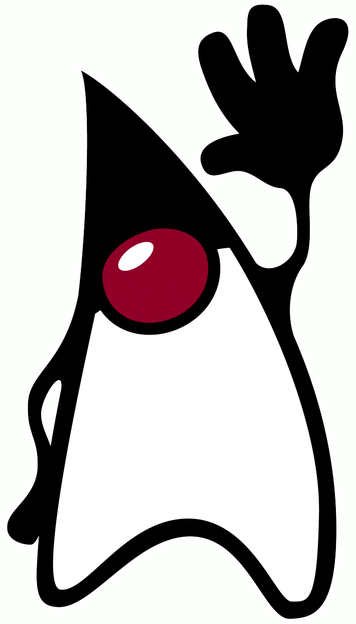
UNIVERSIDADE
FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Collections

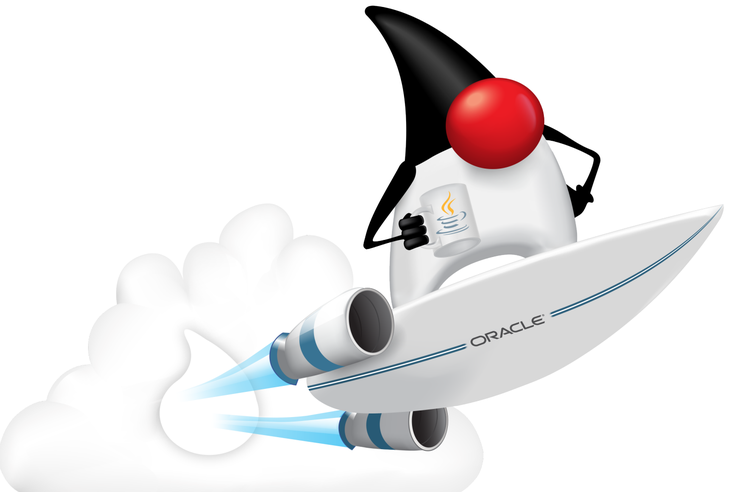
QXD0007 - Programação Orientada a Objetos



Prof. Bruno Góis Mateus (brunomateus@ufc.br)

Conteúdo

- Introdução
- A interface Collection
- Set
- List
- Map


ORACLE

Introdução

Introdução

- A muito tempo atrás, pré Java 2 (1.2), tínhamos as Dictionary, Vector, Stack e Properties para armazenar e manipular grupos de objetos
  - Apesar de úteis essas classes eram limitadas
    - Baseadas em array e difíceis estendê-las
  - Não compartilhavam algo em comum

JAVA 2

- A Sun criou o que nós conhecemos como Collections Framework
  - Conjunto de classes e interfaces que reside no pacote java.util

Introdução

Objetivos

- Prover estruturas de dados de alto desempenho
- Permitir que diferente tipos de coleções funcionem de forma similar, favorecendo a interoperabilidade entre elas
- Permitir a fácil adaptação e customização

Introdução

Collection

- Um objeto que representa um grupo de objetos
  - Ex: Vector

O framework Collections

- Uma arquitetura unificada para representar e manipular coleções (*collections*) independentemente dos detalhes de implementação

Introdução

Vantagens

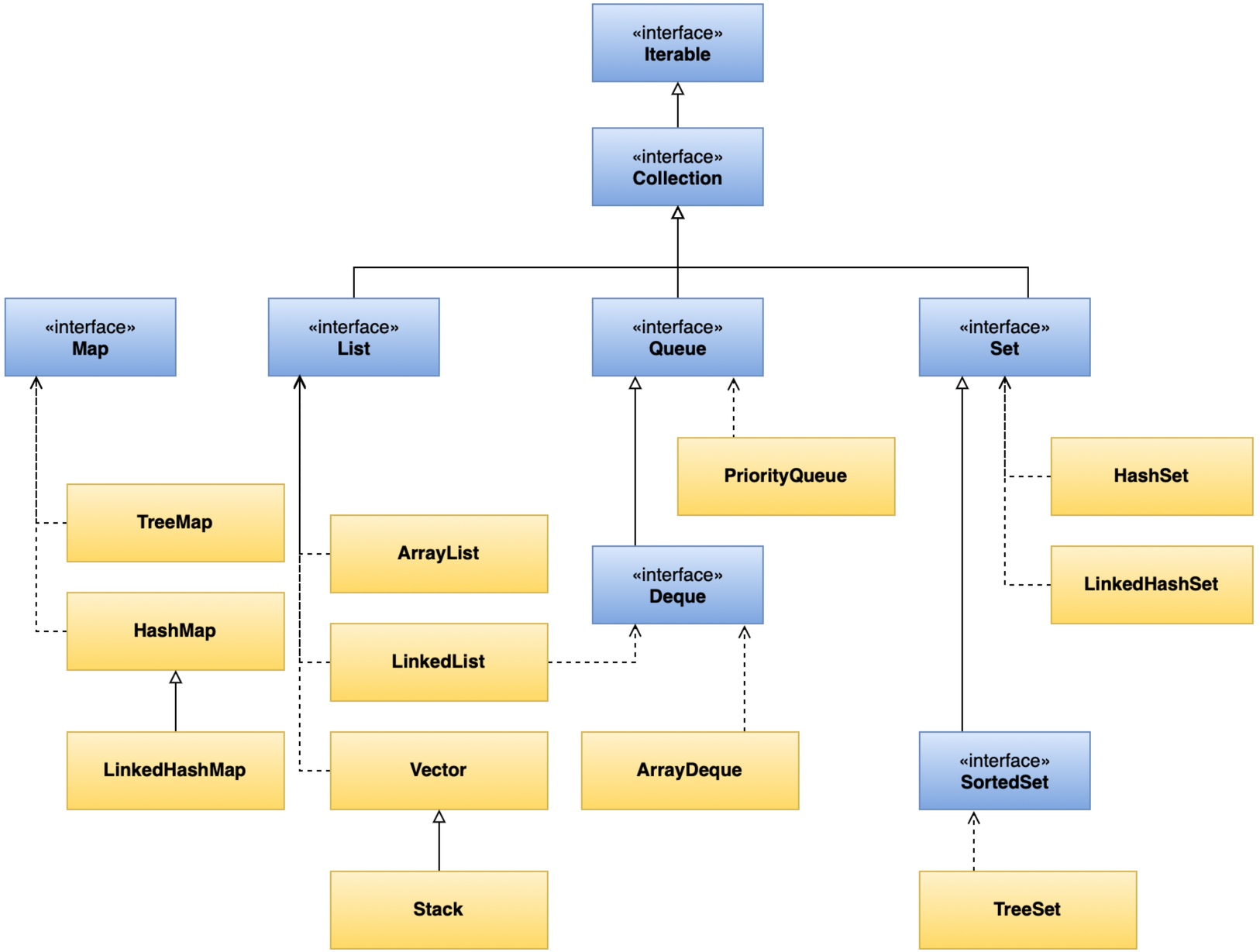
- Reduz o esforço em programação ao prover estruturas de dados e algoritmos prontos
- Aumento de desempenho ao prover implementações de algoritmos e estruturas de dados de alto desempenho (intercambiáveis)
- Reduz o esforço necessário para aprender múltiplas APIs de collections
- Reduz o esforço relacionado ao design e implementação de APIs de collections
- Favorece o reuso de código por meio das interfaces padrões e algoritmos utilizados para manipular as collections

Introdução

Collections framework

- Interfaces que representam os diferentes tipos de collections, com Set, Lists e Maps
- Implementações de caráter gerais
- Implementações legadas de collections vindas de versões anteriores ao framework (Vector e HashTable)
- Implementações para situações mais específicas
- Implementações projetas para o uso em programação concorrente
- Algoritmos, métodos estáticos que fornecem funcionalidades úteis, como ordenação

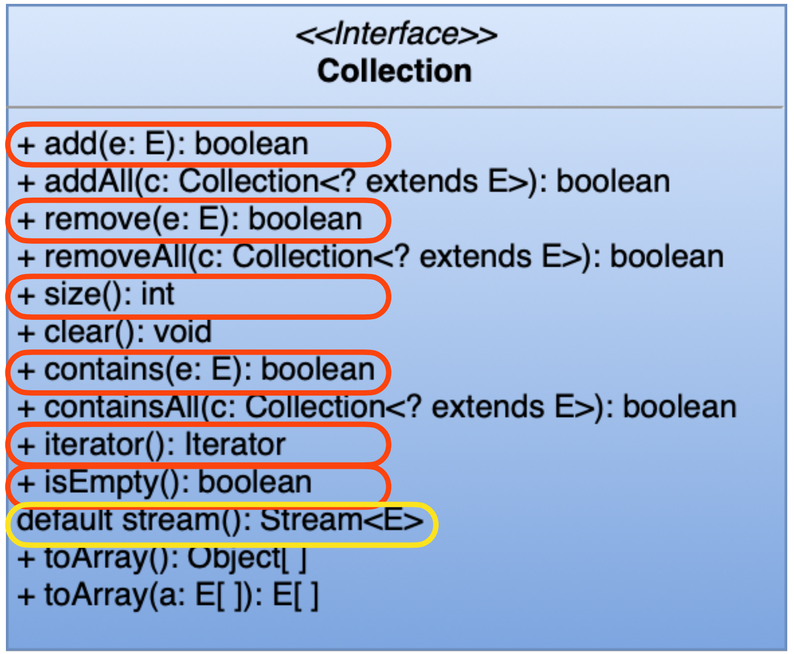
Introdução


«interface»
Iterable
«interface»
Collection
«interface»
Map
«interface»
List
«interface»
Queue
«interface»
Set
PriorityQueue
HashSet
TreeMap
ArrayList
«interface»
Deque
LinkedHashSet
HashMap
LinkedList
LinkedHashMap
Vector
ArrayDeque
«interface»
SortedSet
Stack
TreeSet

A interface Collection

A interface Collections

- Representa um grupo de objetos e seus elementos
- Usada para passar uma coleção de objetos de forma mais genérica possível
- Coleções de caráter geral possui construtores conversores



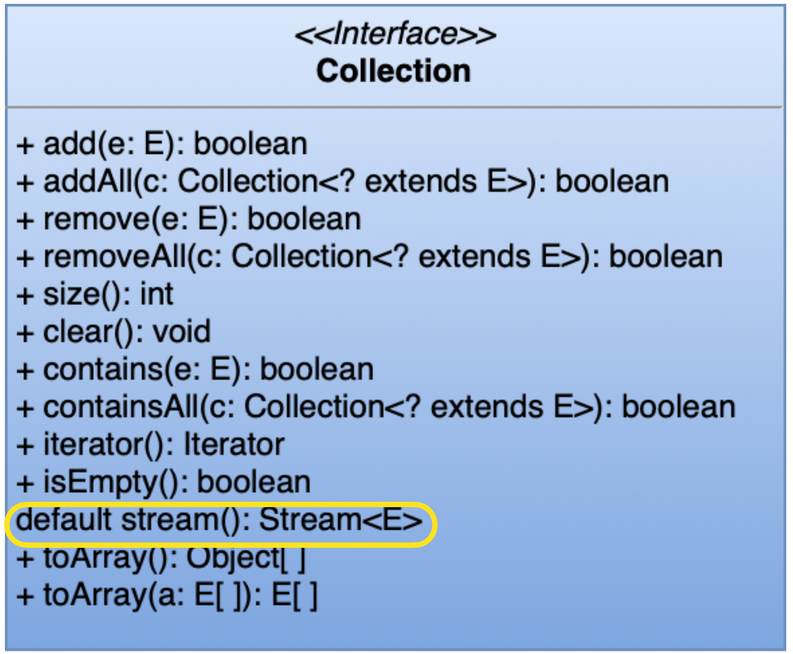

```java
Collection<String> c = //alguma collection;
List<String> list = new ArrayList<String>(c);
List<String> list = new ArrayList<>(c); // A partir do Java 7

Set<String> list = new HashSet<String>(c);
Set<String> list = new HashSep<>(c); // A partir do Java 7
```

A interface Collections

Percorrendo uma collection

- Operações de agregação
  - Melhor maneira para iterar sobre um coleção
  - JDK 8 ou mais recente
  - Utiliza a API de stream junto com o poder das expressões lambdas



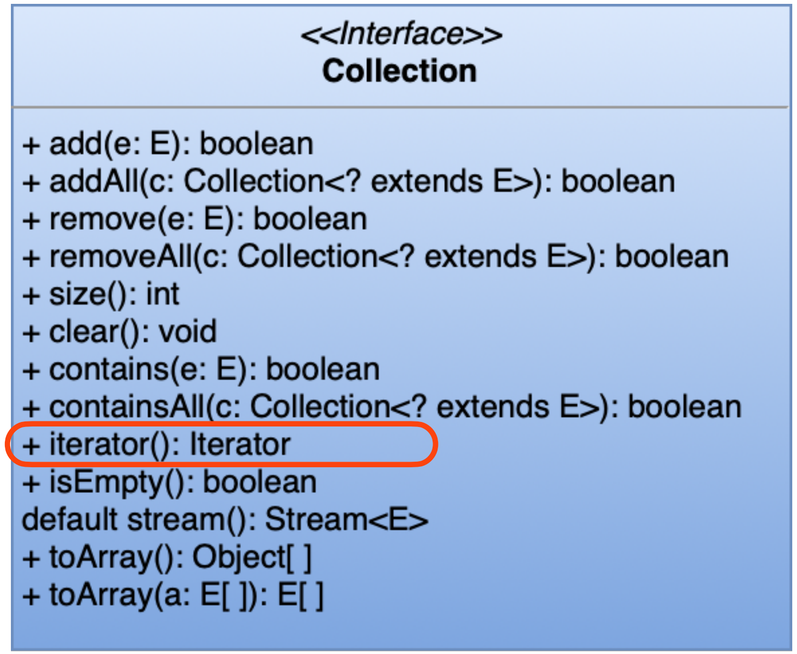

```
myShapesCollection.stream()
.filter(e -> e.getColor() == Color.RED)
.forEach(e -> System.out.println(e.getName()));
```

```
int total = employees.stream()
.collect(
  Collectors.summingInt(Employee::getSalary))
);
```

A interface Collections

Percorrendo uma collection

- For-each
  - Maneira concisa de percorrer uma collection usando um for loop

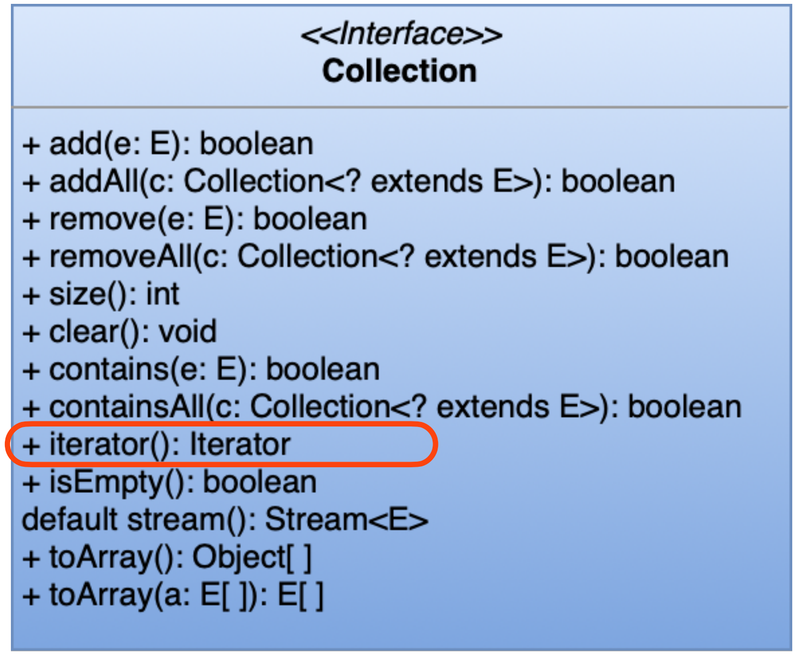



```
for (Object o : collection)
    System.out.println(o);
```

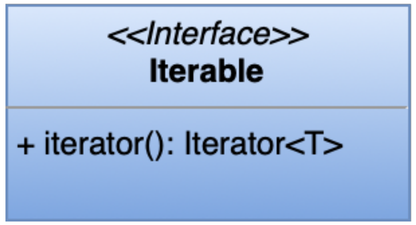
A interface Collections

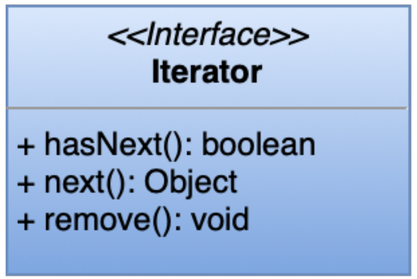
Percorrendo uma collection

- Iterator
  - Permite que você percorra e ao mesmo tempo remova elementos desejados
  - Para recuperar o Iterator é necessário chamar o método iterator()



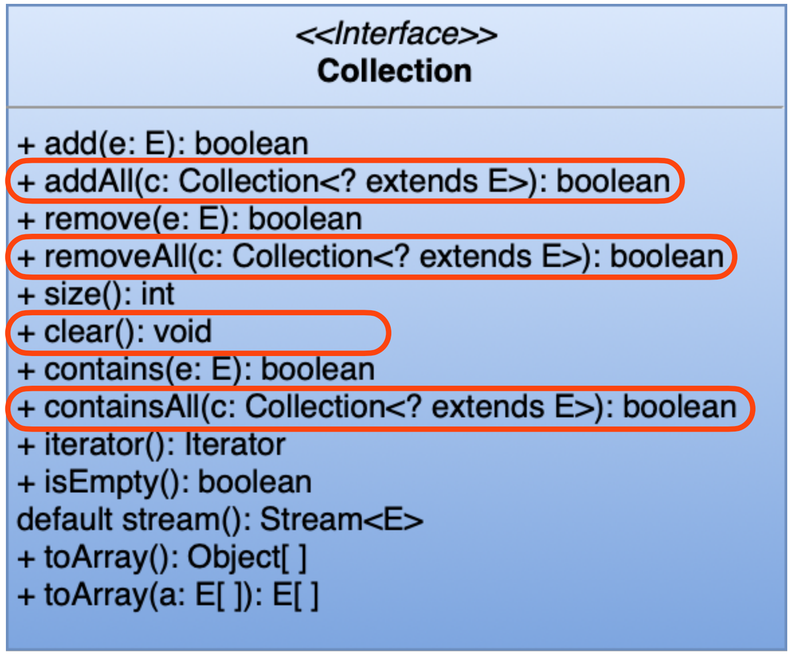

```
Iterator<Integer> it = numbers.iterator();
while(it.hasNext()) {
      Integer i = it.next();
}
```





A interface Collections

- Operações em massa
  - Podem ser implementadas com as operações básicas, porém em geral resultam em código menos eficiente




```
c.removeAll(Collections.singleton(e));
c.removeAll(Collections.singleton(null));
```

Set

Set

Set
• É uma coleção que não contém elementos duplicados • É uma abstração dos conjuntos da matemática • Herdam todos os métodos de Collection e impõe a restrição de duplicidade (*equals e hashCode)* • Dois conjuntos são iguais se ambos possuem os mesmos elementos

Set

- HashSet
  - Armazena os elementos em uma tabela de dispersão
  - Possui o melhor desempenho
  - Não garante a ordem de iteração
- TreeSet
  - Armazena os elementos em um árvore rubro-negra
  - Ordena os elementos de acordo com seus valores (compareTo ou Comparator)
  - Mais lenta que o HashSet
- LinkedHashSet
  - Combina uma tabela de dispersão com um lista encadeada
  - Mantém a ordem de inserção
  - Um pouco mais custoso que o HashSet

Set

```
public class FindDups {
  public static void main(String[] args) {
    Set<String> distinctWords = Arrays.asList(args).stream().collect(Collectors.toSet());
    System.out.println(distinctWords.size()+ " distinct words: " +  distinctWords);
  }
}
```

```
public class FindDups {
  public static void main(String[] args) {
      Set<String> s = new HashSet<String>();
      for (String a : args)
        s.add(a);
      System.out.println(s.size() + " distinct words: " + s);
  }
}
```

Entrada: java FindDups i came i saw i left
Saída: 4 distinct words: [left, came, saw, i]

Set

Operações massa

```
public static void main(String[] args) {
  s1.containsAll(s2); // Equivalente “s2 é subconjunto de s1”
  s1.addAll(s2);      // s1 união s2
  s1.retainAll(s2);   // s1 interseção s2
  s1.removeAll(s2);   // s1 – s2
}
```

List

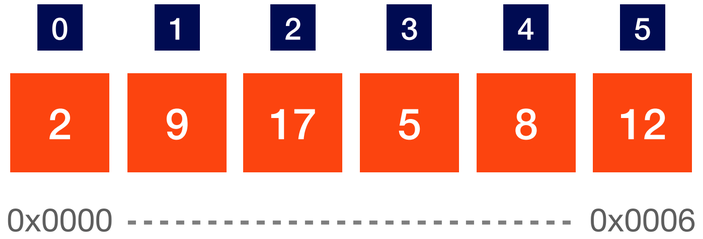
List

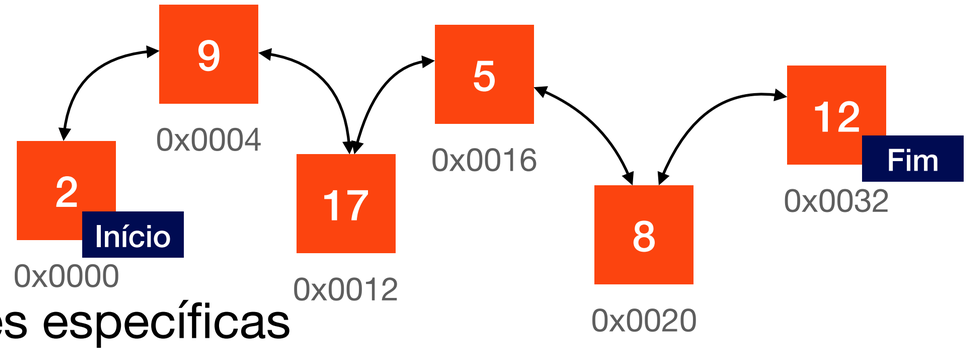
List

- É uma coleção ordenada
- Pode conter elementos duplicados
- Herdam todos os métodos de Collection e provêm ainda:
  - Acesso posicional (*get, set, add, addAll, remove*)
  - Busca por um elemento na lista e retorna sua posição (*indexOf, lastIndexOf*)
  - Sub-listas
- Duas listas são igual se tiverem os mesmos elementos e na mesma ordem

List

- ArrayList
  - Possui o melhor desempenho
  - Inserção e exclusão (linear)
  - Acesso rápido ao elemento
- LinkedList
  - Melhor desempenho em situações específicas
  - Buscas mais lentas (*linear)*
  - Inserções e exclusões rápidas
- Vector e Stack são legadas





add e addAll: Sempre adicionam no final da lista
remove: Sempre remove a primeira ocorrência do objeto

List

- A maioria dos algoritmos disponíveis no framework Collections podem ser aplicados em listas

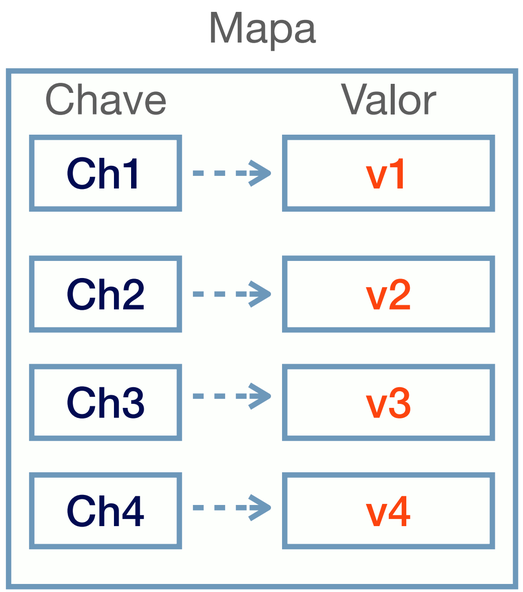
sort	Ordena a lista usando *mergeSort*
shuffle	Embaralha a lista randomicamente
reverse	Inverte a ordem dos elementos da lista
swap	Troca dois elementos de posição
replaceAll	Substituir todas as ocorrências de um elemento por outro
binarySearch	Procura por um elemento na lista utilizando o algoritmo de busca binária

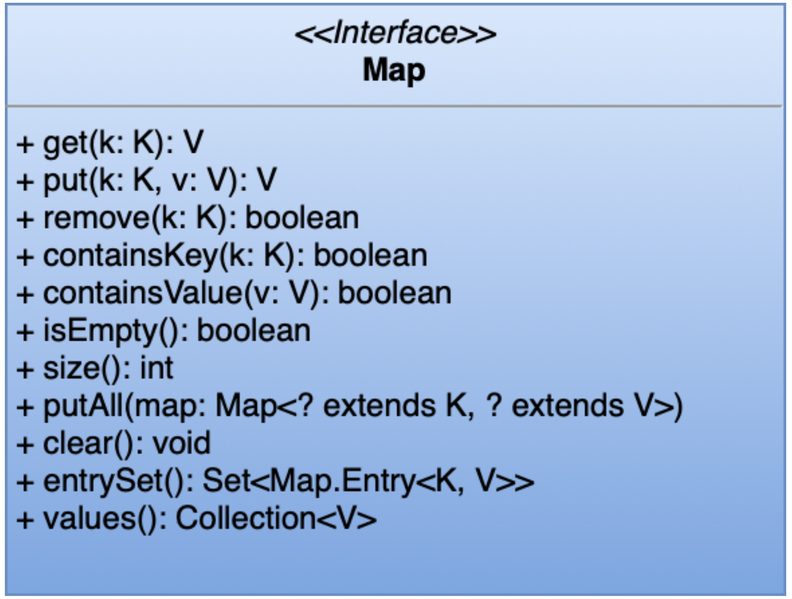
Map

Map

Map

- É um objeto que mapeia chaves a valores
- Não podem conter chaves duplicadas
- Uma chave é mapeada para apenas um valor





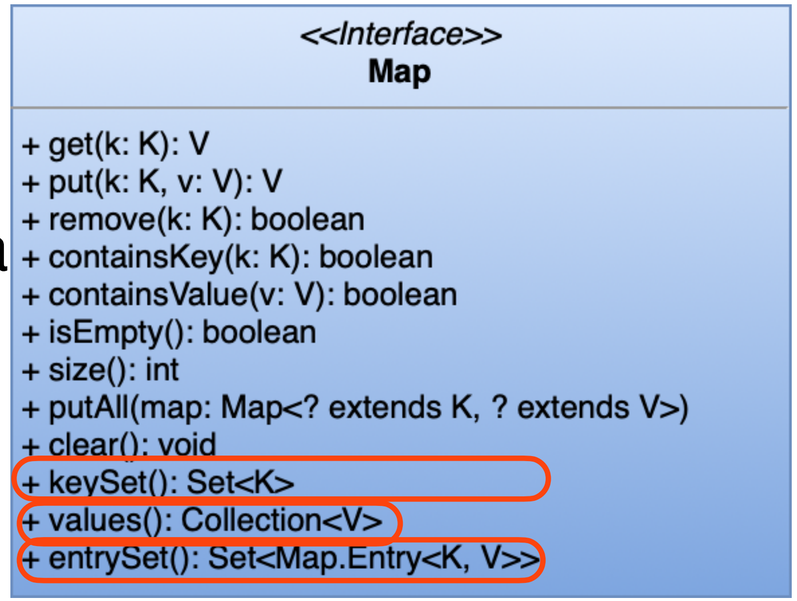
Map

- HashMap
  - Permite chaves null
  - Não garante a ordem dos dados
  - Melhor performance
- TreeMap
  - Garante que o mapa será ordenado na ordem crescentes das chaves
  - É possível especificar outra ordem
- LinkedHashMap
  - Mantém um lista duplamente encadeada entre os itens
  - Ordem de iteração é a ordem de inserção

Map

Percorrendo um mapa

- Os método *collection view* permitem olhar para um mapa de 3 maneiras diferentes
  - São as únicas maneiras de iterar sobre um mapa




```
for (KeyType key : m.keySet())
  System.out.println(key);
```

```
for (ValueType value : m.values())
  System.out.println(value);
```

```
for (Map.Entry<KeyType, ValType> e : m.entrySet())
  System.out.println(e.getKey() + ": " + e.getValue());
```

Por hoje é só