

Programação

OO

III

I

Parte I

O que é uma coleção ?

- ▶ Uma coleção (também denominada *container*)
 - É simplesmente um objeto que agrupa múltiplos elementos dentro de uma única unidade.
- ▶ Em outras palavras...
 - Conjunto de classes que permitem o agrupamento e processamento de grupos de objetos:
- ▶ São utilizadas para armazenar, recuperar e manipular dados,
 - Além de transmitir dados de um método para outro.

Array []

► Características de Array []

- Forma simples de armazenar informações...

```
public class MeuArray {  
  
    public static void main( String args[]){  
        int[] meuArray;  
        meuArray = new int[10];  
        meuArray[0] = 22;  
        meuArray[1] = 2;  
        meuArray[2] = 242;  
        meuArray[3] = 4552;  
        meuArray[4] = 36;  
  
        for (int i = 0; i < meuArray.length; i++){  
            System.out.println("posição " + i + " = " + meuArray[i]);  
        }  
    }  
}
```

- ..mas...

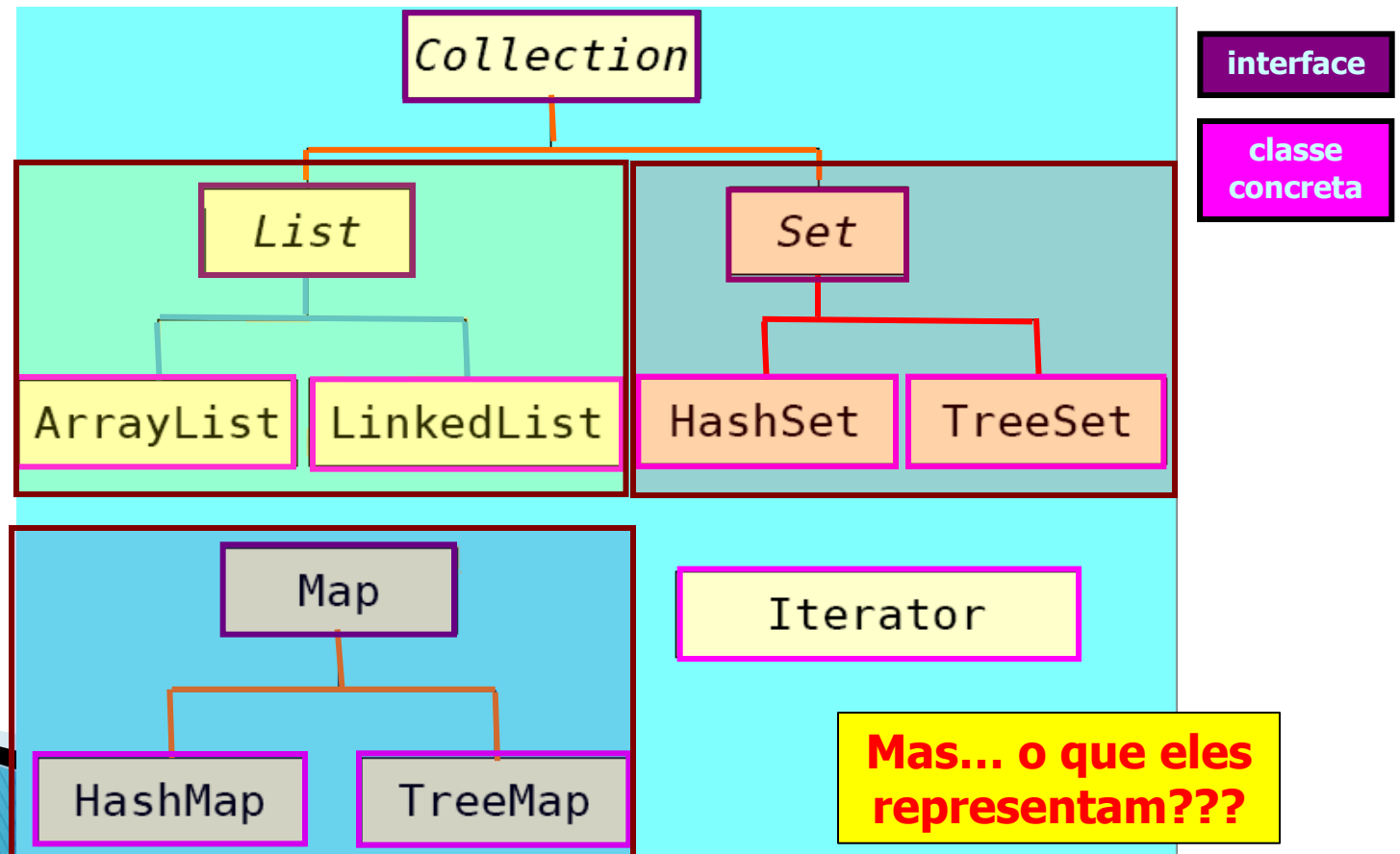
- Tem tamanho limitado;
- Não podemos redimensionar uma array..
- É impossível buscar diretamente por um determinado elemento para o qual não se sabe o índice;
- etc..... ☺

Java Collection Framework

- ▶ Blz...
 - Tudo é muito legal, mas..
 -agora vamos ver como podemos facilitar ainda mais a nossa vida quando temos que lidar com esses tipos de estruturas...

Java Collection Framework

- ▶ Hierarquia (vamos falar mais sobre isso logo logo..)
 - Os elementos que compreendem a estrutura de coleções estão no pacote java.util.



Java Collection Framework

► Hierarquia – Interfaces – Em poucas palavras...

- List

- É uma coleção onde a ordem é mantida, cada objeto pode ser manipulado através de seu índice

- Set

- Uma coleção que não pode ter objetos duplicados

- Map

- Mapeia objetos chaves para objetos, não são permitidas chaves duplicadas

Vamos conversar um pouco sobre essas paradas.... 😊



➡ List – Coleções indexadas

- ▶ O primeiro recurso que a API de Collections traz são Listas ou Coleções indexadas
- ▶ Uma lista é uma coleção que permite:
 - Elementos duplicados,
 - Mantém uma ordenação específica entre os elementos,
 - E os elementos podem ser acessados pelos seus respectivos índices dentro da lista
 - Ela resolve todos os problemas que levantamos em relação a array (busca, remoção, tamanho “infinito”,...).
 - Esse código já está pronto! 😊

List – Coleções indexadas

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

- ▶ Possui as seguintes implementações
 - ArrayList: usa vetores
 - Trabalha com um array interno para gerar uma lista
 - lista de objetos armazenados em um vetor interno
 - Ela é mais rápida na pesquisa (desempenho melhor);
 - Melhor se você precisa de acesso com índice
 - LinkedList: usa lista encadeada
 - Lista de objetos armazenados em uma lista encadeada
 - Normalmente ordenada pela ordem de inserção.
 - É mais rápida na inserção e remoção de itens nas pontas.

List – Coleções indexadas

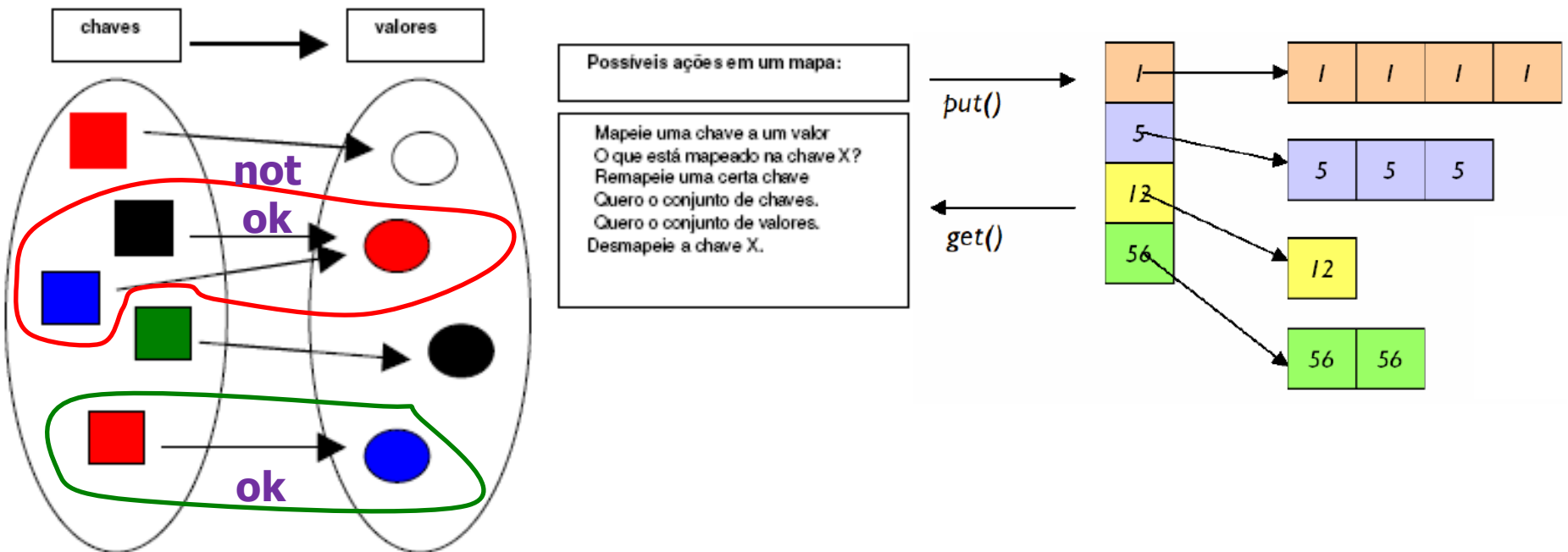
- ▶ Métodos principais:
 - `add(Object)`, `add(int, Object)`, `addAll(Collection)`;
 - `clear()`, `remove(int)`, `removeAll(Collection)`;
 - `contains(Object)`, `containsAll(Collection)`;
 - `get(int)`, `indexOf(Object)`, `set(int, Object)`;
 - `isEmpty()`, `toArray()`, `subList(int, int)`, `size()`.

➡ Map – Coleções pares chave x valor

- ▶ Outro recurso que a API de Collections traz são os mapeamentos (Map) ou Coleções de pares chave x valor
 - É basicamente o conceito de hash mas para o Collection Framework
- ▶ Um mapa é composto de uma associação de um objeto chave a um objeto valor.
 - É equivalente ao conceito de dicionário usado em várias linguagens
 - Exemplo: Mapeia “Vinicius” à chave “CPF.”

Map – Coleções pares chave x valor

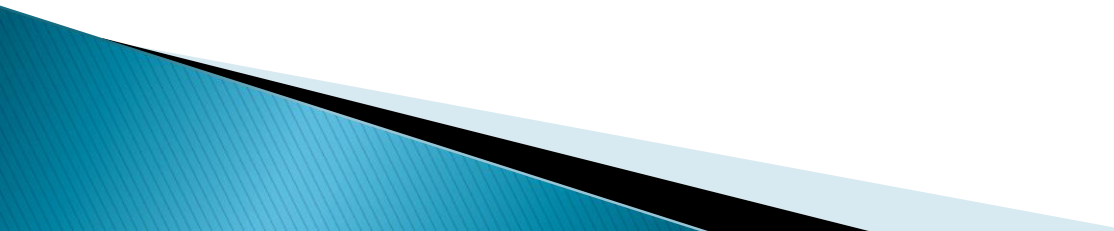
- ▶ Objetos Map não podem conter chaves duplicadas
- ▶ Cada chave só pode mapear um valor apenas
- ▶ Valores podem ser repetidos para chaves diferentes



Map – Coleções pares chave x valor

- ▶ Possui as seguintes implementações
 - HashMap: Usa tabela *hash*;
 - Correspondências chave-valor, onde as chaves não estão ordenadas.
 - Permite elementos e chaves nulos
 - Usa hashCode() para otimizar a busca por uma chave
 - TreeMap: Usa árvore e chaves é ordenada
 - Usado quando preciso que os elementos sejam ordenados (definido pela chave)
 - Objetos devem implementar Comparable ou Comparator
 - Os objetos precisam implementar a interface Comparable ou Comparator (vamos ver mais pra frente...)

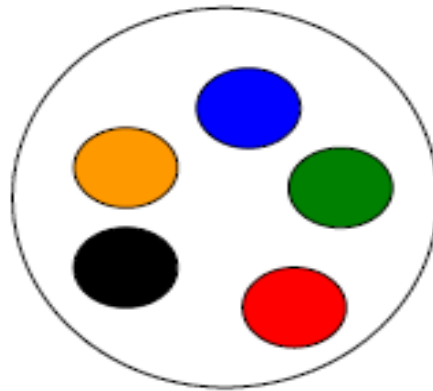
Map – Coleções pares chave x valor

- ▶ Métodos principais:
 - remove(Object), clear();
 - containsKey(Object), containsValue(Object);
 - isEmpty(), size();
 - put(Object key, Object), get(Object key), putAll(Map);
 - entrySet(), keySet(), values().
- 

➡ Set – Coleções não indexadas

- ▶ Outro recurso que a API de Collections traz são os conjuntos (Set) ou Coleções não indexadas
- ▶ Representa uma coleção “desordenada” de dados e não permite elementos duplicados.
 - “Desordenada” pois a priori a ordem não importa...

Não
pode haver
dois objetos
iguais



bolsa....

Set – Coleções não indexadas

- ▶ 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
 - Os objetos precisam implementar a interface Comparable ou Comparator (vamos ver mais pra frente...)
 - Mais rápidas que os outros conjuntos $O(\log(n))$

Set – Coleções não indexadas

- ▶ Métodos principais:
 - `add(Object)`, `addAll(Collection)`;
 - `clear()`, `remove(Object)`, `removeAll(Collection)`,
 - `retainAll(Collection)`;
 - `contains(Object)`, `containsAll(Collection)`;
 - `isEmpty()`, `toArray()`, `size()`.
- ▶ Cadê o `get`????

Iteradores – Coleções

- ▶ Podemos também usar o iterator ou o enhanced for ou forEach

→ Iterator i = numeros.iterator();

coleção

→ while (i.hasNext())
→ System.out.println(i.next());

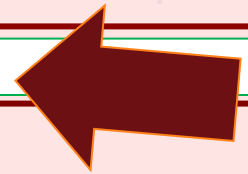
System.out.println("\n");

→ for (Object o : numeros)
→ System.out.println(o);

coleção

coleção

→ list.forEach((item) -> {
→ System.out.println("Item FOR: "+item);
});



E como usamos... sintaxe

- ▶ Repare no uso de um parâmetro ao lado de List e ArrayList:
 - Ele indica que nossa lista foi criada para trabalhar exclusivamente com objetos do tipo ContaCorrente.

➡ List<ContaCorrente> contas = new ArrayList<ContaCorrente>();

contas.add(c1);

contas.add(c3);

contas.add(c2);

➡ contas.get(i); // sem casting!



Implica que o get não vai retornar um Object e sim uma ContaCorrente

- ▶ Isso também nos traz uma segurança em tempo de compilação:

➡ contas.add("uma string"); // isso não compila mais!!

Tipos Genéricos e Coleções

- ▶ Genérico e Set
 - Sintaxe

```
➡ Set<String> conjunto = new HashSet<String>();  
   conjunto.add("item 1");  
   conjunto.add("item 2");  
   conjunto.add("item 3");  
  
   // retorna o iterator  
➡ for(String palavra : conjunto) {  
    ➡ System.out.println(palavra);  
}
```

Tipos Genéricos e Coleções

▶ Genérico e Map

- Assim como as coleções, um mapa é parametrizado.
- O interessante é que ele recebe dois parâmetros:
 - A chave e o valor:

```
ContaCorrente c1 = new ContaCorrente();  
c1.deposita(10000);  
  
ContaCorrente c2 = new ContaCorrente();  
c2.deposita(3000);  
  
// cria o mapa  
➡ Map<String, ContaCorrente> mapaDeContas = new  
HashMap<String, ContaCorrente>();  
  
// adiciona duas chaves e seus valores  
➡ mapaDeContas.put("diretor", c1);  
mapaDeContas.put("gerente", c2);  
// qual a conta do diretor? (sem casting!)  
➡ ContaCorrente contaDoDiretor = mapaDeContas.get("diretor");
```

- Se você tentar colocar algo diferente de String na chave e ContaCorrente no valor...

Vai ter um erro de compilação.

Exercício – Juntos....

- ▶ A) Qual a saída do código abaixo? B) O que acontece quando a linha 11 é descomentada?

```
1  import java.util.*;
2
3  public class testCollection2 {
4
5      private void testCollection() {
6          ➡ List<String> list = new ArrayList<String>();
7
8          list.add(new String("Hello world!"));
9          ➡ list.add(new String("Good bye!"));
10         list.add("Blz");
11         // list.add(new Integer(95));
12
13         printCollection(list);
14     }
15
16     ➡ private void printCollection(List<String> c) {
17
18         ➡ for (String item : c) {
19             System.out.println("Item FOR: "+item);
20         }
21     }
22
23
24     public static void main(String argv[]) {
25
26         testCollection2 e = new testCollection2();
27         e.testCollection();
28     }
29 }
```

**Implementemos
juntos**



- enhanced for
- forEach

Exercício – Juntos....

- ▶ 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.

AgendaTelefônica
- colecao : Map
+ inserir(nome : String, numero : String) : void
+ buscarNumero(nome : String) : String
+ remover(nome : String) : void
+ tamanho() : int

- ▶ V2.0 : Objeto Contato

▶ A) Qual a saída do código abaixo?

Exercício – Juntos....

implemente

```
package mybaralho;

import java.util.*;

class MyBaralho {
    public static void main(String args[]) {

        int jogadores = 3;
        int qtdCartasPorJogador = 3;

        List<String> naipesC = new ArrayList<>();
        naipesC.add("espadas"); naipesC.add("copas");
        naipesC.add("ouros"); naipesC.add("paus");

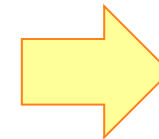
        String[] Arraycartas = new String[]
            {"as", "2", "3", "4", "5", "6", "7", "8", "9", "10", "valete", "dama", "rei"};
        List<String> cartasC = new ArrayList<>(Arrays.asList(Arraycartas));

        List<String> monteC = new ArrayList<>();

        for (String naipe : naipesC)
            for (String carta : cartasC)
                monteC.add(carta + " de " + naipe);

        Collections.shuffle(monteC);

        int i;
        for (i = 0; i < jogadores; i++)
            System.out.println(darCartas(monteC, qtdCartasPorJogador));
    }
}
```



Exercício – Juntos....

- ▶ A) Qual a saída do código abaixo?

```
public static List darCartas(List monte, int qCartasPorJogador) {  
    int tamanhoMonte = monte.size();  
    System.out.println("!! tamanhoMonte !! : " + tamanhoMonte);  
  
    List topoDoMonte = monte.subList(tamanhoMonte-qCartasPorJogador, tamanhoMonte);  
    List mao = new ArrayList(topoDoMonte);  
    topoDoMonte.clear();  
    return mao;  
}
```

implemente