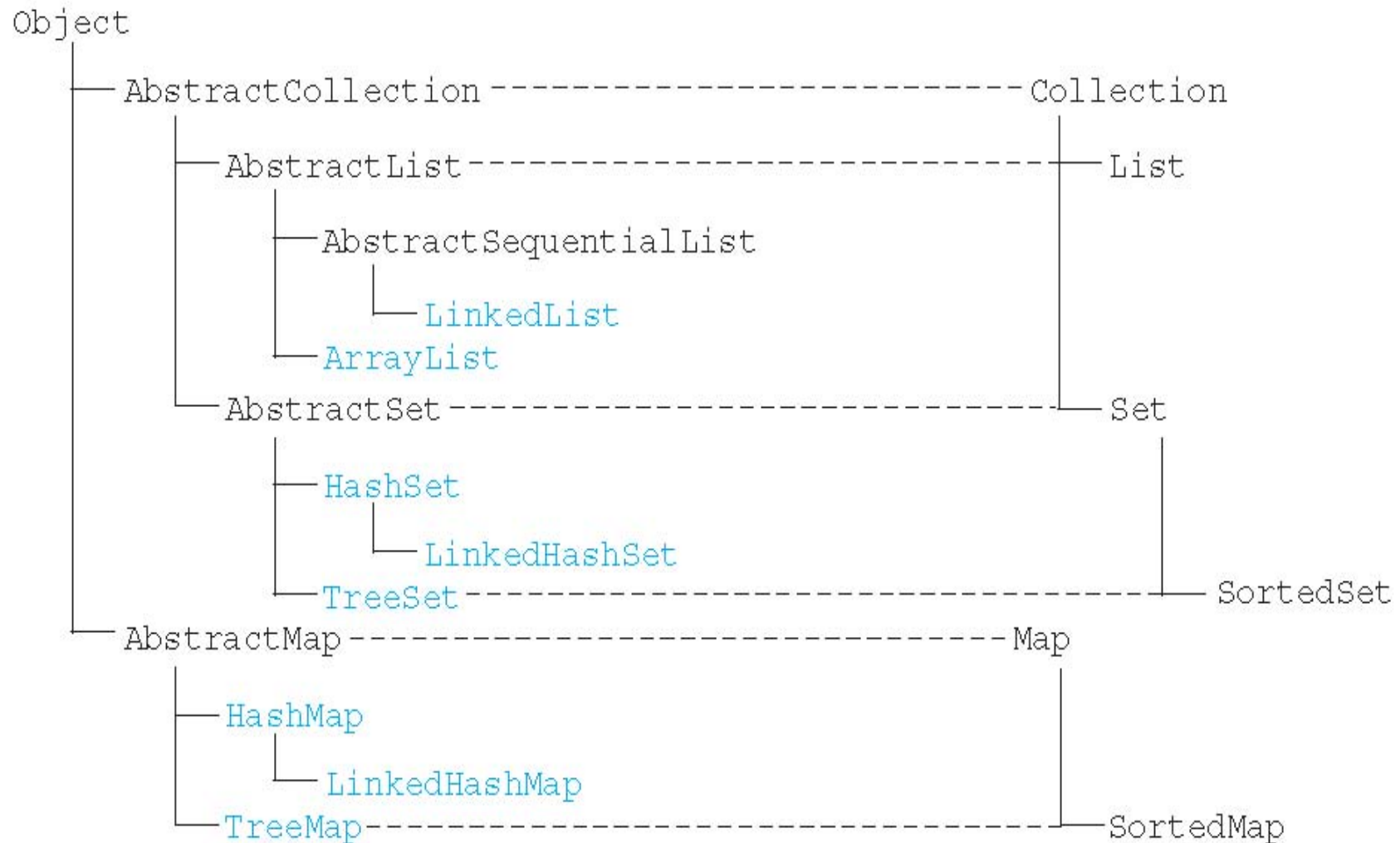


7. 컬렉션 & 8. 리스트

7.1 Java 컬렉션 프레임워크의 계승 계층

- 컬렉션(collection) : 효율적인 접근을 위해 원소들을 관리하는 일반적인 자료구조



8.1 Java 컬렉션 프레임워크의 리스트 클래스

- 리스트 :
 - 선형적인 컬렉션(즉, 원소의 시퀀스) : 선형 구조
 - Java에서 리스트 원소는 배열처럼 0, 1, 2 등 번호를 부여함
 - 리스트 원소를 접근하는 정수 변수를 *인덱스(index)*라 함
 - 배열처럼 리스트는 중복된 참조와 null 참조를 가짐
- 리스트 클래스
 - 2개의 ArrayList와 LinkedList가 제공
 - ArrayList 클래스 : 원소 저장에 배열 사용
 - 탐색 : 상수 시간
 - 삽입과 삭제 : 선형 시간
 - LinkedList 클래스 : 원소 저장에 연결 리스트 사용
 - 탐색 : 선형 시간
 - 삽입과 삭제 : 상수 시간

8.3 java.util.List 인터페이스

```
1 package java.util;
2 public interface List extends Collection {
3     public boolean add(Object object);
4     public void add(int index, Object object);
5     public boolean addAll(Collection collection);
6     public boolean addAll(int index, Collection collection);
7     public void clear();
8     public boolean contains(Object object);
9     public boolean containsAll(Collection collection);
10    public boolean equals(Object object);
11    public Object get(int index);
12    public int hashCode();
13    public int indexOf(Object object);
14    public boolean isEmpty();
15    public Iterator iterator();
16    public int lastIndexOf(Object object);
17    public ListIterator listIterator();
    :
    :
28}
```

ArrayList 사용

```
import java.util.ArrayList;
class IntroArrayList{
    public static void main(String[ ] args){
        ArrayList<Integer> list= new ArrayList<Integer>( );
        list.add(new Integer(11));
        list.add(new Integer(22));
        list.add(new Integer(33));

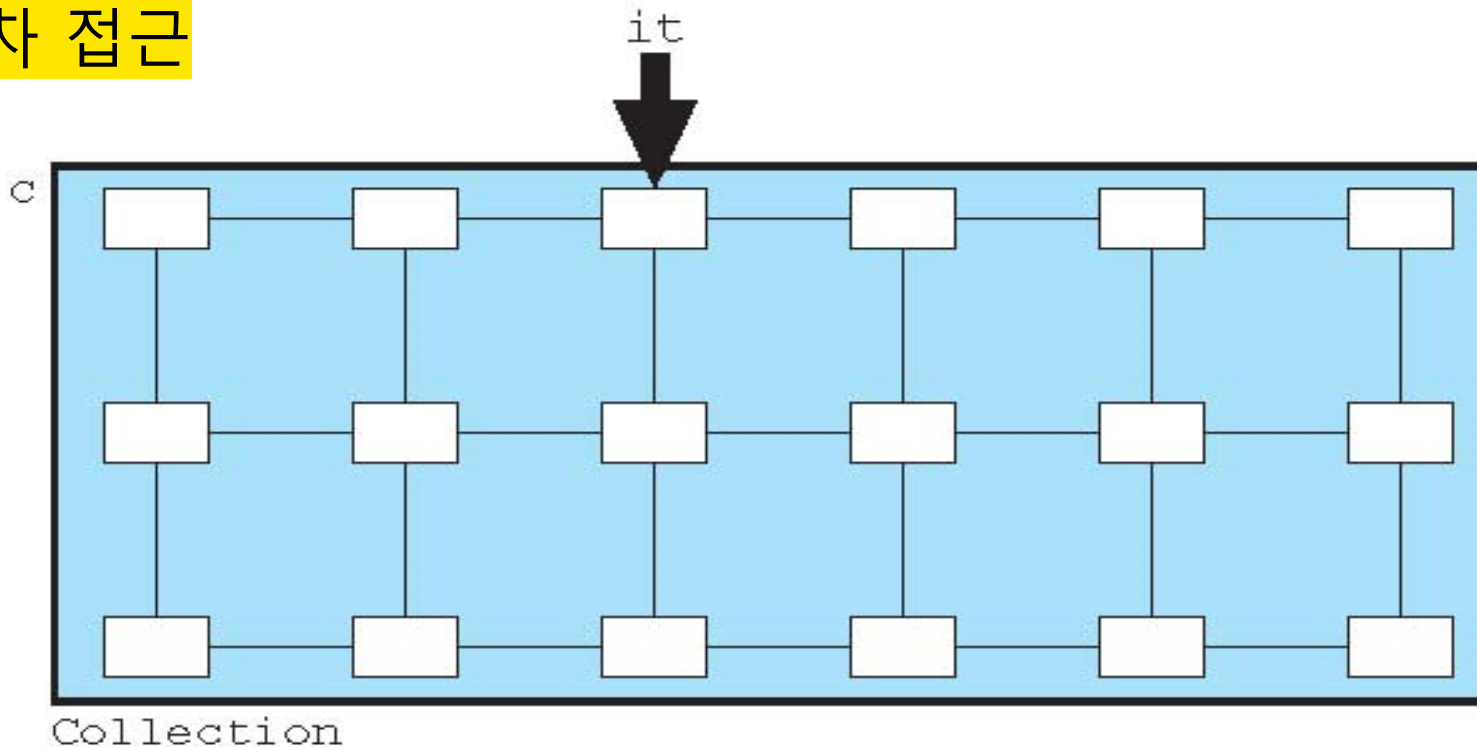
        System.out.println("1차 참조");
        for (int=0; i<list.size( ); i++)
            System.out.println(list.get(i));

        list.remove(0);
        System.out.println("2차 참조");
        for (int=0; i<list.size( ); i++)
            System.out.println(list.get(i));
    }
}
```

7.3 반복자

- 반복자(Iterator) :

- 해당 자료 구조상에서 이동하면서 접근할 개개 원소의 위치를 나타내는 커서 또는 포인터 역할을 하는 객체
- 배열 인덱스와 유사하나, 배열은 직접 접근, 반복자는 순차 접근



Iterator 인터페이스

- LISTING 7.2: An Iterator Interface

```
1 public interface Iterator {  
2     public boolean hasNext();  
3     public Object next();  
4     public void remove();  
5 }
```

next(): 현재 반복자가 가리키고 있는 원소에 대한 참조를 리턴하고 반복자를 다음 원소로 전진시킨다

remove(): next() 에 대한 마지막 호출에 의해 리턴된 원소를 제거한다.

Iterator 객체 사용 예

```
for (Iterator it = c.iterator(); it.hasNext(); ) {  
    Object obj=it.next(); // 현재 원소를 리턴하고 전진  
    // 객체를 이용한 다른 작업 수행  
}
```

* Collection 객체 c를 순회하기 위해 iterator() 메소드에 의해 Iterator 객체를 사용

* Iterator 객체는 처음에 iterator() 메소드에 의해 호출될 때 컬렉션의 시작 위치로 초기화된다

8.2 양방향 리스트 반복자

- LISTING 8.1: The java.util.ListIterator Interface

```
1 package java.util;
2 public interface ListIterator extends Iterator {
3     public void      add(Object object);
4     public boolean    hasNext();
5     public boolean    hasPrevious();
6     public Object     next();
7     public int        nextIndex();
8     public Object     previous();
9     public int        previousIndex();
10    public void        remove();
11    public void        set(Object object);
12}
```

- `next()`: 커서를 전방으로 이동하기 전의 현재원소를 리턴
- `previous()`: 커서를 후방으로 이동한 후의 현재원소를 리턴
- 리스트반복자를 가진 컬렉션은 원소에 0,1,2 등으로 번호를 부여하는 배열과 같다.
- 인덱스 번호는 리스트 반복자의 `nextindex()`와 `previousIndex()`에 의해 리턴된다.
- `nextIndex()` : `next()`의 다음 호출에 의해 리턴될 원소의 인덱스를 리턴
- `previousIndex()` : `previous()`의 다음 호출에 의해 리턴될 원소의 인덱스를 리턴
- `set()`: 변경할 원소를 지정하기 위해 먼저 `next()` 또는 `previous()`를 호출한다. 이 원소를 가변 원소(`mutable element`)라고 한다.

java.util.ListIterator 인터페이스의 테스트

```
1 import java.util.*;
2 public class TestListIterator2 {
3     public static void main(String[] args) {
4         List list = Arrays.asList( new String[]{"AT", "DE", "ES", "FR"});
5         System.out.println("list: " + list);
6         ListIterator it=list.listIterator();
7         System.out.println("it.next(): " + it.next());
8         System.out.println("it.next(): " + it.next());
9         System.out.println("it.previous(): " + it.previous());
10        System.out.println("it.next(): " + it.next());
11        System.out.println("it.next(): " + it.next());
12        System.out.println("it.previous(): " + it.previous());
13        System.out.println("it.previous(): " + it.previous());
14        System.out.println("it.previous(): " + it.previous());
15        System.out.println("it.hasPrevious(): " + it.hasPrevious());
16        System.out.println("it.hasNext(): " + it.hasNext());
17    } }
```

- 출력 결과

```
list: [AT, DE, ES, FR]  
it.next(): AT  
it.next(): DE  
it.previous(): DE  
it.next(): DE  
it.next(): ES  
it.previous(): ES  
it.previous(): DE  
it.previous(): AT  
it.hasPrevious(): false  
it.hasNext(): true
```

```
5 System.out.println("list: "+list);
```

```
list: [AT, DE, ES, FR]
```

```
6 ListIterator it=list.listIterator();
```

```
7 System.out.println("it.next(): "+it.next());
```

```
it.next(): AT
```

```
8 System.out.println("it.next(): "+it.next());
```

```
it.next(): DE
```

```
9 System.out.println("it.previous(): "+it.previous());
```

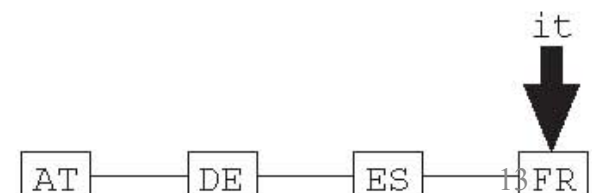
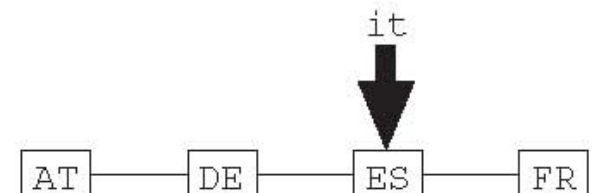
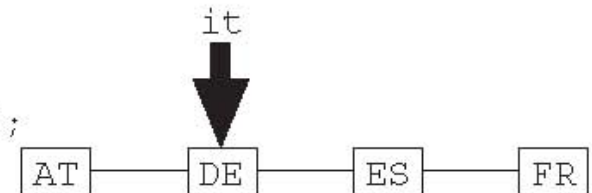
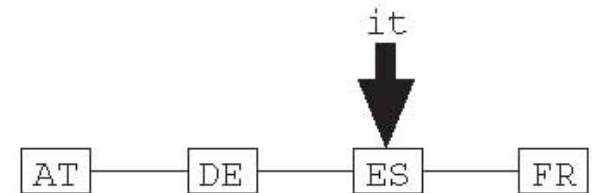
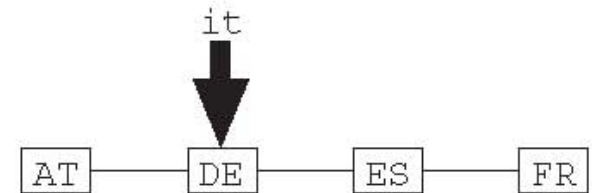
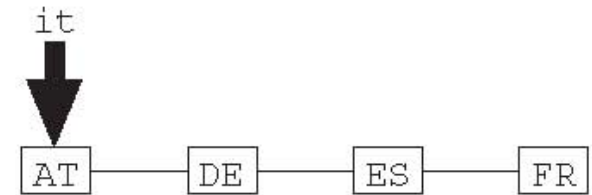
```
it.previous(): DE
```

```
10 System.out.println("it.next(): "+it.next());
```

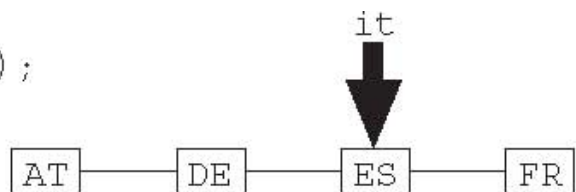
```
it.next(): DE
```

```
11 System.out.println("it.next(): "+it.next());
```

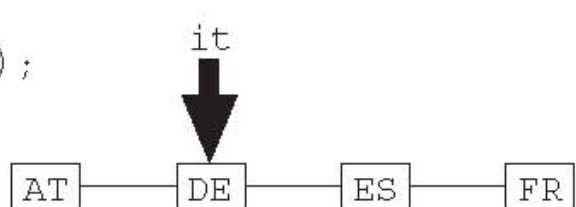
```
it.next(): ES
```



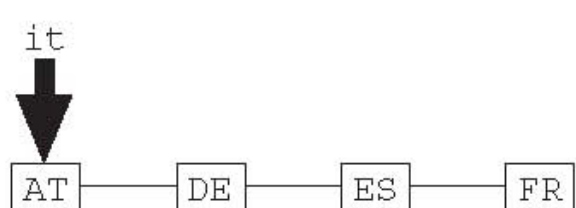
```
12  System.out.println("it.previous(): "+it.previous());  
    it.previous(): ES
```



```
13  System.out.println("it.previous(): "+it.previous());  
    it.previous(): DE
```



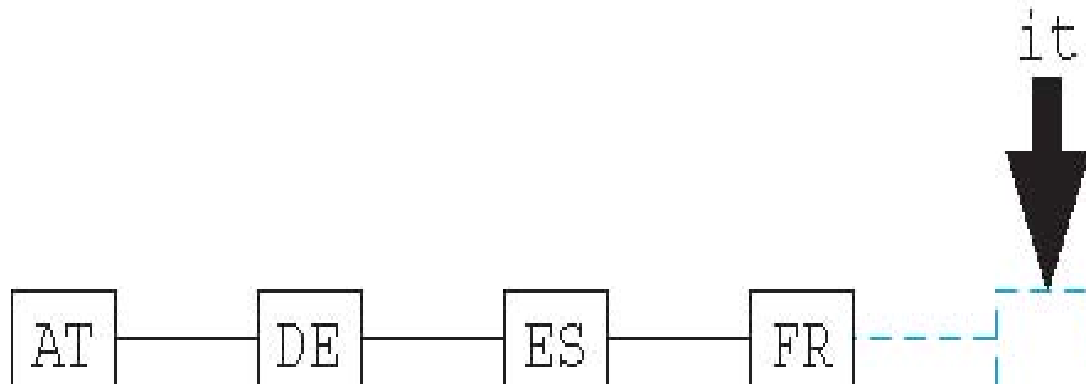
```
14  System.out.println("it.next(): "+it.next());  
    it.next(): AT
```



끝을 지난 반복자

-- 만약, 반복자가 리스트의 끝을 지나 이동했을 때 ::

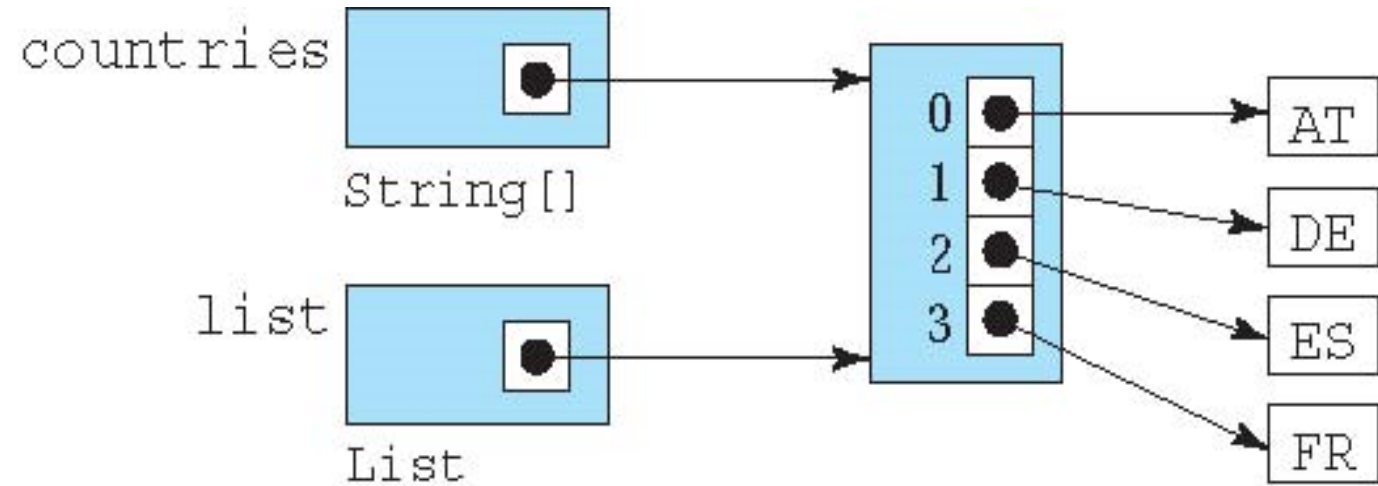
`it.hasPrevious()`는 `true`를 리턴, `it.hasNext()`는 `false`를 리턴함



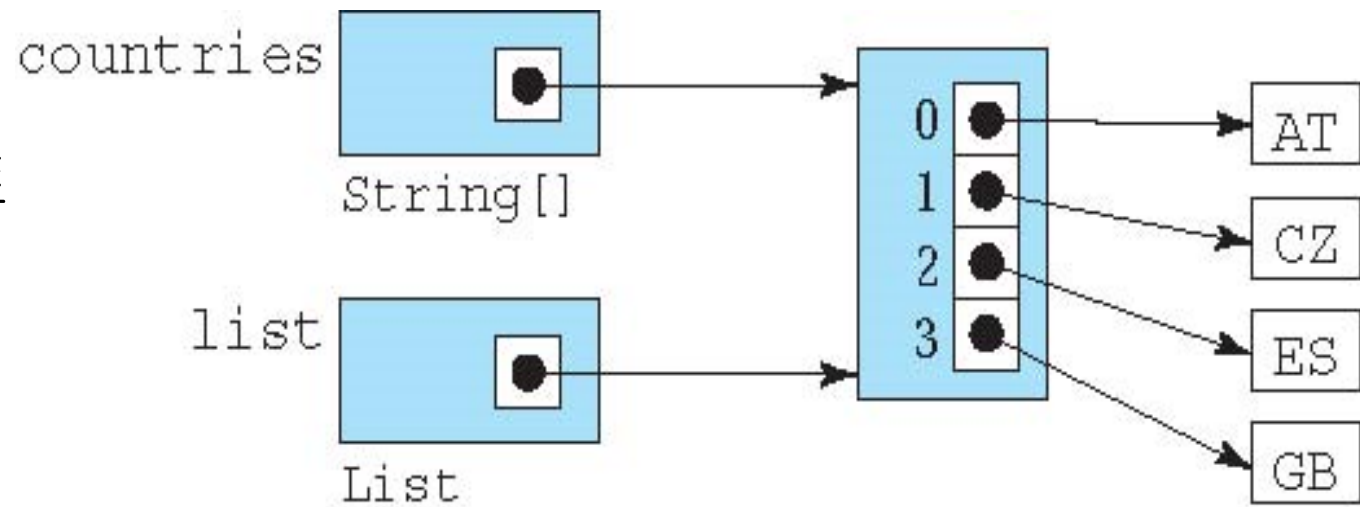
- LISTING 8.4: Testing the java.util.ListIterator Interface

```
1 import java.util.*;
2 public class TestListIterator3 {
3     public static void main(String[] args) {
4         String[] countries = {"AT", "DE", "ES", "FR"};
5         List list = Arrays.asList(countries);
6         System.out.println("list: " + list);
7         ListIterator it=list.listIterator();
8         it.next();
9         it.next();
10        it.set("CZ"); // change DE to CZ
11        System.out.println("list: " + list);
12        System.out.println("countries[1]: "+countries[1]);
13        countries[3] = "GB"; // change FR to GB
14        System.out.println("list: " + list);
15    }
16 }
```


리스트와
문자열 배열



수정된 리스트



- 출력 결과
- list: [AT, DE, ES, FR]
list: [AT, CZ, ES, FR]
countries[1]: CZ
list: [AT, CZ, ES, GB]