

```
1 using System.Collections.Generic;
2 using System.Security.Cryptography.X509Certificates;
3
4 namespace DAMLib
5 {
6     public class DictionaryCollection<K, V>
7     {
8         private Item[] _item = new Item[0];
9
10        public delegate bool DelegateFilterKeyValue(K key, V value);
11        public delegate bool DelegateFilterWithoutParameters();
12        public delegate bool DelegateFilterKey(K key);
13
14        private class Item
15        {
16            public K key;
17            public V value;
18
19            public Item(K key, V value)
20            {
21                this.key = key;
22                this.value = value;
23            }
24
25            public K Key
26            {
27                get { return key; }
28                set { key = value; }
29            }
30            public V Value
31            {
32                get { return value; }
33                set { this.value = value; }
34            }
35        }
36
37        public int Count => _item.Length;
38        public bool IsEmpty => _item.Length < 0;
39
40        // Funcion que añade una Key y un value. La Key no se puede
41        // repetir.
42        public void Add(K key, V value)
43        {
44            if(ContainsKey(key))
45                return;
46
47            int count = _item.Length;
48            Item[] setResult = new Item[count + 1];
49            Item element = new Item(default, default);
50            setResult[count] = element;
51
52            for (int i = 0; i < count; i++)
```

```
53         {
54             setResult[i] = _item[i];
55         }
56
57         setResult[count].Key = key;
58         setResult[count].Value = value;
59
60         _item = setResult;
61     }
62
63     //Funcion que elimina el Item que ocupa la posicion del indice  ➤
64     //indicado en parametros.
65     public void RemoveAt(int index)
66     {
67         if (index < 0 || index > _item.Length)
68             return;
69
70         if (index == -1)
71             return;
72
73         int count = _item.Length;
74         Item[] arrayResult = new Item[count - 1];
75
76         for (int i = 0; i < index; i++)
77         {
78             arrayResult[i] = _item[i];
79         }
80
81         for (int i = index; i < count - 2; i++)
82         {
83             arrayResult[i] = _item[i + 1];
84         }
85
86         _item = arrayResult;
87     }
88
89     // Funcion que devuelve el indice que ocupa el elemento de  ➤
90     //value V.
91     public int IndexOf(V value)
92     {
93         if (value == null)
94             return 0;
95
96         for (int i = 0; i < _item.Length; i++)
97         {
98             if (_item[i].Value.Equals(value))
99                 return i;
100         }
101         return -1;
102     }
103
104     // Funcion que devuelve el elemento que contiene la key  ➤
105     //indicada.
```

```
103     public V GetElementAt(K key)
104     {
105         if (key == null)
106             return default(V);
107
108         for (int i = 0; i < _item.Length; i++)
109         {
110             if (_item[i].Key.Equals(key))
111                 return _item[i].Value;
112         }
113         return default(V);
114     }
115
116     // Funcion que evalua si el diccionario contiene una Key           ↗
117     // determinada.
118     public bool ContainsKey(K key)
119     {
120         // return IndexOf >= 0;
121         if (key == null)
122             return false;
123
124         for(int i = 0; i < _item.Length;i++)
125         {
126             if (_item[i].Key.Equals(key))
127                 return true;
128         }
129         return false;
130     }
131
132     // Funcion que devuelve si dos objetos son iguales.
133     public override bool Equals(object? obj)
134     {
135         return (this == obj);
136     }
137
138     // Funcion delegada Filter que devuelve un diccionario.
139     public DictionaryCollection<K, V> Filter(DelegateFilterKeyValue ↗
140     del)
141     {
142         DictionaryCollection<K, V> dictionaryResult = new           ↗
143         DictionaryCollection<K, V>();
144
145         for (int i = 0; i < _item.Length; i++)
146         {
147             bool InsertIntoCollection = del(_item[i].Key, _item     ↗
148             [i].Value);
149             if (InsertIntoCollection)
150             {
151                 dictionaryResult.Add(_item[i].Key, _item[i].Value);
152             }
153         }
154
155         return dictionaryResult;
```

```
152     }
153
154     public DictionaryCollection<K, V> Filter(DelegateFilterKey del)
155     {
156         DictionaryCollection<K, V> dictionaryResult = new DictionaryCollection<K, V>();
157
158         for (int i = 0; i < _item.Length; i++)
159         {
160             bool InsertIntoCollection = del(_item[i].Key);
161             if (InsertIntoCollection)
162             {
163                 dictionaryResult.Add(_item[i].Key, _item[i].Value);
164             }
165         }
166
167         return dictionaryResult;
168     }
169     public DictionaryCollection<K, V> Filter
170     (DelegateFilterWithoutParameters del)
171     {
172         DictionaryCollection<K, V> dictionaryResult = new DictionaryCollection<K, V>();
173
174         for (int i = 0; i < _item.Length; i++)
175         {
176             dictionaryResult.Add(_item[i].Key, _item[i].Value);
177         }
178
179         return dictionaryResult;
180     }
181     // Funcion que elimina todo el contenido de un diccionario.
182     public void Clear()
183     {
184         _item = Array.Empty<Item>();
185     }
186
187     // Funcion que devuelve el codigo Hash de un elemento.
188     public override int GetHashCode()
189     {
190         return 133 * 533 * 224 * _item.GetHashCode();
191     }
192
193     public override string ToString()
194     {
195         string result = "";
196         foreach (Item i in _item)
197         {
198             result += $"La key {i.Key}, contiene el value {i.Value} \n";
199         }
200         return result;
```

```
201         }  
202     }  
203 }  
204
```