Collections Framework

2016년 9월 25일 일요일 오전 1:00

배열과 컬렉션즈 프레임워크

배열은 연관된 데이터를 묶어서 관리하는 수단이였다. **컬렉션즈 프레임워크는 배열이 가지고 있는 불편함, 한계를 쉽게 벗어나게 도와주고 인스턴스의 저장 및 참고를 목적으로 하는 프레임워크**

배열의 불편함과 한계

- 배열은 인스턴스 생성 시, 배열의 길이를 정해야 한다.
- 배열의 길이에 초과되게 요소를 저장하면 ERROR

```
import java.util.ArrayList;

public class ArrayListDemo {
     public static void main(String[] args) {
          String[] arrObj = new String[2];
          arrObj[0] = "one";
          arrObj[1] = "two";
          // arrObj[2] = "three"; → ArrayIndexOutOfBoundsException

          for(String e : arrObj){
               System.out.println(e);
          }
```

ArrayList al = new ArrayList();

- **ArrayList는 인스턴스 생성 시 크기를 지정하지 않기 때문에 얼마든지 많은 수의 값을 저장할 수 있다.**
- **new ArrayList<>() 문장으로 인스턴스 선언**
- **import java.util.ArrayList; 필요**

- **ArrayList는 배열이 아니기 때문에 데이터 저장, 참고, 크기반환 방식이 다르다.**
  a. **데이터의 저장 메소드 : add(Object obj)**
  b. **데이터의 참고 메소드 : get(int index)**
  c. **ArrayList의 크기반환 메소드 : size()**

al.add("one");	index 0
al.add("two");	index 1
al.add("three");	index 2

```
          for(int i=0; i<al.size(); i++){
```

```
String value = al.get(i); // ERROR
```

Type mismatch: cannot convert from Object to String
2 quick fixes available:
Add cast to 'String'
Change type of 'value' to 'Object'
Press 'F2' for focus

기본적으로 add(Object obj) 메소드는 인자로 모든 데이터를 저장하기 위해 자료형이 Object 형이므로, get(int index) 메소드로 꺼낼 때도, Object 형 데이터로 반환하기 때문에 Casting ERROR 발생

→ 컬렉션 프레임워크에 제네릭이 도입된 이유이다.

```
               System.out.println(value);
          }
     }
}
```

컬렉션즈 프레임워크의 제네릭 도입

앞선 예제에서 알 수 있듯이, 컬렉션즈 프레임워크에 데이터를 저장하고, 꺼낼 때 Object 형 기반으로 하기 때문에,
타입 안정성이 떨어지고, 매번 형변환 해야하는 불편함이 있다.
그래서 컬렉션즈 프레임워크 클래스를 제네릭 클래스로 만들어서 내부에서 사용할 자료형을 인스턴스 생성시 지정!

```
import java.util.ArrayList;

public class ArrayListDemo {
     public static void main(String[] args) {
          ArrayList<String> al = new ArrayList<String>();
          al.add("one");
          al.add("two");
          al.add("three");

          for(int i=0; i<al.size(); i++){
               String value = al.get(i); // Compile OK!
               System.out.println(value);
          }
     }
}
```

전체적인 컬렉션즈 프레임워크 살펴보기

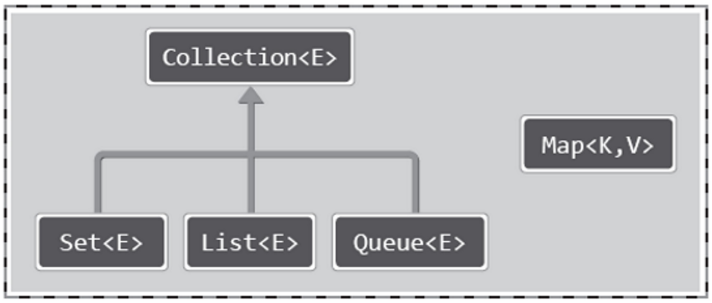
- 컬렉션즈 프레임워크의 인스턴스는 다른 말로 컨테이너라고 하는데, 이는 인스턴스 데이터을 담는 그릇이라는 의미이다.

- 최상위에 Collection, Map 제네릭 인터페이스가 있다.
- 어떤 인터페이스를 구현하느냐에 따라서 데이터 저장과 참조방식이 달라진다.

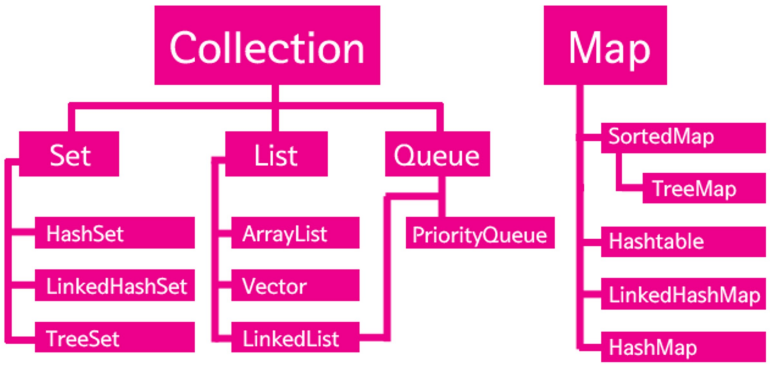
최상위 제네릭 인터페이스

- **Collection<E> : 인스턴스 단위의 데이터 저장 및 참조기능 제공**
- **Map<K,V> : Key - Value 구조의 인스턴스 저장 기능 제공**

❖ 컬렉션 프레임워크의 인터페이스 구조



제네릭 인터페이스를 구현하는 제네릭 클래스들



ArrayList를 찾아보자. Collection-List에 속해있다. ArrayList는 LIst라는 성격으로 분류되고 있는 것이다.
List는 인터페이스이다. 그리고 List 하위의 클래스들은 모두 List 인터페이스를 구현하기 때문에 모두 같은 API를 가지고 있다.
클래스의 취지에 따라서 구현방법과 동작방법은 다르지만 공통의 조작방법을 가지고 있는 것이다.

---

java.util

Interface Collection<E>

Type Parameters:

E - the type of elements in this collection

All Superinterfaces:

Iterable<E>

All Known Subinterfaces:

BeanContext, BeanContextServices, BlockingDeque<E>, BlockingQueue<E>, Deque<E>, List<E>, NavigableSet<E>, Queue<E>, Set<E>, SortedSet<E>, TransferQueue<E>

All Known Implementing Classes:

AbstractCollection, AbstractList, AbstractQueue, AbstractSequentialList, AbstractSet, ArrayBlockingQueue, ArrayDeque, ArrayList, AttributeList, BeanContextServicesSupport, B
DelayQueue, EnumSet, HashSet, JobStateReasons, LinkedBlockingDeque, LinkedBlockingQueue, LinkedHashSet, LinkedList, LinkedTransferQueue, PriorityBlockingQueue, Pr

---

java.util

Interface Map<K,V>

Type Parameters:

K - the type of keys maintained by this map

V - the type of mapped values

All Known Subinterfaces:

Bindings, ConcurrentMap<K,V>, ConcurrentNavigableMap<K,V>, LogicalMessageContext, MessageContext, NavigableMap<K,V>, SOAPMessageContext, SortedMap<K,V>

All Known Implementing Classes:

AbstractMap, Attributes, AuthProvider, ConcurrentHashMap, ConcurrentSkipListMap, EnumMap, HashMap, Hashtable, IdentityHashMap, LinkedHashMap, PrinterStateReasons,