Instituto de computação uff

Técnicas de Programação Avançada

TCC-00.174

Prof.: Anselmo Montenegro

www.ic.uff.br/~anselmo

anselmo@ic.uff.br

Conteúdo: Coleções

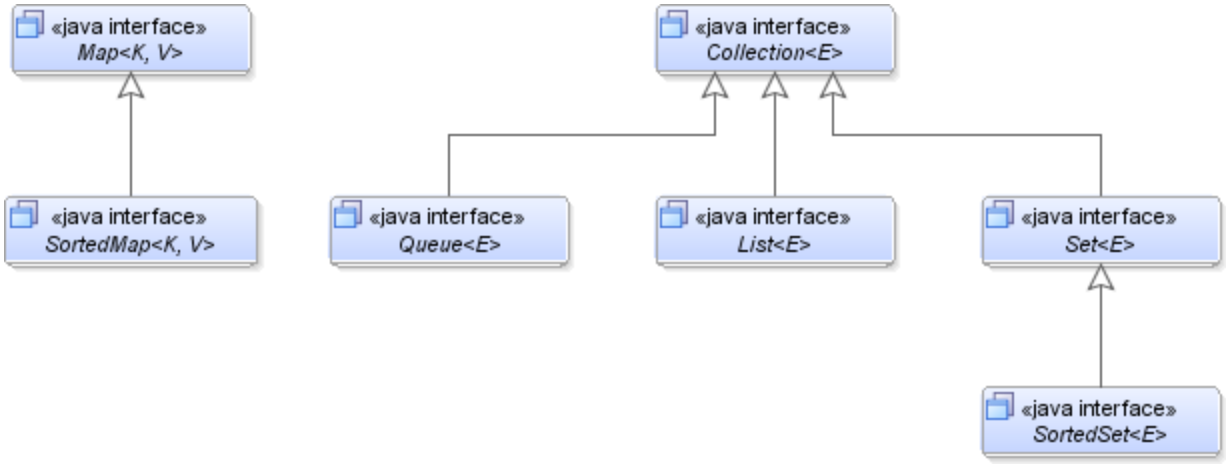
Material elaborado com contribuição do Professor Luiz André

API Collection

Interfaces

Coleções - são estruturas de dados para armazenamento de objetos

Tipos de coleções em Java



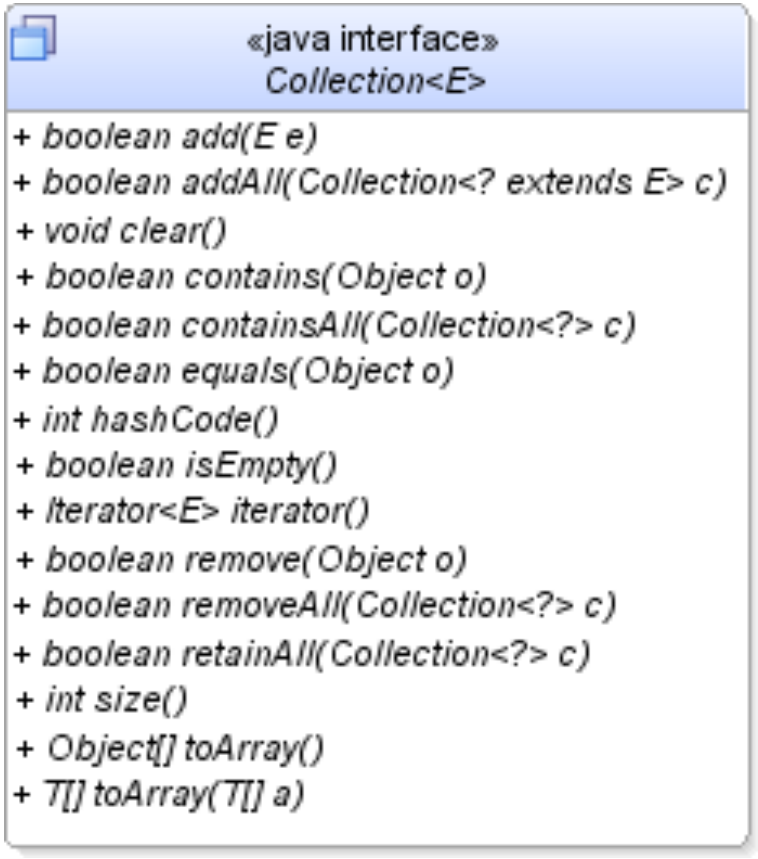
API Collection

Interface Collection

`Collection`

Nível mais alto de abstração

Contém as operações comuns a todas as coleções



API Collection

Interface Collection

For each:

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

Iterator:

```
static void filter(Collection<?> c) {
    for (Iterator<?> it = c.iterator(); it.hasNext(); )
        if (!cond(it.next())) it.remove(); }
```

API Collection

Interface Collection

Como remover todas as instâncias *e* de uma coleção *c*

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

Como remover todos os elementos de tipo null de uma coleção *c*

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

Collections.singleton é um factory method estático que retorna um conjunto (Set) imutável contendo somente o elemento especificado

API Collection

Interface Collection

Como converter uma coleção *c* para um array:

```
Object[] a = c.toArray()
```

```
String[] a = c.toArray(new String[0]);
```

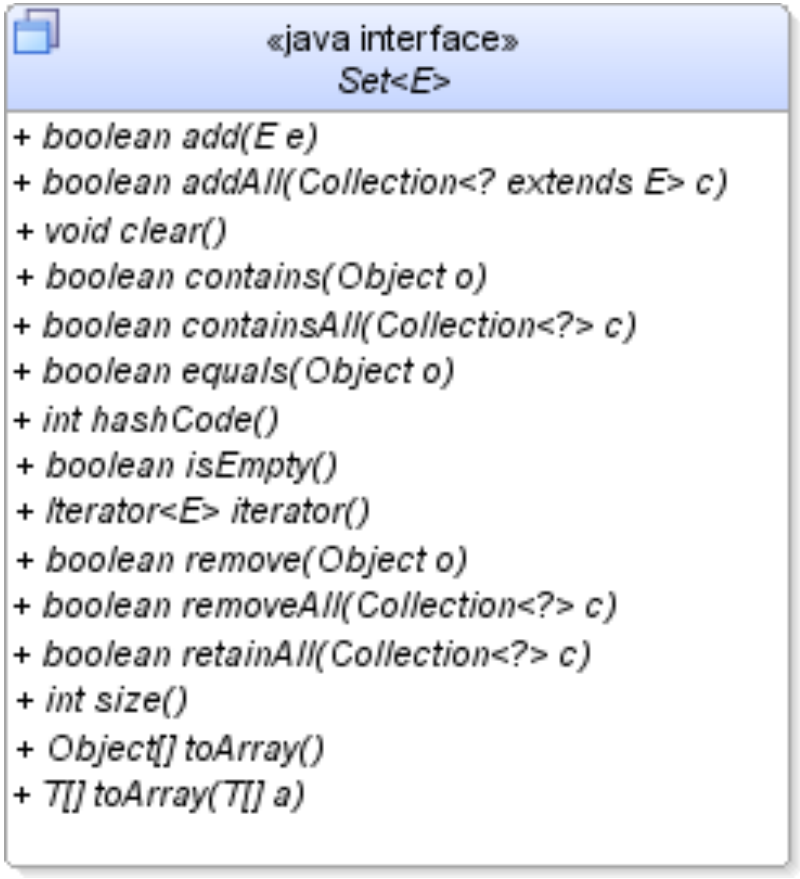
API Collection

Interface Set

`Set`

Conjunto de objetos que não pode conter elementos duplicados

Equivale ao conceito matemático de conjunto



API Collection

Interface Set

Como remover duplicatas de uma coleção usando a interface Set

```
public static <E> Set<E> removeDups(Collection<E> c) {
 return new LinkedHashSet<E>(c);
 }
```

API Collection

Interface Set

Como remover duplicatas de uma coleção usando a interface Set

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

API Collection

Interface Set

União, interseção e diferença não destrutiva de conjuntos:

```
Set<Type> union = new HashSet<Type>(s1); union.addAll(s2);

Set<Type> intersection = new HashSet<Type>(s1); intersection.retainAll(s2);

Set<Type> difference = new HashSet<Type>(s1); difference.removeAll(s2);
```

API Collection

Interface Set

União, interseção e diferença não destrutiva de conjuntos:

```
Set<Type> union = new HashSet<Type>(s1); union.addAll(s2);

Set<Type> intersection = new HashSet<Type>(s1); intersection.retainAll(s2);

Set<Type> difference = new HashSet<Type>(s1); difference.removeAll(s2);
```

API Collection

Interface Set

Elementos únicos, duplicatas e diferença dos dois:

```
import java.util.*;
public class FindDups2 {
    public static void main(String[] args) {
        Set<String> uniques = new HashSet<String>();
        Set<String> dups = new HashSet<String>();

        for (String a : args)
            if (!uniques.add(a))
                dups.add(a);
        Set<String> difference = new HashSet<String>(uniques);
        difference.removeAll(dups);
        System.out.println("Unique words: " + uniques);
        System.out.println("Duplicate words: " + dups);
        System.out.println("Different words:"+difference);
    }
}
```

Diferença simétrica

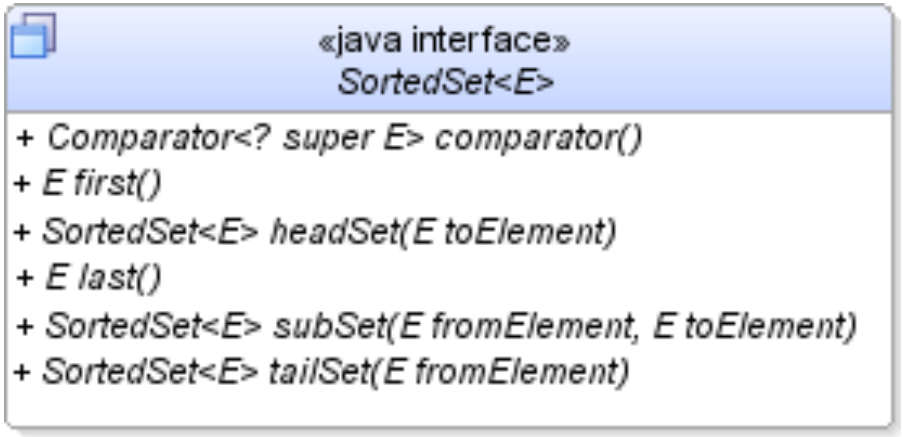
```
Set<Type> symmetricDiff = new HashSet<Type>(s1);
symmetricDiff.addAll(s2);
 Set<Type> tmp = new HashSet<Type>(s1);
 tmp.retainAll(s2));
 symmetricDiff.removeAll(tmp);
```

API Collection

Interface Sorted Set

`SortedSet`

Conjunto objetos classificados em ordem crescente



API Collection

Interface List

`List`

Conjunto ordenado de objetos

Pode conter elementos duplicados

API Collection

Interface List

Concatenação de duas listas:

```
list1.addAll(list2);
```

Versão não destrutiva:

```
List<Type> list3 = new ArrayList<Type>(list1);
list3.addAll(list2);
```

Swap genérico

```
public static <E> void swap(List<E> a, int i, int j) {
 E tmp = a.get(i); a.set(i, a.get(j)); a.set(j, tmp);
}
```

Shuffle genérico:

```
public static void shuffle(List<?> list, Random rnd) {
    for (int i = list.size(); i > 1; i--)
            swap(list, i - 1, rnd.nextInt(i));
}
```

Swap genérico

```
public static <E> void swap(List<E> a, int i, int j) {
 E tmp = a.get(i); a.set(i, a.get(j)); a.set(j, tmp);
}
```

Shuffle genérico:

```
public static void shuffle(List<?> list, Random rnd) {
    for (int i = list.size(); i > 1; i--)
            swap(list, i - 1, rnd.nextInt(i));
}
```

Shuffle genérico II:

```
import java.util.*;
public class Shuffle {
    public static void main(String[] args) {
        List<String> list = new ArrayList<String>();
                for (String a : args)
                    list.add(a);
        Collections.shuffle(list, new Random());
        System.out.println(list);
    }
 }
```

Shuffle genérico III:

```
import java.util.*;
public class Shuffle {
    public static void main(String[] args) {
        List<String> list = Arrays.asList(args);
        Collections.shuffle(list);
    System.out.println(list);
    }
 }
```

API Collection

Interface List

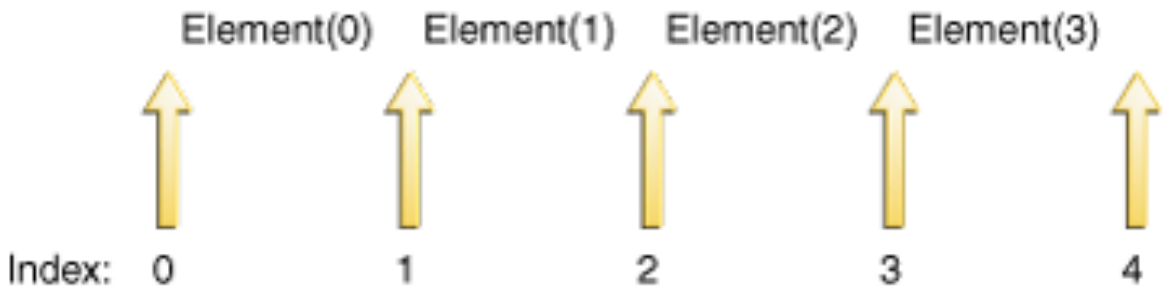
```
public interface ListIterator<E> extends Iterator<E> {
    boolean hasNext();
    E next();
    boolean hasPrevious();
    E previous();
    int nextIndex();
    int previousIndex();
    void remove(); //optional
    void set(E e);   //optional
    void add(E e); //optional
}
```

API Collection

Interface List

Os cursores ficam nos vãos (*gaps*) entre os elementos

Para n elementos existem n+1 gaps



previous () aplicado ao cursor antes de 0 retorna -1

next() aplicado ao cursor após o último elemento retorna o tamanho da lista

API Collection

Interface List

Percorrimento inverso de uma lista

```
for (ListIterator<Type> it = list.listIterator(list.size());
    it.hasPrevious(); ) {
        Type t = it.previous(); ...
}
```

Implementação do indexOf(e)

```
public int indexOf(E e) {
    for (ListIterator<E> it = listIterator(); it.hasNext(); )
        if (e == null ? it.next() == null : e.equals(it.next()))
            return it.previousIndex();
        // Element not found
        return -1;
}
```

API Collection

Interface List

Substitui todas as ocorrências de um determinado valor por outro

```
public static <E> void replace(List<E> list, E val, E newVal) {
    for (ListIterator<E> it = list.listIterator(); it.hasNext(); )
        if (val == null ? it.next() == null : val.equals(it.next()))
            it.set(newVal);
}
```

API Collection

Interface List

Substitui todas as ocorrências de um determinado valor uma sequencia dada por uma outra lista

```
public static <E> void replace(List<E> list, E val, List<? extends E>
    newVals) {
        for (ListIterator<E> it = list.listIterator(); it.hasNext(); ){
            if (val == null ? it.next() == null : val.equals(it.next())) {
            it.remove();
            for (E e : newVals)
                it.add(e);
        }
    }
}
```

API Collection

Interface List

Sublist cria uma *view* dos elementos de uma lista entre dois índices

Remove todos os elementos em um intervalo de índices da lista:

```
list.subList(fromIndex, toIndex).clear();
```

API Collection

Interface List

Retorna o índice da primeira ocorrência de elemento em uma visão da lista, caso exista:

```
int i = list.subList(fromIndex, toIndex).indexOf(o);
```

Retorna o índice da última ocorrência de elemento em uma visão da lista, caso exista:

```
int j = list.subList(fromIndex, toIndex).lastIndexOf(o);
```

Obs.: os índices são índices na view (na sublista), não na lista original!!!

API Collection
Interface Queue

`Queue`

Fila de objetos

Ordenação pode ser FIFO ou determinada por prioridade

API Collection

Interface Queue

Heap sort usando a interface Queue

```
static <E> List<E> heapSort(Collection<E> c) {
   Queue<E> queue = new PriorityQueue<E>(c);
   List<E> result = new ArrayList<E>();
   while (!queue.isEmpty())
     result.add(queue.remove());
   return result;
 }
```

API Collection

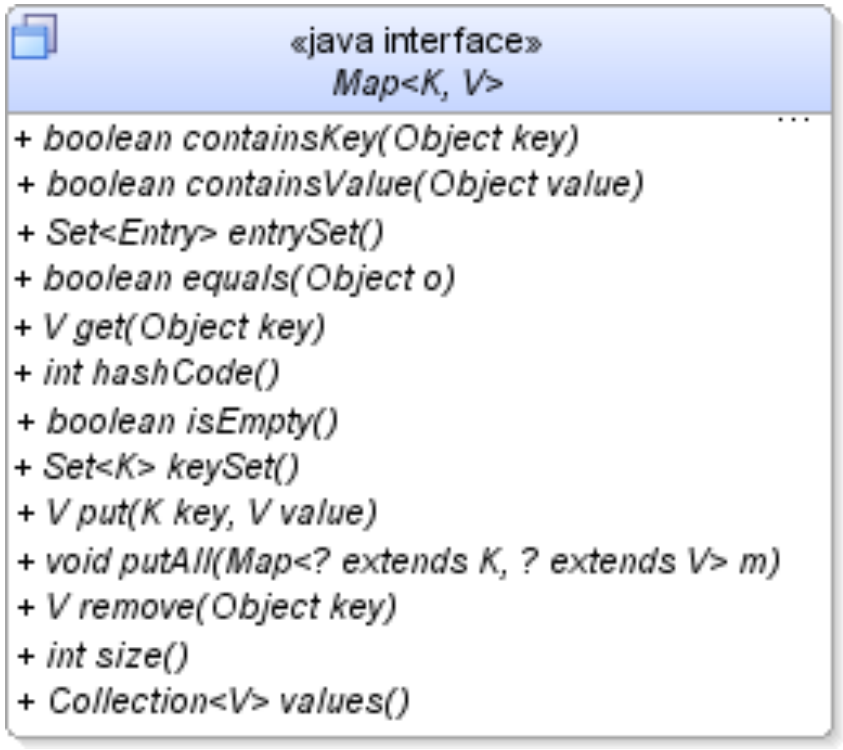
Interface Map

`Map`

Conjunto de objetos acessados por chaves

Não pode conter chave duplicadas

Cada chave está associada a somente um objeto



API Collection

Interface Map

Construir um mapa de frequências de palavras

```
import java.util.*; public class Freq {
    public static void main(String[] args) {
        Map<String, Integer> m = new HashMap<String, Integer>();
        // Initialize frequency table from command line
        for (String a : args) {
            Integer freq = m.get(a);
            m.put(a, (freq == null) ? 1 : freq + 1);
        }
        System.out.println(m.size() + " distinct words:");
        System.out.println(m);
    }
}
```

Operações Bulk em Map

Criando um mapa com valores default

```
static <K, V> Map<K, V> newAttributeMap(Map<K,
    V>defaults, Map<K, V> overrides) { Map<K, V> result =
    new HashMap<K, V>(defaults); result.putAll(overrides);
    return result; }
```

API Collection

Interface Map

Collection View de Map

Iterar sobre todas as chaves de um mapa

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

Filtra elementos de um mapa

```
for (Iterator<Type> it = m.keySet().iterator(); it.hasNext(); )
    if (it.next().isBogus()) it.remove();
```

API Collection

Interface Map

Collection View de Map

Itera sobre todos os pares chave-valor de um mapa

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

A API somente cria o objeto entrySet uma vez para questões de eficiência....

Entry View de Map

Itera sobre todos os pares chave-valor de um mapa

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

A API somente cria o objeto entrySet uma vez para questões de eficiência....

API Collection

Interface Map

Observações:

Entry Set View suporta alteração de valores do Mapa

Collection View somente suporta remove, removeAll e clear

Álgebra de Mapas

Como determinar se um mapa é submapa de um outro

```
if (m1.entrySet().containsAll(m2.entrySet())) { ... }
```

Como determinar se dois mapas tem as mesmas chaves

```
if (m1.keySet().equals(m2.keySet())) { ... }
```

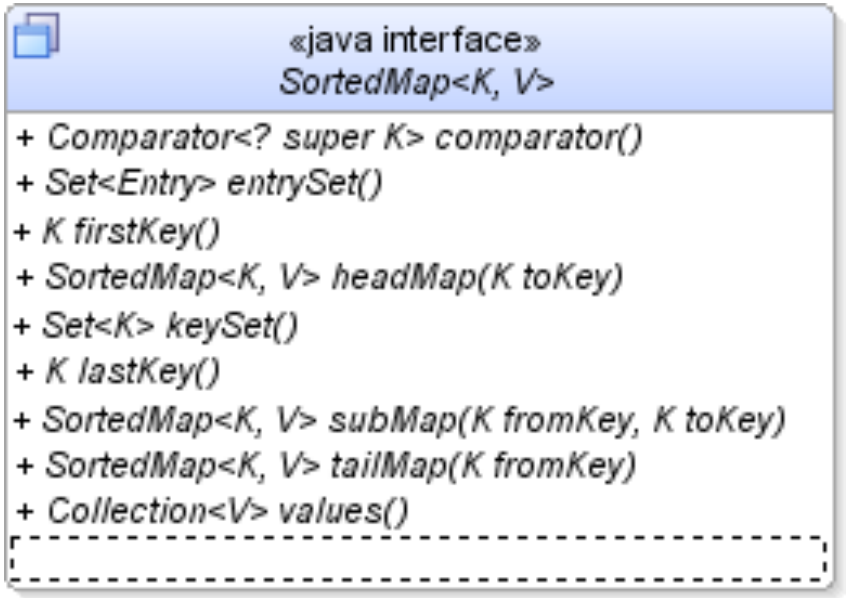
Álgebra de Mapas

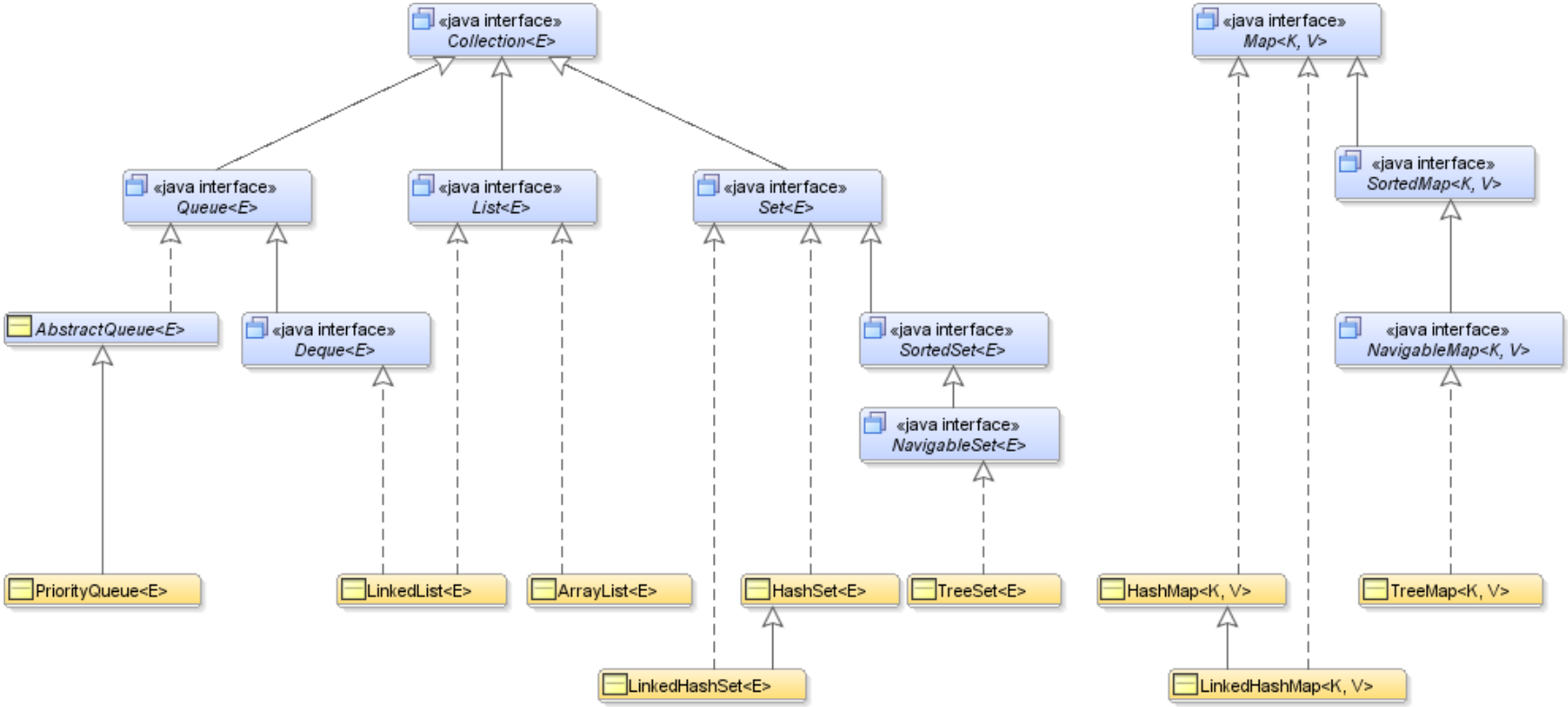
Chaves comuns entre dois mapas

```
Set<KeyType>commonKeys = new
    HashSet<KeyType>(m1.keySet());
    commonKeys.retainAll(m2.keySet());
```

SortedMap

Mantém as chaves classificadas em ordem crescente




«java interface»
Collection<E>
«java interface»
Queue<E>
«java interface»
List<E>
«java interface»
Set<E>
AbstractQueue<E>
«java interface»
Deque<E>
«java interface»
SortedSet<E>
«java interface»
NavigableSet<E>
PriorityQueue<E>
LinkedList<E>
ArrayList<E>
HashSet<E>
TreeSet<E>
LinkedHashSet<E>
«java interface»
Map<K, V>
«java interface»
SortedMap<K, V>
«java interface»
NavigableMap<K, V>
HashMap<K, V>
TreeMap<K, V>
LinkedHashMap<K, V>

Orientação a Objetos

Coleções - Implementações

- `HashSet`
  - Ordena objetos em hash table
- `TreeSet`
  - Ordena objetos em árvore
  - Objetos armazenados devem implementar interface `Comparable`
- `LinkedHashSet`
  - Armazena a ordem de entrada dos elementos em uma lista duplamente encadeada
  - Ordem de iteração preserva ordem de entrada
- `ArrayList`
  - Armazena objetos em lista simples
- `LinkedList`
  - Lista duplamente encadeada
  - FIFO
- `PriorityQueue`
  - Lista

- `TreeMap`
  - Ordena chave em árvore
- `HashMap`
  - Ordena chave em hash table
- `LinkedHashMap`
  - Mantém a ordem em que os elementos foram incluídos no mapa em uma lista duplamente encadeada
  - A ordem de inserção é preservada em iterações sobre as chaves

- Exemplos
  - Dias da semana: segunda, terça, etc.
  - Frutas: morango, mamão, abacaxi, etc.
- A ordem dos valores é definida
- Os elementos são literais constantes
- Definição Java

```
public enum Day {
    SEGUNDA, TERCA, QUARTA, QUINTA, SEXTA, SABADO, DOMINGO;
}
```

- Comparação

```
Day day = Day.SEGUNDA;
System.out.println(day.compareTo(Day.TERCA));
```

Tipos Especiais de Coleções - Enumerações

```
public enum Planet {
  MERCURY (3.303e+23, 2.4397e6),
  VENUS (4.869e+24, 6.0518e6),
  EARTH (5.976e+24, 6.37814e6),
  MARS (6.421e+23, 3.3972e6),
  JUPITER (1.9e+27, 7.1492e7),
  SATURN (5.688e+26, 6.0268e7),
  URANUS (8.686e+25, 2.5559e7),
  NEPTUNE (1.024e+26, 2.4746e7);

  private final double mass;
  private final double radius;
  public static final double G = 6.67300E-11;

...
```

```
...
  Planet(double mass, double radius) {
    this.mass = mass; this.radius =
    radius;
  }
  private double mass() {
    return mass;
  }
  private double radius() {
    return radius;
  }
  public double surfaceGravity() {
    return G * mass / (radius *
    radius);
  }
  public double surfaceWeight(double
  otherMass){
    return otherMass *
    surfaceGravity();
  }
}
```

```
public static void main(String[] args) {
   Planet planet = Planet.MARS;
   System.out.println(planet.surfaceWeight(72));
}
```

- Coleção de valores enumeráveis
  - **class EnumSet<E extends Enum<E>>**
  - **public static <E extends Enum<E>> EnumSet<E> noneOf**
  - **(Class<E> elementType)**

```
Set<Day> myEnum =
EnumSet.allOf(Day.class);

for (Day myday : myEnum)
    System.out.println(myday.toString());
```

- Collections: http://docs.oracle.com/javase/tutorial/collections/
- Collections Interfaces : http://docs.oracle.com/javase/tutorial/collections/interfaces/index.html
- Collections Implementations : http://docs.oracle.com/javase/tutorial/collections/implementations/index.html
- Collections Algorithms: http://docs.oracle.com/javase/tutorial/collections/algorithms/index.html