COPY-ON-WRITE EXAMPLE

Using Proxy in Java implementation

Example Scenario

- Suppose we have a large collection object, such as a hash table, which multiple clients want to access concurrently.
- One of the clients wants to perform a series of consecutive fetch operations while not letting any other client add or remove elements

- Solution 1: Use the collection's lock object. The client method obtains the lock, performs its fetches and then releases the lock.
  - But this method may require holding the collection object's lock for a long period of time, thus preventing other threads from accessing the collection

Example : soln2

- Suppose we have a large collection object, such as a hash table, which multiple clients want to access concurrently.
- One of the clients wants to perform a series of consecutive fetch operations while not letting any other client add or remove elements

- Solution 2: The client clones the collection before performing its fetch operations. It is assumed that the collection provides a clone method that performs a sufficiently deep copy.
  - For example, java.util.Hashtable provides a clone method that makes a copy of the hash table itself, but not the key and value objects

Soln2 cont'd

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

- The collection lock is held while the clone is being created.
- Once the clone is created, the fetch operations are done on the cloned copy, without holding the original collection lock.
- But if no other client modifies the collection while the fetch operations are being done, the expensive clone operation was a wasted effort!

Solution 3: copy on write

- Solution 3: 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.
- We can implement this solution using **proxies**

Copy-on-write with proxies

- Have a proxy for the shared hashtable.
- When the proxy's clone() method is invoked, it returns a copy of the proxy and both proxies refer to the same hash table.
  - Need to count the number of proxies as well.
- When one of the proxies modifies the hash table, the hash table itself is cloned.
  - put() modifies the table.

- Adapted from the book Patterns in Java, Volume 1, Mark Grand

```
// The proxy.
public class LargeHashtable extends Hashtable {
  private ReferenceCountedHashTable theHashTable; //RealSubject

  public LargeHashtable() {
      theHashTable = new ReferenceCountedHashTable();
   }

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

```
/* hash table non-modifying operations */
  public int size() { return theHashTable.size();
  public synchronized Object get(Object key) {
        return theHashTable.get(key); }

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

```
// 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) { //many proxies
     synchronized (theHashTable) {
         theHashTable.removeProxy();
         try {
             theHashTable = (ReferenceCountedHashTable) theHashTable.clone();
         } catch (Throwable e) {
         theHashTable.addProxy();
     }
 } }
```