

Java Collections Framework

Bok, Jong Soon

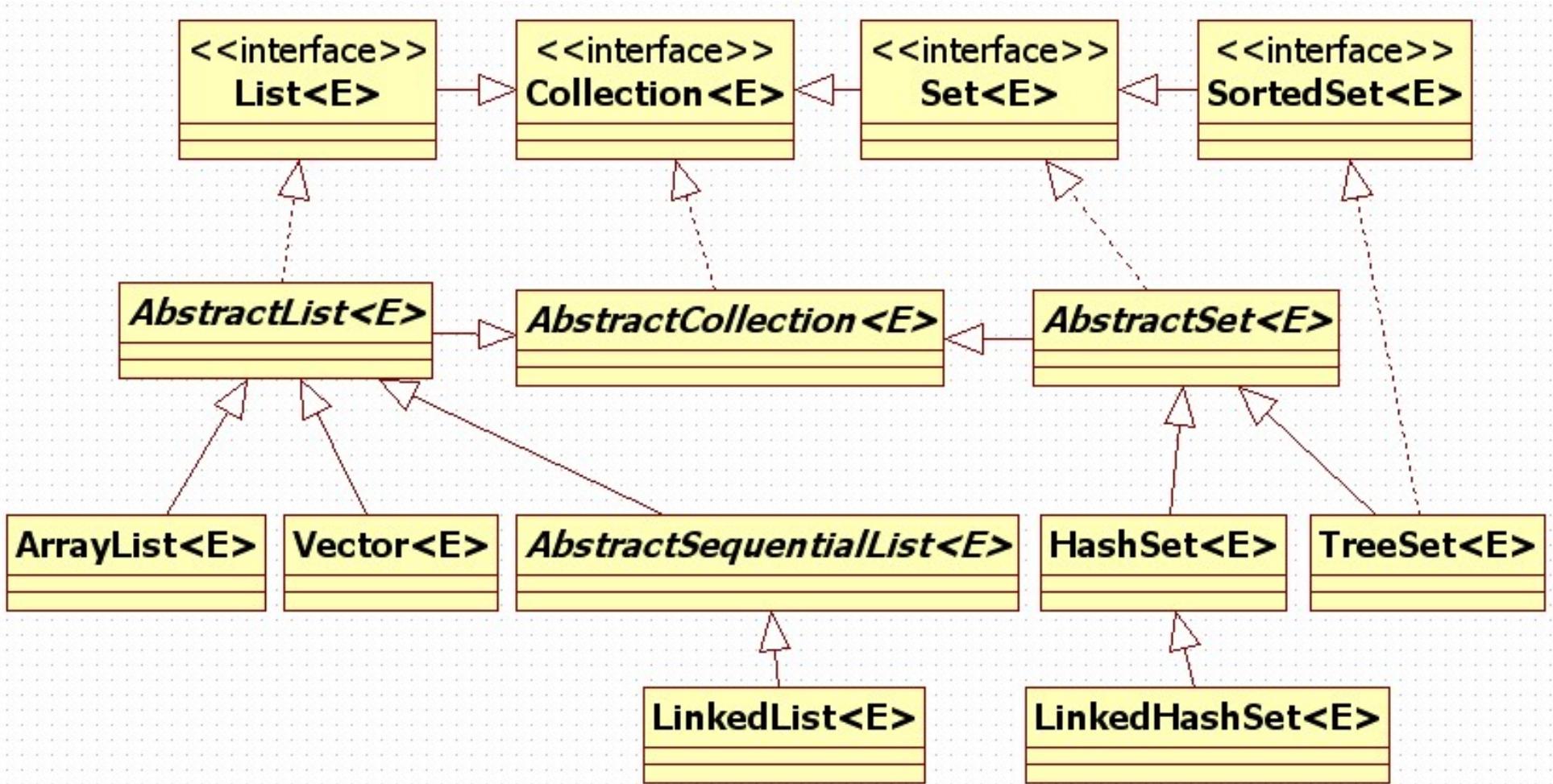
javaexpert@nate.com

<https://github.com/swacademy/Core-Java>

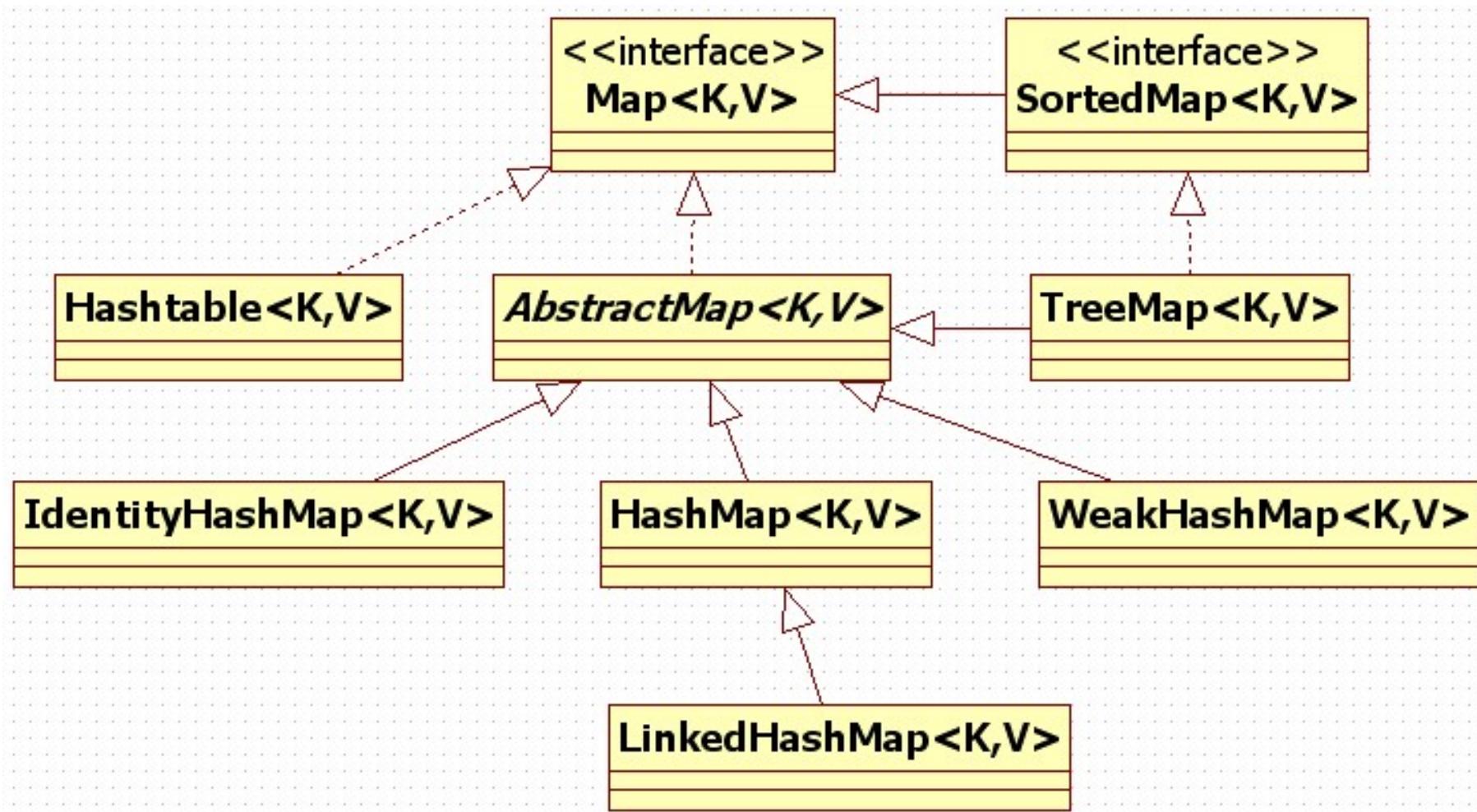
Collection API

- A Collection (or a container) is a single object representing a group of objects known as its elements.
- Collection classes **Vector**, **Bits**, **BitSet**, **Stack**, **Hashtable**, **LinkedList**, and so on are supported.
- The Collection API contains interfaces that maintain objects as a :
 - **Collection** – A group of objects with no specific ordering
 - **Set** – A group of objects with no duplication
 - **SortedSet** – A group of objects with no duplication and being sorted
 - **List** – A group of ordered objects; duplication is permitted.
 - **Map** – key and value mapping
 - **SortedMap** – Map with Sorted key

Collection<E> - related classes



Map<K, V> - related classes



Iterator<E> interface

- An **Iterator** over a collection.
- **Iterator** takes the place of **Enumeration** in the Java collections framework.
- **Iterators** differ from **Enumerations** in two ways:
 - **Iterators** allow the caller to remove elements from the underlying collection during the iteration with well-defined semantics.
 - Method names have been improved.

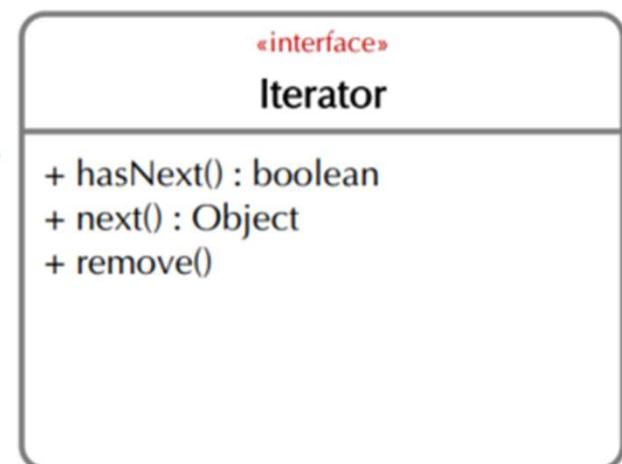
<<interface>>
Iterator<E>

+*hasNext(): boolean*
+*next(): E*
+*remove(): void*

Iterator<E> interface (Cont.)

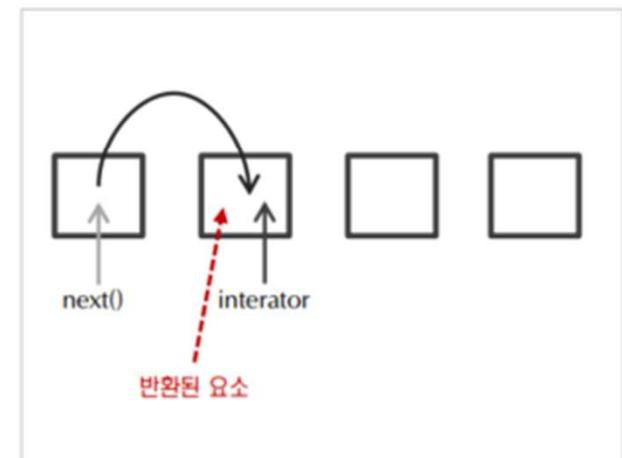
✓ java.util.Iterator 인터페이스

- Collection 원소를 이동하면서 접근하는 포인터 역할을 합니다.
- Collection 원소를 순회하며 원소를 삽입, 삭제할 수 없기 때문에 Iterator 를 사용합니다.
- Iterator 는 Collection 인터페이스의 iterator() 메소드의 리턴 값으로 얻습니다.
- Collection 원소를 순회하기 위한 메소드를 제공합니다.
 - hasNext() : 다음 원소가 존재하는지 여부를 판단합니다.
 - next() : 다음 위치의 원소로 포인터를 이동해 원소를 리턴합니다.
 - remove() : 현재 위치에 있는 원소를 Collection 에서 제거합니다.



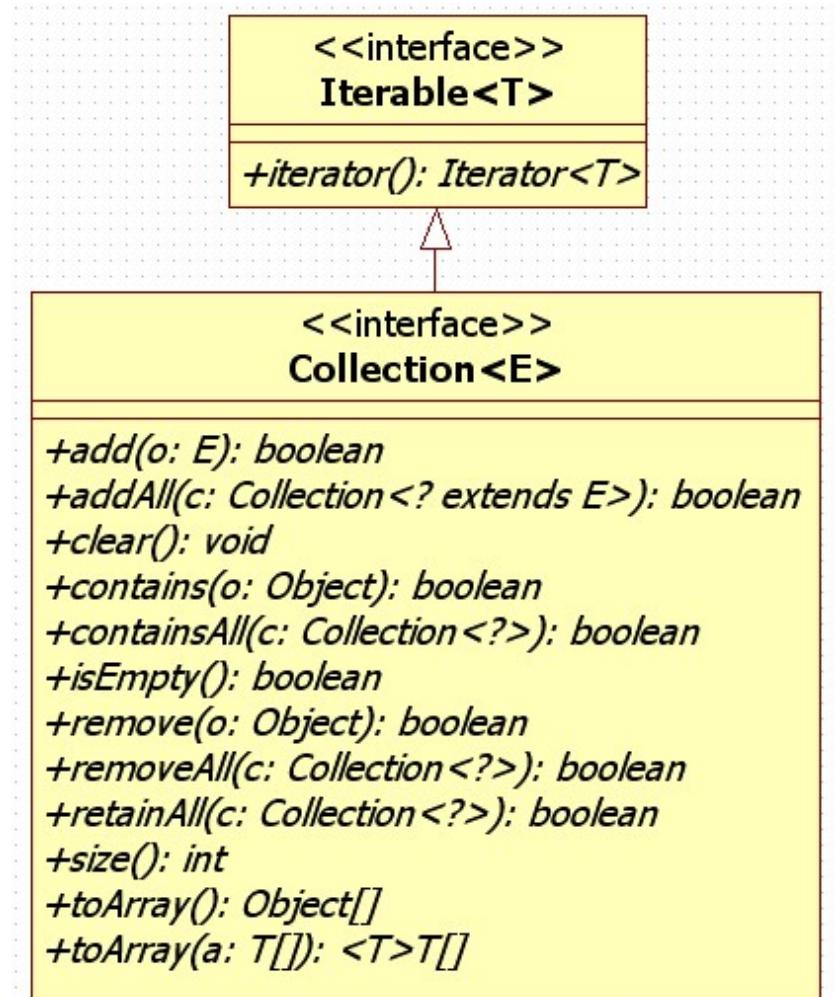
```
List<Student> students = new ArrayList<Student>();  
students.add(new Student("Harold"));  
students.add(new Student("John"));
```

```
Iterator<Student> iter = students.iterator();  
while (iter.hasNext()) {  
    Student student = iter.next();  
  
    // something to do  
}
```



Collection<E> interface

- The root interface in the *collection hierarchy*.



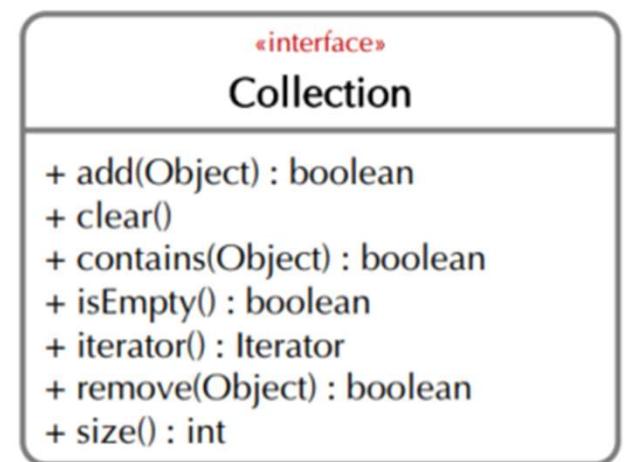
Collection<E> interface (Cont.)

✓ java.util.Collection 인터페이스

- Collection 프레임워크의 최상위 인터페이스입니다.
- Collection 원소에 대한 삽입, 삭제, 탐색의 기능을 정의합니다.

✓ 주요 메소드

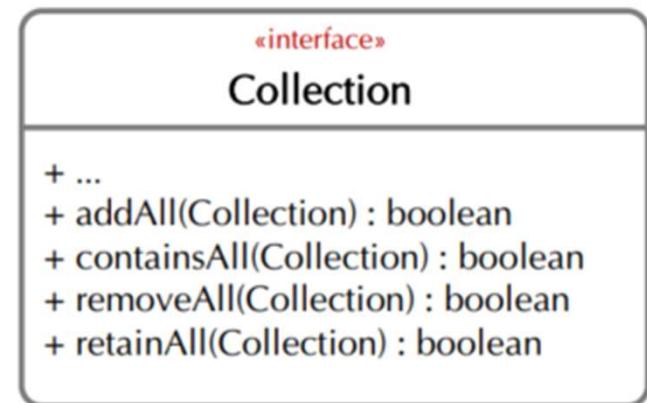
- add() : 새로운 원소를 삽입한다. 중복 원소를 허용하지 않는 경우 false 를 리턴합니다.
- clear() : 모든 원소를 제거한다. 제거 후에 Collection 크기는 0이 됩니다.
- contains() : 주어진 객체가 존재하는지 탐색합니다.
- isEmpty() : Collection 크기가 0인지 여부를 판단합니다.
- remove() : 주어진 객체를 포함하고 있으면 제거한다. Collection이 변경되면 true 를 리턴합니다.
- size() : 모든 원소의 갯수를 리턴합니다.
- iterator() : Collection의 원소를 순회하기 위한 Iterator 객체를 리턴합니다.



Collection<E> interface (Cont.)

✓ java.util.Collection 추가 기능들

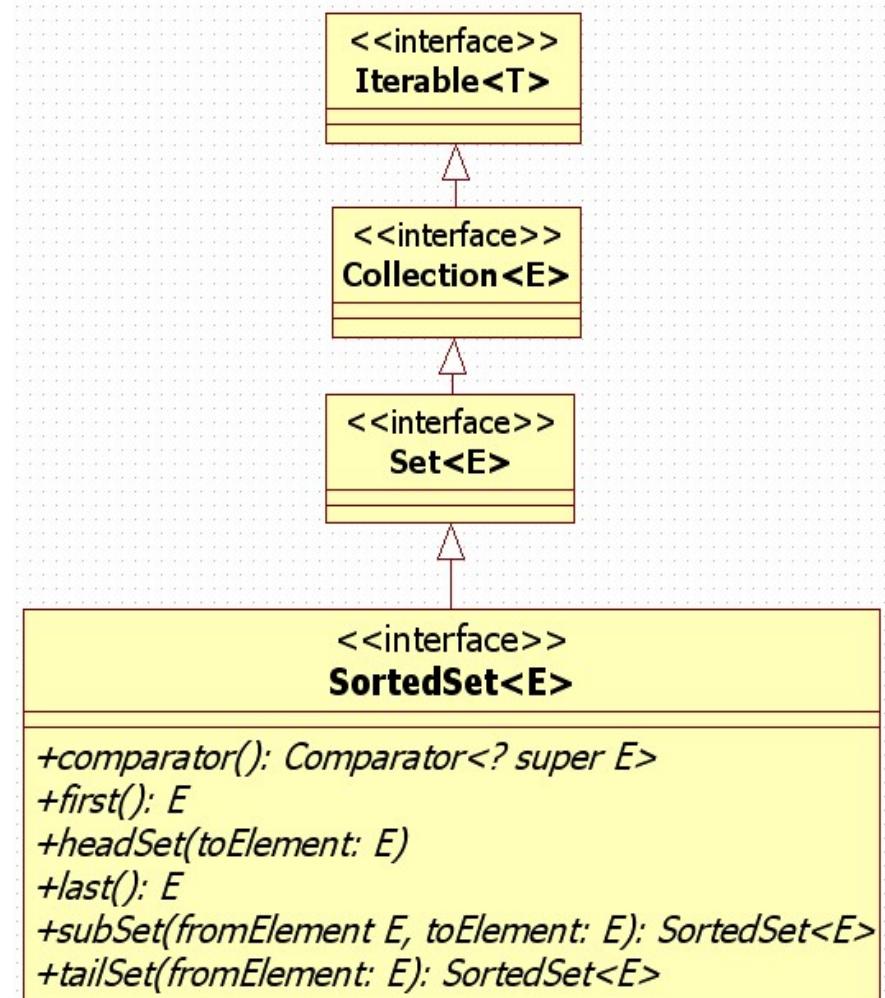
- addAll() : 요청 Collection 의 모든 원소를 추가합니다. (중복된 원소도 추가)
- containsAll() : 요청 Collection 의 모든 원소를 포함할 경우 true 를 리턴합니다.
- removeAll() : 요청 Collection 과 일치하는 원소를 제거합니다.
- retainAll() : 요청 Collection 과 일치하지 않는 원소를 제거합니다.



메소드	집합연산	기호	$x = \{A, B, C, D, E\}$ $y = \{A, E, I, O, U\}$
<code>x.addAll(y)</code>	합집합	$x = x \cup y$	$x = \{A, B, C, D, E, A, E, I, O, U\}$
<code>x.containsAll(y)</code>	부분집합	$x \supseteq y$	false 를 리턴
<code>x.removeAll(y)</code>	상대보수	$x = x - y$	$x = \{B, C, D\}$
<code>x.retainAll(y)</code>	교집합	$x = x \cap y$	$x = \{A, E\}$

Set<E> interface

- A collection that contains no duplicate elements.
- Models the mathematical *set* abstraction.
- Places additional stipulations, beyond those inherited from the **Collection** interface.
- Implementations prohibit *null* elements.



Set<E> interface (Cont.)

✓ Set은 중복을 허용하지 않는 Collection으로 수학에서의 집합 개념을 나타냅니다.

- 구현 클래스 내부에 equals(Object o) 메소드를 이용해서 중복을 체크합니다.

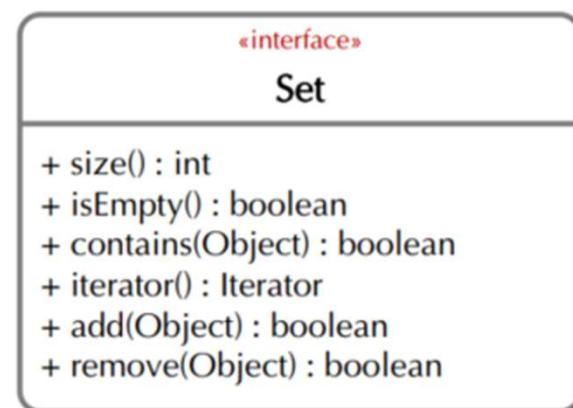
✓ Set 구현클래스

- HashSet : 입력 순서를 유지하지 않습니다.
- LinkedHashSet : 입력 순서를 유지합니다.
- TreeSet : 입력 순서와 상관없이 오름차순 정렬됩니다.

✓ HashSet 예제

```
Random rand = new Random(47);
Set<Integer> intset = new HashSet<Integer>();

for (int i = 0; i < 10000; i++) {
    intset.add(rand.nextInt(30));
}
System.out.println(intset);
```

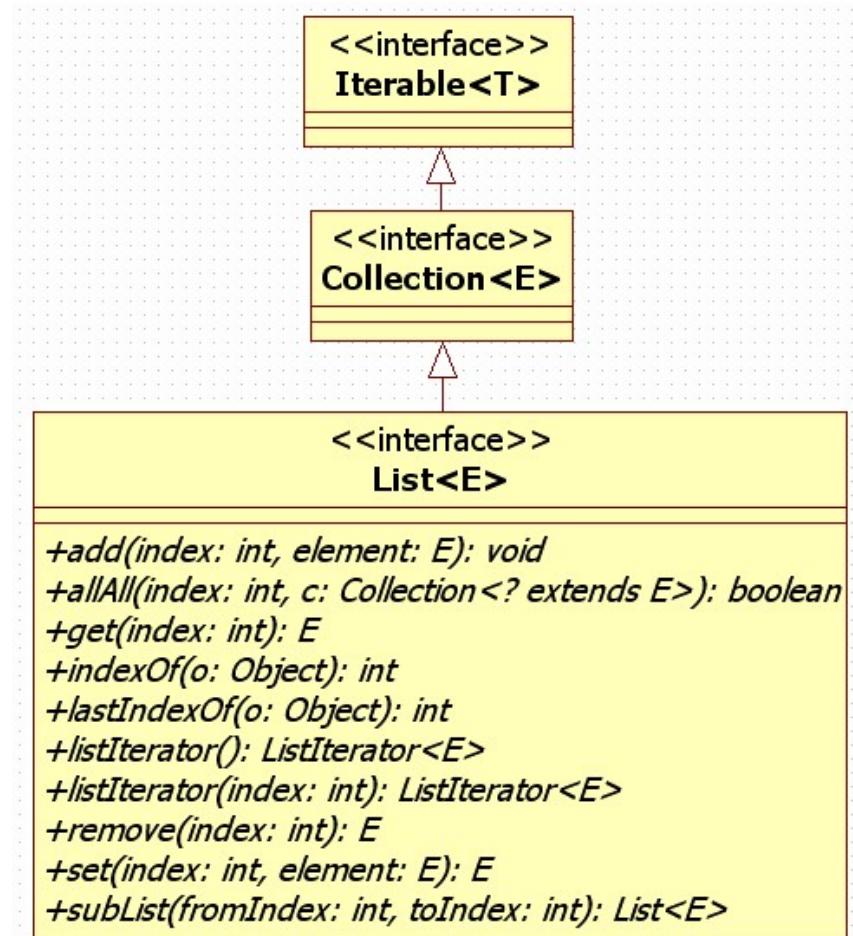


실행결과

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 16, 19,
18, 21, 20, 23, 22, 25, 24, 27, 26, 29, 28]
```

List<E> interface

- An ordered collection (also known as a *sequence*).
- The user can access elements by their integer *index* (position in the list), and search for elements in the list.
- Unlike **Sets**, typically allow duplicate elements.



List<E> interface (Cont.)

✓ java.util.List 인터페이스

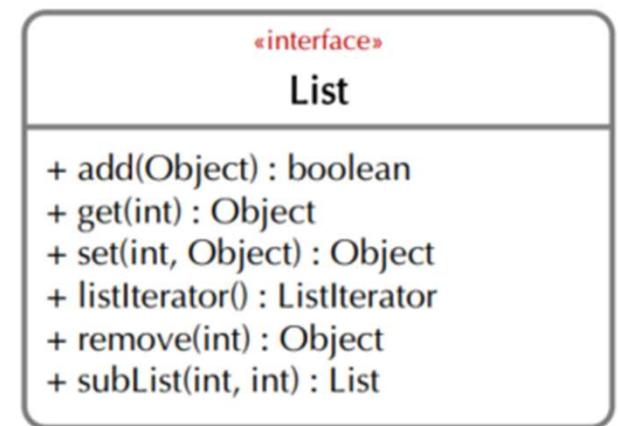
- 일련의 원소를 저장하는 자료구조입니다.
- List는 중복된 값과 null 값을 가질 수 있습니다.
- 인덱스에 의한 원소 접근이 가능합니다.
 - 해당하는 인덱스 위치의 원소를 탐색, 제거, 추가합니다.

✓ 주요 메소드

- add() : 리스트에 하나의 객체를 추가합니다.
- get() : 인덱스에 해당하는 객체를 리턴합니다.
- set() : 해당하는 인덱스 위치에 객체를 세팅합니다. (기존에 객체가 있는 경우 대체됨)
- listIterator() : 리스트를 순회하기 위한 iterator 객체를 리턴합니다.
- remove() : 인덱스에 해당하는 객체를 삭제합니다.
- subList() : 해당 인덱스 범위의 객체목록을 리스트로 리턴합니다.

✓ 구현 클래스

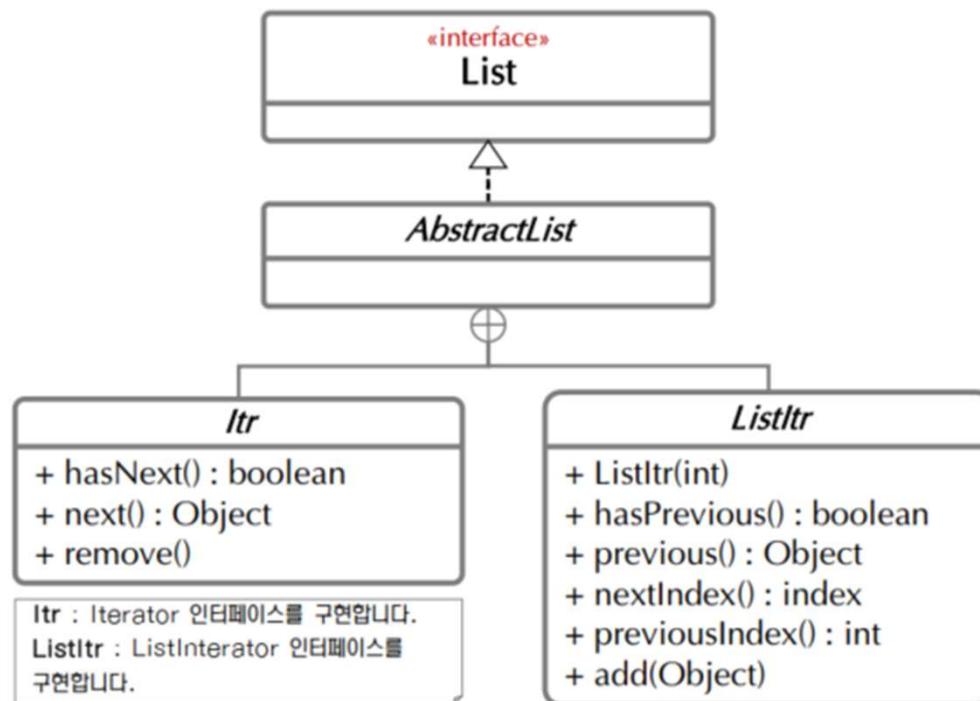
- ArrayList
- LinkedList



List<E> interface (Cont.)

✓ java.util.List 탐색

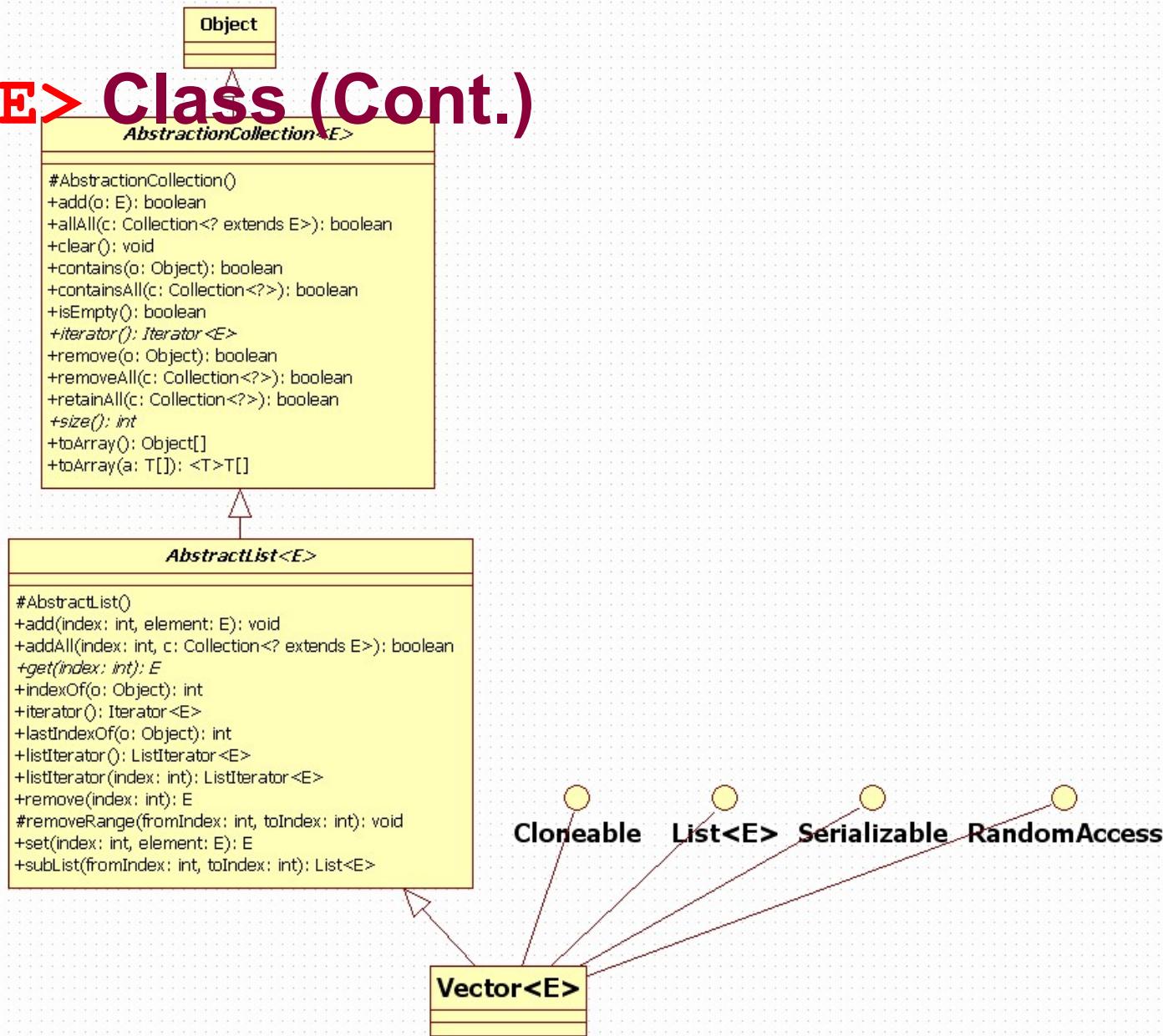
- AbstractList 객체에 내부 클래스를 정의합니다.
- Iterator 인터페이스 기능을 제공합니다.
- 추가로 ListIterator 인터페이스 기능을 제공합니다.
 - 인덱스 번호로 순회할 요소를 지정하여 iterator를 생성할 수 있습니다.
 - 이전이나 다음 요소로의 순회 기능을 제공합니다.



Vector<E> Class

- Implements a growable array of objects.
- Like an array, it contains components that can be accessed using an integer index.
- The size can grow or shrink as needed to accommodate adding and removing items.
- Each vector tries to optimize storage management by maintaining a **capacity** and a **capacityIncrement**.

Vector<E> Class (Cont.)



Vector's Constructors & Fields

Vector<E>
#capacityIncrement: int
#elementCount: int
#elementData: Object[]
+Vector()
+Vector(c: Collection<? extends E>)
+Vector(initialCapacity: int)
+Vector(initialCapacity: int, capacityIncrement: int)

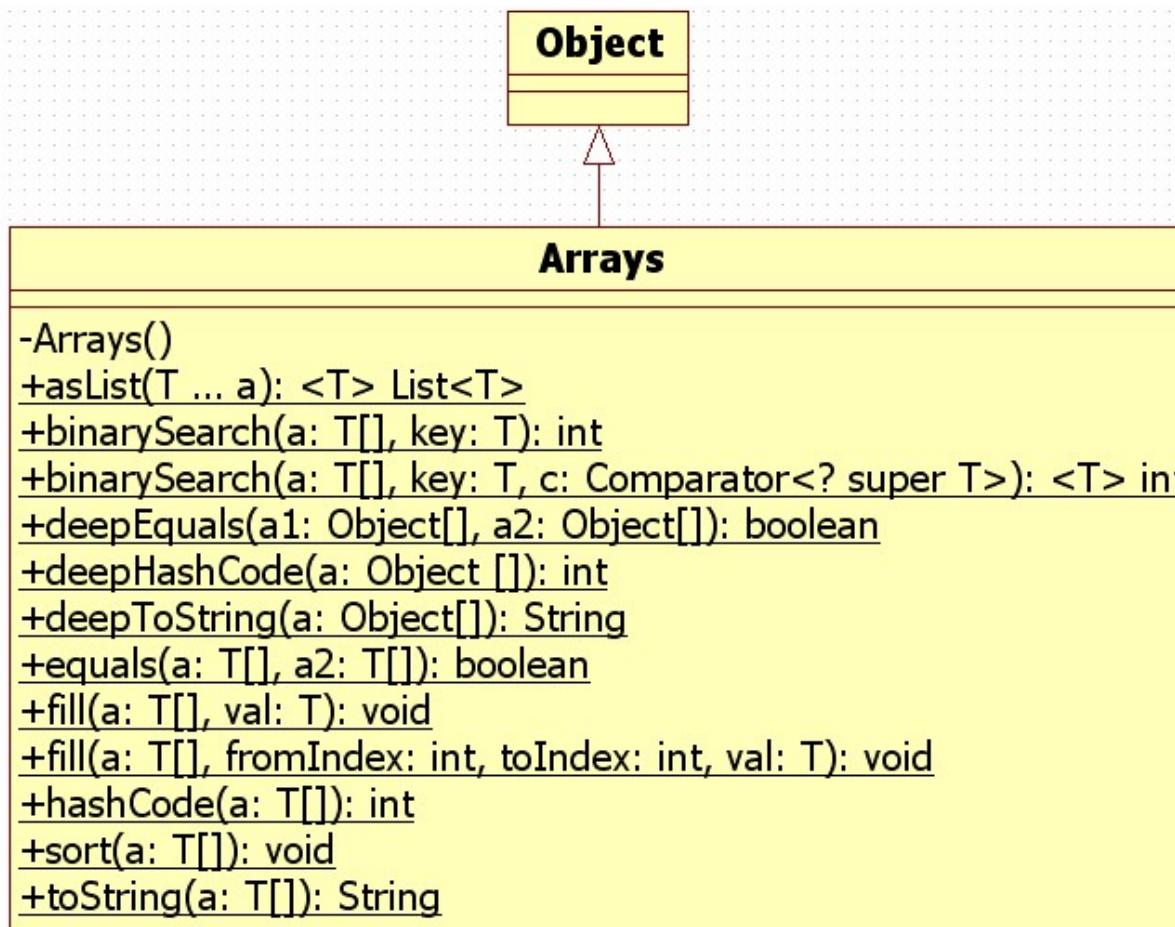
Vector's Methods

```
+addElement(obj: E): void  
+capacity(): int  
+clone(): Object  
+copyInto(anArray: Object[]): void  
+elementAt(index: int): E  
+elements(): Enumeration<E>  
+ensureCapacity(minCapacity: int): void  
+firstElement(): E  
+get(index: int): E  
+indexOf(elem: Object, index: int): int  
+insertElementAt(obj: E, index: int): void  
+lastElement(): E  
+lastIndexOf(elem: Object, index: int): int  
+removeAllElements(): void  
+removeElement(obj: Object): boolean  
+removeElementAt(index: int): void  
+setElementAt(obj: E, index: int): void  
+setSize(newSize: int): void  
+size(): int  
+trimToSize(): void
```

Arrays class

- Contains various methods for manipulating arrays.
- Contains a static factory that allows arrays to be viewed as lists.
- The methods all throw a **NullPointerException** if the specified array reference is **null**, except where noted.
- Is a member of the *Java Collections Framework*.

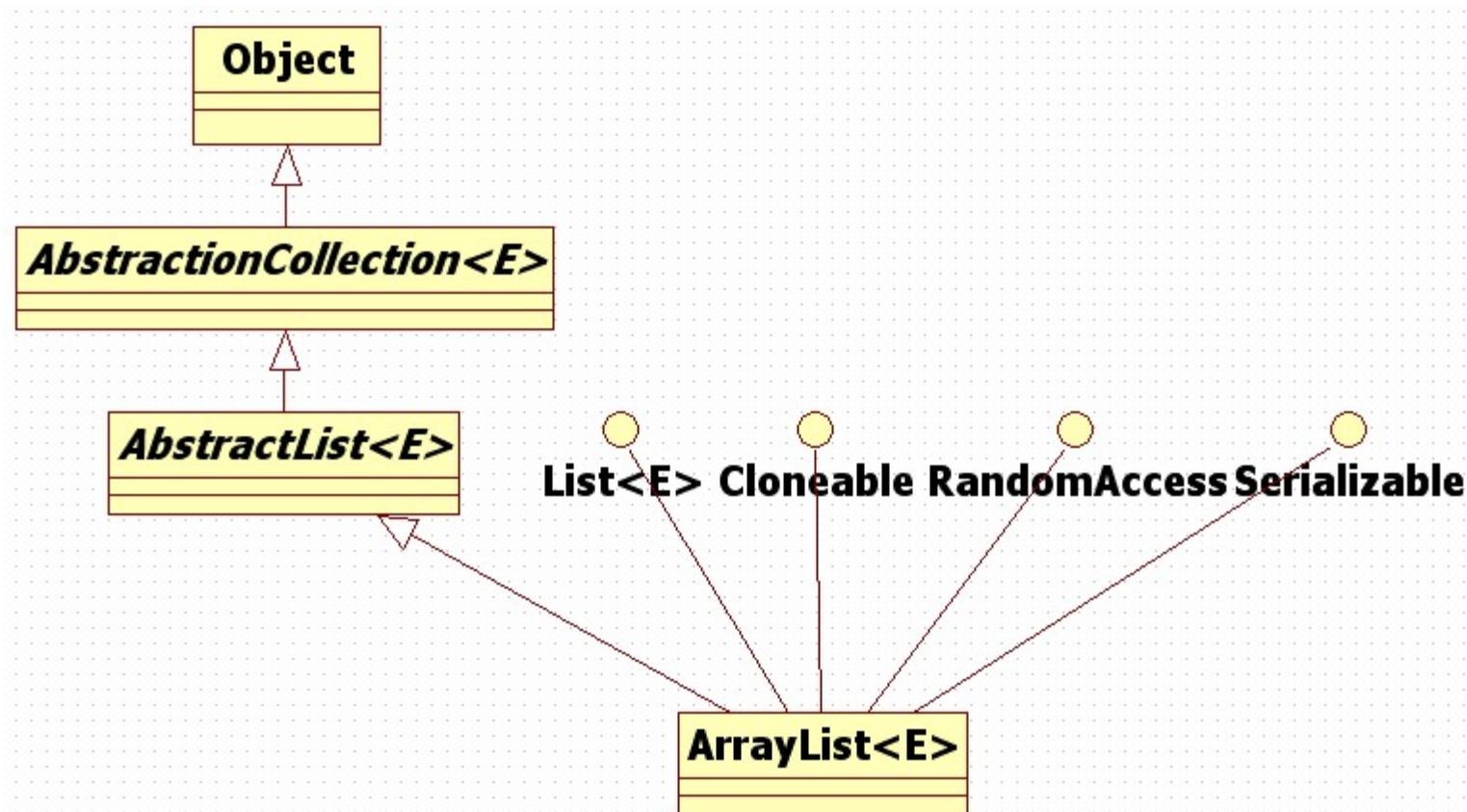
Arrays class (Cont.)



ArrayList<E> class

- Resizable-array implementation of the **List** interface.
- Implements all optional list operations, and permits all elements, including **null**.
- Provides methods to manipulate the size of the array that is used internally to store the list.
- Is a member of the **Java Collections Framework**.

ArrayList<E> class (Cont.)



ArrayList<E> class (Cont.)

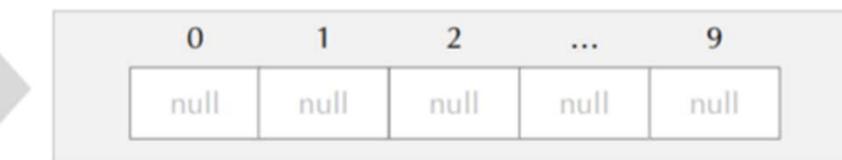
ArrayList<E>
+ArrayList()
+ArrayList(c: Collection<? extends E>)
+ArrayList(initialCapacity: int)
+add(o: E): boolean
+add(index: int, element: E): void
+addAll(c: Collection<? extends E>): boolean
+addAll(index: int, c: Collection<? extends E>): boolean
+clear(): void
+clone(): Object
+contains(elem: Object): boolean
+ensureCapacity(minCapacity: int): void
+get(index: int): E
+indexOf(elem: Object): int
+isEmpty(): boolean
+lastIndexOf(elem: Object): int
+remove(index: int): E
+remove(o: Object): boolean
#removeRange(fromIndex: int, toIndex: int): void
+set(index: int, element: E): E
+size(): int
+toArray(): Object[]
+toArray(a: T[]): <T>T[]
+trimToSize(): void

ArrayList<E> class (Cont.)

- ✓ ArrayList 클래스는 List 인터페이스를 구현하며, 배열과 같이 인덱스를 통해 요소에 접근합니다.

- ArrayList 객체 생성

```
// create a ArrayList instance  
List skills = new ArrayList();
```



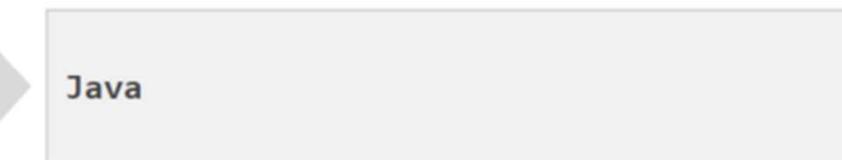
- 리스트에 요소 추가

```
skills.add("Java");  
skills.add("SQL");
```



- 리스트 요소 조회

```
// print element of index 0  
System.out.println(skills.get(0));
```



- 리스트 요소 삭제

```
// remove element of index 0  
skills.remove(0);
```



ArrayList<E> class (Cont.)

✓ 리스트 객체 추가 (without Generic)

- 리스트에 Object 타입의 객체를 저장할 수 있습니다.
- 어떠한 객체가 추가되어도 컴파일 에러가 발생하지 않습니다.

✓ 리스트 객체 사용 (without Generic)

- 사용하려는 객체와 타입이 맞지 않으면 런타임 시 예외가 발생합니다.

```
class Apple {  
  
    private static long counter;  
    private final long id = counter++;  
  
    public long id() {  
        return id;  
    }  
}
```

```
class Orange {  
  
}
```

```
public class ApplesAndOrangesWithoutGenerics {  
    public static void main(String[] args) {  
        ArrayList apples = new ArrayList();  
        for (int i = 0; i < 3; i++) {  
            apples.add(new Apple());  
        }  
        // [!] 의도하지 않은 타입이 추가됨  
        apples.add(new Orange());  
  
        for (int i = 0; i < apples.size(); i++) {  
            ((Apple) apples.get(i)).id();  
            // 런타임에서야 예외가 발생함  
        }  
    }  
}
```

ApplesAndOrangesWithoutGenerics.java

Exception in thread "main" java.lang.ClassCastException: Orange cannot be cast to Apple
at ApplesAndOrangesWithoutGenerics.main(ApplesAndOrangesWithoutGenerics.java:19)

실행결과

ArrayList<E> class (Cont.)

✓ 리스트 객체 추가 (with Generic)

- 리스트에 지정된 타입이 아닌 요소가 추가되는 것을 컴파일 시점에서 방지할 수 있습니다.
- 리스트의 요소를 꺼낼 때, 캐스팅이 필요 없습니다.

✓ 리스트 객체 사용 (with Generic)

- 객체를 사용하는 시점에 별다른 처리 없이 사용 가능하며 아무런 예외가 발생하지 않습니다.

```
public class ApplesAndOrangesWithGenerics {
    public static void main(String[] args) {
        ArrayList<Apple> apples = new ArrayList<Apple>();

        for (int i = 0; i < 3; i++) {
            apples.add(new Apple());
        }
        // Compile-time error:
        // apples.add(new Orange());

        for (int i = 0; i < apples.size(); i++) {
            System.out.println(apples.get(i).id());
        }
        // Using foreach:
        for (Apple c : apples) {
            System.out.println(c.id());
        }
    }
}
```

ApplesAndOrangesWithGenerics.java

ArrayList<E> class (Cont.)

- ✓ 참조변수를 위해 구현 클래스보다 상위 클래스나 인터페이스를 사용합니다.
 - ArrayList 대신 List로 업캐스팅 하면 향후 구현클래스를 간단하게 교체할 수 있습니다.

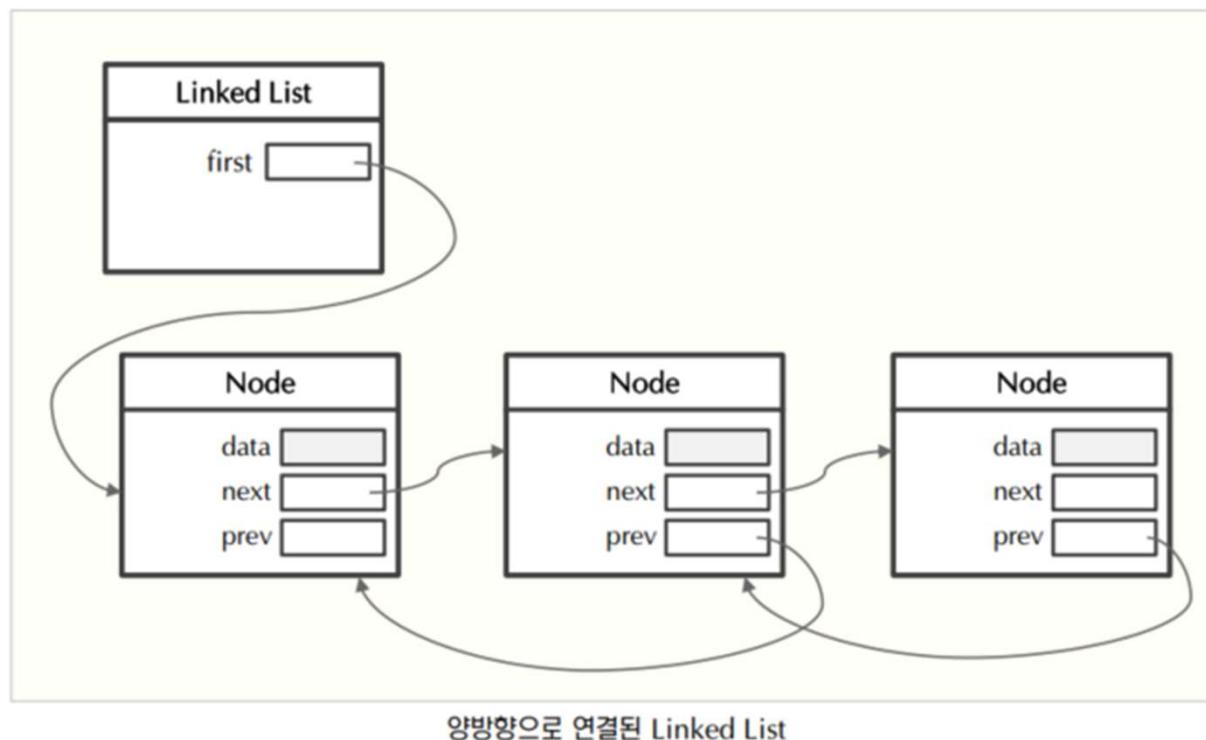
```
class GrannySmith extends Apple {  
}  
  
class Gala extends Apple {  
}  
  
class Fuji extends Apple {  
}  
  
class Braeburn extends Apple {  
}
```

```
public class GenericsAndUpcasting {  
    public static void main(String[] args) {  
        List<Apple> apples = new ArrayList<Apple>();  
        // List<Apple> apples = new LinkedList<Apple>();  
        apples.add(new GrannySmith());  
        apples.add(new Gala());  
        apples.add(new Fuji());  
        apples.add(new Braeburn());  
        for (Apple c : apples) {  
            System.out.println(c);  
        }  
    }  
}
```

GenericAndUpcasting.java

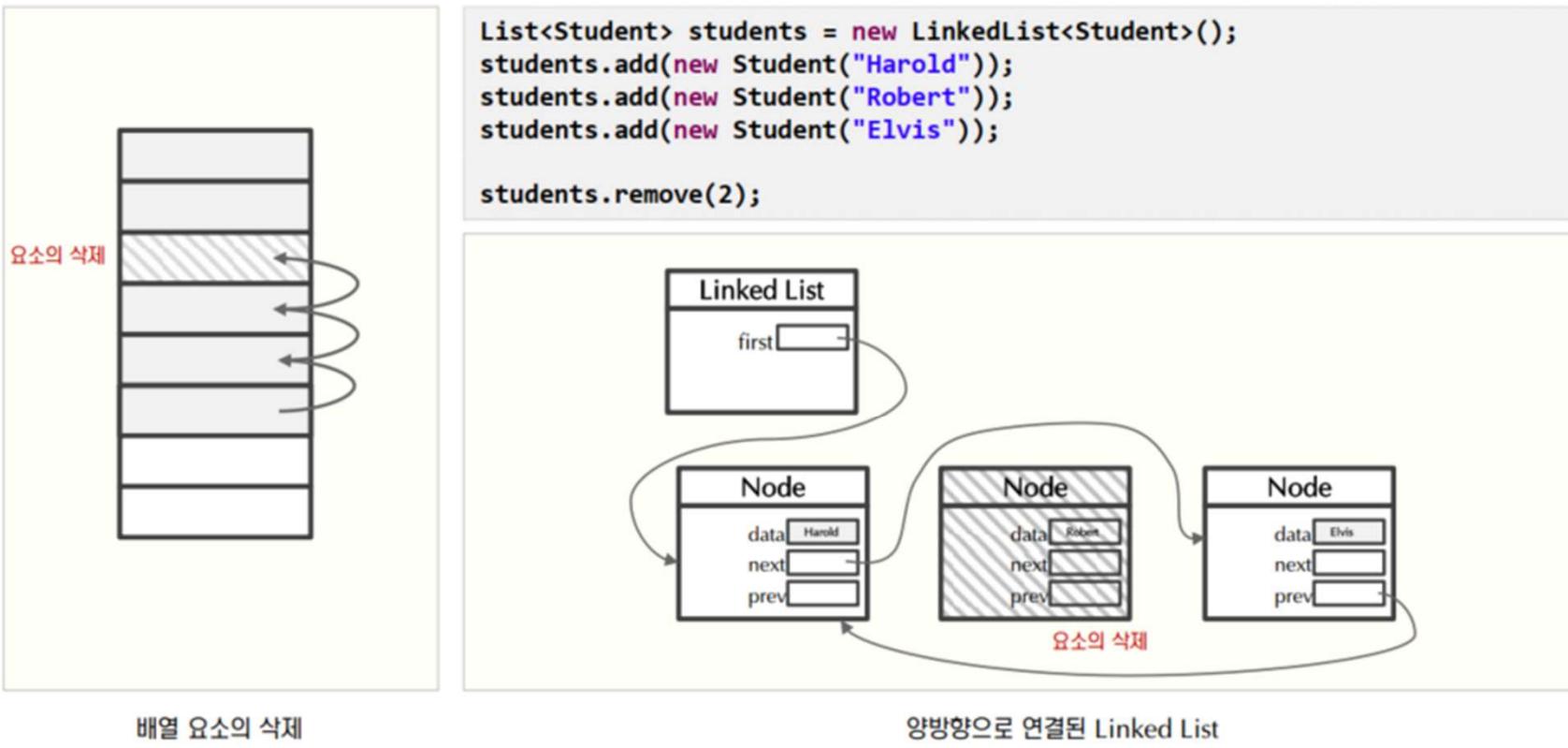
LinkedList<E> class

- ✓ LinkedList 는 요소들을 노드로 표현하고 각 노드를 서로 연결하여 리스트를 구성합니다.
- ✓ 각 노드는 내부적으로 이전 노드와 다음 노드에 대한 참조 값을 가지고 있습니다.



LinkedList<E> class (Cont.)

- ✓ 배열 또는 ArrayList 는 리스트 중간에 있는 요소가 삭제되거나 추가될 때 오버헤드가 발생합니다.
- ✓ LinkedList 는 ArrayList 에서 발생할 수 있는 문제점을 해결합니다.
- ✓ LinkedList 는 중간 요소의 잦은 추가/삭제로 인한 오버헤더가 많은 곳에 적합합니다.



Map<E> interface

✓ 맵(Map)은 키와 값으로 이루어져 있습니다.

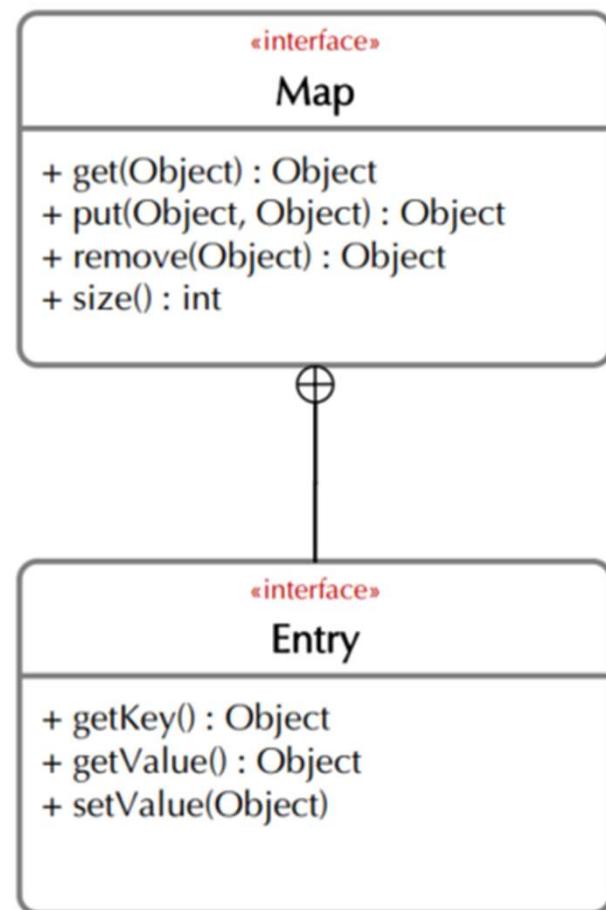
- get() 과 remove() 메소드에서 null 값이 리턴되면, 해당 키가 없음을 의미합니다.
- put() 메소드는 Map에 요소를 삽입하거나 수정합니다.
 - 새로운 값을 삽입할 경우 null을 리턴합니다.
 - 기존 값을 수정할 경우 기존 값을 리턴합니다.

✓ 엔트리(Entry)

- 키와 값을 가진 객체의 순서 쌍입니다.
- 키 컴포넌트는 getter (판독-전용) 접근으로 제한하고 있습니다.
- 값 컴포넌트는 getter(판독자), setter(설정자) 메소드를 정의합니다.

✓ 구현 클래스

- HashMap
- LinkedHashMap
- TreeMap



HashMap<E> class

✓ HashMap은 해시테이블을 기반으로 Map 인터페이스를 구현한 클래스입니다.

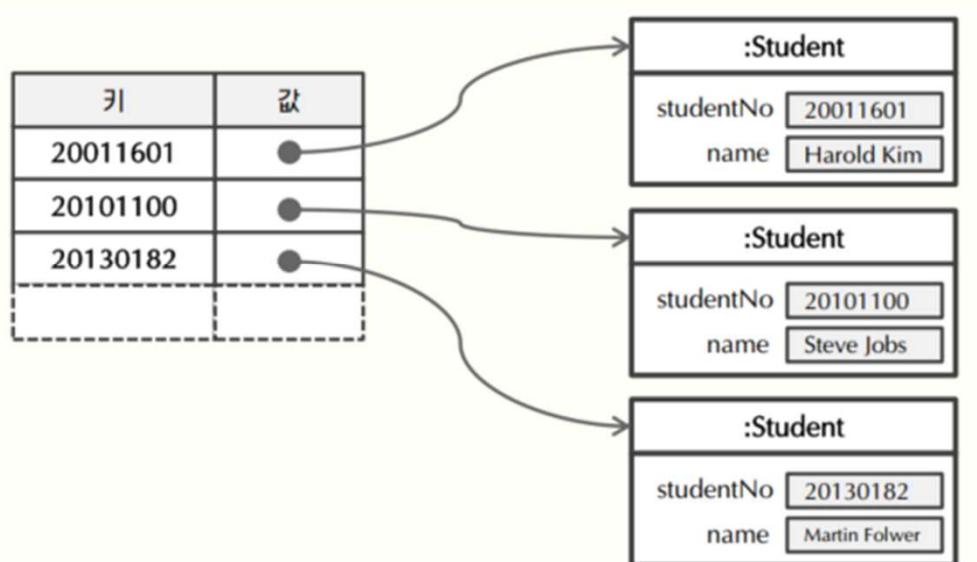
- HashMap은 데이터의 순서를 보장하지 않습니다.

```
Map<String, Student> studentsMap = new HashMap<String, Student>();  
  
studentsMap.put("20130182", new Student("20130182", "Martin Fowler"));  
studentsMap.put("20011601", new Student("20011601", "Harold Kim"));  
studentsMap.put("20101100", new Student("20101100", "Steve Jobs"));  
  
System.out.println(studentsMap);
```

HashMap.java

실행결과

```
{20011601=[studentNo:20011601,  
name:Harold Kim]  
,20130182=[studentNo:20130182,  
name:Martin Fowler]  
,20101100=[studentNo:20101100,  
name:Steve Jobs]}
```



Map은 키와 값의 쌍으로 데이터를 관리한다.

HashMap이 내부적으로 데이터를 관리하는 방식은 도식화된 것처럼 단순하지 않습니다.

해시테이블은 linked list의 배열로 구현되는데,
각각의 배열요소를 버킷이라고 합니다.
버킷에는 하나의 값이 담길 수도 있고 또 다른
버킷을 가질 수도 있습니다.

해시테이블의 기본의 자료구조를 사용할 때,
버킷에 값들이 균등하게 담길 수 있도록 해시 키를
구성하는 것이 핵심입니다.

LinkedHashMap<E> & TreeMap<E> class

✓ LinkedHashMap

- LinkedHashMap은 해시테이블과 연결 리스트를 기반으로 Map 인터페이스를 구현한 클래스입니다.
- LinkedHashMap은 추가된 순서를 보장합니다.

```
Map<String, Student> studentsMap = new LinkedHashMap<String, Student>();  
  
studentsMap.put("20130182", new Student("20130182", "Martin Fowler"));  
studentsMap.put("20011601", new Student("20011601", "Harold Kim"));  
studentsMap.put("20101100", new Student("20101100", "Steve Jobs"));  
  
System.out.println(studentsMap);
```

실행결과

```
{20130182=[studentNo:20130182,  
name:Martin Fowler],  
20011601=[studentNo:20011601,  
name:Harold Kim],  
20101100=[studentNo:20101100,  
name:Steve Jobs]}
```

✓ TreeMap

- TreeMap은 Red-black 트리 기반으로 Map 인터페이스를 구현한 클래스입니다.
- TreeMap은 키에 대한 정렬된 순서를 보장합니다.
- 키 정렬을 위하여 모든 키는 Comparable 인터페이스를 구현하고 있어야 합니다.

```
Map<String, Student> studentsMap = new TreeMap<String, Student>();  
  
studentsMap.put("20130182", new Student("20130182", "Martin Fowler"));  
studentsMap.put("20011601", new Student("20011601", "Harold Kim"));  
studentsMap.put("20101100", new Student("20101100", "Steve Jobs"));  
  
System.out.println(studentsMap);
```

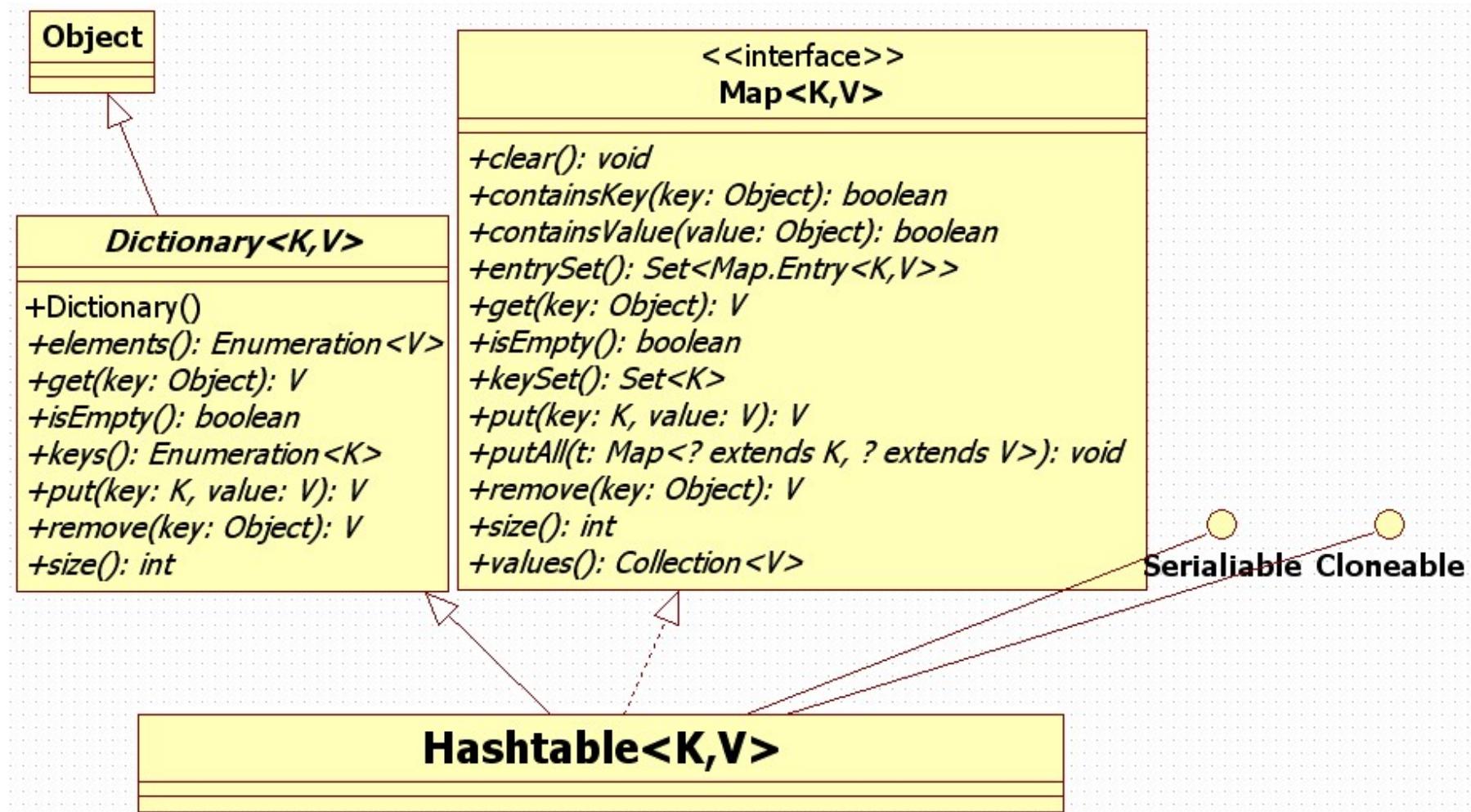
실행결과

```
{20011601=[studentNo:20011601,  
name:Harold Kim],  
20101100=[studentNo:20101100,  
name:Steve Jobs],  
20130182=[studentNo:20130182,  
name:Martin Fowler]}
```

Hashtable<K, V> class

- Implements a hashtable, which maps keys to values.
- To successfully store and retrieve objects, the objects used as keys must implement the **hashCode** method and the **equals** method.
- Generally, the default load factor (.75) offers a good tradeoff between time and space costs.
- Higher values decrease the space overhead but increase the time cost to look up an entry.

Hashtable<K, V> class (Cont.)



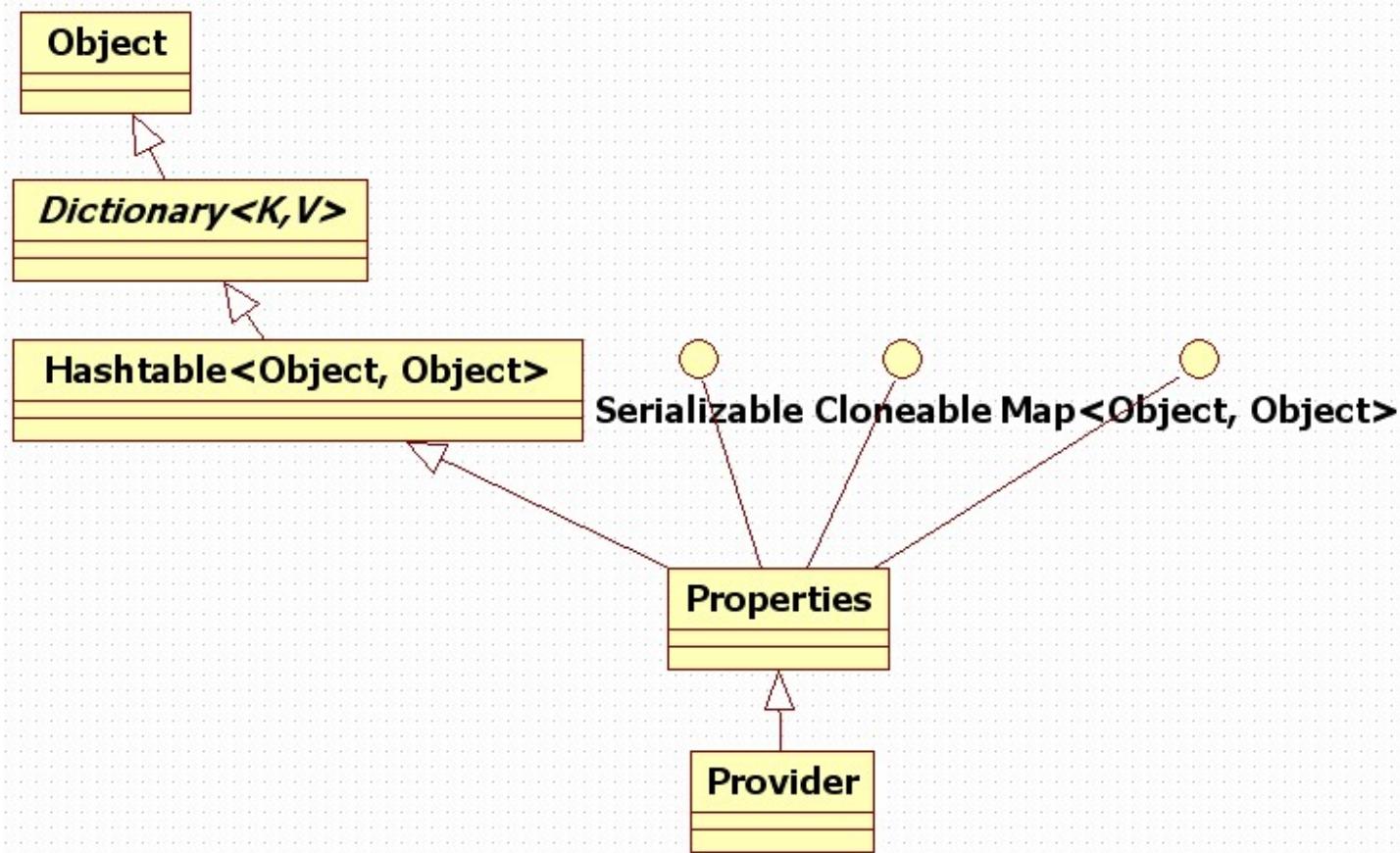
Hashtable<K, V> class (Cont.)

Hashtable<K,V>
<code>+Hashtable() +Hashtable(initialCapacity: int) +Hashtable(initialCapacity: int, loadFactor: float) +Hashtable(t: Map<? extends K, ? extends V>) +clear(): void +clone(): Object +contains(value: Object): boolean +containsKey(key: Object): boolean +containsValue(value: Object): boolean +elements(): Enumeration<V> +entrySet(): Set<Map.Entry<K,V>> +get(key: Object): V +isEmpty(): boolean +keys(): Enumeration<K> +keySet(): Set<K> +put(key: K, value: V): V +putAll(t: Map<? extends K, ? extends V>): void +remove(key: Object): V +size(): int +values(): Collection<V></code>

Properties class

- Represents a persistent set of properties.
- Each key and its corresponding value in the property list is a string.
- Inherits from **Hashtable**.
- The **setProperty** method should be used instead.

Properties class (Cont.)



Properties class (Cont.)

Properties
#defaults: Properties
+Properties()
+Properties(defaults: Properties)
+getProperty(key: String): String
+getProperty(key: String, defaultValue: String): String
+list(out : PrintStream): void
+list(out : PrintWriter): void
+load(inStream: InputStream): void
+loadFromXML(: InputStream): void
+propertyNames(): Enumeration<?>
+save(out : OutputStream, commects: String): void
+setProperty(key: String, value: String): Object
+store(out : OutputStream, commects: String): void
+storeToXML(os: OutputStream, commemt: String): void
+storeToXML(os: OutputStream, comment: String, encoding: String): void

Sample Code I

```
1 import java.util.*;
2 import static java.lang.System.out;
3
4 public class LinkedListDemo {
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         LinkedList<String> list = new LinkedList<String>();
8         list.add("Handphone");
9         list.add("Notebook");
10        list.add("Computer");
11        Iterator<String> it = list.iterator();
12        while(it.hasNext()){
13            String product = it.next();
14            out.println(product);
15        }
16        list.remove("Notebook");
17        out.println("-----");
18        it = list.iterator();
19        while(it.hasNext()){
20            String product = it.next();
21            out.println(product);
22        }
23    }
24 }
```

Handphone
Notebook
Computer

Handphone
Computer

Sample Code II

```
1 import java.util.*;
2 import static java.lang.System.out;
3
4 public class HashSetDemo {
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         HashSet<Number> set = new HashSet<Number>();
8         set.add(new Integer(5));
9         set.add(10);
10        set.add(4.5);
11        set.add(10);
12        Iterator<Number> it = set.iterator();
13        while(it.hasNext()){
14            out.println(it.next());
15        }
16    }
17 }
```

```
4 . 5
10
5
```

Sample Code III

```
1 import java.util.*;
2 import static java.lang.System.out;
3
4 public class HashSetDemo {
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         String [] array = {"Java", "EJB", "Java", "XML"};
8         HashSet<String> hs1 = new HashSet<String>();
9         HashSet<String> hs2 = new HashSet<String>();
10        for(String str : array){
11            if(!hs1.add(str)) hs2.add(str);
12        }
13        out.println("hs1 : " + hs1);
14        hs1.removeAll(hs2);
15        out.println("hs1 : " + hs1);
16        out.println("hs2 : " + hs2);
17    }
18 }
```

hs1 : [Java, XML, EJB]
hs1 : [XML, EJB]
hs2 : [Java]

Sample Code IV

```
1 import java.util.Vector;
2 import static java.lang.System.out;
3
4 public class VectorDemo {
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         Vector<String> v = new Vector<String>(2,5);
8         out.printf("Capacity = %d, Size = %d\n", v.capacity(), v.size());
9
10        v.add("Handphone");
11        v.add("Notebook");
12        v.add("Computer");
13
14        out.printf("Capacity = %d, Size = %d\n", v.capacity(), v.size());
15    }
16 }
```

Capacity = 2, Size = 0
Capacity = 7, Size = 3

Sample Code V

```
1 import java.util.Vector;
2 import static java.lang.System.out;
3
4 public class VectorDemo {
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         Vector<String> v = new Vector<String>(2,5);
8         v.add("Sonata");
9         v.add("Tico");
10        v.addElement("Canival");
11        for(String str : v) out.printf("%s, ", str);
12        out.printf("\n");
13        String search = "Tico";
14        int idx = v.indexOf(search);
15        if(idx != -1) out.printf("%s is %dth item.\n", search, idx);
16        else out.printf("%s is not found\n", search);
17        out.println("=====Before Deletion=====");
18        out.printf("Capacity = %d, Size = %d\n", v.capacity(), v.size());
19        String del = "Canival";
20        if(v.contains(del)) v.remove(del);
21        out.println("=====After Deletion=====");
22        out.printf("Capacity = %d, Size = %d\n", v.capacity(), v.size());
23        v.trimToSize();
24        out.println("=====After Trimming=====");
25        out.printf("Capacity = %d, Size = %d\n", v.capacity(), v.size());
26    }
27 }
```

```
Sonata, Tico, Canival,
Tico is 1th item.
=====Before Deletion=====
Capacity = 7, Size = 3
=====After Deletion=====
Capacity = 7, Size = 2
=====After Trimming=====
Capacity = 2, Size = 2
```

Sample Code VI

```
1 import static java.lang.System.out;
2
3 public class ArraysDemo {
4     public static void main(String[] args) {
5         // TODO Auto-generated method stub
6         String [] array = {"Sonata", "Tico", "Canival"};
7         java.util.Arrays.fill(array, "Java");
8         for(String str : array)
9             out.printf("%s, ", str);
10        out.printf("\n");
11        java.util.Arrays.fill(array, 1,2, "XML");
12        for(String str : array)
13            out.printf("%s, ", str);
14            out.printf("\n");
15    }
16 }
```

Java, Java, Java,
Java, XML, Java,

Sample Code VII

```
1 import java.util.*;
2 import static java.lang.System.out;
3
4 public class PropertiesDemo {
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         Properties prop = System.getProperties();
8         Enumeration<Object> keys = prop.keys();
9         while(keys.hasMoreElements()){
10             String key = keys.nextElement().toString();
11             out.printf("%s = %s\n", key, System.getProperty(key));
12         }
13     }
14 }
```