JAVA 프로그래밍

Chapter 13 컬렉션 프레임워크

목차

- 자료구조
- 자료구조 클래스들
- 컬렉션 프레임워크
- Collection과 Map

자료구조

❖ 자료구조

- 데이터를 효율적으로 사용할 수 있도록 만들어진 구조.
- 데이터 추가, 삭제, 검색의 효율성을 제공

❖ 자료구조 종류

- 리스트(List) : 데이터를 1차원으로 늘어놓은 형태의 자료구조.
- - 배열리스트, 연결리스트
- 스택(Stack) : 마지막에 넣은 데이터부터 순서대로 꺼낼 수 있는 자료구조.
- 큐(Queue) : 입력된 순서대로 데이터를 꺼낼 수 있는 자료구조.
- 해쉬테이블(Hash table) : 키(key)를 이용해서 데이터를 검색하는 자료구조.
- 집합(Set) : 키(key)를 이용해서 데이터를 검색하는 구조이긴 하나 데이터를 중복 저장하지 않는 자료구조.

자료구조 클래스들

자료구조	클래스 이름
리스트	ArrayList LinkedList Vector
스택	Stack LinkedList
큐	Queue LinkedList
해쉬 테이블	HashMap Hashtable
집합	HashSet

컬렉션 프레임워크

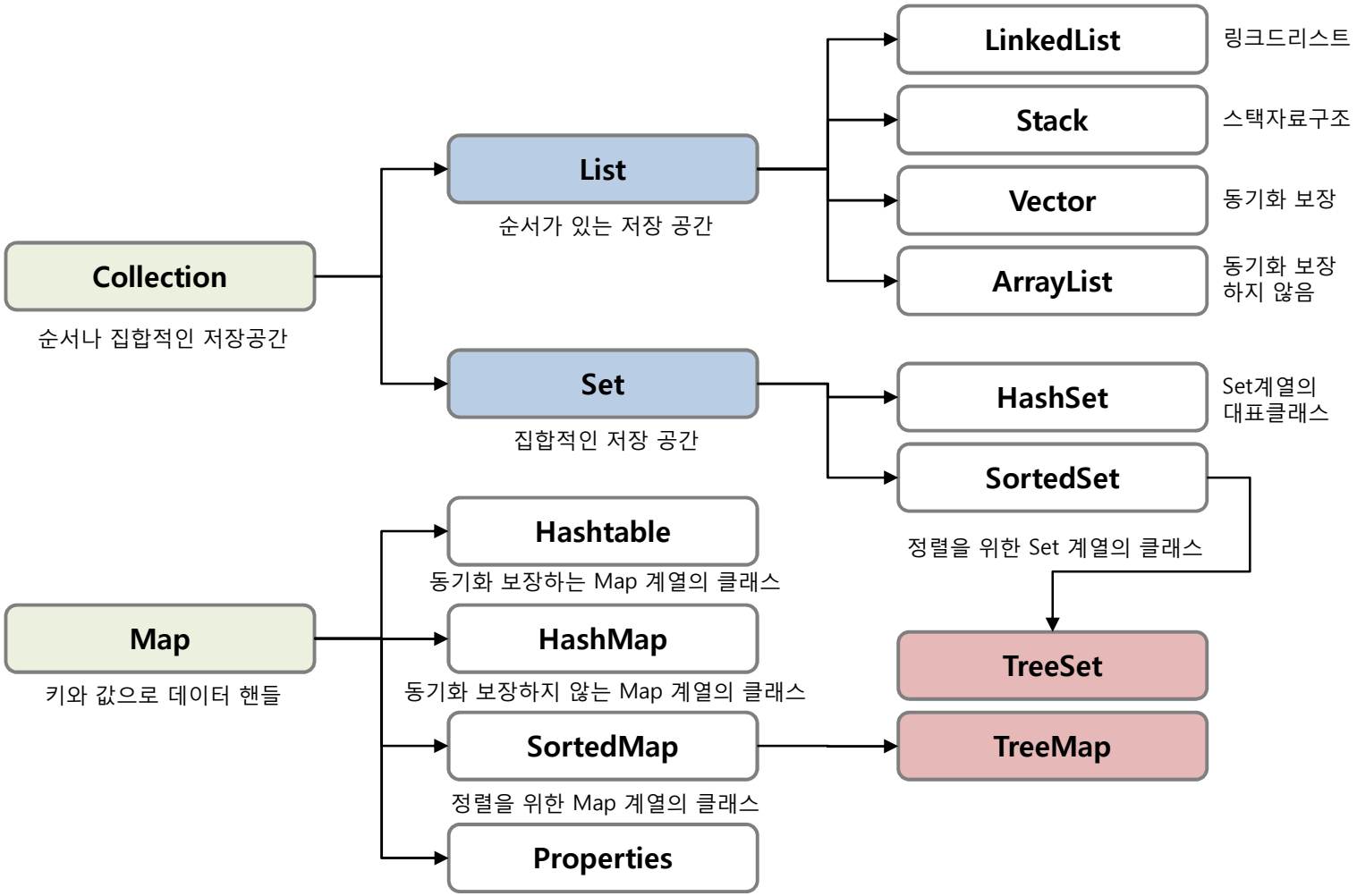
- 컬렉션 프레임워크
  - 다수의 데이터를 쉽게 처리할 수 있는 표준화된 방법을 제공하는 클래스들의 집합.
  - 배열의 단점을 보완한 배열의 발전된 모델.
  - 데이터 군을 저장하는 클래스들을 표준화한 설계이다.
  - 컬렉션(Collection)과 맵(Map) 계열의 클래스

- 배열과 컬렉션이나 맵계열의 클래스와의 차이점
  - 배열은 크기를 동적으로 늘릴 수 없지만 컬렉션이나 맵 계열의 클래스는 동적으로 메모리를 확장할 수 있다.

- Collection류와 Map류의 클래스
  - 자료구조적인 기능의 클래스

- Collection과 Map 인터페이스
  - Collection과 Map은 인터페이스이기 때문에 메서드의 프로토타입만 존재한다.
  - Collection을 구현해서 사용하면 집합적인 저장공간으로서의 기능을 구현한 클래스가 된다.
  - Map을 구현해서 사용하면 검색적인 저장공간으로서의 기능을 구현한 클래스가 된다.

Collection과 Map의 상속구조



Collection과 Map

❖ 컬렉션 클래스들의 특징

인터페이스 분류		특징	구현 클래스
Collection	List	- 순서를 유지하고 저장 - 중복 저장 가능	ArrayList, Vector, LinkedList, Stack
	Set	- 순서를 유지하지 않고 저장 - 중복 저장 안 됨	HashSet, TreeSet
Map		-키와 값의 쌍으로 저장 - 키는 중복 저장 안 됨	HashMap, Hashtable, TreeMap, Properties

Collection 인터페이스

- List와 Set 인터페이스의 많은 공통된 부분을 Collection 인터페이스에 정의하고, 두 인터페이스는 그것을 상속받는다.
- 컬렉션을 다루는데 가장 기본적인 동작들을 정의하고 그 메서드를 제공한다.
- 이것을 구현한 클래스들은 모두 집합적인 저장공간으로서의 기능을 가진다.

기능	메서드	설명
객체 추가	boolean add(E e)	주어진 객체를 추가
객체 검색	boolean contains(Object o)	주어진 객체가 저장되어 있는지 여부
	isEmpty()	컬렉션이 비어 있는지 여부
	int size()	포함된 객체의 수를 리턴
	T[] toArray()	포함된 모든 객체들을 배열 형태로 리턴
	Iterator<E> iterator()	해당 컬렉션의 반복자(Iterator)를 반환함.
객체 삭제	void clear()	해당 컬렉션의 모든 요소를 제거함
	boolean remove(Object o)	주어진 객체를 제거

List 컬렉션

- List 컬렉션은 객체를 일렬로 늘어놓은 구조를 가진다.
- 인덱스로 객체를 검색, 삭제할 수 있는 기능을 제공한다.
- ArrayList, Vector, LinkedList 등이 있다.

기능	메서드	설명
객체 추가	boolean add(E e)	주어진 객체를 맨 끝에 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()	저장되어 있는 전체 객체 수를 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

ArrayList

- ArrayList는 List 인터페이스 구현 클래스이다.
- 객체들을 순차적으로 저장하고 중복 저장이 가능하다.
- List<E> list = new ArrayList<E>();

```
// List 객체 생성
List<String> list = new ArrayList<String>();
// 객체 저장
list.add(“홍길동”);
list.add(“김길동”);
list.add(“박길동”);
list.add(“이길동”);

System.out.println(“총 객체 수 : ” + set.size());

String name = list.get(2);
System.out.println(name);

list.remove(2);
```

Vector

- Vector는 ArrayList와 동일한 내부 구조를 가진다.
- ArrayList와 다른 점은 Vector는 동기화된(synchronized) 메서드로 구성되어 멀티 스레드가 동시에 이 메서드들을 실행할 수 없다.
- List<E> list = new Vector<E>();

```
// List 객체 생성
List<String> list = new Vector<String>();
// 객체 저장
list.add(“홍길동”);
list.add(“김길동”);
list.add(“박길동”);
list.add(“이길동”);

System.out.println(“총 객체 수 : ” + set.size());

String name = list.get(2);
System.out.println(name);

list.remove(2);
```

AutoBoxing(자바 5.0)

❖ 자바 1.4까지

- Vector v = new Vector(); //객체 생성
- v.addElement(new Integer(100)); //Wrapper 클래스의 사용
- v.addElement(new Integer(200)); //Wrapper 클래스의 사용
- Integer t0 = (Integer)v.elementAt(0);
- Integer t1 = (Integer)v.elementAt(1);

❖ 자바 5.0 이후

- Vector<Integer> v = new Vector<Integer>();
- v.addElement(100); //AutoBoxing 발생
- v.addElement(200); //AutoBoxing 발생
- int a0 = v.elementAt(0);//AutoUnBoxing 발생
- int a1 = v.elementAt(1);//AutoUnBoxing 발생

자바 5.0 이상에서는 컴파일러가 내부에서 자동으로 Wrapper 클래스의 객체로 변경시켜 준다.

자바 5.0 이상에서는 컴파일러가 내부에서 자동으로 객체를 적절한 상수값으로 변경시켜 준다.

Set 컬렉션

- List 컬렉션은 저장 순서를 유지하지만, Set 컬렉션은 저장 순서가 유지되지 않는다.
- 객체를 중복 저장할 수 없다. (수학의 집합과 유사)
- HashSet, LinkedHashSet, TreeSet 등이 있다.

기능	메서드	설명
객체 추가	boolean add(E e)	주어진 객체를 저장, 객체가 성공적으로 저장되면 true를 리턴하고 중복 객체면 false를 리턴
객체 검색	boolean contains(Object o)	주어진 객체가 저장되어 있는지 여부
	Iterator<E> iterator()	저장된 객체를 한 번씩 가져오는 반복자 리턴
	isEmpty()	컬렉션이 비어 있는지 조사
	int size()	저장되어 있는 전체 객체 수를 리턴
객체 삭제	void clear()	저장된 모든 객체를 삭제
	boolean remove(Object o)	주어진 객체를 삭제

HashSet

- HashSet은 Set 인터페이스 구현 클래스이다.
- 객체들을 순서 없이 저장하고 중복 저장하지 않는다.
- Set<E> set = new HashSet<E>();

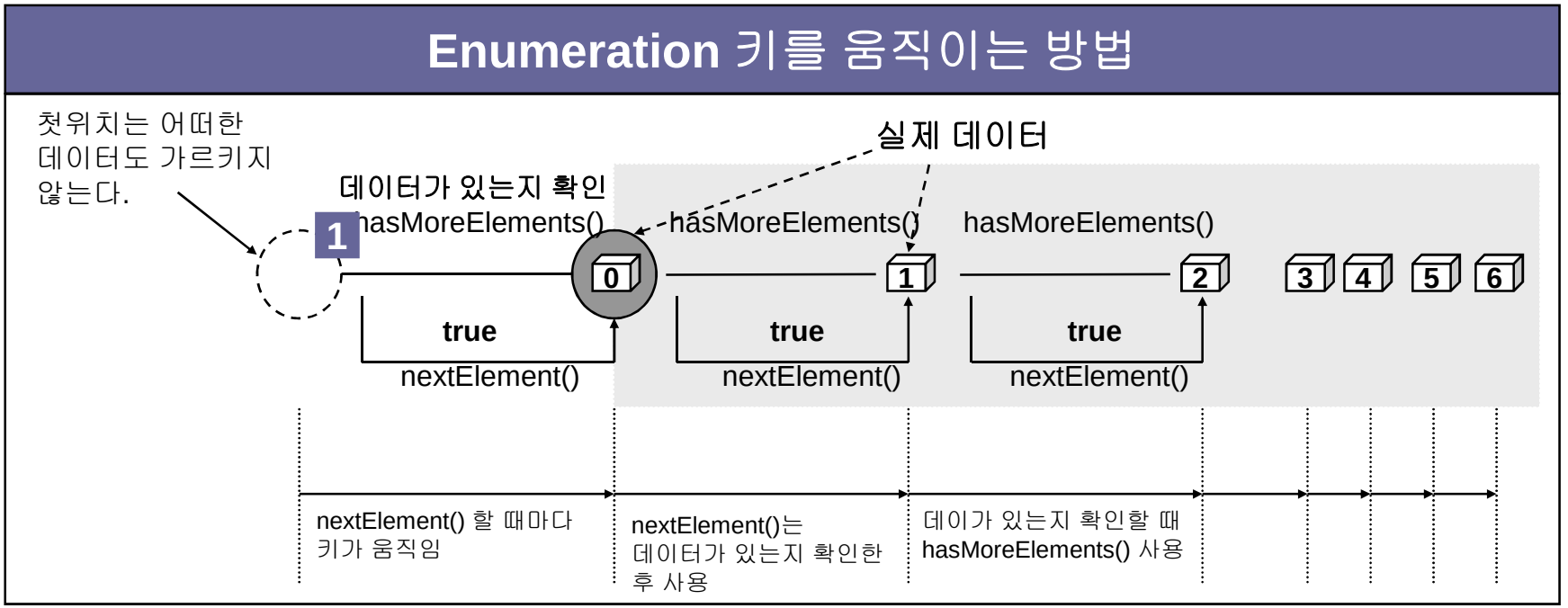
```
// Set 객체 생성
Set<String> set = new HashSet<String>();
// 객체 저장
set.add("홍길동");
set.add("김길동");
set.add("박길동");
set.add("이길동");

System.out.println("총 Entry 수 : " + set.size());
set.remove("홍길동");
set.contains("김길동");

Iterator<String> iter = set.iterator();
while(iter.hasNext()){
    String temp = iter.next();
    System.out.print(temp + ", ");
}
```

컬렉션 검색

- 컬렉션 검색을 위한 인터페이스
  - Enumeration과 Iterator
  - Advanaced for(foreach)
- Enumeration과 Iterator의 특징
  - 데이터의 마지막에 상관하지 않고 모든 데이터에 접근할 수 있다.
  - Iterator는 자바 1.2에서 제공, Enumeration의 대체용
- 컬렉션 검색을 위한 제어문(자바 5.0)
  - Advanced for(일명 for each)
  - 자바 5.0에 새롭게 추가된 컬렉션을 위한 반복 제어문
  - 데이터의 마지막에 상관하지 않고 검색하기 위한 제어문

```
String[] ar = new String[]{"안녕1", "안녕2", "안녕3"};
for (String tmp : ar  ){
    System.out.println(  tmp  );
}
```

Vector에서의 Enumeration




```
// Vector 생성과 데어터 삽입
Vector<String> v = new Vector<String>();
v.addElement(new String("망아지"));
v.addElement(new String("송아지"));
v.addElement(new String("강아지"));
v.addElement(new String("병아리"));
```

```
// Enumeration 얻어내기
Enumeration<String> en = v.elements();
Enumeration을 이용해서 전체 데이터 추출
while(en.hasMoreElements()){
    String temp = en.nextElement();
    System.out.println(temp);
}
```

Map 컬렉션

- Collection과 달리 Map은 검색적인 개념을 담고 있는 인터페이스.
- Map은 키(key)와 값(value)으로 구성된 Entry 객체를 저장하는 구조를 가진다.
- 동일한 키로 값을 저장하면 기존의 값은 없어지고 새로운 값으로 대치된다.

기능	메서드	설명
객체 추가	V put(K key, V value)	주어진 키와 값을 추가, 저장되면 값을 리턴
객체 검색	boolean containsKey(Object key)	주어진 키가 있는지 여부
	boolean containsValue(Object value)	주어진 값이 있는지 여부
	Set<Map.Entry<K,V>> entrySet()	키와 값의 쌍으로 구성된 모든 Map.Entry 객체를 Set에 담아서 리턴
	V get(Object key)	주어진 키의 값을 리턴
	boolean isEmpty()	컬렉션이 비어 있는지 여부
	Set<K> keySet()	모든 키를 Set 객체에 담아서 리턴
	int size()	저장된 키의 총 수를 리턴
	Collection<V> values()	저장된 모든 값을 Collection에 담아서 리턴
객체 삭제	void clear()	모든 Map.Entry(키와 값)를 제거
	V remove(Object key)	키(key)를 이용해서 데이터를 제거

HashMap 클래스

- HashMap은 Map 인터페이스를 구현한 대표적인 Map 컬렉션이다.
- HashMap의 키로 사용할 객체는 hashCode()와 equals() 메서드를 재정의해서 동등 객체가 될 조건을 정해야 한다.
- Map<K, V> map = new HashMap<K, V>();

```
// Map 객체 생성
Map<String, Integer> map = new HashMap<String, Integer>();
// 객체 저장
map.put("홍길동", 90);
map.put("김길동", 80);
map.put("박길동", 70);
map.put("이길동", 60);

System.out.println("총 Entry 수 : " + map.size());

System.out.println("홍길동 : " + map.get("홍길동"));
```

Hashtable 클래스

- Hashtable 은 HashMap과 동일한 내부 구조를 가지고 있다.
- Hashtable의 키로 사용할 객체는 hashCode()와 equals() 메서드를 재정의해서 동등 객체가 될 조건을 정해야 한다.
- HashMap과의 차이점은 Hashtable은 동기화된(synchronized) 메서드로 구성되어 있어 멀티 스레드가 동시에 이 메서드를 실행할 수 없다.(thread safe)
- Map<K, V> map = new Hashtable<K, V>();

```
// Map 객체 생성
Map<String, Integer> map = new Hashtable<String, Integer>();
// 객체 저장
map.put(“홍길동”, 90);
map.put(“김길동”, 80);
map.put(“박길동”, 70);
map.put(“이길동”, 60);

System.out.println(“총 Entry 수 : ” + map.size());

System.out.println(“홍길동 : ” + map.get(“홍길동”));
```

Properties 클래스

❖ Properties 클래스

- Properties클래스는 말 그대로 속성들을 모아서 하나의 객체로 만들기 위해 제공되는 클래스이다
- Properties는 키와 값을 String 타입으로 제한한 컬렉션이다.
- 어플리케이션의 옵션 정보, 데이터베이스 연결 정보, 국제화(다국어) 정보 등을 저장한 *.properties 파일을 읽을 때 주로 사용한다.

❖ Properties 클래스의 필요성

- 일반적으로 사용하는 각자의 컴퓨터도 마찬가지이며 그 안에서 구동 되는 모든 프로그램들도 각각의 속성들을 가지고 있다.
- 컴퓨터가 켜질 때 또는 프로그램이 시작되기 전에 여러 개의 속성들 중 원하는 속성들을 미리 인식되게 하여 전반적인 실행 환경을 조율하고 보다 신속한 처리속도를 가져오는데 목적을 두고 있다.

Properties

❖ Properties 클래스의 주요 메서드

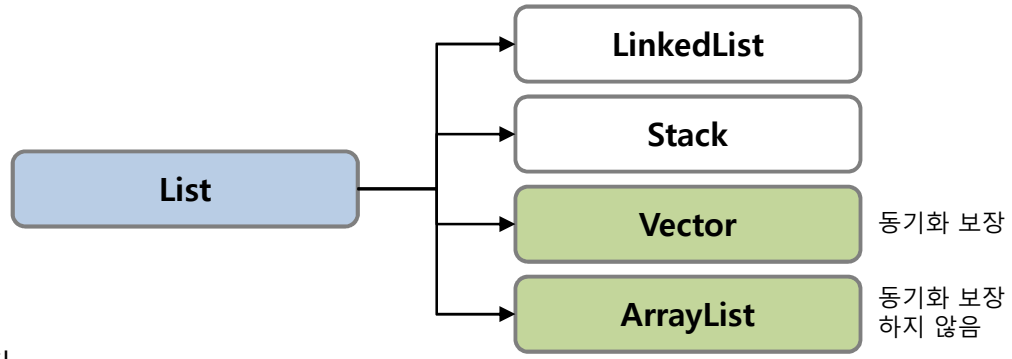
메소드		기능
String	getProperty(String key)	인자로 전달된 key를 가지는 속성값을 찾아 반환한다.
void	list(PrintWriter out)	인자로 전달된 출력 스트림을 통해 속성 목록들을 출력한다.
void	load(InputStream inStream)	인자로 전달된 입력 스트림으로부터 키와 요소가 한 쌍으로 구성된 속성 목록들을 읽어들여 현 Properties 객체에 저장한다.
Enumeration<?>	propertyName()	속성 목록에 있는 모든 속성의 key값들을 열거형 객체로 반환한다.
Object	setProperty(String key, String value)	현 Properties 객체의 속성 목록에 인자로 전달된 key 와 value를 한 쌍으로 구성되어 저장한다. 내부적으로는 Hashtable의 put()메소드가 호출된다.
void	store(OutputStream out, String comments)	모든 속성들을 load() 메소드를 사용해 Properties 테이블에 로드하고 적절한 포맷과 인자로 전달된 출력 스트림을 통해 출력한다.

컬렉션 클래스 비교

Vector와 ArrayList의 비교

❖ Vector와 ArrayList의 공통점

- 순서가 있는 Collection이다.
- List 인터페이스를 구현하고 있다.
- 데이터를 중복해서 포함할 수 있다.



❖ Vector와 ArrayList의 차이점

- Vector는 자동으로 동기화를 보장해준다.
- ArrayList는 동기화를 보장해주지 않는다.

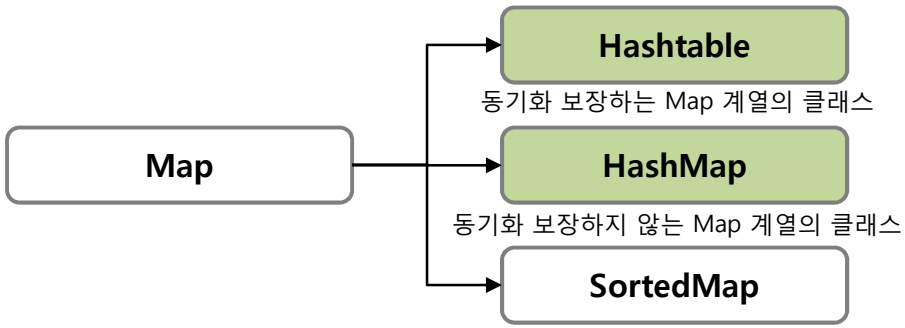
❖ ArrayList의 동기화 지원 방법

- List list = Collections.synchronizedList(new ArrayList(...));

❖ Vector와 ArrayList의 기능

- ArrayList는 배열에 동적 메모리 증가 기능을 구현한 클래스이다.
- Vector는 ArrayList에 동기화가 보장되도록 최적화한 클래스이다.

Hashtable, HashMap의 비교



- ❖ Hashtable, HashMap의 공통점
  - 내부적으로 모두 Hash 기법을 이용한다.
  - Map 인터페이스를 구현하고 있다.
  - 키(Key)와 값(Value)을 이용해서 데이터를 관리한다.
- ❖ Hashtable과 HashMap
  - Hashtable은 동기화가 보장 되지만 HashMap은 동기화가 보장되지 않는다.
- ❖ HashMap의 동기화 지원 방법
  - Map m = Collections.synchronizedMap(new HashMap(...));
- ❖ Hashtable, HashMap과 HashSet과의 관계
  - Hashtable과 HashMap은 둘다 Map 인터페이스를 구현하고 있다.
  - HashSet은 내부적으로 Hash기법을 사용하지만 Set 인터페이스를 구현하고 있다.

ArrayList와 HashSet의 비교

❖ ArrayList와 HashSet의 공통점

- 객체의 저장공간이다.
- 동기화를 보장하지 않는다.

❖ ArrayList와 HashSet의 차이점

- ArrayList

  List 인터페이스를 구현하는 클래스이다.

  순서의 개념이 있다.

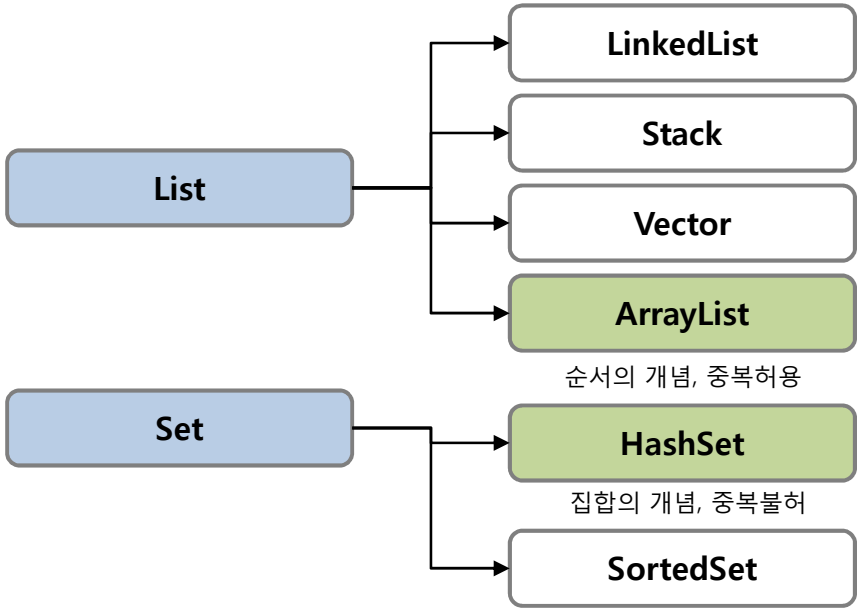
  데이터의 중복을 허용한다.

- HashSet

  Set 인터페이스를 구현하는 클래스이다.

  순서의 개념이 아니라 집합의 개념이 있다.

  데이터의 중복을 허용하지 않는다.



❖ ArrayList와 HashSet의 기능

- 순서가 중요한 의미를 가진다면 ArrayList를, 순서는 중요하지 않고 데이터가 중복되지 않기를 원한다면 HashSet을 이용하면 된다.