

# The Proxy Pattern



**Human Computer Interaction Research  
University of Nevada, Reno**



# Iterator Pattern

## ★ Structural Patterns

- » adapter
- » façade
- » composite
- » proxy

## ★ Creational Patterns

- » factory method
- » abstract factory
- » singleton

## ★ Behavioral Patterns

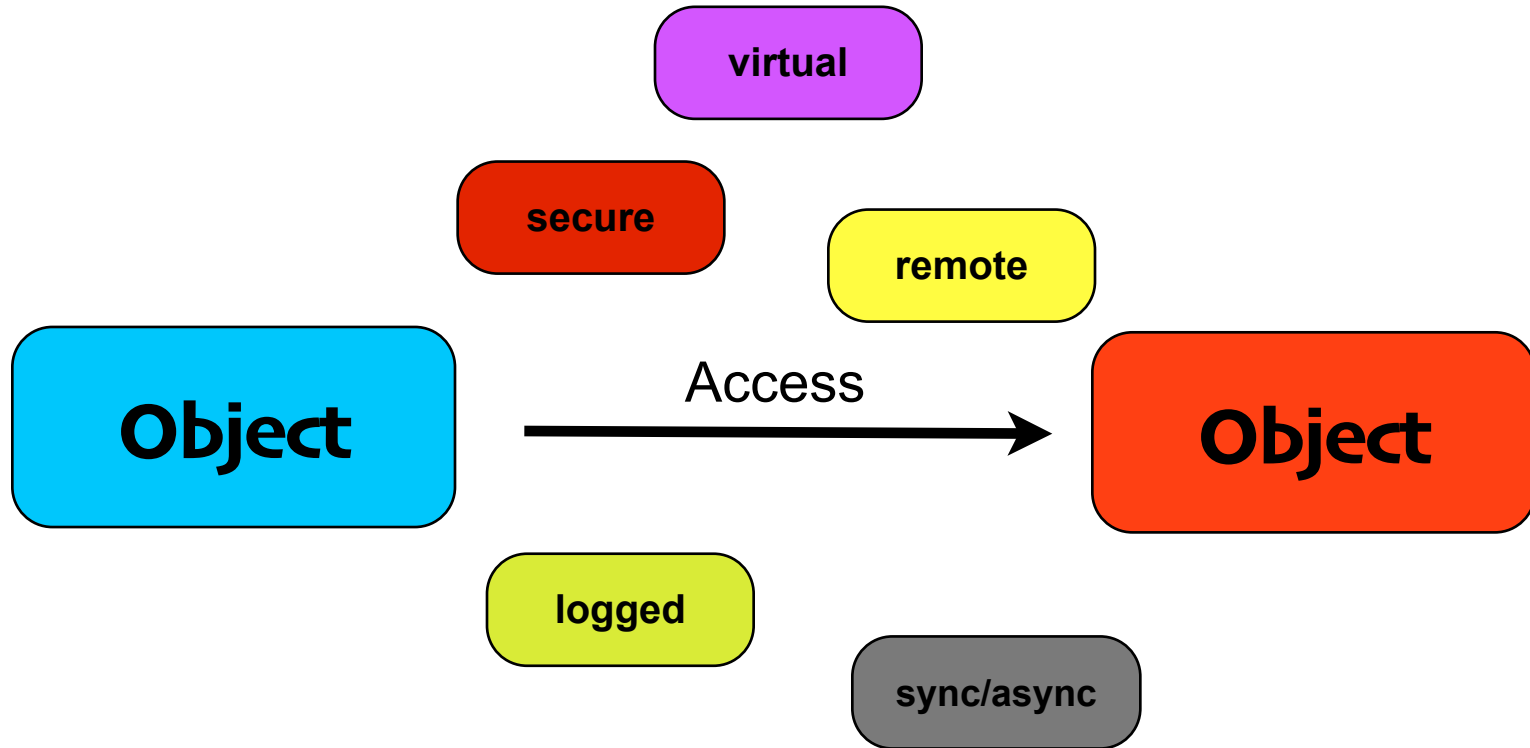
- » strategy
- » observer
- » decorator
- » command
- » template method
- » iterator
- » state

# Problem

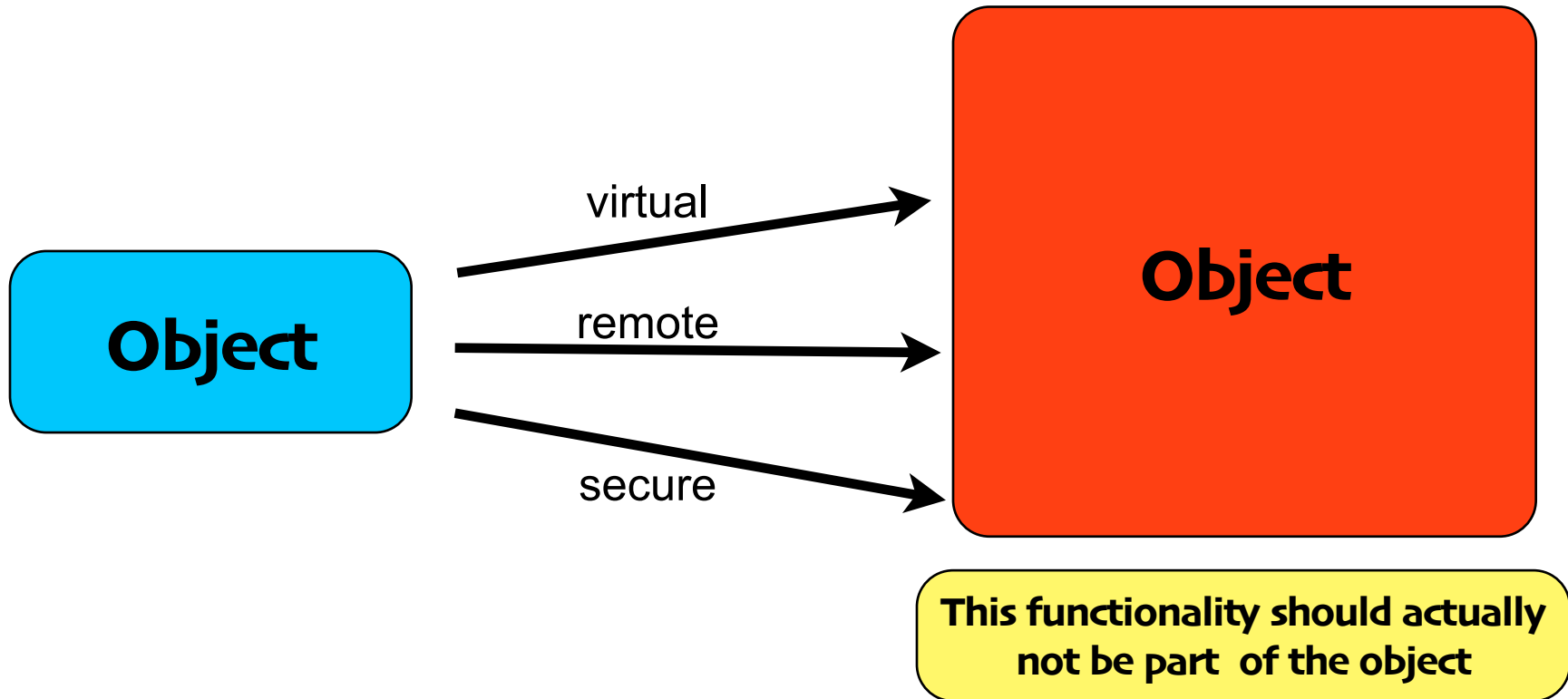
**Objects talk to each other using an interface that has been overburdened with the needs of networking, security, access coherence, or historic versions of the interface.**

**an object demands too much of another object**

# Object Access Control



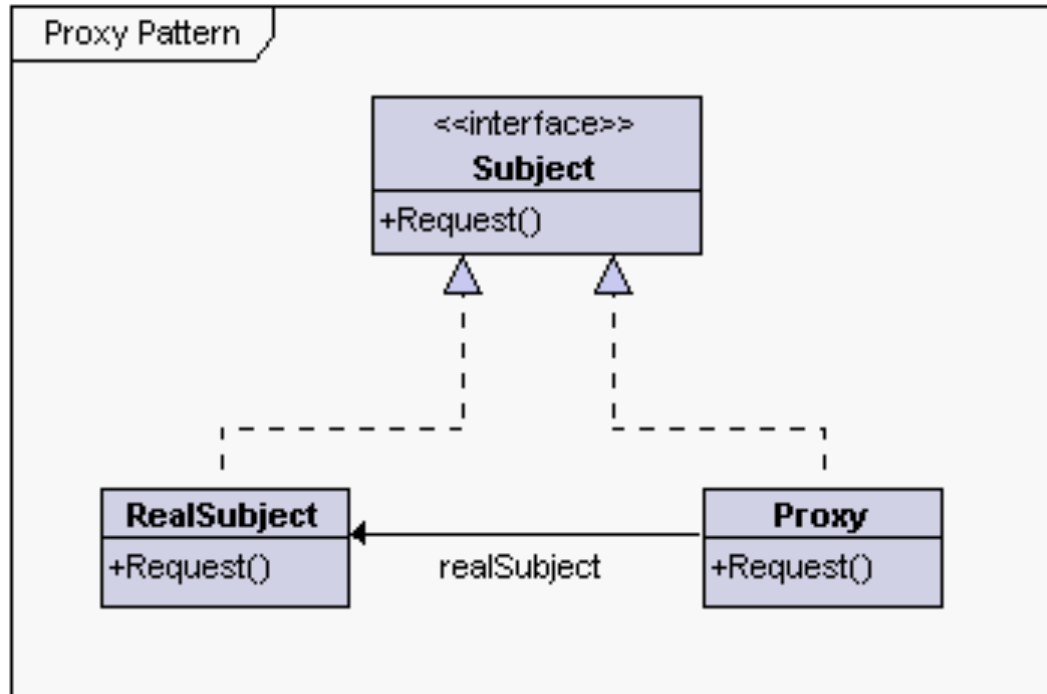
# Object Access Control



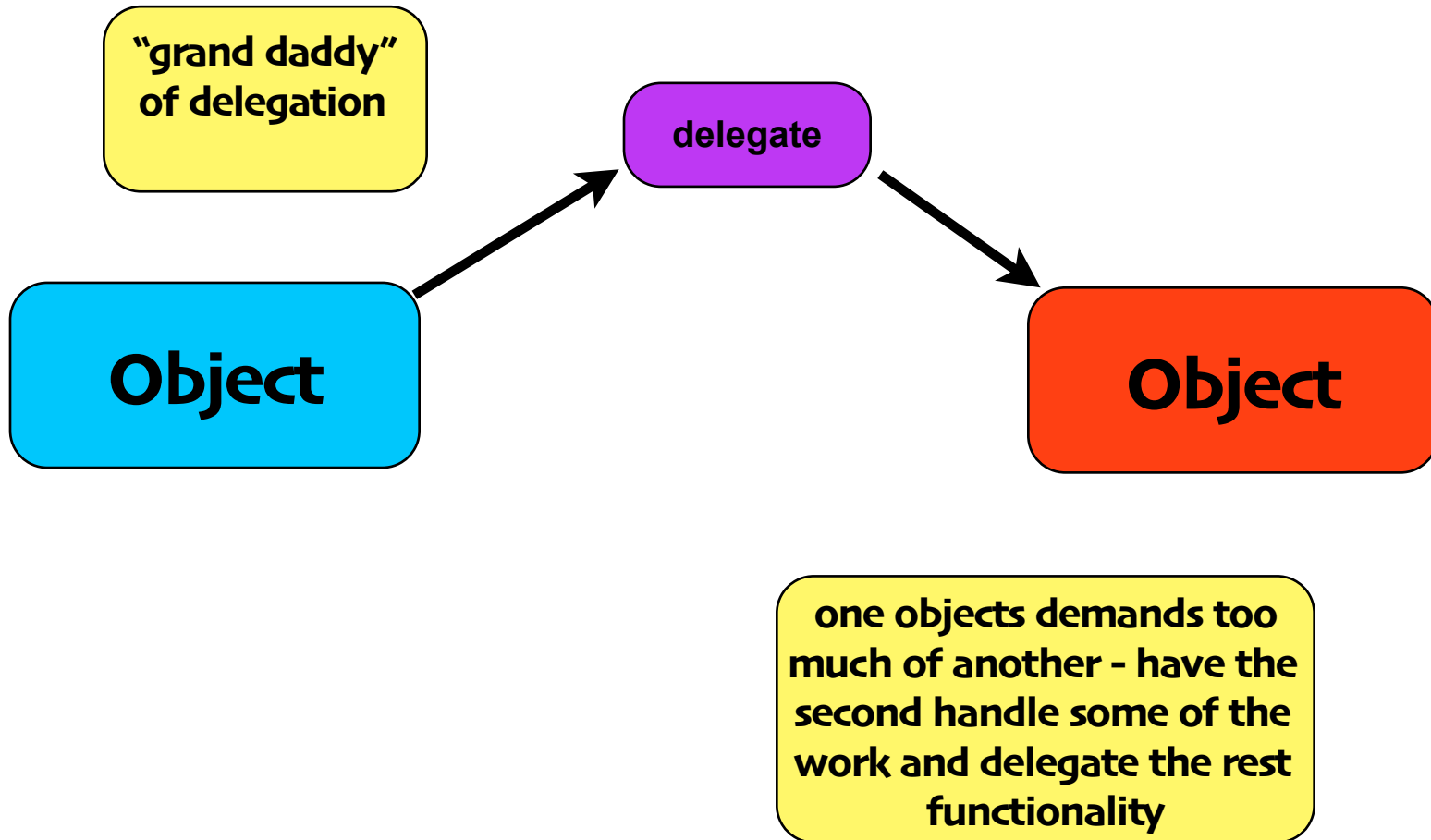
# Solution

**The Proxy Pattern** provides a surrogate or placeholder for another object to control access to it.

# Class Diagram



# Proxy pattern



# Types of proxies

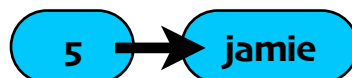
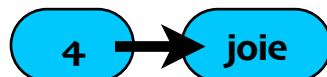
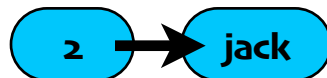
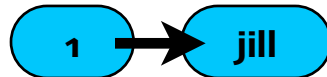
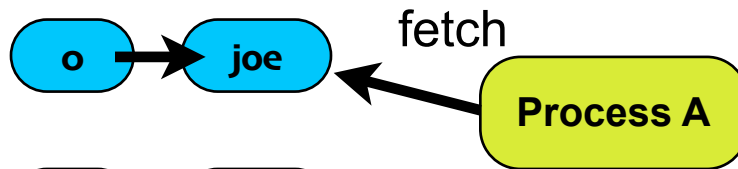
- ★ **Remote Proxy** - Provides a reference to an object located in a different address space
- ★ **Virtual Proxy** - Allows the creation of a memory intensive object on demand
- ★ **Protection Proxy** - provides different clients with different levels of access
- ★ **Cache Proxy** - Temporary stores results of expensive target operations

# Types of proxies

- ★ **Fire Wall Proxy** - protects targets from bad clients
- ★ **Synchronization proxy** - provides multiple access to a target object
- ★ **Smart reference Proxy** - counting the number of references to an object

# Example: Copy-on Write Proxy

## Hash-table



large

A wants to fetch all objects without any other client adding or removing elements

# Solution 1: locking

```
public void doFetches(Hashtable ht)
{ synchronized(ht) {
  // Do fetches using ht reference.
}
}
```

**Collection cannot be accessed by other fetches until lock is released**

# Solution 2: cloning

```
public void doFetches(Hashtable ht) {  
    Hashtable newht = (Hashtable) ht.clone();  
    // Do fetches using newht reference.  
}
```

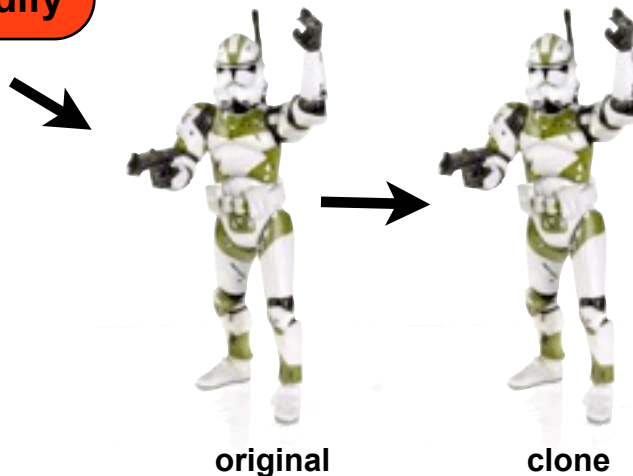
always clones

cloning is expensive

lock while clone

modify

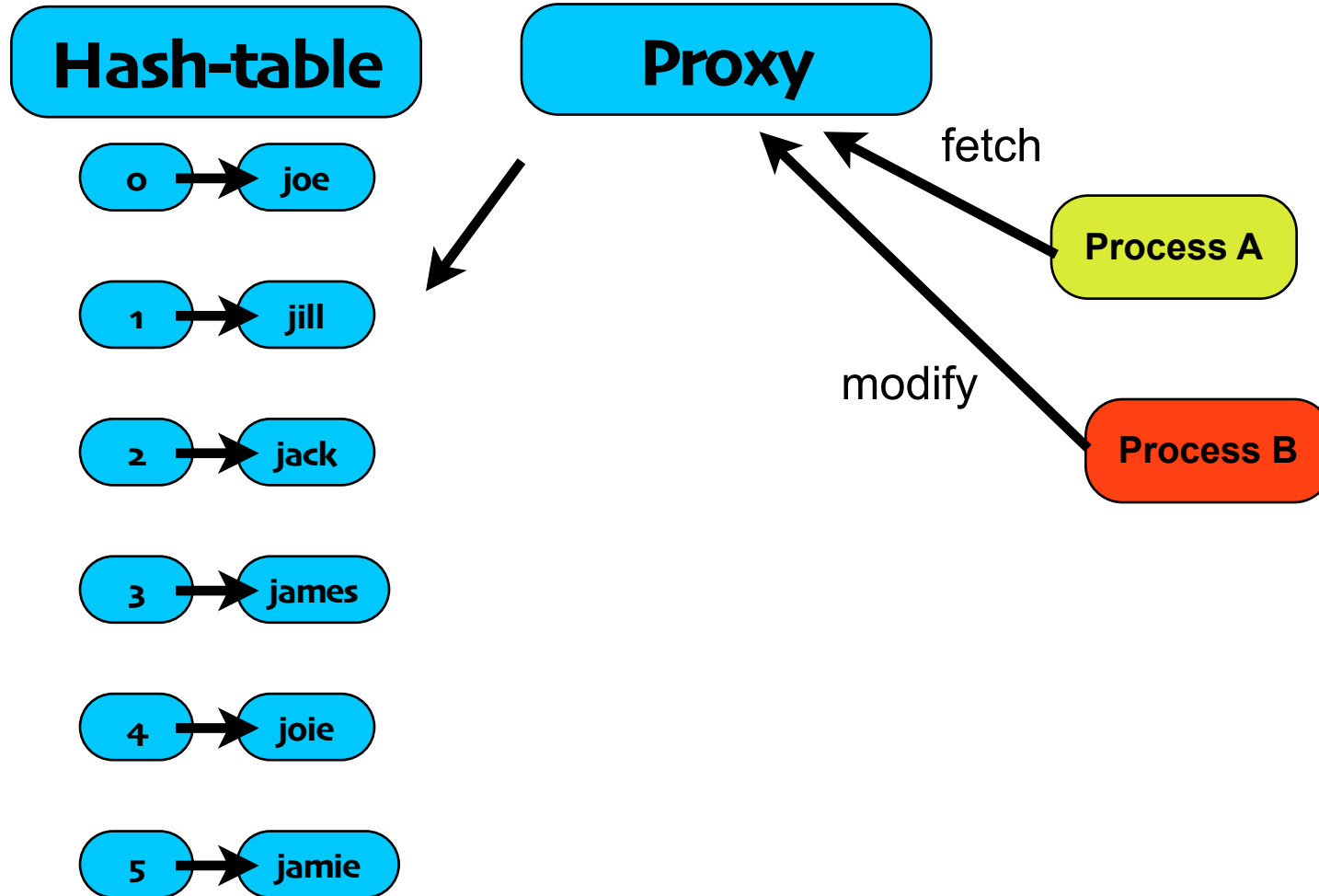
if no other client accesses  
the collection the clone was  
a wasted effort



# Copy on Write Definition

- ★ It would be nice if we could actually clone the collection only when we need to, that is when some other client has modified the collection.
- ★ For example, it would be great if the client that wants to do a series of fetches could invoke the clone() method, but no actual copy of the collection would be made until some other client modifies the collection.
- ★ This is a copy-on-write cloning operation

# Access through proxy



# How it works

★ When the proxy's `clone()` method is invoked, it returns a copy of the proxy and both proxies refer to the same hash table. When one of the proxies modifies the hash table, the hash table itself is cloned. The `referenceCountedHashTable` class is used to let the proxies know they are working with a shared hash table. This class keeps track of the number of proxies using the shared hash table

# Implement using proxy

★ **example from:** Patterns in Java, Volume 1


```
// The proxy.
```

```
public class LargeHashtable extends Hashtable {
```

```
// The ReferenceCountedHashTable that this is a proxy for.  
private ReferenceCountedHashTable theHashTable;
```

```
// Constructor public LargeHashtable() {  
    theHashTable = new ReferenceCountedHashTable();  
}
```


```
// Return the number of key-value pairs in this hashtable.  
public int size() {  
    return theHashTable.size();  
}
```



```
// Return the value associated with the specified key.
public synchronized Object get(Object key) {
    return theHashTable.get(key);
}


// Add the given key-value pair to this Hashtable.
public synchronized Object put(Object key, Object value) {
    copyOnWrite();
    return theHashTable.put(key, value);
}

// Return a copy of this proxy that accesses the same Hashtable.
public synchronized Object clone() {
    Object copy = super.clone();
    theHashTable.addProxy();
    return copy;
}
```




`// This method is called before modifying the  
underlying // Hashtable. If it is being shared then this  
method clones it.`

```
private void copyOnWrite() {  
    if (theHashTable.getProxyCount() > 1) {  
        synchronized (theHashTable) {  
            theHashTable.removeProxy();  
            try {  
                theHashTable = (ReferenceCountedHashTable)  
                    theHashTable.clone();  
            } catch (Throwable e)  
            { theHashTable.addProxy();  
            }  
        }  
    }  
}
```



```
// Private class to keep track of proxies sharing the hash table.
private class ReferenceCountedHashTable extends Hashtable
{ private int proxyCount = 1;
// Constructor public ReferenceCountedHashTable() {
    super();
}

// Return a copy of this object with proxyCount set back to 1.
public synchronized Object clone() {
    ReferenceCountedHashTable copy;
    copy = (ReferenceCountedHashTable) super.clone();
    copy.proxyCount = 1;
    return copy;
}
```



```
/ Return the number of proxies using this object.
synchronized int getProxyCount() {
    return proxyCount;
}
// Increment the number of proxies using this object by one.
synchronized void addProxy() {
    proxyCount++;
}

// Decrement the number of proxies using this object by one.
synchronized void removeProxy() {
    proxyCount--;
}
}
```

# Exercise

