```
using System.Reflection.PortableExecutable;
using System.Xml.Linq;

namespace DAMLib
{
    public class SetWithHash<T>
    {
        public T[] _set;
        public int[] _hash;


        public bool Empty => _set.Length == 0;
        public int Count
        {
            get
            {
                if (_set == null)
                    return 0;
                else
                    return _set.Length;
            }
        }


        public SetWithHash()
        {
            _set = new T[0];
            _hash = new int[0];
        }

        // Funcion que añade un elemento generico al Set. Incluye un
          numero entero para el hash del elemento.
        public void Add(T element)
        {
            if (element == null)
                return;

            if (!Contains(element))
            {
                int count = _set.Length;

                T[] setResult = new T[count + 1];
                int[] hashResult = new int[count + 1];

                for (int i = 0; i < count; i++)
                {
                    setResult[i] = _set[i];
                    hashResult[i] = _hash[i];
                }

                setResult[count] = element;
                hashResult[count] = GetHashCode();

```

```
53                  _set = setResult;
54                  _hash = hashResult;
55              }
56          }
57
58          // Funcion que elimina un elemento generico del Set
59          public void Remove(T element)
60          {
61              int index = IndexOf(element);
62
63              if (index == -1)
64                  return;
65
66              int count = _set.Length;
67              T[] arrayResult = new T[count - 1];
68              int[] hashResult = new int[count - 1];
69
70              for (int i = 0; i < count; i++)
71              {
72                  if (i == index)
73                  {
74                      continue;
75                  }
76                  else
77                  {
78                      arrayResult[i] = _set[i];
79                      hashResult[i] = _hash[i];
80                  }
81              }
82
83              _set = arrayResult;
84              _hash = hashResult;
85          }
86
87          // Funcion que devuelve el numero Hash del elemento.
88          public int HashWithIndex(int index)
89          {
90              return _hash[index];
91          }
92
93          // Funcion que devuelve el indice que ocupa el elemento en el
              Set.
94          public int IndexOf(T element)
95          {
96              if (element == null)
97                  return -1;
98
99              int hash = element.GetHashCode();
100
101             for (int i = 0; i < _set.Length; i++)
102             {
103                 if (hash == _set[i].GetHashCode() &&
104                     _set[i].Equals(element))
```

```
105                     return i;
106             }
107             return -1;
108         }
109
110         // Funcion que devuelve verdadero o falso si el Set contiene el elemento generico.
111         public bool Contains(T element)
112         {
113             if (element == null)
114                 return false;
115
116             int index = IndexOf(element);
117
118             if (index == -1)
119                 return false;
120
121             if (_hash[index].GetHashCode() == element.GetHashCode())
122                 return true;
123             return false;
124         }
125
126
127         // Funcion que devuelve un booleano si dos objetos son iguales
128         public override bool Equals(object? obj)
129         {
130             return this == obj;
131         }
132
133         // Funcion que devuelve verdadero si dos objetos son iguales y tienen identicos atributos.
134         public bool EqualsDeep(object? obj)
135         {
136             if (this == obj)
137                 return true;
138
139             if (obj is not SetWithHash<T>)
140                 return false;
141
142             SetWithHash<T> testObj = (SetWithHash<T>)obj;
143
144             if (this._set == testObj._set && this._hash == testObj._hash)
145                 return true;
146             return false;
147         }
148
149         // Funcion que sobreescribe la funcion que devuelve el codigo hash de un objeto.
150         public override int GetHashCode()
151         {
152             return 133 * 533 * 224 * _set.GetHashCode();
153         }
```

```
154
155         public override string ToString()
156         {
157             string result = "";
158
159             for (int i = 0; i < _set.Length; i++)
160             {
161                 result += string.Format("El valor {0} ocupa la posicion {1} y cuyo hash es el numero {2}\n", _set[i], i, _hash[i]);
162             }
163
164             return result;
165         }
166     }
167 }
168
```