Java 프로그래밍 길잡이

✓ 원리를 알면 IT가 맛있다

Java Programming for Beginners

chapter 10.

다중 데이터 처리

❑ 학습목표

- 컬렉션 자료구조를 이용한 데이터 관리.
- HashSet자료구조를 이용한 데이터 관리.
- ArrayList 자료구조를 이용한 데이터 관리.
- Map 자료구조를 이용한 데이터 관리.
- 제네릭(generic)를 이용한 컬렉션 사용법.
- Iterator와 Enumeration 사용법

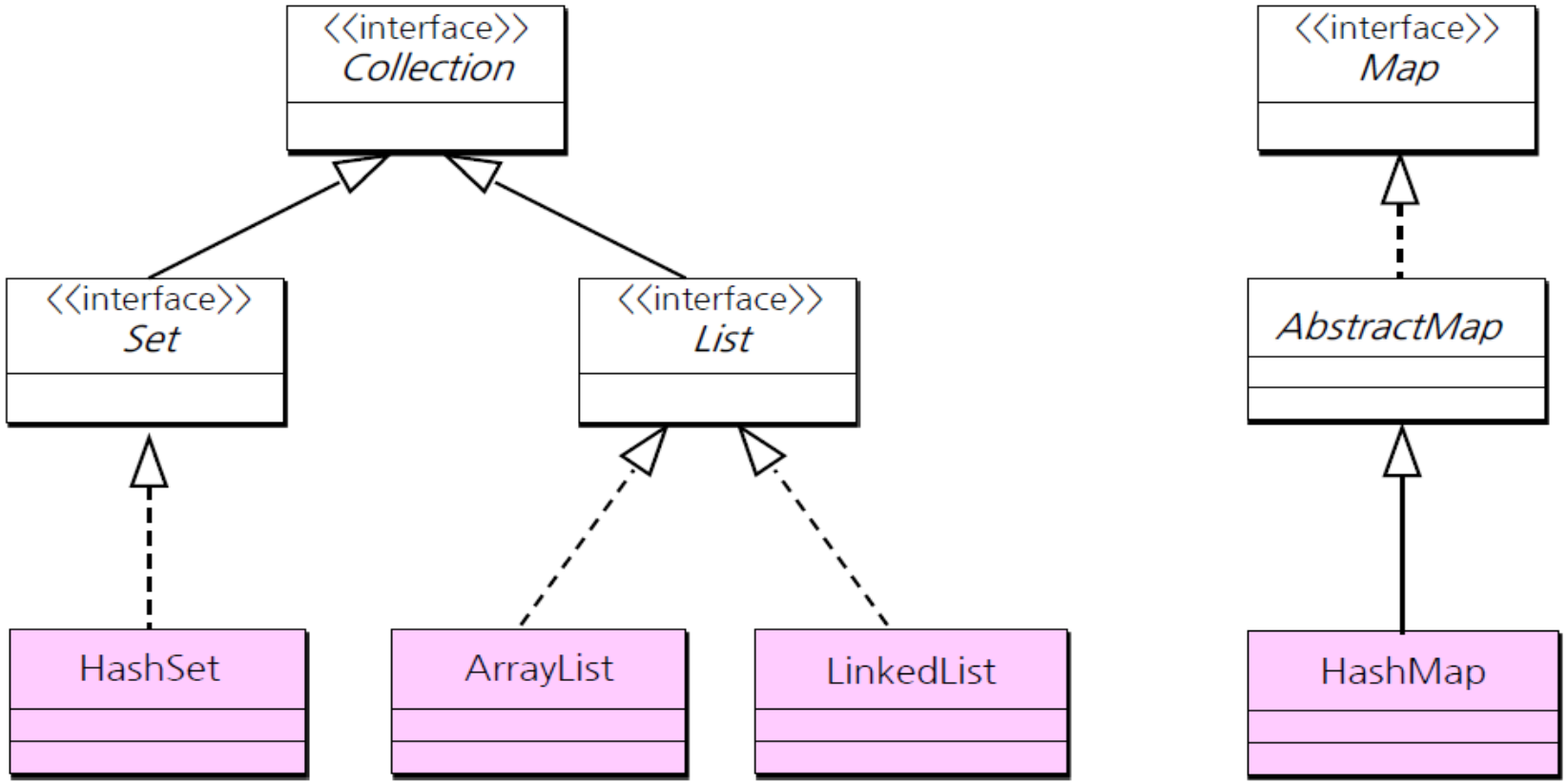
❑ 컬렉션 API

- 컬렉션 API 이란?
  - 다수의 데이터를 쉽게 처리할 수 있는표준화 된 방법을 제공하는 클래스들.
  - 크게 List, Set, Map 의 3가지 타입 API를 제공.

- 컬렉션 특징
  - 객체만 저장 가능하다.
  - 객체를 저장할 때마다 자동으로 크기가 늘어난다.
  - **저장된 객체를 삭제, 수정**이 가능하고 삽입도 가능하다.
  - **Set 계열 :** 순서가 없기 때문에 중복이 허용되지 않는다.
  - **List 계열: 순서가 있기 때문에 중복이 허용된다.**
  - Map 계열: 키와 값의 쌍으로 저장된다.

컬렉션 API

- 컬렉션 API



- 컬렉션 API

❖ Collection 인터페이스

Method	설명
boolean add(Object o) boolean addAll(Collection c)	지정된 객체 또는 Collection의 객체들을 Collection에 추가
void clear()	Collection의 모든 객체를 삭제
boolean isEmpty()	Collection이 비어있는지 확인
int size()	Collection에 저장된 객체의 개수를 반환
Object[] toArray()	Collection에 저장된 객체를 배열로 반환

❖ Map 인터페이스

Method	설명
Object put(Object key, Object value) void putAll(Map m)	Key객체를 id로 value객체를 저장 Map의 모든 key-value를 추가
void clear()	Map의 모든 객체를 삭제
boolean isEmpty()	Map이 비어있는지 확인
int size()	Map에 저장된 객체의 개수를 반환
Set keySet()	Map에 저장된 모든 Key객체를 반환

❑ 컬렉션 API - Set 계열

- Set 계열

❖ HashSet

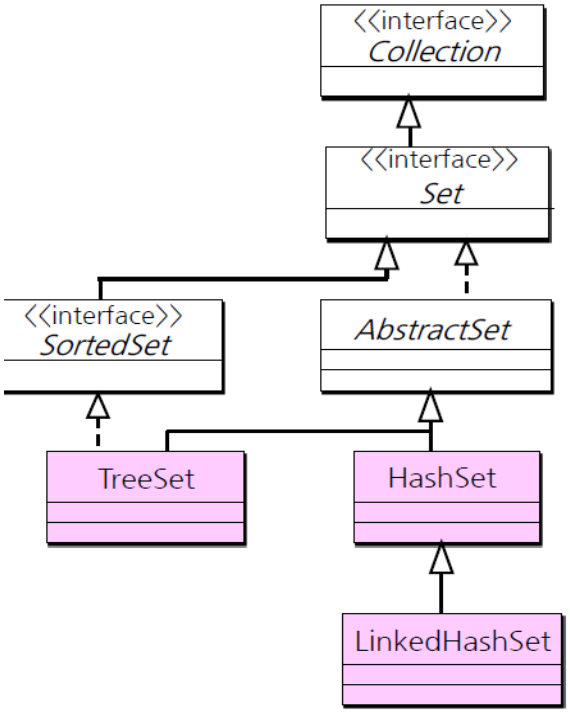
- Set에 객체를 저장하는데 Hash를 사용하여 처리 속도가 빠르다.

❖ LinkedHashSet

- HashSet과 거의 같다. 차이점은 Set에 추가되는 순서를 유지한다는 점

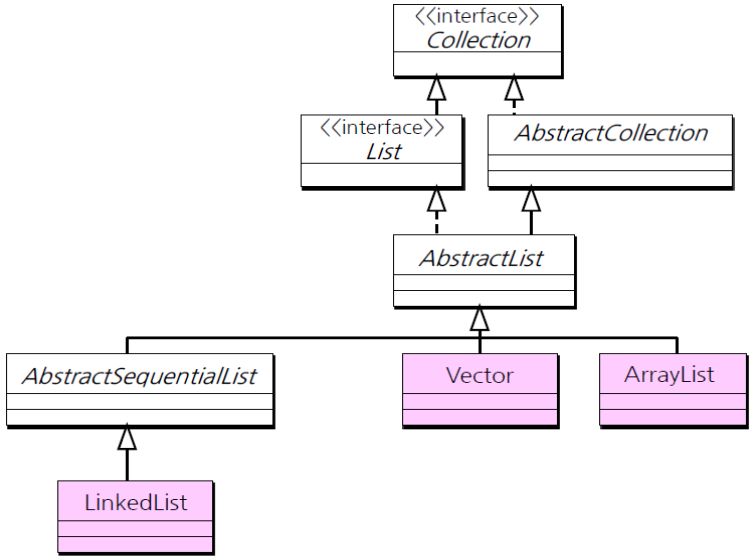
❖ TreeSet

- 객체의 Hash값에 의한 오름차순의 정렬 유지



컬렉션 API - List 계열

- List 계열



❖ List

- List의 요소에는 순서를 가진다.

❖ ArrayList

- List에서 객체를 얻어내는데 효율적
- 동기화 (Synchronization) 를 제공하지 않는다.

❖ LinkedList

- List에서 앞뒤의 데이터를 삽입하거나 삭제하는데 효율적이다.
- 동기화를 제공하지 않는다.

❖ Vector

- 기본적으로 ArrayList와 동등하지만 Vector에서는 동기화를 제공한다.
- 그래서 List객체들 중에서, 가장 성능이 좋지 않다.

- Map 계열

❖ HashMap

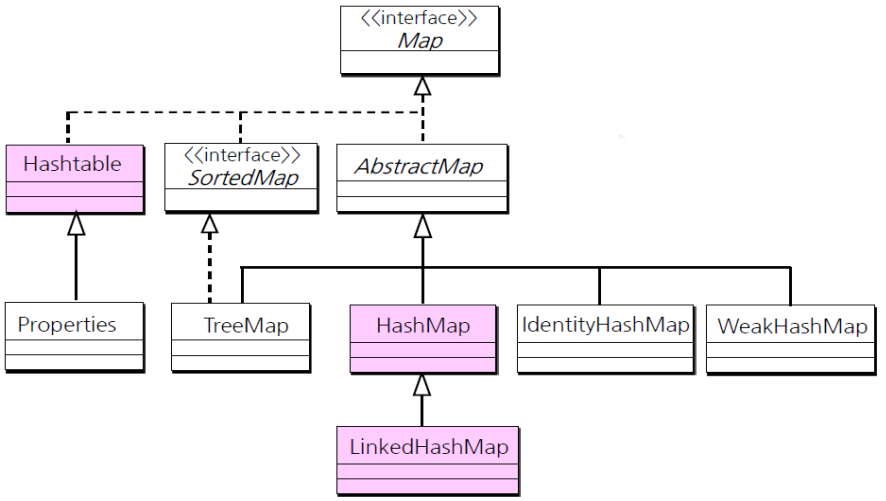
- Map에 키를 저장하는데 hash를 사용하여 성능이 좋다.
- 저장되는 순서가 유지 되지 않는다.
- 오직 하나의 null키를 가질 수 있다.

❖ LinkedHashMap

- HashMap과 거의 같다. 차이점은 Map에 추가되는 순서를 유지한다는 점

❖ HashTable

- 동기화(Synchronization)를 제공한다.
- null키와 null 값을 저장할 수 없다.



❑ 제네릭스 [Generics]

❖ 제네릭스의 특징

- Collection 선언 시에 < > 안에 매개변수 타입을 선언, 컴파일러에게 사용 타입을 전달
- 캐스팅이 필요 없고, 보다 안전한 코드를 작성 가능
- 메소드가 받아들일 수 있는 타입을 제한함으로서 에러검사, 타입검사 생략
- java.util 패키지의 컬렉션 클래스들은 기본적으로 제네릭스를 지원
- 매개변수 타입에는 primitive 타입을 사용할 수 없음

```
List intList = new ArrayList();
list.add("String type");

List<Integer> intList = new ArrayList<Integer>();
list.add("String type");
```

int 타입을 저장하는 List에 String을 저장해도 문제가 안 생긴다.!

컴파일 시 에러

❖ List에서의 제네릭스 사용

- 제네릭스는 타입을 제한하여 List가 특정 타입만을 받도록 함
- 아래 예제의 strList 객체는 String 객체 만을 저장 가능함

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

- Integer형의 제네릭스 사용 예제

```
List<Integer> intList = new ArrayList<Integer>( );
```

❖ Map에서의 제네릭스 사용

- java.util.Map은 키(Object) 와 값(Object)을 갖는 collection
- 값을 넣고 빼는데 있어서 형 변환이 필요 없음
- 아래 예제에서 map은 키와 값을 모두 Integer 타입으로 선언

```
Map<Integer, Integer> map = new HashMap<Integer, Integer>( );
map.put(10, 100);
System.out.println( map.get(10) );
```

- 가능한 제네릭스 사용 예제

```
Map<String, String> map = new HashMap<String, String>( );
Map<String, Object> map = new HashMap<String, Object>( );
Map<Long, String> map = new HashMap<Long, String>( );
```

❑ 5) Enumeration & Iterator

컬렉션에 저장된 데이터를 얻기 위한 API.

가. java.util.Enumeration

- hasMoreElements()
- nextElement()

나. java.util.Iterator

- hasNext()
- next()
- remove()

Thank you