco?
  - Precisaremos procurar por um espaço vazio?
  - Guardaremos em alguma estrutura de dados externa, as posições vazias?
  - E se não houver espaço vazio?
  - Teríamos de criar um array maior e copiar os dados do antigo para ele?

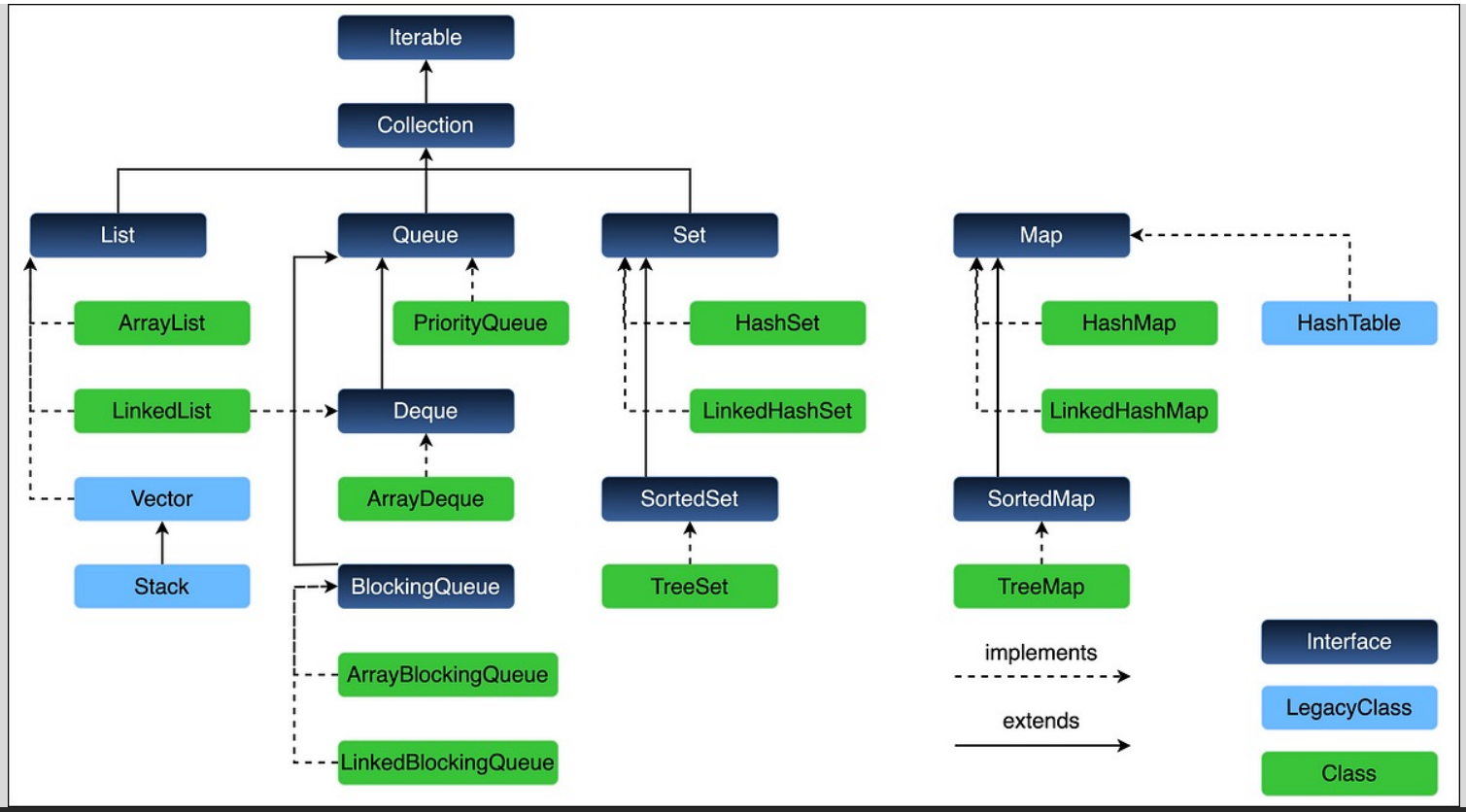


Retire a quarta Conta

conta[3] = **null**;

Coleções

- Com esses e outros objetivos em mente, foi criado um conjunto de classes e interfaces conhecido como Collections Framework.
- A API do Collections é robusta e possui diversas classes que representam estruturas de dados avançadas.


Iterable
Collection
List
Queue
Set
Map
ArrayList
PriorityQueue
HashSet
HashMap
HashTable
LinkedList
Deque
LinkedHashSet
LinkedHashMap
Vector
ArrayDeque
SortedSet
SortedMap
Stack
BlockingQueue
TreeSet
TreeMap
ArrayBlockingQueue
LinkedBlockingQueue
implements
extends
Interface
LegacyClass
Class

Coleções

Coleções - Listas

- 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.
- Em outras palavras, você tem a garantia de que, quando percorrer a lista, os elementos serão encontrados em uma ordem pré-determinada, definida hora da inserção dos mesmos.
- Lista resolve todos os problemas que levantamos em relação ao array (busca, remoção, tamanho “infinito”,...).

Coleções - Listas

- A API de Collections traz a interface java.util.List, que específica o que uma classe deve ser capaz de fazer para ser uma lista.
- Há diversas implementações disponíveis, cada uma com uma forma diferente de representar uma lista.
- A implementação mais utilizada da interface List é a ArrayList, que trabalha com um array interno para gerar uma lista.
  - ArrayList é mais rápida na pesquisa do que sua concorrente.
  - LinkedList é mais rápida na inserção e remoção de itens nas pontas.

Coleções - Listas

ArrayList não é um Array!

- 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 você não tem como acessá-lo.
- Repare, também, que você não pode usar [] com uma ArrayList, nem acessar atributo length. Não há relação!

Coleções - Listas

- Coleções indexadas (ordem é importante):
  - ArrayList: usa vetores (desempenho geral melhor);
  - LinkedList: usa lista encadeada (mais rápida na insertção e remoção nas pontas).

- Destaques da API:
  - add(Object), add(int, Object), addAll(Collection);
  - clear(), remove(int), removeAll(Collection);
  - contains(Object), containsAll(Collection);
  - get(int), indexOf(Object), set(int, Object);
  - isEmpty(), toArray(), subList(int, int),
  - size().

Coleções - Listas

```java
import java.util.*;
public class Teste {
    public static void main(String[] args) {
        List impares = new ArrayList();
        impares.add(1); impares.add(3); impares.add(5);

        List pares = new LinkedList();
        pares.add(2); pares.add(4); pares.add(6);

        for (int i = 0; i < impares.size(); i++)
            System.out.println(impares.get(i));

        for (int i = 0; i < pares.size(); i++)
            System.out.println(pares.get(i));
    }
}
```

Coleções - Listas

- Para criar um ArrayList, basta chamar o construtor:
  - ArrayList lista = new ArrayList();
- É sempre possível abstrair a lista a partir da interface List:
  - List lista = new ArrayList();
- Para criar uma lista de nomes (String), podemos fazer:
  - List lista = new ArrayList();
  - lista.add("Manoel");
  - lista.add("Joaquim");
  - lista.add("Maria");

Coleções - Listas

- A interface List possui dois métodos add:
  - recebe o objeto a ser inserido e o coloca no final da lista
    - lista.add("Manoel");
  - permite adicionar o elemento em qualquer posição da lista
    - lista.add(2,"Manoel");
- Note que, em momento algum, dizemos qual é o tamanho da lista

Coleções - Listas

- Toda lista (na verdade, toda Collection) trabalha do modo mais genérico possível.
- Todos os métodos trabalham com Object.
  - ContaCorrente c1 = new ContaCorrente();
  - c1.deposita(100);
  - ContaCorrente c2 = new ContaCorrente();
  - c2.deposita(200);
  - List contas = new ArrayList();
  - contas.add(c1);
  - contas.add(c2);

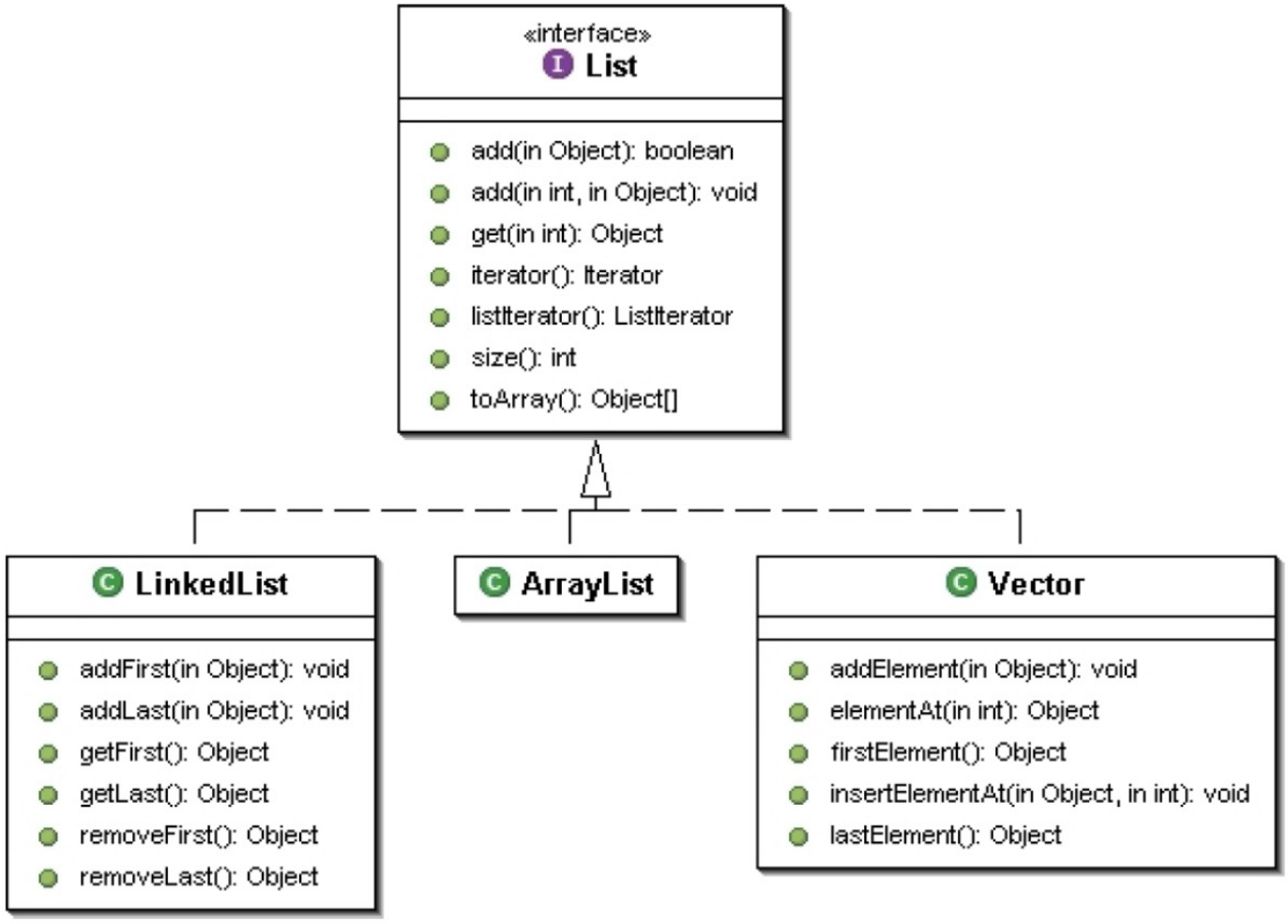
Coleções - Listas

- Para saber quantos elementos há na lista, usamos o método size():
- System.out.println(contas.size());
- Há ainda um método get(int) que recebe como argumento o índice do elemento que se quer recuperar. Através dele, podemos fazer um for para iterar na lista de contas:
  - for (int i = 0; i < contas.size(); i++) {
  - contas.get(i); // código não muito útil....
  - }
- Mas como fazer para imprimir o saldo dessas contas?

Coleções - Listas

- Podemos acessar o getSaldo() diretamente após fazer contas.get(i)?
- Não podemos; lembre-se que toda lista trabalha sempre com Object. Assim, a referência devolvida pelo get(i) é do tipo Object, sendo necessário o cast para ContaCorrente se quisermos acessar o getSaldo()

```
for (int i = 0; i < contas.size(); i++) {
ContaCorrente cc = (ContaCorrente) contas.get(i);
System.out.println(cc.getSaldo());
}// note que a ordem dos elementos não é alterada
```


«interface»
List
add(in Object): boolean
add(in int, in Object): void
get(in int): Object
iterator(): Iterator
listIterator(): ListIterator
size(): int
toArray(): Object[]
LinkedList
addFirst(in Object): void
addLast(in Object): void
getFirst(): Object
getLast(): Object
removeFirst(): Object
removeLast(): Object
ArrayList
Vector
addElement(in Object): void
elementAt(in int): Object
firstElement(): Object
insertElementAt(in Object, in int): void
lastElement(): Object

Coleções - Listas

Accesso Aleatório e Percorrendo Listas com Get

- Algumas listas, como a ArrayList, têm acesso aleatório aos seus elementos:
  - A busca por um elemento em uma determinada posição é feita de maneira imediata, sem que a lista inteira seja percorrida (que chamamos de acesso sequencial).
  - Neste caso, o acesso através do método get(int) é muito rápido.
  - Caso contrário, percorrer uma lista usando um for como esse que acabamos de ver, pode ser desastroso.

Coleções - Conjuntos

- Coleções não indexadas sem duplicação (não pode haver dois objetos iguais):
  - HashSet: usa tabela hash (dispersão);
  - TreeSet: usa árvore e é ordenado (Comparable).

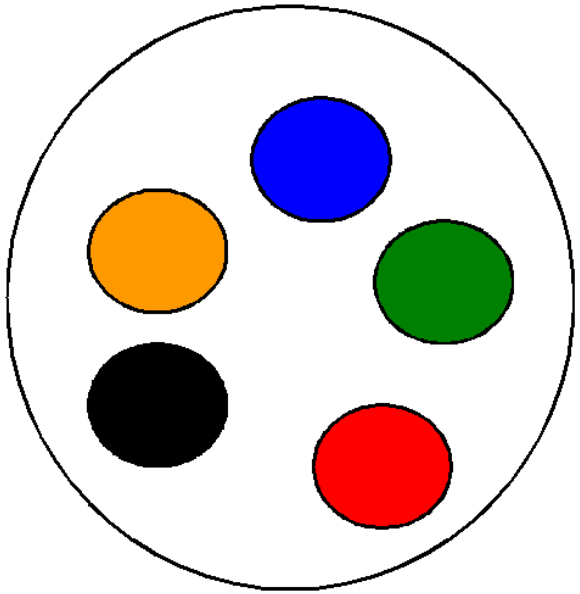
- Destaques da API:
  - add(Object), addAll(Collection);
  - clear(), remove(int), removeAll(Collection),
  - retainAll(Collection);
  - contains(Object), containsAll(Collection);
  - isEmpty(), toArray(), size().

Coleções - Conjuntos

Outro recurso que a API de Collections traz são os conjuntos (Set) ou Coleções não indexadas.

Um conjunto (Set) funciona de forma análoga aos conjuntos da matemática, ele é uma coleção que não permite elementos duplicados.

Coleções - Conjuntos



Possíveis ações em um conjunto:

- A camiseta Azul está no conjunto?
- Remova a camiseta Azul.
- Adicione a camiseta Vermelha.
- Limpe o conjunto.

- Não existem elementos duplicados!

- Ao percorrer um conjunto, sua ordem não é conhecida!

Coleções - Conjuntos

Aqui, o segundo Diretor não será adicionado e o método add lhe retornará false.

```
Set<String> cargos = new HashSet<>();

cargos.add("Gerente");
cargos.add("Diretor");
cargos.add("Presidente");
cargos.add("Secretária");
cargos.add("Funcionário");
cargos.add("Diretor"); // repetido!

// imprime na tela todos os elementos
System.out.println(cargos);
```

Coleções - Conjuntos

Possui as seguintes implementações

- HashSet: usa tabela hash;
  - Conjunto de objetos armazenados em uma tabela hash
- LinkedHashSet:
  - Armazenamento do conjunto de objetos em uma lista encadeada
- TreeSet: usa árvore e é ordenado.
  - Conjunto de objetos armazenados em árvore binária
  - O armazenamento dos objetos pode ser ordenados
  - Mais rápidas que os outros conjuntos

A escolha de qual classe usar vai depender de fatores como desempenho e facilidade de uso

Coleções - Conjuntos

- O uso de um Set pode parecer desvantajoso:
  - Ele não armazena a ordem, e não aceita elementos repetidos.
  - Não há métodos que trabalham com índices, como o get(int) que as listas possuem.
- A grande vantagem do Set é que
  - Existem implementações, como a HashSet, que possui uma performance incomparável com as Lists quando usado para pesquisa (método contains por exemplo).

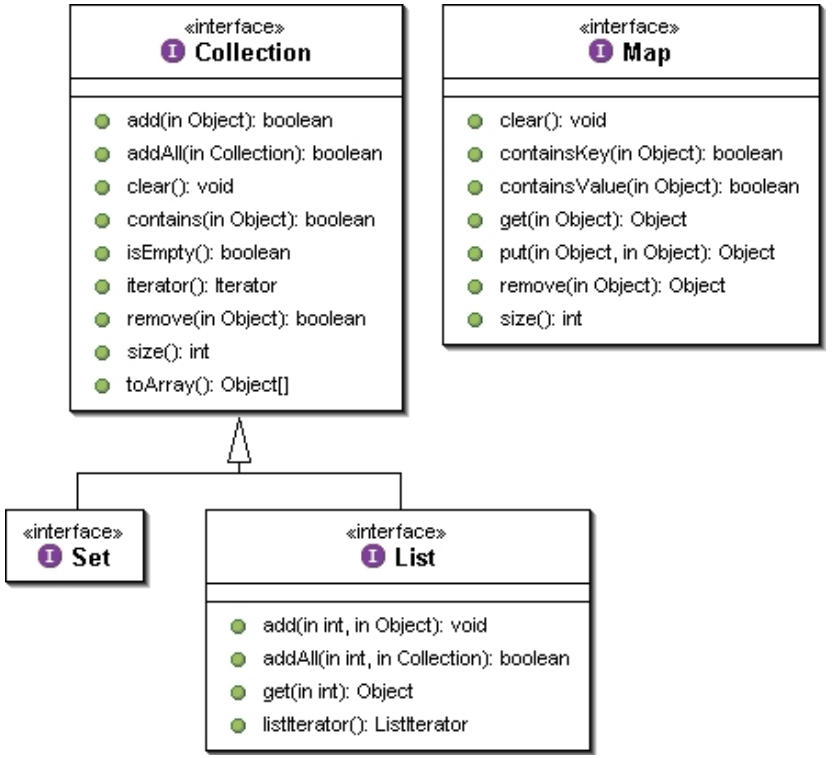
Coleções - Conjuntos

- As coleções têm como base a interface Collection, que define métodos para adicionar e remover um elemento, e verificar se ele está na coleção, entre outras operações, como mostra a tabela a seguir:

`boolean add(Object)`	Adiciona um elemento na coleção. Como algumas coleções não suportam elementos duplicados, este método retorna `true` ou `false` indicando se a adição foi efetuada com sucesso.
`boolean remove(Object)`	Remove determinado elemento da coleção. Se ele não existia, retorna `false`.
`int size()`	Retorna a quantidade de elementos existentes na coleção.
`boolean contains(Object)`	Procura por determinado elemento na coleção, e retorna verdadeiro caso ele exista. Esta comparação é feita baseando-se no método `equals()` do objeto, e não através do operador ==.
`Iterator iterator()`	Retorna um objeto que possibilita percorrer os elementos daquela coleção.

Coleções - Conjuntos

- Uma coleção pode implementar diretamente a interface Collection, porém normalmente se usa uma das duas sub interfaces mais famosas: justamente Set e List.



Coleções - Conjuntos

- Como percorrer os elementos de uma coleção?
- Se for uma lista, podemos sempre utilizar um laço for, invocando o método get para cada elemento.
- Mas e se a coleção não permitir indexação?
- Por exemplo, um Set não possui um método para pegar o primeiro, o segundo ou o quinto elemento do conjunto, já que um conjunto não possui o conceito de “ordem”
- Podemos usar o enhanced-for (o “foreach”) para percorrer qualquer Collection sem nos preocupar com isso.

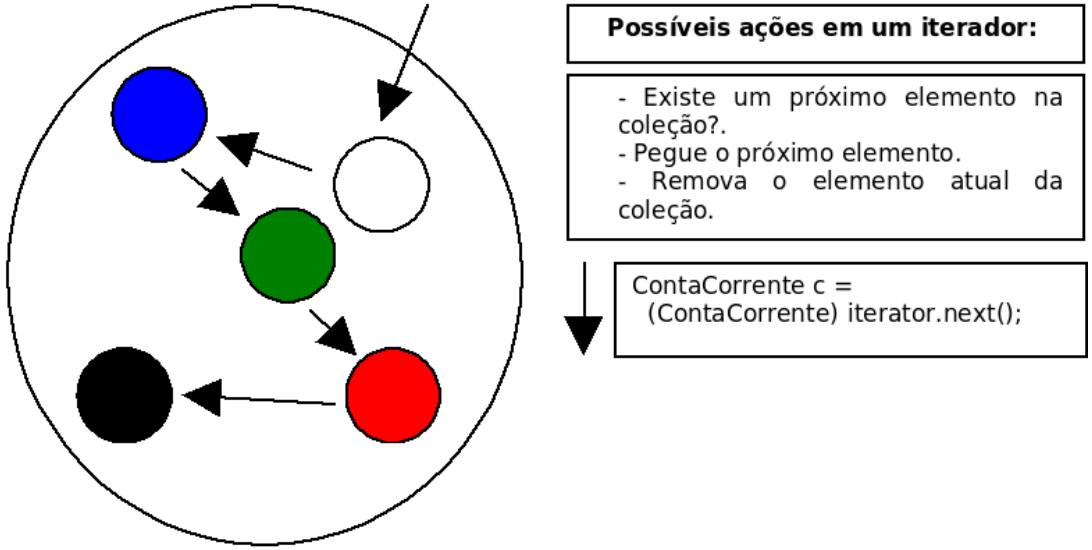
Coleções - Conjuntos

```
Set<String> conjunto = new HashSet<>();

conjunto.add("java");

conjunto.add("raptor");

conjunto.add("scala");

for (String palavra : conjunto) {

System.out.println(palavra);

}
```

- Em que ordem os elementos serão acessados?
- Em um conjunto, a ordem depende da implementação da interface Set: você muitas vezes não vai saber ao certo em que ordem os objetos serão percorridos

Coleções - Conjuntos

Toda coleção fornece acesso a um iterator, um objeto que implementa a interface Iterator, que conhece internamente a coleção e dá acesso a todos os seus elementos, como a figura baixo mostra.



Coleções - Conjuntos

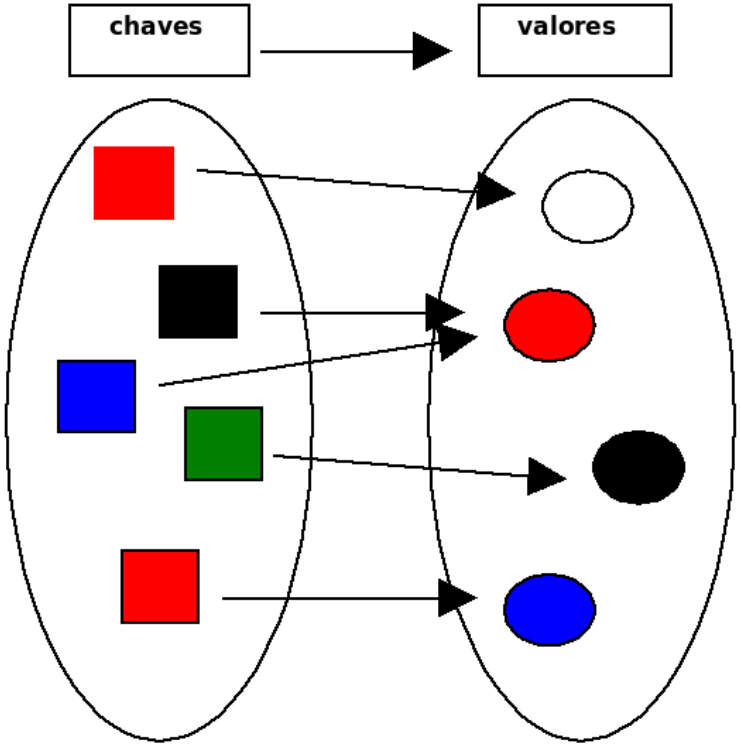
```
Iterator i = numeros.iterator();
// O while só termina quando todos os elementos do conjunto forem percorridos,
// isto é, quando o método hasNext mencionar que não existem mais itens.
while (i.hasNext())
    System.out.println(i.next());


// A partir do Java 5.0, surgiu uma nova sintaxe para laços que usam iteradores;
System.out.println("\n");
for (Object o : numeros)
    System.out.println((String)o);
```

Coleções - Mapas

- Muitas vezes queremos buscar rapidamente um objeto dado alguma informação sobre ele.
- Um exemplo seria:
  - Dada a placa do carro, obter todos os dados do carro.
  - Poderíamos utilizar uma lista para isso e percorrer todos os seus elementos, mas isso pode ser péssimo para a performance, mesmo para listas não muito grandes.

Coleções - Mapas

Um mapa é composto por um conjunto de associações entre um objeto chave a um objeto valor.



Possíveis ações em um mapa:

Mapeie uma chave a um valor
O que está mapeado na chave X?
Remapeie uma certa chave
Quero o conjunto de chaves.
Quero o conjunto de valores.
Desmapeie a chave X.

Coleções - Mapas

- O método **put(object, object)** da interface Map recebe a chave e o valor de uma nova associação.
- Para saber o que está associado a um determinado objetochave, passa-se esse objeto no método get(Object).
- Essas são as duas operações principais e mais frequentes realizadas sobre um mapa

```
ContaCorrente c1 = new ContaCorrente();
c1.deposita(10000);
ContaCorrente c2 = new ContaCorrente();
c2.deposita(3000);

// cria o mapa
Map<String, ContaCorrente> mapaDeContas = new HashMap<>();
// adiciona duas chaves e seus respectivos valores
mapaDeContas.put("diretor", c1);
mapaDeContas.put("gerente", c2);

// qual a conta do diretor? (sem casting!)
ContaCorrente contaDoDiretor = mapaDeContas.get("diretor");
System.out.println(contaDoDiretor.getSaldo());
```

Coleções - Mapas

- Criamos duas contas correntes e as colocamos em um mapa associando-as aos seus donos.

Coleções - Mapas

- Um mapa é muito usado para “indexar” objetos de acordo com determinado critério, para podermos buscar esse objetos rapidamente. Um mapa costuma aparecer juntamente com outras coleções, para poder realizar essas buscas.
- Suas principais implementações são:
  - HashMap
  - TreeMap
  - Hashtable.

EXERCÍCIOS - Coleções - Mapas

- V1.0 : Crie um programa em Java para testar a classe AgendaTelefonica abaixo
  - Teste a classe com pelo menos 5 contatos diferentes na agenda de telefones.

- V2.0 : Objeto Contato