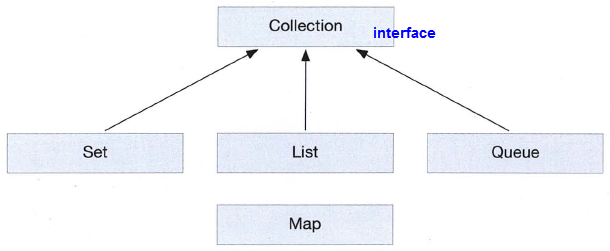
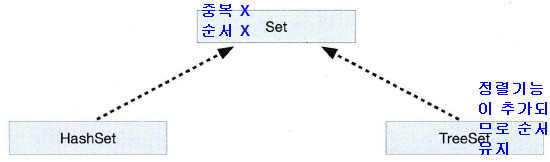
Collection Test

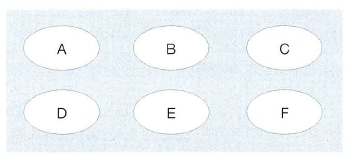


**1. Set Interface**



(1)HashSet

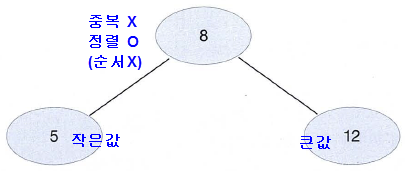
저장 영역에 객체 덩어리를 순서 없이 담는 형태로 요소를 저장히는 클래스이다. 중복된 데이터를 허용하지 않기 때문에 add(Object obj) 메소드를 이용해 새로운 요소를 추가할 때 동일한 객체가 이미 존재하면 추가는 실패하며 결과 값으로 false를 반환한다



|  |
| --- |
| **package** Set;  **import** java.util.HashSet;  **import** java.util.Iterator;  **public** **class** HashSetTest {    **public** **static** **void** main(String[] args) {    **if**(hs.add("korea"))//add()를 통해서 성공여부  System.***out***.println("첫 번째 korea 추가 성공");  System.***out***.println("첫 번째 korea 추가 실패");  **if**(hs.add("japan"))  System.***out***.println("japan 추가 성공");  System.***out***.println("japan 추가 실패");  **if**(hs.add("america"))  System.***out***.println("america 추가 성공");  System.***out***.println("america 추가 실패");  **if**(hs.add("britain"))  System.***out***.println("britain 추가 성공");  System.***out***.println("britain 추가 실패");  **if**(hs.add("korea")){//중복추가이므로 실패함  System.***out***.println("두 번째 korea 추가 성공");  System.***out***.println("두 번째 korea 추가 실패");  }  //Interator인터페이스 요소는 특정 컬렉션 요소에 접근해서 반복적으로 작업을 할때 편리    }  }  } |
| 첫 번째 korea 추가 성공  japan 추가 성공  america 추가 성공  britain 추가 성공  두 번째 korea 추가 실패  japan  america  korea  britain |

(2) TreeSet

(예) 범위 검색 작업에 효과적

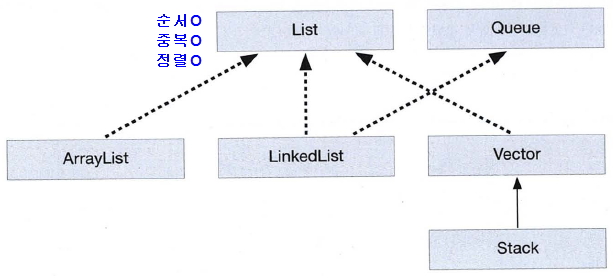


|  |
| --- |
| **package** Set;  **import** java.util.Iterator;  **import** java.util.TreeSet;  //정렬기능으로 인해서 순서적 출력됨  **public** **class** TreeSetTest {  **public** **static** **void** main(String[] args) {      **if**(hs.add("korea"))  System.***out***.println("첫 번째 korea 추가 성공");  System.***out***.println("첫 번째 korea 추가 실패");  **if**(hs.add("japan"))  System.***out***.println("japan 추가 성공");  System.***out***.println("japan 추가 실패");  **if**(hs.add("america"))  System.***out***.println("america 추가 성공");  System.***out***.println("america 추가 실패");  **if**(hs.add("britain"))  System.***out***.println("britain 추가 성공");  System.***out***.println("britain 추가 실패");  **if**(hs.add("korea"))  System.***out***.println("두 번째 korea 추가 성공");  System.***out***.println("두 번째 korea 추가 실패");      System.***out***.println("##정렬 기능때문에 순서적으로 출력된다.##");  }  } |
| 첫 번째 korea 추가 성공  japan 추가 성공  america 추가 성공  britain 추가 성공  두 번째 korea 추가 실패  america  britain  japan  korea  ##정렬 기능때문에 순서적으로 출력된다.## |

**2. List Interface**

List 인터페이스는 저장되는 요소들의 순서가 유지되며 중복된 데이터를 허용한다. 그리고 인텍스 변호에 의해서 정렬된다





① ArrayList

|  |
| --- |
| **import** java.util.ArrayList;  **public** **class** ArrayListTest {  **public** **static** **void** main(String[] args) {      System.***out***.println("ArrayList에 요소 추가하고 출력");  **for**( ){    }  System.***out***.println();  System.***out***.println();    array.remove(0);  array.remove("japan");    System.***out***.println("ArrayList에서 요소 제거 후 출력");    }  } |
| ArrayList에 요소 추가하고 출력  korea  japan  america  britain  ArrayList에서 요소 제거 후 출력  america  britain |

② Vector

Vector 클래스는 ArrayList의 구 버전으로 ArrayList 사용법과 같다. 차이점이 있다면 컬렉션에 작업을 수행할

때 Vector의 경우 멀티스레드 작업이 아니어도 무조건 동기화가 걸리고， ArrayList의 경우에는 기본적으로

동기화가 적용되지 않고 동기회가 펼요한 경우에만 메소드를 이용해서 동기화 처리를 해준다.

|  |
| --- |
| **import** java.util.Enumeration;  **import** java.util.Iterator;  **import** java.util.Vector;  **public** **class** VectorTest {    **public** **static** **void** main(String[] args) {    System.***out***.println();    System.***out***.println("Iterator로 요소 출력");    System.***out***.println();    System.***out***.println("Vector객체의 요소들을 Enumeration이용하여 출력");    } |
| vt.get(0) = 서울  vt.get(1) = 춘천  vt.get(2) = 광주  vt.get(3) = 대구  vt.get(4) = 부산  Iterator로 요소 출력  서울  춘천  광주  대구  부산  Vector객체의 요소들을 Enumeration이용하여 출력  서울  춘천  광주  대구  부산 |

③ LinkedList

|  |
| --- |
| **import** java.util.LinkedList;  **public** **class** LinkedListTest1 {    **public** **static** **void** main(String[] args) {      }  } |
| Santafe  소나타  K9  Sorento  아우디  에쿠스 |

④ Queue

|  |
| --- |
| **import** java.util.LinkedList;  **import** java.util.Queue;  **public** **class** LinkedTest2 {    **public** **static** **void** main(String[] args) {      }  } |
| 강남구  노원구  은평구  도봉구  서초구 |

⑤ Stack

|  |
| --- |
| **import** java.util.Stack;  **public** **class** StackTest {  **public** **static** **void** main(String[] args) {      }  } |
| 이문동  불광동  서초동  상계동 |

**3. Map Interface**

① hashable

|  |
| --- |
| **import** java.util.Hashtable;  **import** java.util.Iterator;  **import** java.util.Set;  **public** **class** HashtableTest {  **public** **static** **void** main(String[] args) {        hashtable.put("driver", "orcle.jdbc.driver.OracleDriver");  hashtable.put("url", "jdbc:oracle:thin:@localhost:1521:orcl");  hashtable.put("username", "java");  hashtable.put("password", "java");    //keyset,iterator()사용하기    }  } |
| key = password, value = java  key = url, value = jdbc:oracle:thin:@localhost:1521:orcl  key = driver, value = orcle.jdbc.driver.OracleDriver  key = username, value = java |

② Properties

Properties 클래스는 Hashtable을 싱속받은 클래스로서 다른 Map 클래스들은 카와 값을 객체 타입으로 다루

는데 Properties 클래스는 키와 값을 String 타입으로 다루므로 각 속성 값을 다루기에 편리한 클래스이다

|  |
| --- |
| **import** java.util.Enumeration;  **import** java.util.Properties;  **public** **class** PropertiesTest {  **public** **static** **void** main(String[] args) {  p.put("name", "홍길동");  p.put("addr", "역삼동");  p.put("age", "10");  p.put("nation", "대한민국");  // propertyNames()      }  } |
| name = age, value = 10  name = name, value = 홍길동  name = nation, value = 대한민국  name = addr, value = 역삼동 |