```
using System.Collections.Generic;
using System.Security.Cryptography.X509Certificates;

namespace DAMLib
{
    public class DictionaryCollection<K, V>
    {
        private Item[] _item = new Item[0];

        public delegate bool DelegateFilterKeyValue(K key, V value);
        public delegate bool DelegateFilterWithoutParameters();
        public delegate bool DelegateFilterKey(K key);

        private class Item
        {
            public K key;
            public V value;

            public Item(K key, V value)
            {
                this.key = key;
                this.value = value;
            }

            public K Key
            {
                get { return key; }
                set { key = value; }
            }
            public V Value
            {
                get { return value; }
                set { this.value = value; }
            }
        }

        public int Count => _item.Length;
        public bool IsEmpty => _item.Length < 0;


        // Funcion que añade una Key y un value. La Key no se puede repetir.
        public void Add(K key, V value)
        {
            if(ContainsKey(key))
                return;

            int count = _item.Length;
            Item[] setResult = new Item[count + 1];
            Item element = new Item(default, default);
            setResult[count] = element;

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

            setResult[count].Key = key;
            setResult[count].Value = value;

            _item = setResult;
        }

        //Funcion que elimina el Item que ocupa la posicion del indice
          indicado en parametros.
        public void RemoveAt(int index)
        {
            if (index < 0 || index > _item.Length)
                return;

            if (index == -1)
                return;

            int count = _item.Length;
            Item[] arrayResult = new Item[count - 1];

            for (int i = 0; i < index; i++)
            {
                arrayResult[i] = _item[i];
            }

            for (int i = index; i < count - 2; i++)
            {
                arrayResult[i] = _item[i + 1];
            }

            _item = arrayResult;
        }

        // Funcion que devuelve el indice que ocupa el elemento de
          value V.
        public int IndexOf(V value)
        {
            if (value == null)
                return 0;

            for (int i = 0; i < _item.Length; i++)
            {
                if (_item[i].Value.Equals(value))
                    return i;
            }
            return -1;
        }

        // Funcion que devuelve el elemento que contiene la key
          indicada.
        public V GetElementAt(K key)
        {
            if (key == null)
                return default(V);

            for (int i = 0; i < _item.Length; i++)
            {
                if (_item[i].Key.Equals(key))
                    return _item[i].Value;
            }
            return default(V);
        }

        // Funcion que evalua si el diccionario contiene una Key determinada.
        public bool ContainsKey(K key)
        {
            // return IndexOf >= 0;
            if (key == null)
                return false;

            for(int i = 0; i < _item.Length;i++)
            {
                if (_item[i].Key.Equals(key))
                    return true;
            }
            return false;
        }

        // Funcion que devuelve si dos objetos son iguales.
        public override bool Equals(object? obj)
        {
            return (this == obj);
        }

        // Funcion delegada Filter que devuelve un diccionario.
        public DictionaryCollection<K, V> Filter(DelegateFilterKeyValue del)
        {
            DictionaryCollection<K, V> dictionaryResult = new DictionaryCollection<K, V>();

            for (int i = 0; i < _item.Length; i++)
            {
                bool InsertIntoCollection = del(_item[i].Key, _item[i].Value);
                if (InsertIntoCollection)
                {
                    dictionaryResult.Add(_item[i].Key, _item[i].Value);
                }
            }

            return dictionaryResult;
        }

        public DictionaryCollection<K, V> Filter(DelegateFilterKey del)
        {
            DictionaryCollection<K, V> dictionaryResult = new DictionaryCollection<K, V>();

            for (int i = 0; i < _item.Length; i++)
            {
                bool InsertIntoCollection = del(_item[i].Key);
                if (InsertIntoCollection)
                {
                    dictionaryResult.Add(_item[i].Key, _item[i].Value);
                }
            }

            return dictionaryResult;
        }
        public DictionaryCollection<K, V> Filter(DelegateFilterWithoutParameters del)
        {
            DictionaryCollection<K, V> dictionaryResult = new DictionaryCollection<K, V>();

            for (int i = 0; i < _item.Length; i++)
            {
                dictionaryResult.Add(_item[i].Key, _item[i].Value);
            }

            return dictionaryResult;
        }

        // Funcion que elimina todo el contenido de un diccionario.
        public void Clear()
        {
            _item = Array.Empty<Item>();
        }

        // Funcion que devuelve el codigo Hash de un elemento.
        public override int GetHashCode()
        {
            return 133 * 533 * 224 * _item.GetHashCode();
        }

        public override string ToString()
        {
            string result = "";
            foreach (Item i in _item)
            {
                result += $"La key {i.Key}, contiene el value {i.Value}\n";
            }
            return result;
        }
    }
}

```