```
package maps

import maps.lineardatastructures.ResizableArray

// This is part of the extension.
class ArrayBasedMap<K, V> : CustomMutableMap<K, V> {
    val contents = ResizableArray<Entry<K, V>>()

    var count: Int = 0
        private set(value) {
            if (value == contents.size) contents.resize(contents.size * 2)
            field = value
        }

    override val entries: Iterable<Entry<K, V>>
        get() = contents

    override fun get(key: K): V? = contents.getFirst { key == it.key }?.value

    override fun remove(key: K): V? {
        val prev = contents.removeFirst { key == it.key }?.value
        if (prev != null) --count
        return prev
    }

    override fun put(key: K, value: V): V? {
        var added = false
        var prev: V? = null
        for (i in 0 until contents.size) {
            val item = contents[i]
            if (item != null && item.key == key) {
                prev = item.value
                contents[i] = null
            }
            if (!added && contents[i] == null) {
                added = true
                contents[i] = Entry(key, value)
            }
        }
        if (prev == null) {
            count++
        }
        return prev
    }
}

private fun <E : Any> ResizableArray<E>.addToFirstFree(item: E) {
    var added = false
    for (i in 0 until size) {
        if (!added && get(i) == null) {
            added = true
            set(i, item)
        }
    }
}

private fun <E : Any> ResizableArray<E>.removeFirst(predicate: (E) -> Boolean): E? =
    getIndexOfFirst(predicate)?.let(::remove)

private fun <E : Any> ResizableArray<E>.getFirst(predicate: (E) -> Boolean): E? =
    getIndexOfFirst(predicate)?.let(::get)

private fun <E : Any> ResizableArray<E>.getIndexOfFirst(predicate: (E) -> Boolean):
Int? {
    for (i in 0 until size) {
        if (get(i)?.let(predicate) == true) {
            return i
        }
    }
    }
    return null
}
```

```
 1: package maps
 2:
 3: data class Entry<K, V>(val key: K, val value: V)
 4:
 5: interface CustomMutableMap<K, V> {
 6:     // Provides read access to all entries of the map
 7:     val entries: Iterable<Entry<K, V>>
 8:
 9:     // Provides read access to all keys of the map
10:     val keys: Iterable<K>
11:         get() = entries.map(Entry<K, V>::key).toSet()
12:
13:     // Provides read access to all values of the map
14:     val values: Iterable<V>
15:         get() = entries.map(Entry<K, V>::value)
16:
17:     // Returns the value at 'key', or null if there is no such value.
18:     // This operator allows array-like indexing.
19:     operator fun get(key: K): V?
20:
21:     // Operator version of 'put' to allow array-like indexing.
22:     operator fun set(key: K, value: V): V? = put(key, value)
23:
24:     // Associates 'value' with 'key'. Returns the previous value associated with
25:     // 'key', or null if there is no such previous value.
26:     fun put(key: K, value: V): V?
27:
28:     // Associates the value of 'entry' with the key of 'entry'. Returns the previous
29:     // value associated with this key, or null if there is no such previous value.
30:     fun put(entry: Entry<K, V>): V? = put(entry.key, entry.value)
31:
32:     // Removes entry with key 'key' from the map if such an entry exists, returning
33:     // the associated value if so. Otherwise, returns null.
34:     fun remove(key: K): V?
35:
36:     // Returns true if and only if there is some value associated with 'key' in the
map
37:     fun contains(key: K): Boolean = get(key) != null
38: }
```



```
 1: package maps
 2:
 3: private const val POW2_SIZE = true
 4: private const val LOAD_FACTOR = 0.75
 5: private const val DEFAULT_SIZE = 32
 6:
 7: typealias BucketFactory<K, V> = () -> CustomMutableMap<K, V>
 8:
 9: abstract class GenericHashMap<K, V>(private val bucketFactory: BucketFactory<K, V>)
: CustomMutableMap<K, V> {
10:
11:     private var buckets: Array<CustomMutableMap<K, V>> = Array(DEFAULT_SIZE) {
bucketFactory() }
12:
13:     private val numBuckets
14:         get() = buckets.size
15:
16:     private var numElements: Int = 0
17:         set(value) {
18:             field = value
19:             // Using Java's approach to resizing Hashmaps - i.e: never shrinks,
doubles when
20:             // reaches load factor. Could definitely set up cases where shrinking
is required - i.e:
21:             // verify memory usage goes back down after removing all
22:             if (numElements > numBuckets * LOAD_FACTOR) resize(numBuckets * 2)
23:         }
24:
25:     private fun resize(size: Int) {
26:         val oldEntries = entries
27:         buckets = Array(size) { bucketFactory() }
28:         oldEntries.forEach { put(it) }
29:     }
30:
31:     private fun K.bucketIndex(): Int =
32:         if (POW2_SIZE) hashCode() and (numBuckets - 1) else
Math.floorMod(hashCode(), numBuckets)
33:
34:     private fun K.bucket(): CustomMutableMap<K, V> = buckets[bucketIndex()]
35:
36:     override val entries: Iterable<Entry<K, V>>
37:         get() = buckets.asIterable().flatMap { it.entries }
38:
39:     override fun put(key: K, value: V): V? {
40:         val prev = key.bucket().put(key, value)
41:         if (prev == null) numElements++
42:         return prev
43:     }
44:
45:     override fun get(key: K): V? = key.bucket().get(key)
46:
47:     override fun remove(key: K): V? {
48:         // this version using extension functions allows a chain of method calls as
below...
49:         val result = key.bucket().remove(key)
50:         numElements--
51:         return result
52:     }
53: }
```

```
1: package maps
2:
3: class HashMapBackedByArrays<K, V> : GenericHashMap<K, V>({ ArrayBasedMap() })
```



```
1: package maps
2:
3: class HashMapBackedByLists<K, V> : GenericHashMap<K, V>({ ListBasedMap() })
```

```
    1: package maps.lineardatastructures
    2:
    3: interface Node<T> {
    4:     var next: ValueNode<T>?
    5: }
    6:
    7: class ValueNode<T>(val item: T, override var next: ValueNode<T>?) : Node<T>
    8:
    9: class RootNode<T>(override var next: ValueNode<T>?) : Node<T>
   10:
   11: class CustomLinkedList<T> : MutableIterable<T> {
   12:     private val root: RootNode<T> = RootNode(null)
   13:
   14:     private var head: ValueNode<T>?
   15:         get() = root.next
   16:         set(value) {
   17:             root.next = value
   18:         }
   19:
   20:     val isEmpty: Boolean
   21:         get() = head == null
   22:
   23:     fun add(item: T) {
   24:         head = ValueNode(item, head)
   25:     }
   26:
   27:     fun peek(): T? = head?.item
   28:
   29:     fun remove(): T? {
   30:         val toRemove = head ?: return null
   31:         head = toRemove.next
   32:         return toRemove.item
   33:     }
   34:
   35:     // Uses the remove() function in the MutableIterator (below). This could
   36:     // be defined as an extension function on MutableIterable.
   37:     fun remove(pred: (T) -> Boolean): T? {
   38:         val iterator = iterator()
   39:         while (iterator.hasNext()) {
   40:             val item = iterator.next()
   41:             if (pred(item)) {
   42:                 iterator.remove()
   43:                 return item
   44:             }
   45:         }
   46:         return null
   47:     }
   48:
   49:     override fun iterator(): MutableIterator<T> = object : MutableIterator<T> {
   50:         var prev: Node<T>? = null
   51:         var curr: Node<T> = root
   52:         var next: ValueNode<T>? = root.next
   53:         var canRemove: Boolean = false
   54:
   55:         override fun hasNext(): Boolean = next != null
   56:
   57:         override fun next(): T {
   58:             canRemove = true
   59:             val newCurr = next ?: throw NoSuchElementException("Called next on
empty iterator!")
   60:             prev = curr
   61:             curr = newCurr
   62:             next = newCurr.next
   63:             return newCurr.item
   64:         }
   65:
   66:         override fun remove() {
   67:             if (!canRemove) throw UnsupportedOperationException("Called remove
```



```
before next!")
   68:             canRemove = false
   69:             val newCurr = prev!!
   70:             newCurr.next = next
   71:             curr = newCurr
   72:         }
   73:     }
   74: }
```

```
 1: package maps.lineardatastructures
 2:
 3: // This is part of the extension.
 4: class ResizableArray<E>(initialSize: Int = 1) : Iterable<E> {
 5:     private var elements: Array<Any?> = Array(initialSize) { null }
 6:
 7:     val size: Int
 8:         get() = elements.size
 9:
10:     fun remove(index: Int): E? {
11:         val result = elements[index]
12:         for (i in index..<elements.size - 1) {
13:             elements[i] = elements[i + 1]
14:         }
15:         return result as E?
16:     }
17:
18:     operator fun get(index: Int): E? = elements[index] as E?
19:
20:     operator fun set(index: Int, item: E?): E? {
21:         val prev: E? = get(index)
22:         elements[index] = item
23:         return prev
24:     }
25:
26:     fun resize(newSize: Int) {
27:         elements = elements.copyOf(newSize)
28:     }
29:
30:     override fun iterator(): Iterator<E> = object : Iterator<E> {
31:         private var index = 0
32:
33:         override fun hasNext(): Boolean = get(index) != null
34:
35:         override fun next(): E = get(index++) ?: throw
NoSuchElementException("Called next on empty iterator!")
36:     }
37: }
```



```
 1: package maps
 2:
 3: import maps.lineardatastructures.CustomLinkedList
 4:
 5: class ListBasedMap<K, V> : CustomMutableMap<K, V> {
 6:
 7:     private val contents: CustomLinkedList<Entry<K, V>> = CustomLinkedList()
 8:
 9:     override val entries: Iterable<Entry<K, V>>
10:         get() = contents
11:
12:     // Looks like this overrides the operator function; the operator keyword seems
optional...
13:     override fun get(key: K): V? = contents.find { it.key == key }?.value
14:
15:     override fun put(key: K, value: V): V? {
16:         val prev = get(key)
17:         remove(key)
18:         contents.add(Entry(key, value))
19:         return prev
20:     }
21:
22:     override fun remove(key: K): V? {
23:         val keysMatch = { e: Entry<K, V> -> e.key == key }
24:         return contents.remove(keysMatch)?.value
25:     }
26: }
```

```
1: package maps
2:
3: class ArrayBasedMapTest : CustomMutableMapTest() {
4:     override fun <K, V> emptyMap(): CustomMutableMap<K, V> =
5:         ArrayBasedMap()
6: }
```



```
1: package maps
2:
3: class HashMapBackedByArraysTest : CustomMutableMapTest() {
4:     override fun <K, V> emptyMap(): CustomMutableMap<K, V> = HashMapBackedByArrays()
5: }
```

```
1: package maps
2:
3: class HashMapBackedByListsTest : CustomMutableMapTest() {
4:     override fun <K, V> emptyMap(): CustomMutableMap<K, V> = HashMapBackedByLists()
5: }
```



```
1: package maps
2:
3: class ListBasedMapTest : CustomMutableMapTest() {
4:     override fun <K, V> emptyMap(): CustomMutableMap<K, V> =
5:         ListBasedMap()
6: }
```