# Collection<E> → List<E> interface

## Collection<E> → List<E> 인터페이스를 구현하는 컬랙션즈 클래스

---

### List<E> Interface

- 인스턴스를 저장하는 컨테이너
- 인스턴스의 중복저장을 허용한다.
- 인스턴스의 저장순서 유지

- List<E>의 대표적인 메소드

| | |
|---|---|
| add(E e) | 인자로 전달된 인스턴스 데이터 저장 |
| get(int idx) | 인자로 전달된 인덱스값에 위치한 인스턴스 반환 |
| remove(int idx) | 인자로 전달된 인덱스값에 위치한 인스턴스 삭제 |
| int size() | List<E>에 저장된 인스턴스 개수 반환 |

### i. ArrayList<E>

이름에서 의미하듯이 ArrayList<E>는 배열 기반으로 데이터를 저장한다.

```
import java.util.ArrayList;

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

          // 인스턴스 데이터의 저장
          list.add(new Integer(11));
          list.add(new Integer(11));
          list.add(new Integer(33));

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

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

| Index | 0 | 1 | 2 |
|---|---|---|---|
| Data | 11 | 11 | 22 |

↓ list.remove(0);

| Index | 0 | 1 | 2 |
|---|---|---|---|
| Data | 11 | 22 | |

```
    }
}
```

### ii. LinkedList<E>

**이름에서 의미하듯이 LinkedList<E>는 리스트 자료구조 기반으로 데이터를 저장한다.**

**대부분 ArrayList<E> 로 대체 가능하다.**

```
public class LinkedListDemo {
    public static void main(String[] args) {
        LinkedList<Integer> list = new LinkedList<Integer>();

        // 인스턴스 데이터의 저장
        list.add(new Integer(11));
        list.add(new Integer(11));
        list.add(new Integer(33));

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

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

- **ArrayList<E> 와 LinkedList<E>의 차이점**

**ArrayList<E>**

- **배열기반**
- **저장공간을 늘리는 과정에서 많은 시간 소요 단점**
- **데이터의 삭제에 필요한 연산과정이 매우 길다. 단점**
- **인덱스를 이용한 데이터의 참조가 용이해서 빠른 참조 가능 장점**

**LinkedList<E>**

- **자료구조 - 리스트 기반**
- **저장공간을 늘리는 것이 간단 장점**
- **데이터의 삭제가 간단 장점**
- **데이터의 참조가 다소 불편하다. 단점**

---

## Iterator<E> Interface (반복자)

- 모든 Collection<E> 인터페이스를 구현하는 컬렉션즈 클래스들이 가지고 있는 Iterator<E> 인터페이스
- **컨테이너에 담긴 값들을 순차적으로 하나하나씩 꺼내서 처리를 해주는 역할**

java.util

## Interface Iterator<E>

**Type Parameters:**

E - the type of elements returned by this iterator

**All Known Subinterfaces:**

ListIterator<E>, XMLEventReader

**All Known Implementing Classes:**

BeanContextSupport.BCSIterator, EventReaderDelegate, Scanner

```
import java.util.Iterator;
import java.util.LinkedList;

public class IteratorDemo {
     public static void main(String[] args) {
          LinkedList<String> list = new LinkedList<String>();
          list.add("first");
          list.add("second");
          list.add("third");
          list.add("fourth");

          System.out.println("iterator를 사용한 순차검색");

          Iterator<String> iterator = list.iterator();

          // 현재 참조할 요소가 있으면 true (0번째 부터 시작)
          while(iterator.hasNext()){

               // 현재 위치의 요소를 반환하고, 다음 순서로 이동
               String str = iterator.next();
               System.out.println(str);

               // next()가 다음번 순서로 이동하기 전 순서에서 데이터 비교 후 같으면 삭제
               if(str.compareTo("third") == 0)
                    iterator.remove();
          }

          System.out.println("삭제 후 iterator를 사용한 순차검색");

          iterator = list.iterator();

          while(iterator.hasNext())
               System.out.println(iterator.next());
     }
}
```

| | |
|---|---|
| Iterator<E> iterator() | Collection<E> 인터페이스를 구현하는<br>컬렉션 클래스의 인스턴스가 가지고 있는 iterator 인스턴스의 참조값 반환 |
| boolean hasNext() | 참조할 요소(Element)가 있으면 true, 더 이상 없으면 false 반환<br>(0 번째 요소부터 시작) |
| E next() | 현재 위치의 요소(Element)를 반환하고 다음 순서로 이동한다. |
| void remove() | 현재 위치의 요소 삭제 |

---