```java
package lrs;

/**
 * Represents the abstract list state.
 * @author Dung X. Nguyen  Copyright 2000 - All rights reserved.
 * @author Alan L. Cox
 * @since 02/13/01
 */
abstract class ANode
{
    /**
     * Returns the first data object of the referencing LRStruct.
     *
     * @param owner the LRStruct referencing this ANode.
     * @return the tail LRStruct of owner.
     * @throw java.util.NoSuchElementException if empty.
     */
    abstract Object getFirst(LRStruct owner);

    /**
     * Sets a new first data object for the referencing LRStruct.
     *
     * @param data the new data object for this ANode.
     * @param owner the LRS referencing this ANode.
     * @throw java.util.NoSuchElementException if empty.
     */
    abstract void setFirst(Object data, LRStruct owner);

    /**
     * Returns the tail LRStruct of the referencing LRStruct.
     *
     * @param owner the LRStruct referencing this ANode.
     * @return the tail LRStruct of owner.
     * @throw java.util.NoSuchElementException if empty.
     */
    abstract LRStruct getRest(LRStruct owner);

    /**
     * Sets a new tail for the referencing LRStruct.
     *
     * @param tail the new tail for the owner LRStruct.
     * @param owner the LRS referencing this ANode.
     * @throw java.util.NoSuchElementException if empty.
     */
    abstract void setRest(LRStruct tail, LRStruct owner);

    /**
     * Inserts a data object at the front of the LRStruct owner.
     *
     * @param data the object to be inserted at the front.
     * @param owner the LRS referencing this ANode.
     */
    abstract void insertFront(Object data, LRStruct owner);

    /**
     * Removes and returns the first data object for the referencing LRStruct.
     *
     * @param owner the LRS referencing this ANode.
     * @return the front data of the LRStruct owner.
     * @throw java.util.NoSuchElementException if empty.
     */
    abstract Object removeFront(LRStruct owner);

    /**
     * Executes an algorithm and returns the output.
     *
     * @param visitor the algorithm to be executed.
     * @param input the input needed by the algorithm.
     * @param owner the LRStruct referencing this ANode.
     */
    abstract Object execute(IAlgo visitor, Object input, LRStruct owner);
}
```

```java
package lrs;

/**
 * Represents the empty state. Singleton pattern.
 * @author Dung X. Nguyen  Copyright 2000 - All rights reserved.
 * @author Alan L. Cox
 * @since 02/13/01
 */
class EmptyNode extends ANode
{
    /**
     * Singleton Pattern.
     */
    final static EmptyNode singleton = new EmptyNode();

    private EmptyNode()
    {
    }

    /**
     * @throw java.util.NoSuchElementException.
     */
    Object getFirst(LRStruct owner)
    {
        throw new java.util.NoSuchElementException("Empty list has no first.");
    }

    /**
     * @throw java.util.NoSuchElementException.
     */
    void setFirst(Object data, LRStruct owner)
    {
        throw new java.util.NoSuchElementException("Empty list has no first.");
    }

    /**
     * @throw java.util.NoSuchElementException.
     */
    LRStruct getRest(LRStruct owner)
    {
        throw new java.util.NoSuchElementException("Empty list has no first.");
    }

    /**
     * @throw java.util.NoSuchElementException.
     */
    void setRest(LRStruct tail, LRStruct owner)
    {
        throw new java.util.NoSuchElementException("Empty list has no tail.");
    }

    /**
     * Inserts a data object at the front of the LRStruct owner.
     *
     * @param data the object to be inserted at the front.
     * @param owner the LRS referencing this EmptyNode.
     */
    void insertFront(Object data, LRStruct owner)
    {
        owner.setHead(new NonEmptyNode(data));
    }

    /**
     * @throw java.util.NoSuchElementException.
     */
    Object removeFront(LRStruct owner)
    {
        throw new java.util.NoSuchElementException("Empty list has no front.");
    }

    /**
     * Executes the visitor's empty case and returns the output.
     *
     * @param visitor the algorithm to be executed.
     * @param input the input needed by the algorithm.
     * @param owner the LRStruct referencing this EmptyNode.
     */
    Object execute(IAlgo visitor, Object input, LRStruct owner)
    {
        return visitor.emptyCase(owner, input);
    }
}
```

```java
package lrs;

/**
 * Represents an abstract algorithm on a LRStruct.  Visitor pattern.
 * @author Dung X. Nguyen  Copyright 2000 - All rights reserved.
 * @since 02/15/00
 * @dependency lrs.LRStruct uses
 */
public interface IAlgo
{
    public abstract Object emptyCase(LRStruct host, Object input);

    public abstract Object nonEmptyCase(LRStruct host, Object input);
}
```

```java
package lrs;

/**
 * Mutable linear recursive structure.
 * @author Dung X. Nguyen  Copyright 2000 - All rights reserved.
 * @author Alan L. Cox
 * @since 02/13/01
 * @dependency lrs.IAlgo
 */
public class LRStruct
{
    private ANode _head;

    public LRStruct()
    {
        _head = EmptyNode.singleton;
    }

    /**
     * @return the object at the front of the list
     */
    public final Object getFirst()
    {
        return _head.getFirst(this);
    }

    /**
     * @param data the object that replaces the one at the front of the list
     */
    public final void setFirst(Object data)
    {
        _head.setFirst(data, this);
    }

    /**
     * @return the rest of the list
     */
    public final LRStruct getRest()
    {
        return _head.getRest(this);
    }

    /**
     * @param tail the list that becomes the rest of the list
     */
    public final void setRest(LRStruct tail)
    {
        _head.setRest(tail, this);
    }

    /**
     * @param data an object to insert at the front of the list
     */
    public final void insertFront(Object data)
    {
        _head.insertFront(data, this);
    }

    /**
     * @return the object removed from the front of the list
     */
    public final Object removeFront()
    {
        return _head.removeFront(this);
    }

    /**
     * The hook required by the visitor pattern.
     *
     * @param visitor The visitor object.
     * @param input An input that is passed to the visitor.
     * @return The result of the visitor.
     */
    public final Object execute(IAlgo visitor, Object input)
    {
        return _head.execute(visitor, input, this);
    }

// Package access only:

    /**
     * Initializes this LRStruct object with the given head node.
     *
     * @param head null != head.
     */
    LRStruct(ANode head)
    {
        _head = head;
    }

    /**
     * This is the ``set state'' method required
     * by the state pattern.
     *
     * @param head The ANode that becomes the head of the list.
     */
    final void setHead(ANode head)
    {
        _head = head;
    }

    /**
     * This is the ``get state'' method required
     * by the state pattern.
     *
     * @return The ANode at the head of the list.
     */
    final ANode getHead()
    {
        return _head;
    }
}
```

```java
package lrs;

/**
 * Represents the non-empty state of a LStruct.
 * @author Dung X. Nguyen  Copyright 2000 - All rights reserved.
 * @author Alan L. Cox
 * @since 02/13/01
 */
class NonEmptyNode extends ANode
{
    private Object _data;
    private LRStruct _tail;

    /**
     * Initializes this NonNullNode to contain dat and an empty tail list.
     *
     * @param data the data object to be stored in this NonNullNode.
     */
    NonEmptyNode(Object data)
    {
        _data = data;
        _tail = new LRStruct();
    }

    /**
     * Initializes this NonNullNode to contain dat and a given tail list.
     *
     * @param data the data object to be stored in this NonNullNode.
     * @param tail the LRStruct tail of this NonNullNode.
     */
    NonEmptyNode(Object data, LRStruct tail)
    {
        _data = data;
        _tail = tail;
    }

    /**
     * Returns the first data object of the referencing LRStruct.
     *
     * @param owner the LRStruct referencing this NonEmptyNode.
     * @return the tail LRStruct of owner.
     */
    Object getFirst(LRStruct owner)
    {
        return _data;
    }

    /**
     * Sets a new first data object for the referencing LRStruct.
     *
     * @param data the new data object for this NonEmptyNode.
     * @param owner the LRS referencing this NonEmptyNode.
     */
    void setFirst(Object first, LRStruct owner)
    {
        _data = first;
    }

    /**
     * Returns the tail LRStruct of the referencing LRStruct.
     *
     * @param owner the LRStruct referencing this NonEmptyNode.
     * @return the tail LRStruct of owner.
     */
    LRStruct getRest(LRStruct owner)
    {
        return _tail;
    }

    /**
     * Sets a new tail for the referencing LRStruct.
     *
     * @param tail the new tail for the owner LRStruct.
     * @param owner the LRS referencing this NonEmptyNode.
     */
    void setRest(LRStruct tail, LRStruct owner)
    {
        _tail = tail;
    }

    /**
     * Inserts a data object at the front of the LRStruct owner.
     *
     * @param data the object to be inserted at the front.
     * @param owner the LRS referencing this NonNullNode.
     */
    void insertFront(Object data, LRStruct owner)
    {
        owner.setHead(new NonEmptyNode(data, new LRStruct(this)));
        /* Details:
           LRStruct coOwner = new LRStruct(this);
           NonNullNode newNode = new NonNullNode(dat, coOwner);
           owner.setHead(newNode);
           "old" style: owner._head = newNode - cannot be done.
         */
    }

    /**
     * Removes and returns the first data object for the referencing LRStruct.
     *
     * @param owner the LRStruct referencing this NonEmptyNode.
     * @return the front data of the LRStruct owner.
     */
    Object removeFront(LRStruct owner)
    {
        owner.setHead(_tail.getHead()); // owner._head = _tail._head
        return _data;
    }

    /**
     * Executes the visitor’s non-empty case and returns the output.
     *
     * @param visitor the algorithm to be executed.
     * @param input the input needed by the algorithm.
     * @param owner the LRStruct referencing this NonEmptyNode.
     */
    Object execute(IAlgo visitor, Object input, LRStruct owner)
    {
        return visitor.nonEmptyCase(owner, input);
    }
}
```