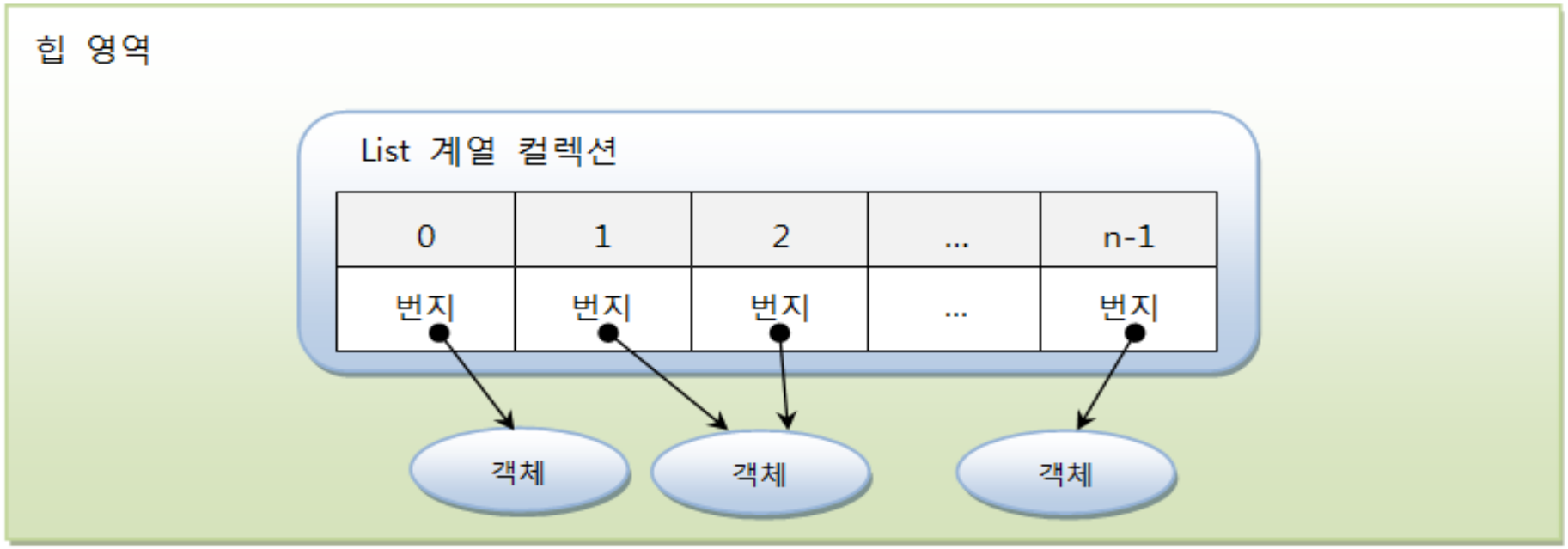
List 컬렉션

❖ List 컬렉션

- 특징
  - 인덱스로 관리
  - 중복해서 객체 저장 가능
- 구현 클래스
  - `ArrayList`
  - `Vector`
  - `LinkedList`



List 컬렉션

❖ List 컬렉션의 주요 메소드

기능	메소드	설명
객체 추가	boolean add(E e)	주어진 객체를 맨끝에 추가
	void add(int index, E element)	주어진 인덱스에 객체를 추가
	set(int index, E element)	주어진 인덱스에 저장된 객체를 주어진 객체로 바꿈
객체 검색	boolean contains(Object o)	주어진 객체가 저장되어 있는지 여부
	E get(int index)	주어진 인덱스에 저장된 객체를 리턴
	isEmpty()	컬렉션이 비어 있는지 조사
	int size()	저장되어있는 전체 객체수를 리턴
객체 삭제	void clear()	저장된 모든 객체를 삭제
	E remove(int index)	주어진 인덱스에 저장된 객체를 삭제
	boolean remove(Object o)	주어진 객체를 삭제

❖ List 컬렉션의 주요 메소드

```
List<String> list = …;
list.add("홍길동");              //맨끝에 객체 추가
list.add(1, "신용권");           //지정된 인덱스에 객체 삽입
String str = list.get(1);       //인덱스로 객체 찾기
list.remove(0);                 //인덱스로 객체 삭제
list.remove("신용권");           //객체 삭제
```

❖ List 컬렉션의 순회

```
List<String> list = …;
for(int i=0; i<list.size(); i++) {
    String str = list.get(i);
}
```

저장된 총 객체 수만큼 루핑

i 인덱스에 저장된 String 객체를 가져옴

String 객체를 하나씩 가져옴

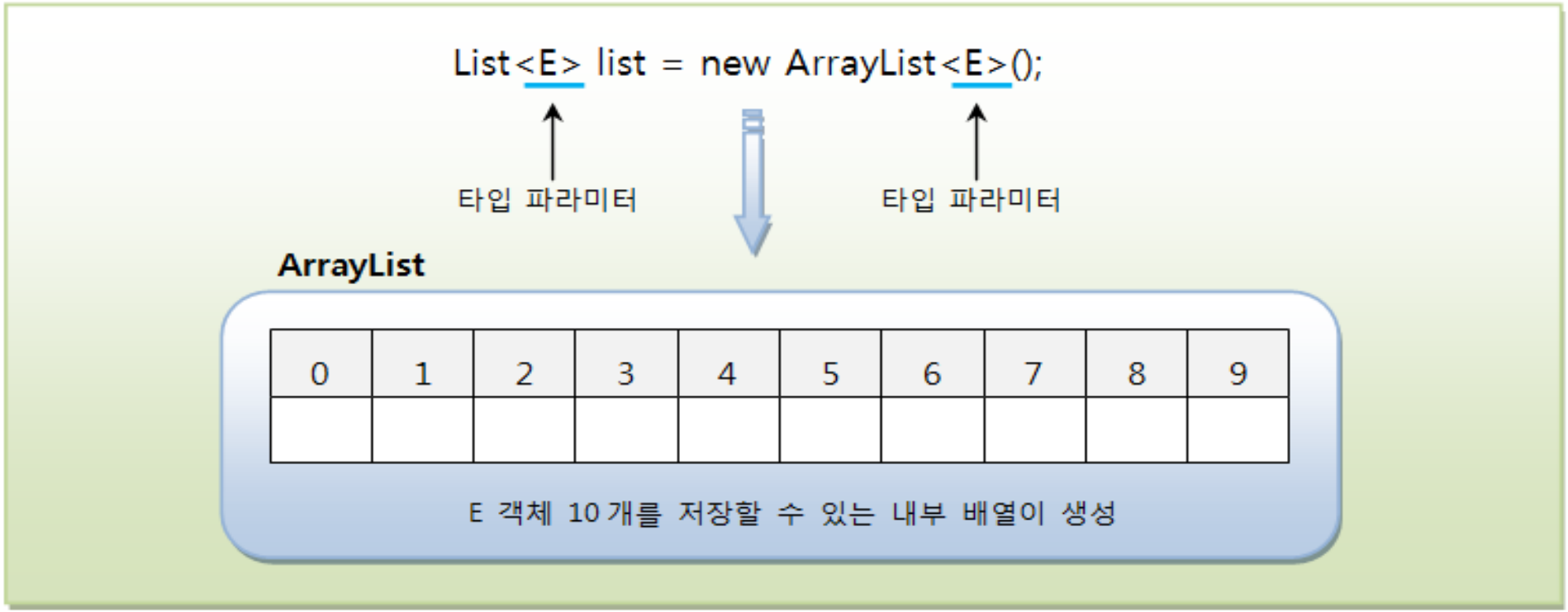
```
for(String str : list) {
}
```

저장된 총 객체 수만큼 루핑

❖ ArrayList

- ○ 저장 용량(capacity)
  - ▪ 초기 용량 : 10 (따로 지정 가능)
  - ▪ 저장 용량을 초과한 객체들이 들어오면 자동적으로 늘어남. 고정도 가능

```
List<String> list = new ArrayList<String>();
```

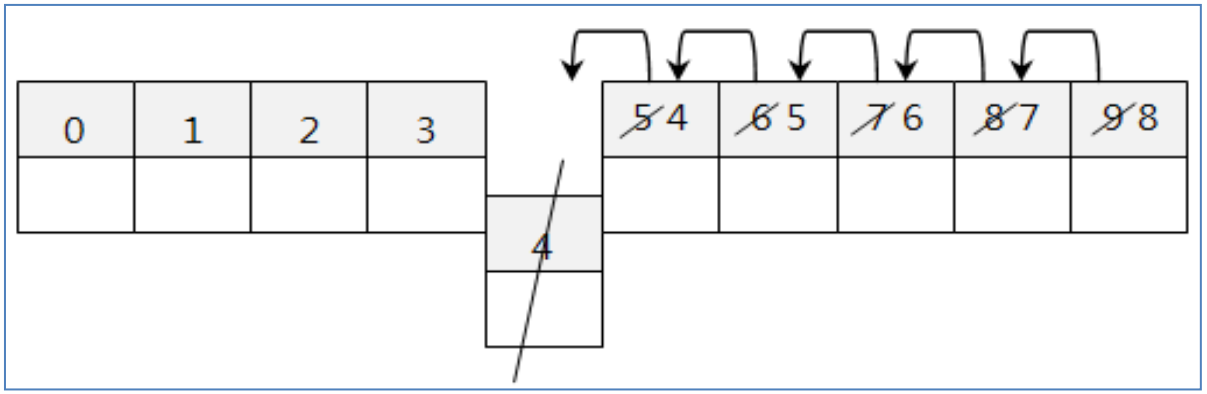


❖ ArrayList

```
list<String> list = new ArrayList<String>();      //컬렉션 생성
list.add("홍길동");                                //컬렉션에 객체를 추가
String name =  list.get(0);                       //컬렉션에서 객체 검색, 홍길동을 바로 얻음
```

❖ ArrayList

- ○ 객체 제거
  - ▪ 바로 뒤 인덱스부터 마지막 인덱스까지 모두 앞으로 1씩 당겨짐



❖ String 객체를 저장하는 ArrayList: ArrayListExample.java

```
import java.util.*;

public class ArrayListExample {
    public static void main(String[] args) {
        List<String> list = new ArrayList<String>();

        list.add("Java");
        list.add("JDBC");
        list.add("Servlet/JSP");
        list.add(2, "Database");
        list.add("iBATIS");

        int size = list.size();
        System.out.println("총 객체수: " + size);
        System.out.println();

        String skill = list.get(2);
        System.out.println("2: " + skill);
        System.out.println();
```

❖ String 객체를 저장하는 ArrayList: ArrayListExample.java

```
        for(int i=0; i<list.size(); i++) {
            String str = list.get(i);
            System.out.println(i + ":" + str);
        }
        System.out.println();

        list.remove(2);
        list.remove(2);
        list.remove("iBATIS");

        for(int i=0; i<list.size(); i++) {
            String str = list.get(i);
            System.out.println(i + ":" + str);
        }
    }
}
```

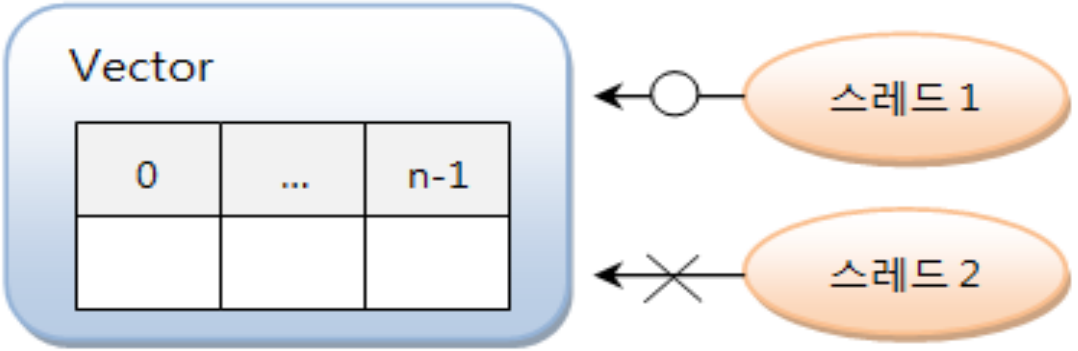
❖ Arrays.asList() 메서드: ArrayListExample.java

```
import java.util.Arrays;
import java.util.List;

public class ArraysAsListExample {
    public static void main(String[] args) {
        List<String> list1 = Arrays.asList("홍길동", "신용권", "감자바");
        for(String name: list1) {
            System.out.println(name);
        }

        List<Integer> list2 = Arrays.asList(1, 2, 3);
        for(int value : list2) {
            System.out.println(value);
        }
    }
```

❖ Vector

```
List<E> list = new Vector<E>();
```

- 특징
  - Vector는 스레드 동기화(synchronization)
  - 복수의 스레드가 동시에 Vector에 접근해 객체를 추가, 삭제하더라도 스레드에 안전(thread safe)



❖ 게시물 정보 객체: ArrayListExample.java

```
public class Board {
    String subject;
    String content;
    String writer;

    public Board(String subject, String content, String writer) {
        this.subject = subject;
        this.content = content;
        this.writer = writer;
    }
}
```

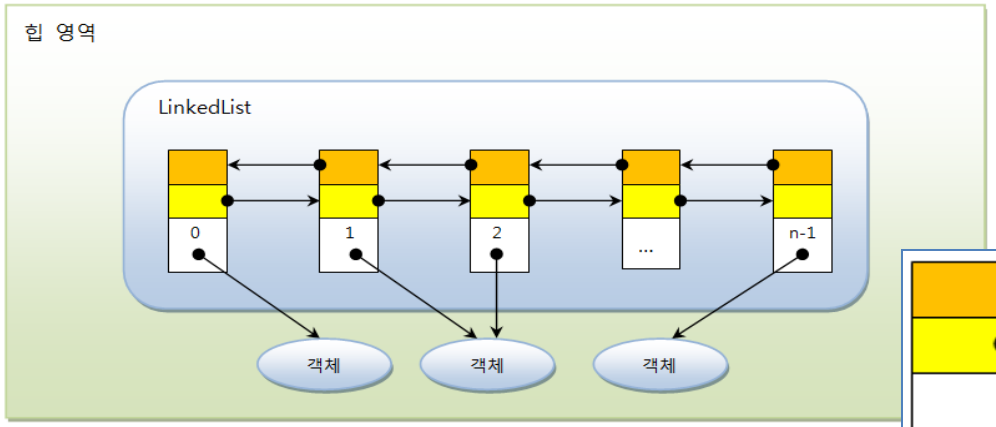
❖ Board 객체를 저장하는 Vector: VectorExample.java

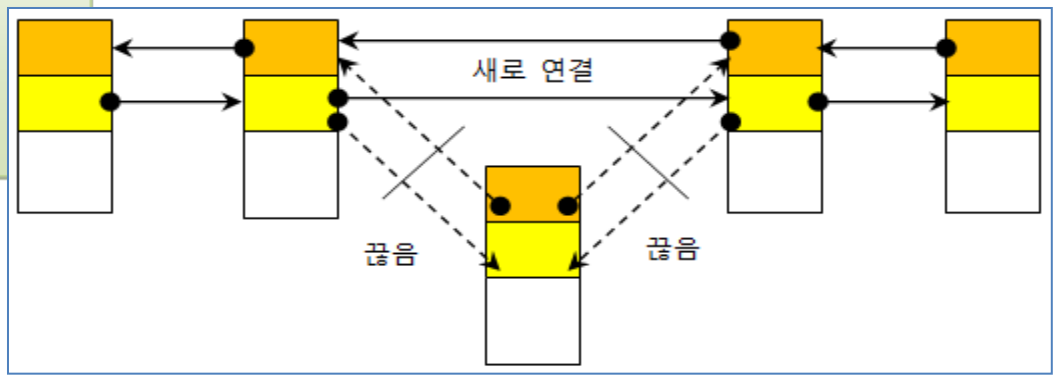
```
import java.util.List;
import java.util.Vector;

public class VectorExample {
    public static void main(String[] args) {
        List<Board> list = new Vector<Board>();

        list.add(new Board("제목1", "내용1", "글쓴이1"));
        list.add(new Board("제목2", "내용2", "글쓴이2"));
        list.add(new Board("제목3", "내용3", "글쓴이3"));
        list.add(new Board("제목4", "내용4", "글쓴이4"));
        list.add(new Board("제목5", "내용5", "글쓴이5"));

        list.remove(2);
        list.remove(3);

        for(int i=0; i<list.size(); i++) {
            Board board = list.get(i);
            System.out.println(board.subject + "\t" + board.content + "\t" +
                               board.writer);
        }
    }
}
```

❖ LinkedList

```
List<E> list = new LinkedList<E>();
```

- 특징
  - 인접 참조를 링크해서 체인처럼 관리
  - 특정 인덱스에서 객체를 제거하거나 추가하게 되면 바로 앞뒤 링크만 변경
  - 빈번한 객체 삭제와 삽입이 일어나는 곳에서는 ArrayList보다 좋은 성능





❖ ArrayList와 LinkedList의 실행 성능 비교: ArrayListExample.java

```
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;

public class LinkedListExample {
    public static void main(String[] args) {
        List<String> list1 = new ArrayList<String>();
        List<String> list2 = new LinkedList<String>();

        long startTime;
        long endTime;

        startTime = System.nanoTime();
        for(int i=0; i<10000; i++) {
            list1.add(0, String.valueOf(i));
        }

        endTime = System.nanoTime();
        System.out.println("ArrayList 걸린시간: " +
                           (endTime-startTime) + " ns");
```

❖ ArrayList와 LinkedList의 실행 성능 비교: ArrayListExample.java

```
        startTime = System.nanoTime();
        for(int i=0; i<10000; i++) {
            list2.add(0, String.valueOf(i));
        }
        endTime = System.nanoTime();
        System.out.println("LinkedList 걸린시간: " +
                           (endTime-startTime) + " ns");
    }
}
```

구분	순차적으로 추가/삭제	중간에 추가/삭제	검색
ArrayList	빠르다	느리다	빠르다
LinkedList	느리다	빠르다	느리다