CURSO DE PROGRAMAÇÃO EM JAVA

Aula 17

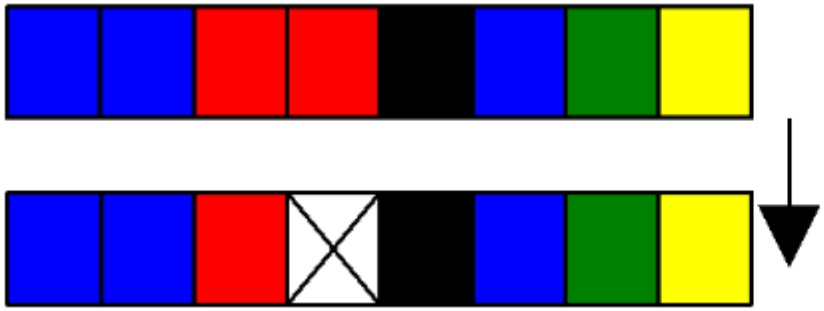
Coleções

1.
Coleções

Arrays são trabalhosos

Como vimos na aula de arrays, manipulá-las 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 sem criar, para isso, métodos auxiliares.



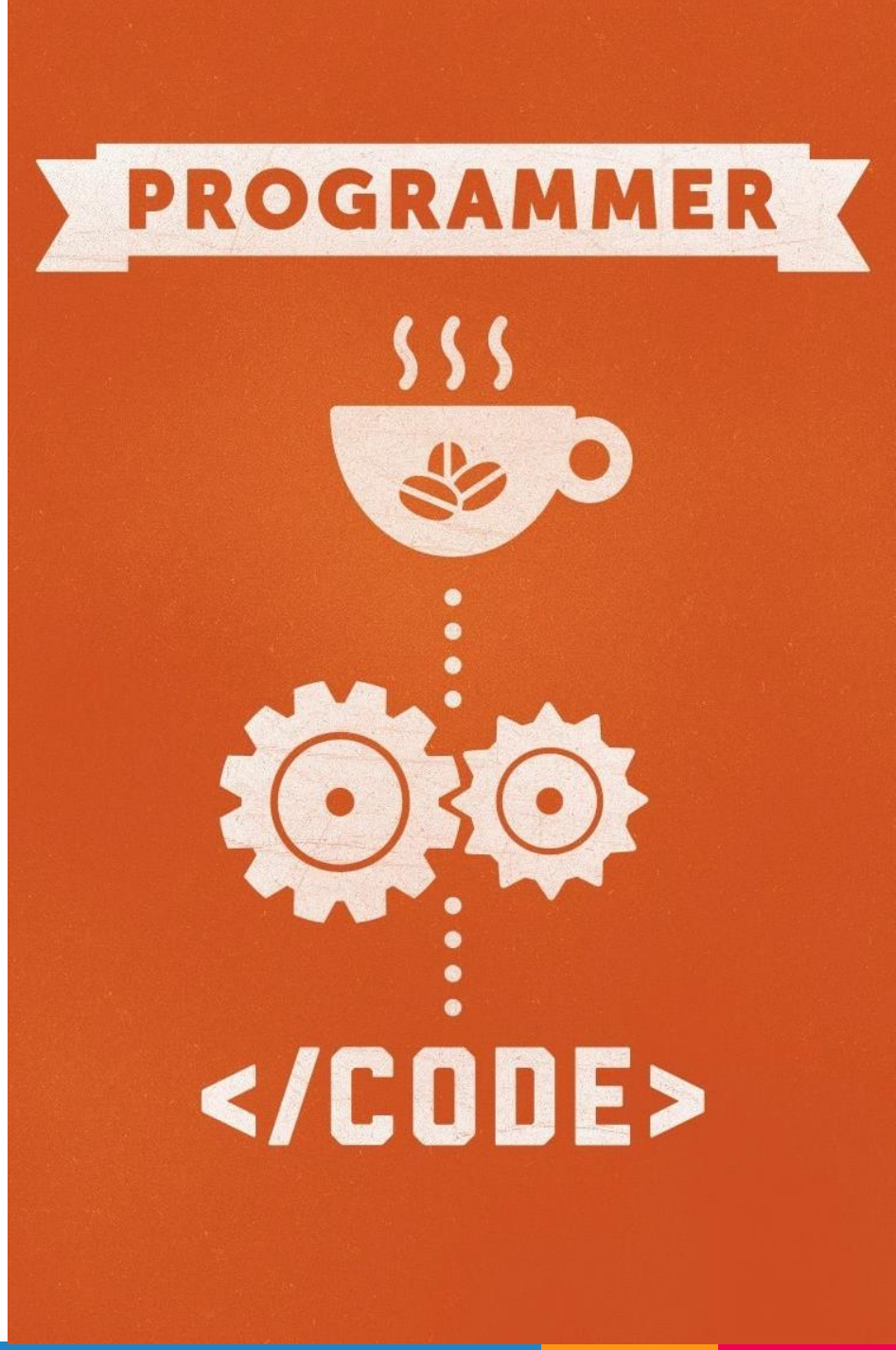
Retire a quarta Conta

conta[3] = **null**;

Collections

Além dessas dificuldades que os arrays apresentavam, faltava um conjunto robusto de classes para suprir a necessidade de estruturas de dados básicas, como listas ligadas e tabelas de espalhamento.

Com esses e outros objetivos em mente, o comitê responsável pelo Java criou um conjunto de classes e interfaces conhecido como **Collections Framework**, que reside no pacote java.util desde o Java2 1.2.



2.

Listas: java.util.List

LISTAS: JAVA.UTIL.LIST

A API de **Collections** traz a interface **java.util.List**, que especifica 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. Portanto, ela é **mais rápida na pesquisa** do que sua concorrente, a **LinkedList**, que é **mais rápida na inserção e remoção** de itens nas pontas.

A interface List possui **dois métodos add**, um que recebe o objeto a ser inserido e o **coloca no final da lista**, e um segundo que permite adicionar o elemento em **qualquer posição** da mesma.

LISTAS: JAVA.UTIL.LIST

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");
```

Note que, em momento algum, dizemos qual é o **tamanho da lista**; podemos **acrescentar quantos elementos quisermos**, que a lista cresce conforme for necessário

LISTAS: JAVA.UTIL.LIST

Toda lista (na verdade, **toda Collection**) trabalha do modo mais **genérico** possível. Isto é, não há uma ArrayList específica para Strings, outra para Números, outra para Datas etc. Todos os métodos **trabalham com Object**.

Assim, é possível criar, por exemplo, uma lista de Contas Correntes:

```
ContaCorrente c1 = new ContaCorrente();
c1.deposita(100);
ContaCorrente c2 = new ContaCorrente();
c2.deposita(200);
ContaCorrente c3 = new ContaCorrente();
c3.deposita(300);

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

LISTAS: JAVA.UTIL.LIST

Para saber quantos elementos há na lista, usamos o método size():

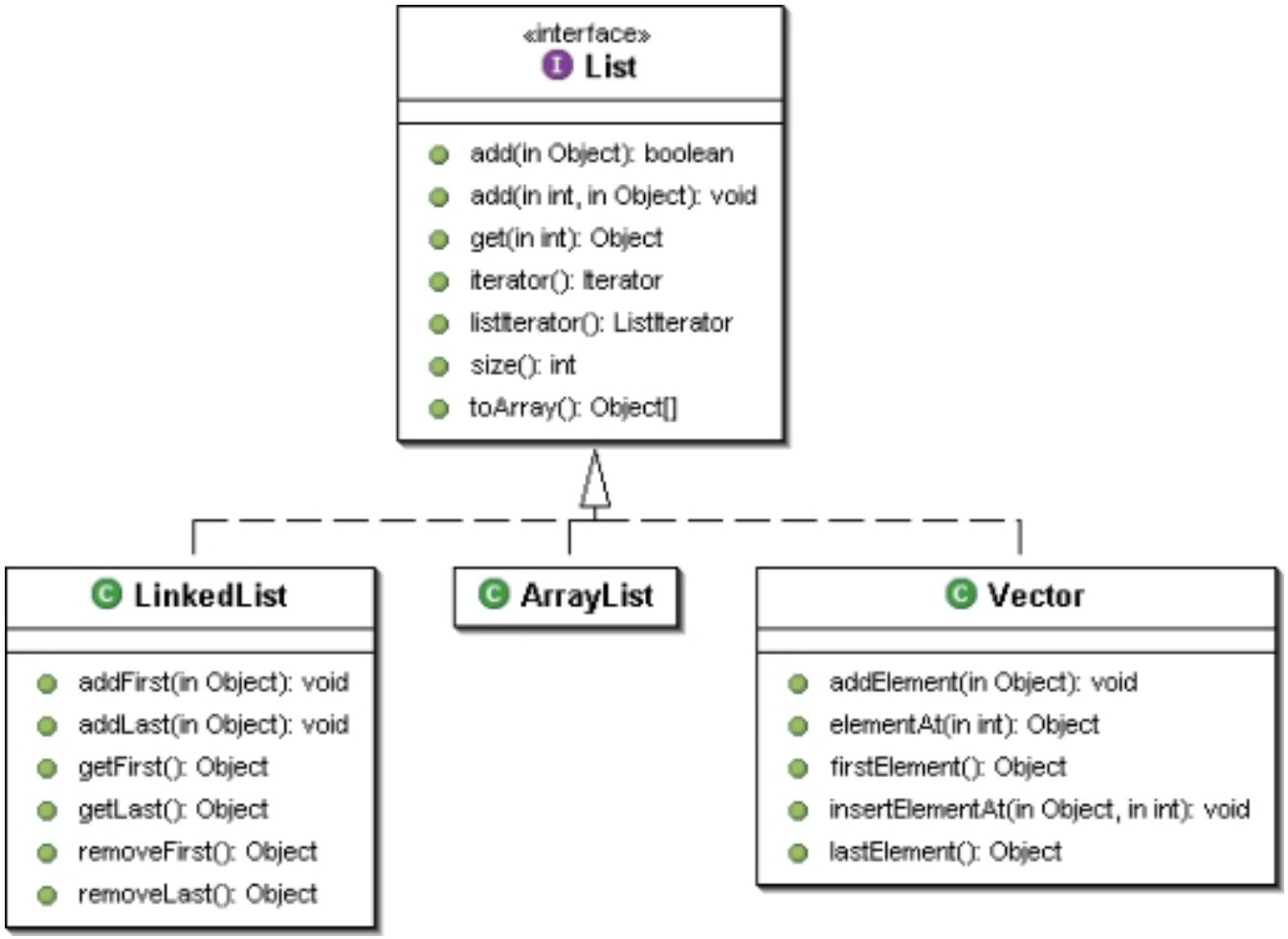
```
System.out.println(contas.size());
```

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:

1)
```
for (int i = 0; i < contas.size(); i++) {
   contas.get(i); // código não muito útil....
}
```

2)
```
for (int i = 0; i < contas.size(); i++) {
   ContaCorrente cc = (ContaCorrente) contas.get(i);
   System.out.println(cc.getSaldo());
}
// note que a referência devolvida pelo get(i) é do tipo Object, sendo necessário o cast para ContaCorrente se quisermos acessar o getSaldo().
```

LISTAS: JAVA.UTIL.LIST



GENERICS

Geralmente, não nos interessa uma lista com vários tipos de objetos misturados; no dia-a-dia, usamos listas como contas correntes. No Java 5.0, 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);
```

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());
}
```

3.
Ordenação: Collections.sort

COLLECTIONS.SORT

A classe Collections traz um método estático **sort** que recebe um **List como argumento** e o ordena por **ordem crescente**. Por exemplo:

```
List<String> lista = new ArrayList<>();
lista.add("Sérgio");
lista.add("Paulo");
lista.add("Guilherme");

System.out.println(lista);
Collections.sort(lista);
System.out.println(lista);
```

Ao testar o exemplo acima, você observará que, primeiro, a lista é impressa na **ordem de inserção** e, depois de invocar o sort, ela é impressa em **ordem alfabética.**

COLLECTIONS.SORT

Mas toda lista em Java pode ser de qualquer tipo de objeto, por exemplo, ContaCorrente. E se quisermos ordenar uma lista de ContaCorrente? Em que ordem a classe Collections ordenará? Pelo saldo? Pelo nome do correntista?

```
ContaCorrente c1 = new ContaCorrente();
c1.deposita(500);
ContaCorrente c2 = new ContaCorrente();
c2.deposita(200);
ContaCorrente c3 = new ContaCorrente();
c3.deposita(150);

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

Collections.sort(contas); // qual seria o critério para esta ordenação?
```

COLLECTIONS.SORT

O método **sort** necessita que todos seus objetos da lista sejam **comparáveis** e possuam um **método** que se **compara** com outra ContaCorrente.

Vamos fazer com que os elementos da nossa coleção **implementem a interface java.lang.Comparable**, que define o método int **compareTo(Object)**. Este método deve retornar zero, se o objeto comparado for igual a este objeto, um número negativo, se este objeto for menor que o objeto dado, e um número positivo, se este objeto for maior que o objeto dado.

Para ordenar as **ContaCorrentes** por s**aldo**, basta implementar o **Comparable**:

COLLECTIONS.SORT

```
public class ContaCorrente extends Conta
                                    implements Comparable<ContaCorrente> {

  // ... todo o código anterior fica aqui

  public int compareTo(ContaCorrente outra) {
    if (this.saldo < outra.saldo) {
      return -1;
    }

    if (this.saldo > outra.saldo) {
      return 1;
    }

    return 0;
  }
}
```

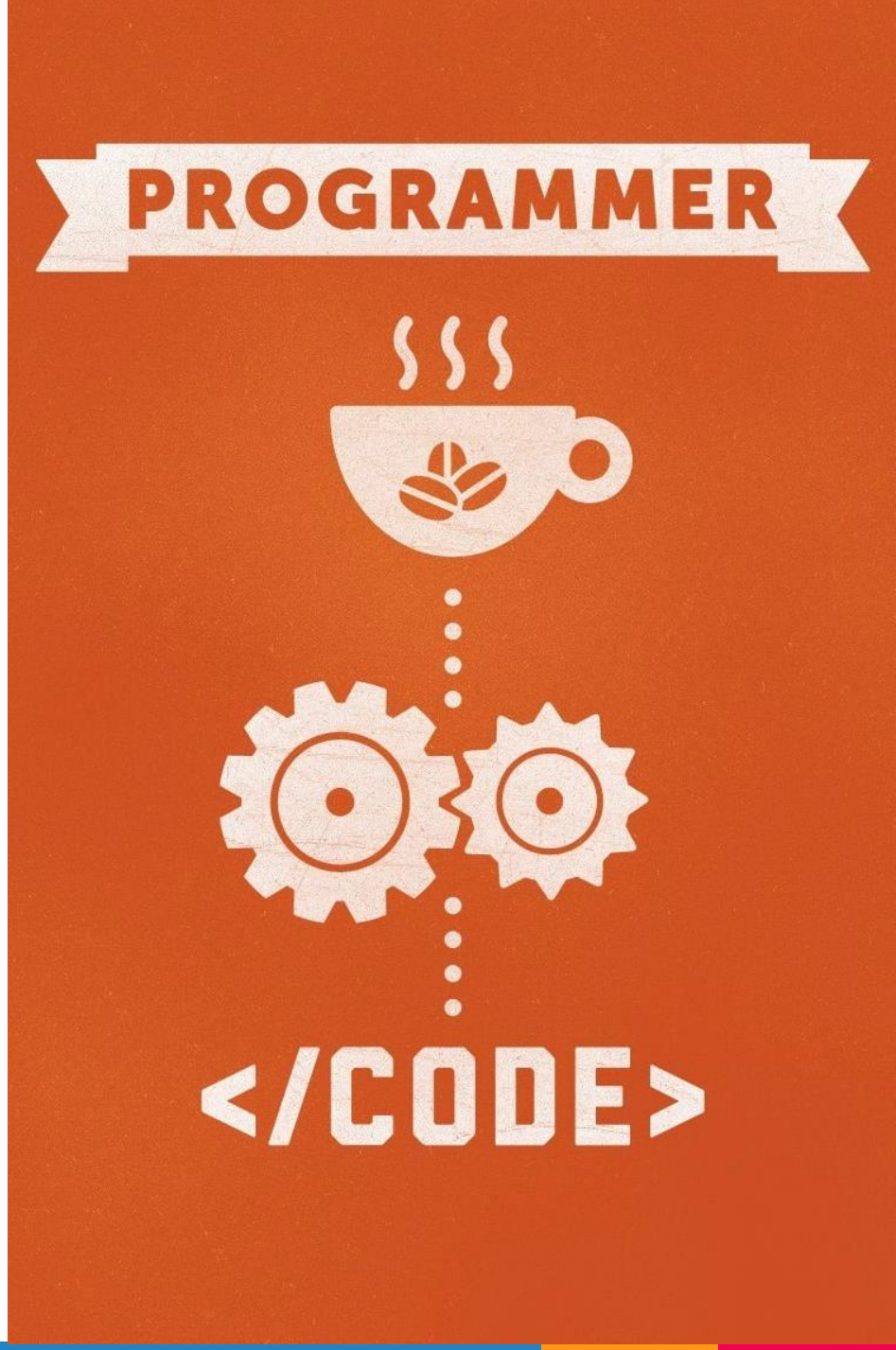
3.

Conjunto: java.util.Set

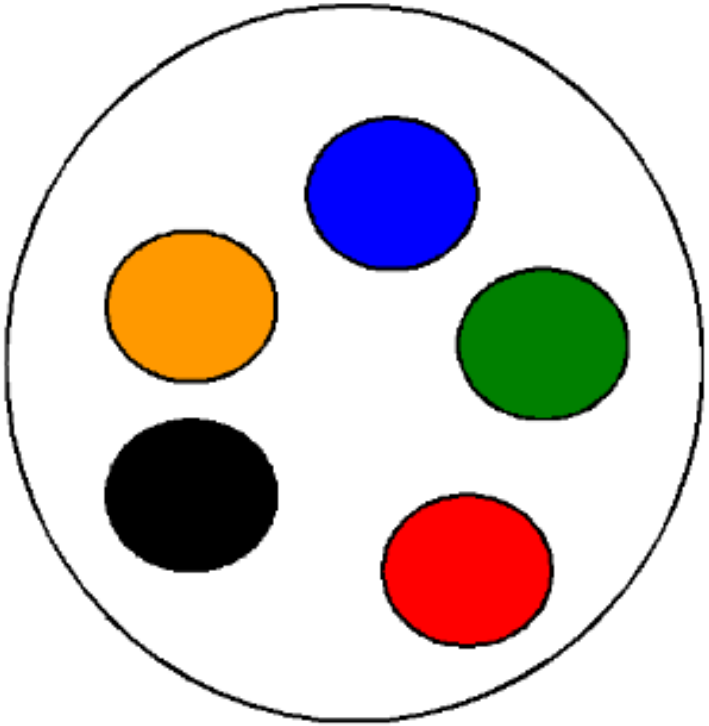
Conjunto

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

Outra característica fundamental dele é o fato de que a ordem em que os elementos são armazenados pode não ser a ordem na qual eles foram inseridos no conjunto. A interface não define como deve ser este comportamento. Tal ordem varia de implementação para implementação.



Conjunto: java.util.Set



Possíveis ações em um conjunto:

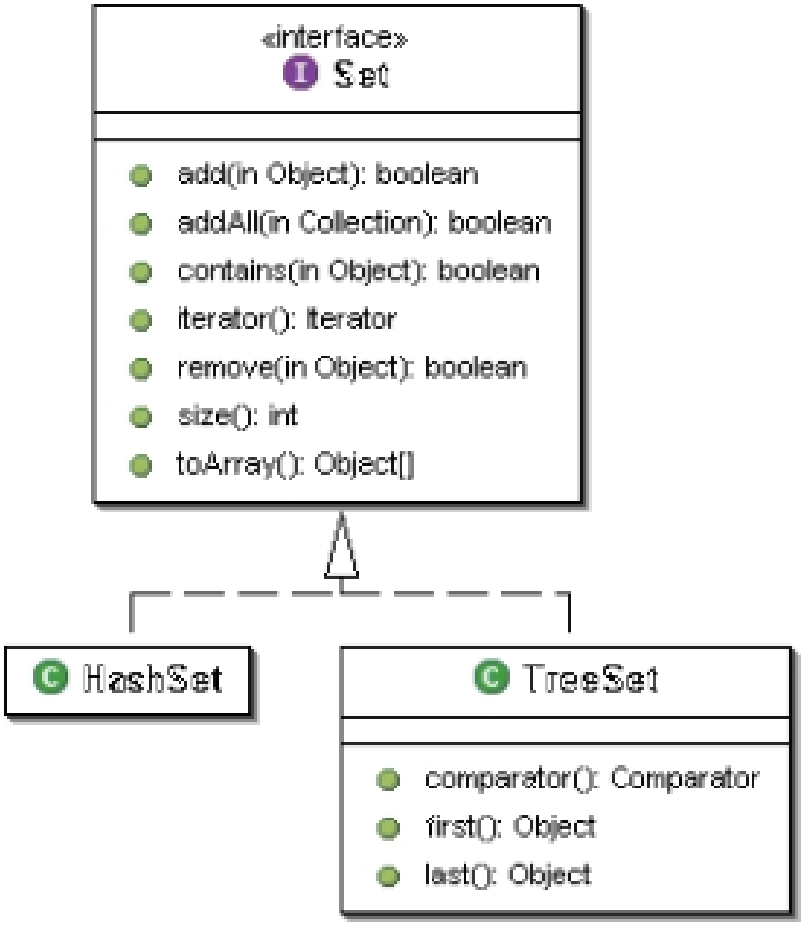
- A camiseta Azul está no conjunto?
- Remova a camiseta Azul.
- Adicione a camiseta Vermelha.
- Limpe o conjunto.

- Não existem elementos duplicados!

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

CONJUNTO: JAVA.UTIL.SET

Um conjunto é representado pela interface Set e tem como suas principais implementações as classes **HashSet**, **LinkedHashSet** e **TreeSet**.



CONJUNTO: JAVA.UTIL.SET

O código a seguir cria um conjunto e adiciona diversos elementos, e alguns repetidos:

```
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);
```

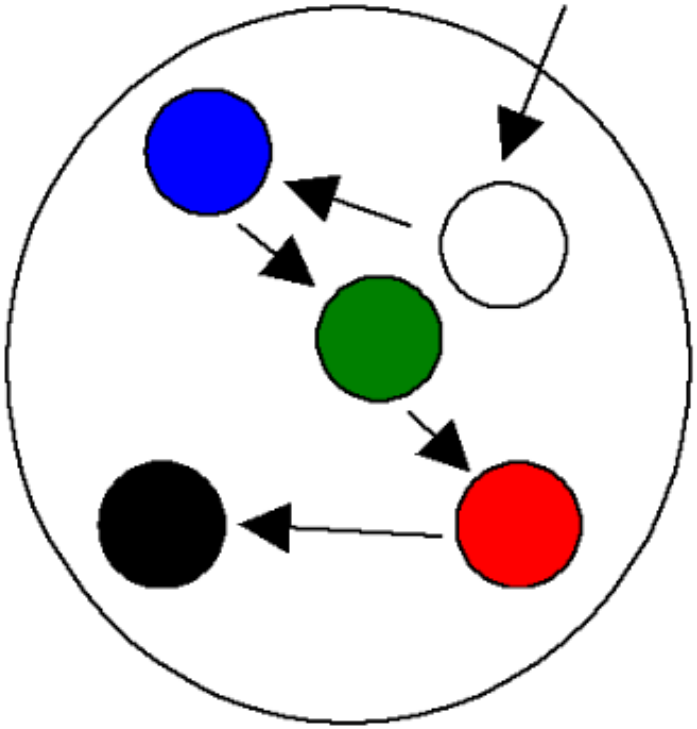
Aqui, o segundo Diretor não será adicionado e o método add lhe retornará false.

4.

java.util.Iterator

JAVA.UTIL.ITERATOR

Antes do Java 5 introduzir o novo enhanced-for, iterações em coleções eram feitas com o Iterator. 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 abaixo mostra.



Possíveis ações em um iterador:

- Existe um próximo elemento na coleção?.
- Pegue o próximo elemento.
- Remova o elemento atual da coleção.

```
ContaCorrente c =
  (ContaCorrente) iterator.next();
```

JAVA.UTIL.ITERATOR

Primeiro criamos um **Iterator** que entra na coleção. A cada chamada do método **next**, o **Iterator** retorna o **próximo objeto** do conjunto. Um iterator pode ser obtido com o método **iterator()** de **Collection**, por exemplo numa lista de String:

```
Iterator<String> i = lista.iterator();
```

A interface Iterator possui dois métodos principais: **hasNext()** (com retorno booleano), indica se ainda **existe** um elemento a ser percorrido; **next(),** retorna o próximo **objeto**.

JAVA.UTIL.ITERATOR

```
Set<String> conjunto = new HashSet<>();
conjunto.add("item 1");
conjunto.add("item 2");
conjunto.add("item 3");

// retorna o iterator
Iterator<String> i = conjunto.iterator();
while (i.hasNext()) {
    // recebe a palavra
    String palavra = i.next();
    System.out.println(palavra);
}
```

O while anterior só termina quando todos os elementos do conjunto forem percorridos, isto é, quando o método hasNext mencionar que não existem mais itens.

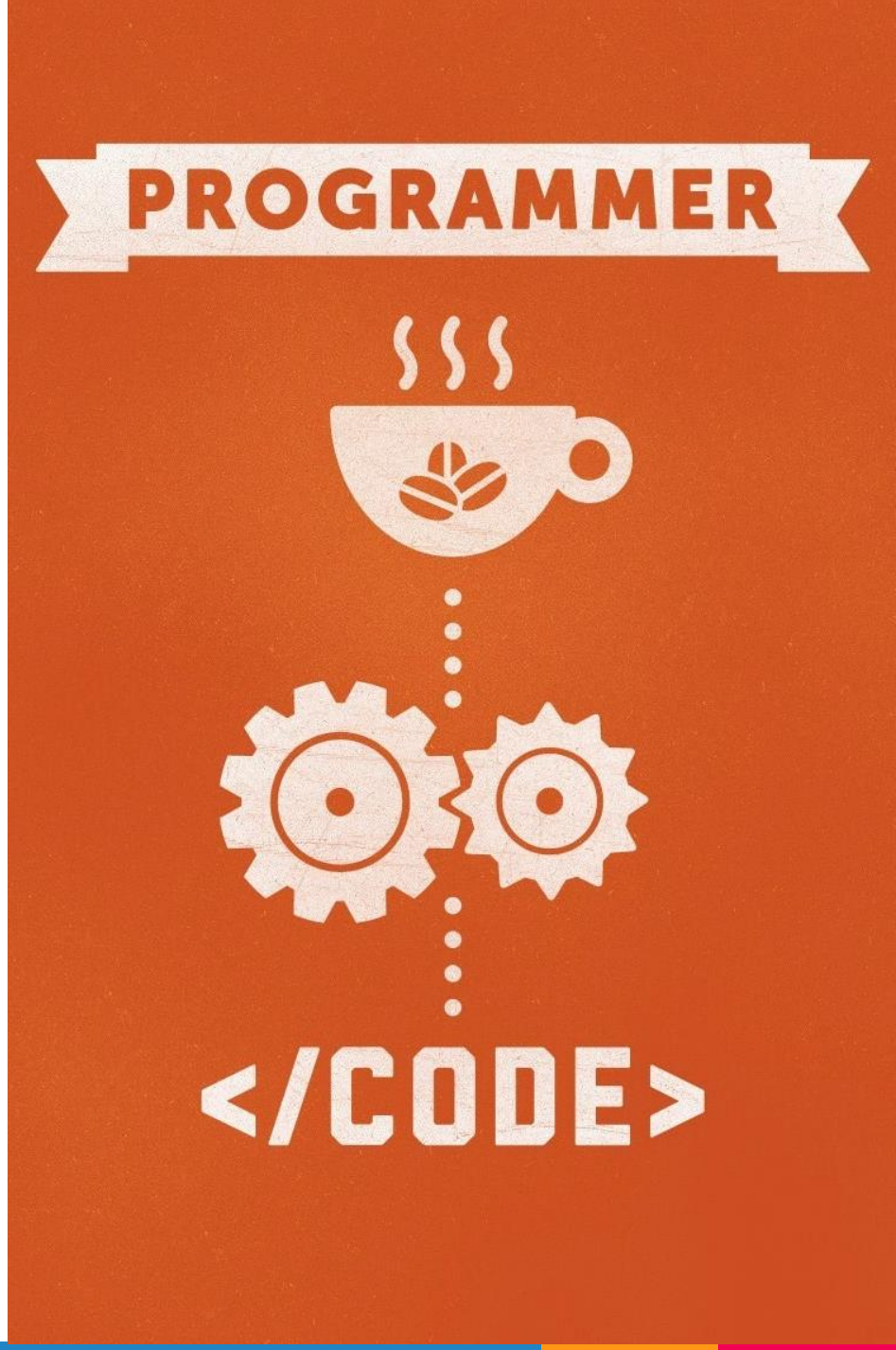
5.

Mapas - java.util.Map

Mapas - java.util.Map

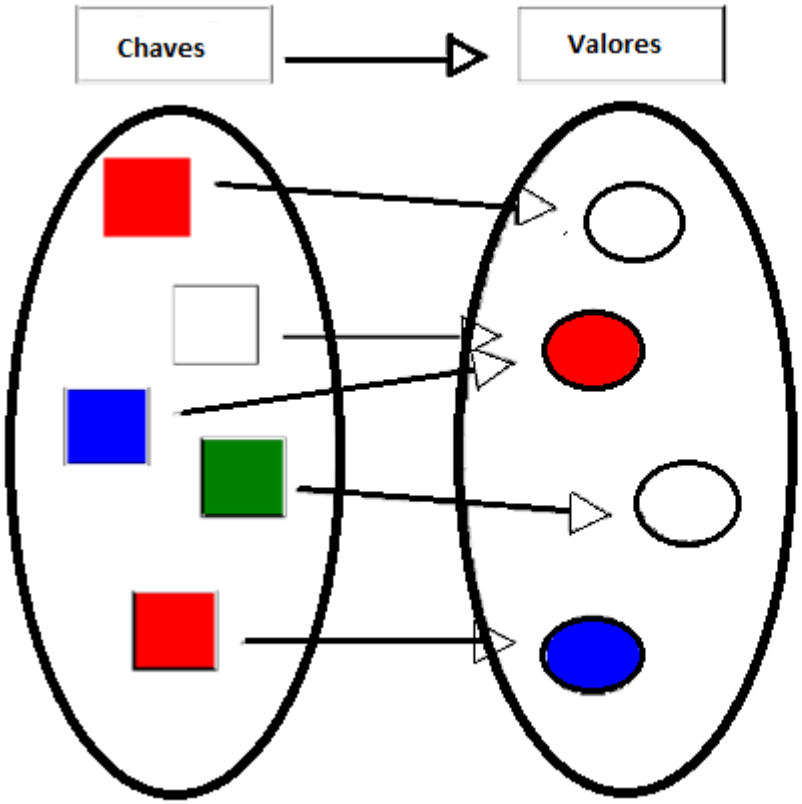
Muitas vezes queremos buscar rapidamente um objeto dado alguma informação sobre ele. Um exemplo seria, dada a placa do carro, obter todos os dados do carro.

Um mapa é composto por um conjunto de associações entre um objeto chave a um objeto valor. É equivalente ao conceito de dicionário, usado em várias linguagens. Algumas linguagens, como Perl ou PHP, possuem um suporte mais direto a mapas, onde são conhecidos como matrizes/arrays associativas.



Mapas - java.util.Map

java.util.Map é um mapa, pois é possível usá-lo para mapear uma chave a um valor, por exemplo: mapeie à chave "empresa" o valor "Caelum", ou então mapeie à chave "rua" ao valor "Vergueiro". Semelhante a associações de palavras que podemos fazer em um dicionário.



Possíveis ações de 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.

MAPAS - JAVA.UTIL.MAP

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 objeto-chave, passa-se esse objeto no método **get(Object**).

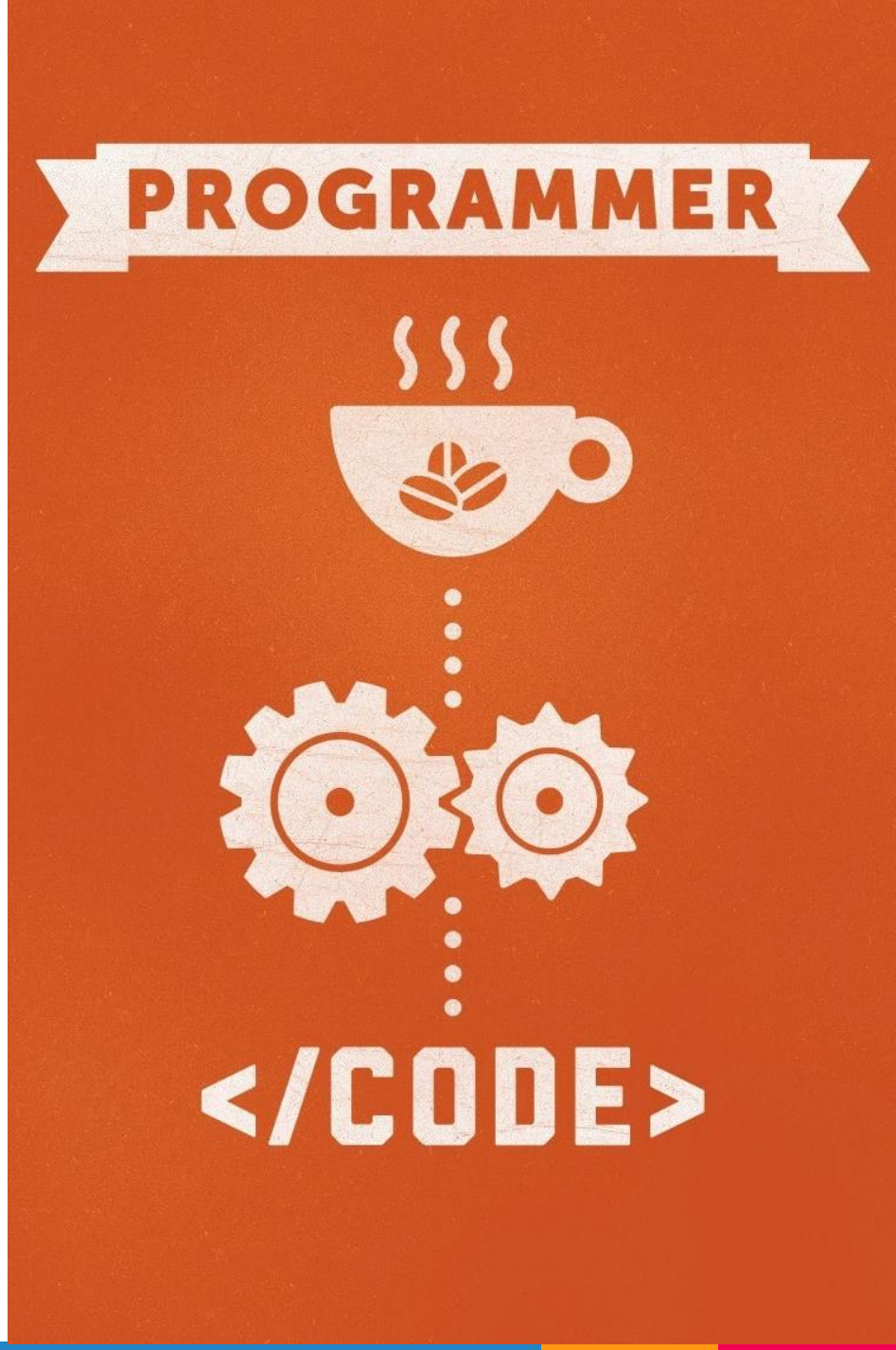
```
ContaCorrente c1 = new ContaCorrente();
c1.deposita(10000);
ContaCorrente c2 = new ContaCorrente();
c2.deposita(3000);

// 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());
```

Mapas - java.util.Map

Mapa trabalha diretamente com **Objects** (tanto na chave quanto no valor), o que tornaria necessário o **casting** no momento que recuperar elementos. Usando os **generics**, como fizemos aqui, não precisamos mais do casting.

Suas principais implementações são o **HashMap**, o **TreeMap** e o **Hashtable**.

O método **keySet**() retorna um Set com as chaves daquele mapa e o método **values**() retorna a **Collection** com todos os valores que foram associados a alguma das chaves.



DESAFIO

E aí, vamos praticar?

Amigos do Habay

Todo final de ano ocorre uma festa na Instituição de Educação Fantástica (IEF). Logo no início de julho, são abertas as inscrições para participar dela. No momento da inscrição, o usuário pode escolher se quer ser "O Amigo do Habay" na festa ou não. O mais lógico seria escolher a opção Sim, afinal, é um privilégio ser O Amigo do Habay, já que ele é a pessoa mais descolada do IEF. Porém, há indivíduos que definitivamente não pretendem ser O Amigo do Habay, e por motivos desconhecidos.

Somente um será o escolhido. Em vista disso, muitos alunos que escolheram a opção Sim realizaram a inscrição diversas vezes para aumentar a própria probabilidade de ser O Amigo do Habay. O organizador geral da festa contratou você para organizar as inscrições do site, pois está havendo um spam de inscrições. O critério para ser o escolhido é a quantidade de letras do primeiro nome, e em caso de empate, vence aquele que realizou primeiro a inscrição. A organização final dos inscritos deverá seguir a ordem de escolha (Sim ou Não), mas respeitando a ordem alfabética.

OBS.: Ninguém que escolheu a opção Não realizou a inscrição mais de uma vez.

Amigos do Habay

Entrada: A entrada contém somente um caso de teste. Cada linha é composta pelo primeiro nome do participante (sem espaços), seguido da opção YES (caso o usuário queira ser O Amigo do Habay) ou NO (caso não queira). A entrada termina assim que o usuário digita "FIM" (sem as aspas).
Saída: Seu programa deverá imprimir os inscritos pela ordem de escolha e por ordem alfabética, seguido do nome do vencedor. Imprima uma linha em branco entre a lista de inscritos e o nome do vencedor.

Exemplo de Entrada	Exemplo de Saída
Joao NO Carlos YES Abner NO Samuel YES Ricardo NO Abhay YES Andres YES Roberto NOSamuel YES Samuel YES Abhay YES Aline YES Andres YES FIM	Abhay Aline Andres Carlos Samuel Abner Joao Ricardo Roberto Amigo do Habay: Carlos

Ajude Girafales

Minutos antes do término das aulas, professor Girafales passa uma lista de presença. Certo dia, ele resolveu conferir as assinaturas e notou que alguns alunos assinavam diferente em algumas aulas e desconfiou que alguém poderia estar assinando por eles. Como o professor possui muitos alunos e pouco tempo (o café com dona Florinda é prioridade), ele pediu sua ajuda para validar as assinaturas. Uma assinatura é considerada falsa se houver mais de uma diferença entre a original e a que estiver sendo checada. Considere diferença uma troca de maiúscula para minúscula ou o contrário.

Entrada: Haverá diversos casos de testes. A primeira linha de cada caso inicia com um inteiros **N** ($1 \leq \mathbf{N} \leq 50$) representando a quantidade de alunos de sua turma. As próximas **N** linhas serão da seguinte forma:

Nome do aluno Assinatura Original

A seguir haverá um inteiro **M** ($0 \leq \mathbf{M} \leq \mathbf{N}$), representando a quantidade de alunos que compareceram a uma aula. **M** linhas seguem, no seguinte formato:

Nome do aluno Assinatura na aula

Todos os alunos possuem apenas o primeiro nome na lista, nenhum nome se repete e todos os nomes contêm no máximo 20 letras (**a-z A-Z**).
A entrada termina com **N** = 0, a qual não deve ser processada.

Ajude Girafales

Saída: Para cada caso, exiba uma única linha, a quantidade de assinaturas falsas encontradas.

Exemplo de Entrada	Exemplo de Saída
4 Chaves ChAvEs Kiko kikO Nhonho NHONHO Chiquinha CHIquinHa 3 Chaves ChAvEs Kiko kIKO Chiquinha CHIquinHA 2 Jadson jadsON Crishna Crishna 2 Crishna CRISHNA Jadson JADson 0	1 2

Obrigado!

Alguma pergunta?

Você pode nomes contatar em:
ywassef@hotmail.com