Using Anonymous Inner Classes: `LRSContainer`

- We can replace `LRSFindVisitor`, `LRSRemoveVisitor`, and `LRSInsertVisitor` by anonymous inner classes.

```
package containers;

import lrs.*;

class LRSFindVisitor implements IAlgo {

  // Singleton pattern

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

Using Anonymous Inner Classes: `LRSContainer` (cont.)

```
public class LRSContainer implements IContainer {
  . . .
  public Object find(Object key)
  {
    return _lrs.execute(new IAlgo() {

      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);
      }
    }, key);
  }
  . . .
```

IContainer *Version 2*

```
package containers;

import java.util.Enumeration;

public interface IContainer {

  public Object find(Object key);

  public Object remove(Object key);

  public void insert(Object key, Object value);

  public Enumeration enumeration();
}
```

java.util.Enumeration

```
package java.util;

public interface Enumeration
{
  public boolean hasMoreElements();

  public Object nextElement();
}
```

java.util.Enumeration **(cont.)**

```
. . .
IContainer c;
. . .
Enumeration e = c.enumeration();

while (e.hasMoreElements())
  System.out.println(e.nextElement());
. . .
```

LRSContainer *Version 2*

```
package containers;

import java.util.Enumeration;
import lrs.*;                              // for LRStruct and IAlgo

public class LRSContainer implements IContainer {
  . . .
  public Enumeration enumeration()
  {
    class LRSEnumeration implements Enumeration {
      private LRStruct _next;

      LRSEnumeration(LRStruct lrs)
      {
        _next = lrs;
      }
```

```
public boolean hasMoreElements()
{
  return Boolean.TRUE == _next.execute(new IAlgo() {

    public Object emptyCase(LRStruct host, Object input)
    {
      return Boolean.FALSE;
    }

    public Object nonEmptyCase(LRStruct host, Object input)
    {
      return Boolean.TRUE;
    }
  }, null);
}

public Object nextElement()
{
  return _next.execute(new IAlgo() {
          public Object emptyCase(LRStruct host, Object input)
          {
            return null;
          }

          public Object nonEmptyCase(LRStruct host, Object input)
          {
            Object object = _next.getFirst();

            _next = _next.getRest();

            return object;
          }
        }, null);
      }
    }

    return new LRSEnumeration(_lrs);
  }
  . . .
```