```java
package dataStructures;

/**
 * @author Ricardo Gaspar Nr. 35277
 * @author Hugo António Nr. 34334 Turno P2 Docente Vasco Amaral
 */
public class SepChainHashTable<K extends Comparable<K>, V> extends
        HashTable<K, V> {

    static final long serialVersionUID = 0L;

    // The array of dictionaries.
    protected Dictionary<K, V>[] table;

    public SepChainHashTable(int capacity) {
        initialize(capacity);

    }

    public SepChainHashTable() {
        this(DEFAULT_CAPACITY);
    }

    /**
     * Returns the hash value of the specified key.
     *
     * @param key
     *            The specified key.
     * @return hash value of the specified key.
     */
    protected int hash(K key) {
        return Math.abs(key.hashCode()) % table.length;
    }

    /*
     * (non-Javadoc)
     *
     * @see dataStructures.HashTable#find(java.lang.Object)
     */
    public V find(K key) {
        return table[this.hash(key)].find(key);
```

```
42      }
43
44      /*
45       * (non-Javadoc)
46       *
47       * @see dataStructures.HashTable#insert(java.lang.Object,
    java.lang.Object)
48       */
49      public V insert(K key, V value) {
50
51          if (this.isFull())
52              this.reHash();
53          V resultValue = table[hash(key)].insert(key, value); //
    hash(key) é a
54                                                                  //
    posição
55          // Caso o tenha adicionado uma nova entrada
56          if (resultValue == null)
57              currentSize++;
58
59          return resultValue;
60      }
61
62      /*
63       * (non-Javadoc)
64       *
65       * @see dataStructures.HashTable#remove(java.lang.Object)
66       */
67      public V remove(K key) {
68
69          if (this.isEmpty())
70              return null;
71          V resultValue = table[hash(key)].remove(key); // hash(key)
    é a
72          // posição
73          // Caso o tenha adicionado uma nova entrada
74          if (resultValue != null)
75              currentSize--;
76
77          return resultValue;
78      }
```

```
79
80      /*
81       * (non-Javadoc)
82       *
83       * @see dataStructures.HashTable#iterator()
84       */
85      public Iterator<Entry<K, V>> iterator() {
86
87          return new SepChainHashTableIterator<K, V>(this.table);
88      }
89
90      /**
91       * Cria uma nova HashTable, com o dobro da capacidade. Copia os
    dados da
92       * antiga e insere-os na nova.
93       */
94      protected void reHash() {
95
96          // Criar o iterador de entries existentes na tabela inicial
97          // Assim é possível iterar a tabela antiga
98          Iterator<Entry<K, V>> it = this.iterator();
99          //Aumentar a dimensão da tabela
100         initialize(maxSize * 2);
101         while (it.hasNext()) {
102             Entry<K, V> entry = it.next();
103             this.insert(entry.getKey(), entry.getValue());
104         }
105
106     }
107
108     /**
109      * Inicializa e redimensiona a HashTable existente para um
    tamanho dado.
110      *
111      * @param newMaxSize
112      *             Novo tamanho da HashTable
113      */
114     private void  initialize(int newMaxSize) {
115
116         int arraySize = HashTable.nextPrime((int) (1.1 *
    newMaxSize));
```

```
117         // Afectar a variável correspondente à tabela existente com
    uma nova tabela
118         // Compiler gives a warning.
119         table = (Dictionary<K, V>[]) new Dictionary[arraySize];
120         for (int i = 0; i < arraySize; i++)
121             table[i] = new OrderedDoublyLL<K, V>();
122 
123         maxSize = newMaxSize;
124         currentSize = 0;
125 
126     }
127 
128 }
129 
```