```
package containers;

public interface IContainer {

    /**
     * If there is an object associated with key
     * then this object is returned else null is returned.
     */
    public Object find(Object key);

    /**
     * Afterwards, find(key) returns null, and if there is
     * an object associated with key then this object is
     * returned else null is returned.
     */
    public Object remove(Object key);

    /**
     * (key, value) is stored in this container with no
     * duplication and such that find(key) returns value.
     */
    public void insert(Object key, Object value);
}
```

```
package containers;

/**
 * The (key,value) pair.
 * @author Alan L. Cox
 * @since 02/15/01
 */
class KeyValuePair {
    private Object _key;
    private Object _value;

    KeyValuePair(Object key, Object value)
    {
        _key = key;
        _value = value;
    }

    /**
     * @return the key
     */
    Object getKey()
    {
        return _key;
    }

    /**
     * @return the value
     */
    Object getValue()
    {
        return _value;
    }
}
```

```
package containers;

import lrs.LRStruct;

/**
 * A container implemented using the LRStruct.
 * @author Alan L. Cox
 * @since 02/15/01
 */
public class LRSContainer implements IContainer {

    private LRStruct _lrs = new LRStruct();

    /**
     * If there is an object associated with key
     * then this object is returned else null is returned.
     *
     * @param key the Object key to find
     * @return the Object associated with the key
     */
    public Object find(Object key)
    {
        return _lrs.execute(LRSFindVisitor.singleton, key);
    }

    /**
     * Afterwards, find(key) returns null, and if there is
     * an object associated with key then this object is
     * returned else null is returned.
     *
     * @param key the Object key to remove
     * @return the Object associated with the key
     */
    public Object remove(Object key)
    {
        return _lrs.execute(LRSRemoveVisitor.singleton, key);
    }

    /**
     * (key, value) is stored in this container with no
     * duplication and such that find(key) returns value.
     *
     * @param key the Object key from the (key,value) to insert
     * @param value the Object value from the (key,value) to insert
     */
    public void insert(Object key, Object value)
    {
        _lrs.execute(LRSInsertVisitor.singleton, new KeyValuePair(key, value));
    }
}
```

```
package containers;

import lrs.*;

class LRSFindVisitor implements IAlgo {

    final static LRSFindVisitor singleton = new LRSFindVisitor();

    private LRSFindVisitor()
    {
    }

    public Object emptyCase(LRStruct host, Object input)
    {
        return null;
    }

    public Object nonEmptyCase(LRStruct host, Object input)
    {
        KeyValuePair pair = (KeyValuePair) host.getFirst();

        if (input.equals(pair.getKey()))
            return pair.getValue();
        else
            return host.getRest().execute(this, input);
    }
}
```

```
package containers;

import lrs.*;

class LRSInsertVisitor implements IAlgo {

    final static LRSInsertVisitor singleton = new LRSInsertVisitor();

    private LRSInsertVisitor()
    {
    }

    public Object emptyCase(LRStruct host, Object input)
    {
        host.insertFront(input);

        return null;
    }

    public Object nonEmptyCase(LRStruct host, Object input)
    {
        KeyValuePair pair = (KeyValuePair) host.getFirst();

        if (((KeyValuePair) input).getKey().equals(pair.getKey()))
            host.setFirst(input);
        else
            host.getRest().execute(this, input);

        return null;
    }
}
```

```
package containers;

import lrs.*;

class LRSRemoveVisitor implements IAlgo {

    final static LRSRemoveVisitor singleton = new LRSRemoveVisitor();

    private LRSRemoveVisitor()
    {
    }

    public Object emptyCase(LRStruct host, Object input)
    {
        return null;
    }

    public Object nonEmptyCase(LRStruct host, Object input)
    {
        KeyValuePair pair = (KeyValuePair) host.getFirst();

        if (input.equals(pair.getKey())) {
            host.removeFront();
            return pair.getValue();
        }
        else
            return host.getRest().execute(this, input);
    }
}
```