Chapter 10 날짜와 시간 **&** 형식화

◈ **Calendar**와 **Date**

- Calendar는 추상 클래스이기 때문에 직접 객체를 생성할 수 없고, 메서드를 통해서 완전히 구현된 클래스의 인스턴스를 얻어서 사용한다.
- 그 이유는 최소한의 변경으로 프로그램이 동작할 수 있도록 하기 위해
  . Calendar cal = new Calendar(); ⇒ 에러!! 추상클래스는 인스턴스 생성 불가
  . Calendar cal = Calendar.getInstance();

◈ **Calendar**에서의 데이터 접근

- get, set 사용하여 접근
- add(int field, int amount): 지정한 필드의 값을 원하는 만큼 증가/감소
- roll(int field, int amount): 지정한 필드의 값을 증가/감소하나 다른 필드는 고정
- 1월: 0부터 시작한다. 즉 1월=0, 12월=11
- 일요일:0, 월요일:1, 화요일:2, 수요일:3, 목요일:4, 금요일:5, 토요일:6

◈ **SimpleDateFormat** 클래스

y	년도	F	월의 몇번째 요일	h	시간(1~12)
M	월	E	요일	m	분(0~59)
w	년의 몇번째 주	a	오전/오후	s	초(0~59)
W	월의 몇번째 주	H	시간(0~23)	S	천분의 1초
D	년의 몇번째 일	k	시간(1~24)	z	Time zone(GMT)
d	월의 몇번째 일	K	시간(0~11)		

◈ **ChoiceFormat** 클래스

- 특정 범위에 속하는 값을 문자열로 변환해주는 클래스
- #: 경계값을 범위에 포함
- <: 경계값을 범위에 포함시키지 않음
  . eg) 60<D|70<C|80<B|90<A ⇒ 0~59는 D, 60~69는 C …

◈ **MessageFormat** 클래스

- 데이터를 정해진 양식에 맞게 출력하는 클래스
- MessageFormat.format(msg, arguments);

◈ **java.time** 패키지

- Date, Calendar가 가지고 있는 단점을 해소하기 위해 생긴 패키지
- String 클래스와 마찬가지로 immutable한 특징. 즉 multi thread 환경에서는 동시에 여러가지 thread가 같은 객체에 접근할 수 있기 때문에, 변경 가능한 객체는 데이터가 잘못된 가능성이 있으며, 이를 thread-safe하지 않다고 함.

java.time	날짜와 시간을 다루는 클래스 제공
java.time.chrono	표준(ISO)이 아닌 달력 시스템 서포트 클래스 제공
java.time.format	날짜와 시간을 parsing하고 형식화 클래스 제공
java.time.temporal	날짜와 시간의 field와 unit을 위한 클래스 제공
java.time.zone	time zone과 관련된 클래스 제공

◈ **java.time** 패키지의 핵심 클래스

- 역시 인스턴스를 얻어서 사용하며, 객체 생성은 now(), of()로 한다.
- Period, Duration 클래스
  . Period: 날짜 간의 차이, Duration: 시간 사이의 차이
- Temporal, TemporalAccessor, TemporalAdjuster를 구현한 클래스
  . LocalDate, LocalTime, LocalDateTime, ZonedDateTime, Instant 등
- TemporalAmount를 구현한 클래스
  . Period, Duration
- TemporalUnit: 날짜와 시간의 단위를 정의해 놓은 인터페이스
  . ChronoUnit이 이 인터페이스를 구현했음.
- TemporalField: 년, 월, 일 등 날짜와 시간의 필드를 정의해 놓은 인터페이스
  . ChronoField가 이 인터페이스를 구현했음.

◈ **LocalDate**와 **LocalTime** 클래스

- LocalDate today = LocalDate.now();
- LocalTime now = LocalTime.now();

※ Calendar와 달리 월의 범위가 1~12이고, 요일은 월요일:1, 화요일:2, 일요일:7

※ 특정 field 값을 가져오는 것은 get(), getXXX(), 자바의정석 p.556 참조

- field 값 변경하기 : with(), plus(), minus()
  . field를 변경하는 메서드들은 항상 새로운 객체를 생성하여 반환하므로, 대입 연산자를 사용해야 한다.
- 날짜와 시간의 비교: isAfter(), isBefore(), isEqual(), Equals()
  . isEqual()은 날짜만 비교하고, Equals()는 날짜 외의 모든 필드 비교

◈ **LocalDateTime**과 **ZonedDateTime** 클래스

- LocalDate + LocalTime = LocalDateTime
- LocalDateTime + 시간대 = ZonedDateTime
- ZoneId: 일광 절약시간(DST, Daylight Saving Time)이 자동적으로 처리된 zone의 id를 얻을 수 있다.
- ZoneOffset: UTC로부터 얼마만큼 떨어져있는지 표현

◈ **ZonedDateTime** 과 **OffsetDateTime**

- ZonedDateTime: ZoneId로 구역을 표현하며, DST가 적용되어 있다.
- OffsetDateTime: 시간대의 차이로만 구분한다. 컴퓨터에게 DST를 적용하는 행위는 위험하므로, 일관된 시간체계를 유지하는 것이 더 안전하다. 같은 지역 내의 컴퓨터 간에 데이터를 주고받을 때, 전송시간을 표현하기에 LocalDateTime이면 충분하지만, 서로 다른 시간대에 존재하는 컴퓨터간의 통신에는 OffsetDateTime이 필요하다.

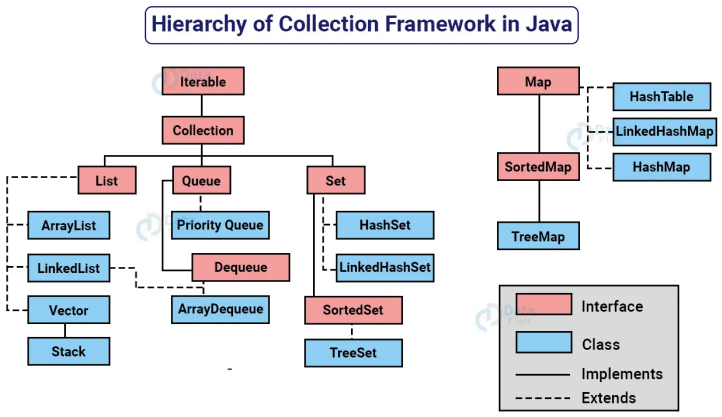
◈ **TemporalAdjusters** 클래스

- 자주 쓰일만한 날짜와 시간 계산들을 대신 해주는 메서드들이 제공됨

firstDayOfNextYear()	다음 해의 첫 날
firstDayOfNextMonth()	다음 달의 첫 날
firstDayOfYear()	올 해의 첫 날
firstDayOfMonth()	이번 달의 첫 날
lastDayOfYear()	올 해의 마지막 날
lastDayOfMonth()	이번 달의 마지막 날
firstInMonth (DayOfWeek dayOfWeek)	이번 달의 첫 번째 ? 요일
lastInMonth (DayOfWeek dayOfWeek)	이번 달의 마지막 ?요일
previous (DayOfWeek dayOfWeek)	지난 ?요일(당일 미포함)
previousOrSame (DayOfWeek dayOfWeek)	지난 ?요일(당일 포함)
next	다음 ?요일(당일 미포함)
nextOrSame (DayOfWeek dayOfWeek)	다음 ?요일(당일 포함)
dayOfWeekInMonth (DayOfWeek dayOfWeek)	이번 달의 n번째 ?요일

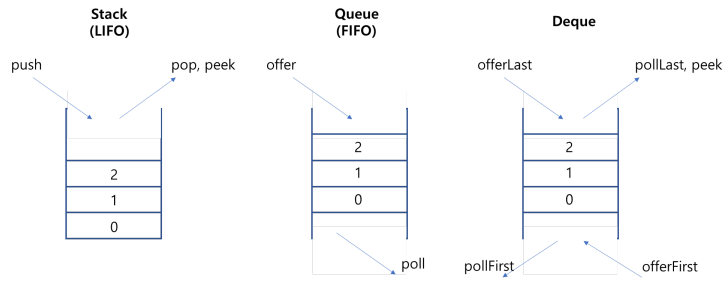
Chapter 11 컬렉션 프레임웍

1. 컬렉션 프레임웍
   - 1.1. 컬렉션 프레임웍의 핵심 인터페이스
   - 1.2. ArrayList
   - 1.3. LinkedList
   - 1.4. Stack과 Queue
   - 1.5. Iterator, ListIterator, Enumeration
   - 1.6. Arrays
   - 1.7. Comparator와 comparable
   - 1.8. HashSet
   - 1.9. TreeSet
   - 1.10. HashMap과 Hashtable
   - 1.11. TreeMap
   - 1.12. Properties
   - 1.13. Collections
   - 1.14. 컬렉션 클래스 정리 & 요약



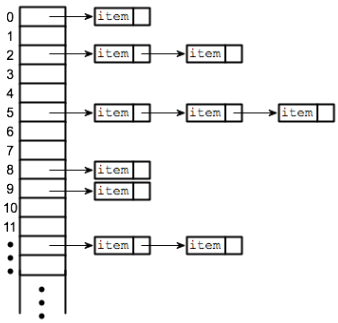
인터페이스	데이터 중복	데이터 순서	예시
List	허용(O)	있음(O)	대기자 명단
Set	비허용(X)	없음(X)	양의 정수 집합
Map	Key는 허용 Value는 비허용	없음(X)	우편번호, 전화번호부

구분	컬렉션	특징
List	Vector	- Vector 대신 가급적이면 ArrayList를 사용할 것 - capacity는 기존 값의 2배가 최소로 설정됨 - 데이터를 읽어오고 저장하는 데는 효율이 좋지만, 용량(배열의 수)을 변경할 때는 효율이 떨어짐
	ArrayList	- Vector보다 개선된 컬렉션 - 배열기반, 데이터의 추가와 삭제에 불리 - 순차적인 추가/삭제는 제일 빠름 - 비순차적인 임의의 요소에 대한 접근성 좋지 않음 - 임의의 요소에 대한 접근성 뛰어남 - 크기 변경 불가, 새로운 배열을 생성해서 복사 필요 - 배열에 추가 저장공간 부족하면, 자동으로 더 큰 새로운 배열 생성하고, 기존 배열을 복사하여 저장함 - 삭제를 위해서는 인덱스의 끝부터 역순으로 접근하는 것이 효율적임. (자리이동 이슈 때문)
	LinkedList	- 연결기반, 중간에 있는 데이터의 추가/삭제에 유리 - 이동 방향이 단방향이라, 다음 요소에 접근은 쉬우나 이전 요소에 대한 접근이 어려움
		<Doubly-linked list> - 전/후 양방향 주소를 가지고 있는 linked list <Doubly circular linked list> - Doubly linked list의 시작과 끝을 연결, 연속성 부여 - 실제로 LinkedList 클래스는 Doubly circular linked list로 구현되어 있음
	Stack	- Vector를 상속받아 구현 - LIFO(Last In First Out) - ArrayList와 같은 배열 기반의 컬렉션이 적합
Queue	Queue	- LinkedList가 Queue 인터페이스를 구현 - FIFO(First In First Out) - 데이터 추가/삭제가 쉬운 LinkedList로 구현하는 것이 더 적합
	Deque	- Double-Ended Queue, 양쪽 끝에 추가/삭제 가능 - Stack과 Queue를 하나로 합쳐놓은 것과 같으며, Stack으로 사용할 수도 있고, Queue로도 사용 가능 - ArrayDeque, LinkedList
	Priority Queue	- 저장 순서에 관계없이 우선순위가 높은 것부터 꺼내는 특징 - heap의 자료구조 형태로 저장, null 저장 불가

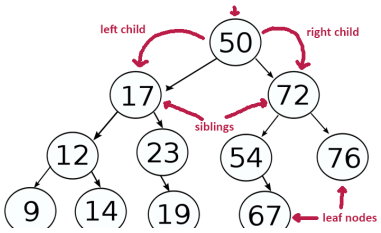
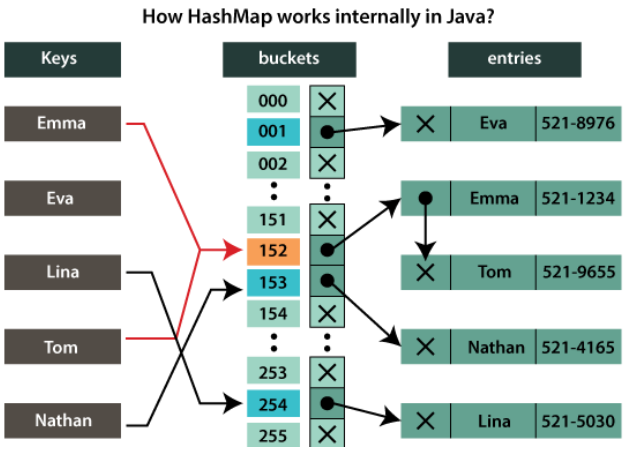


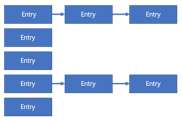
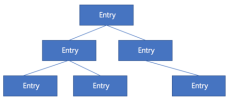
<hash 관련 정리>

- hash: 다양한 길이를 가진 데이터를 고정된 길이를 가진 데이터로 매핑한 값
- entry: key와 value를 묶어서 하나의 데이터(entry)로 저장한다.
- hashing: hash function을 이용해서 hash table에 데이터를 저장하고 검색하는 기법. 배열과 linked list의 조합으로 구성
- hash function: 임의의 길이를 갖는 임의의 데이터에 대해 고정된 길이의 데이터로 mapping하는 함수, 결과값으로는 데이터가 저장되어 있는 장소인 hash 값이 나오기 때문에 원하는 데이터를 빠르게 찾을 수 있다.



Set	HashSet	- 대표적인 Set인터페이스를 구현한 클래스 - 저장 순서를 유지하지 않고, 중복값도 비허용 - equals()를 override하려면, hashCode()도 해야함 - hashing이 구현됨 → 데이터 빠르게 검색
	LinkedHash Set	- 저장 순서를 유지하고, 중복값 비허용 - hashing이 구현됨 → 데이터 빠르게 검색
	TreeSet	- SortedSet(데이터들이 정렬되어 있는 Set)을 상속 - binary search tree의 자료구조 형태로 데이터

		저장 - 정렬, 검색, 범위 검색에 높은 성능을 보임 - 저장 순서를 유지하지 않고, 중복값도 비허용 <binary search tree의 특징> - 모든 노드는 최대 두개의 자식 node를 가진다. - left node의 값은 parent보다 작고, right node의 값은 parent보다 커야 한다. - node의 추가/삭제에 시간이 걸린다.(비순차적) - 검색(범위검색)과 정렬에 유리하다. - 중복된 값을 저장하지 못한다.
TreeSet		
Map	HashTable	- HashMap의 old 버전 - key나 value로 null 비허용 - hashing이 구현됨 → 데이터 빠르게 검색
	HashMap	- HashTable의 개선 버전 - key와 value를 entry 형태로 저장하며, 많은 양의 데이터를 검색하는데 있어서 뛰어난 성능을 보임 - 내부 클래스로 Entry가 정의되어 있음 - key는 유일해야 하며, value는 key와 매핑된 값으로 중복이 허용됨. 따라서 두 개중 어느 쪽을 key로 할 것인지 잘 결정해야 함 - key나 value로 null 허용 - hashing이 구현됨 → 데이터 빠르게 검색
		
	LinkedHash Map	- 저장 순서 유지(key가 입력된 순서가 보장됨) - key는 유일하고, value는 중복값 허용
	TreeMap	- SortedMap(데이터들이 정렬되어있는 map)을 구현 - binary search tree 형태로 key와 value의 쌍으로 이루어진 데이터를 저장함.
	Properties	- Hashtable을 상속받아 구현한 컬렉션 클래스 - 주로 application의 환경 설정과 관련된 property를 저장하는데 사용되며, 데이터를 file로 부터 읽고, 쓰는 편리한 기능 제공

	구조	특징
hashmap		- key 값 비정렬 - 검색 성능 뛰어남 - 비순차 접근 유리
treemap		- key 값 정렬 - 범위검색 성능 뛰어남 - 순차 접근 유리

컬렉션	읽기(접근시간)	추가/삭제	기타
ArrayList	빠르다	느리다	비효율적인 메모리 사용
LinkedList	느리다	빠르다	데이터가 많을수록 접근성 ↓

※ 이 두 가지 클래스의 장점을 조합해서 사용하는 방법을 활용하면 효율성이 좋다. 처음에 작업하기 전에 데이터를 저장할 때는 ArrayList를 사용하고, 작업을 할때는 LinkedList로 데이터를 옮겨서 작업하는 방식

```
ArrayList al = new ArrayList(100000);
for(int i = 0; i < 100000 ; i++) al.add(i+"");

LinkedList ll = new LinkedList(al);
for(int i = 0; i<1000;i++) ll.add(500, "X");
```

◈ **Iterator, ListIterator, Enumeration**
- 컬렉션에 저장된 요소를 접근하는데 사용되는 인터페이스
- Enumeration: Iterator의 구버전, 가능하면 Iterator를 사용할 것
- ListIterator: Iterator의 기능을 향상시킨 버전, 양방향 조회 기능 추가 (List를 구현한 경우)

◈ **Arrays** 클래스
- 배열을 다루는데 유용한 method 정의

기능	**method**	설명
배열 복사	copyOf()	배열 전체 복사
	copyRange()	배열의 일부 복사, 범위끝 미포함
배열 채우기	fill()	모든 요소를 지정된 값으로 채우기
	setAll()	함수형 인터페이스로 채우기
배열의 정렬과 검색	sort()	배열의 정렬
	binarySearch()	지정된 값이 저장된 index return
문자열 비교와 출력	equals()	두 배열에 저장된 요소 비교
	deepEquals()	다차원 배열의 비교
	toString()	배열의 모든 요소를 문자열로 출력
	deepToString()	다차원 배열의 모든 요소를 출력
배열을 List로 변환	asList()	배열을 List에 담아서 return 크기 변경 불가
기타	parallelXXX()	빠른 결과를 위해 여러 thread로 작업

	spliterator()	여러 thread가 처리할 수 있게 하나의 Spliterator를 return
	stream()	컬렉션을 stream으로 변환

◈ **Comparator**와 **Comparable** 인터페이스
- 컬렉션을 정렬하는데 필요한 method를 정의하는 인터페이스
- 즉, 객체를 비교할 수 있도록 만드는 인터페이스이다.
- 같은 타입의 인스턴스끼리 서로 비교할 수 있는 클래스들, 주로 Integer와 같은 wrapper 클래스와String, Date, File과 같은 것들이며, 기본적으로 오름차순으로 구현되어 있음

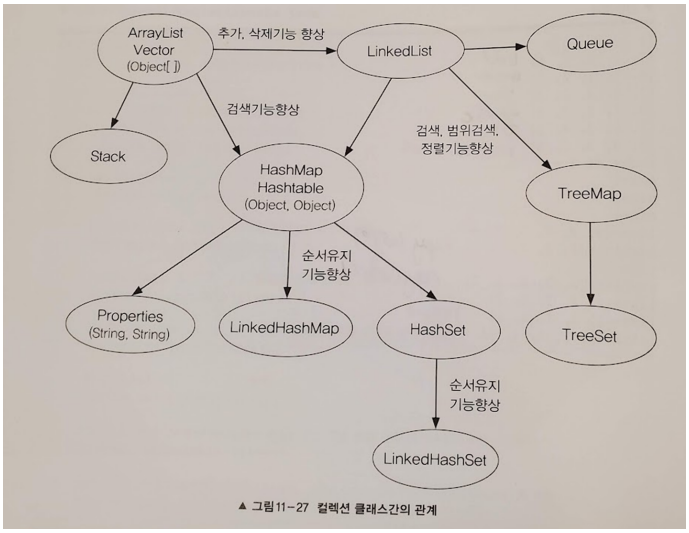
인터페이스	차이점
Comparable	- compareTo(T o) - 자신과 parameter를 비교 - util 패키지 - 기본 정렬 기준(오름차순)을 구현할 때
Comparator	- compare(T o1, T o2) - 두 parameter를 비교 - lang 패키지 - 기본 정렬기준 외에 다른 기준(오름차순 외)으로 정렬할 때

※ 주의: method를 override할 때 overflow, underflow를 방지하기 위해 비교는 부등호로 하는 것이 좋다.

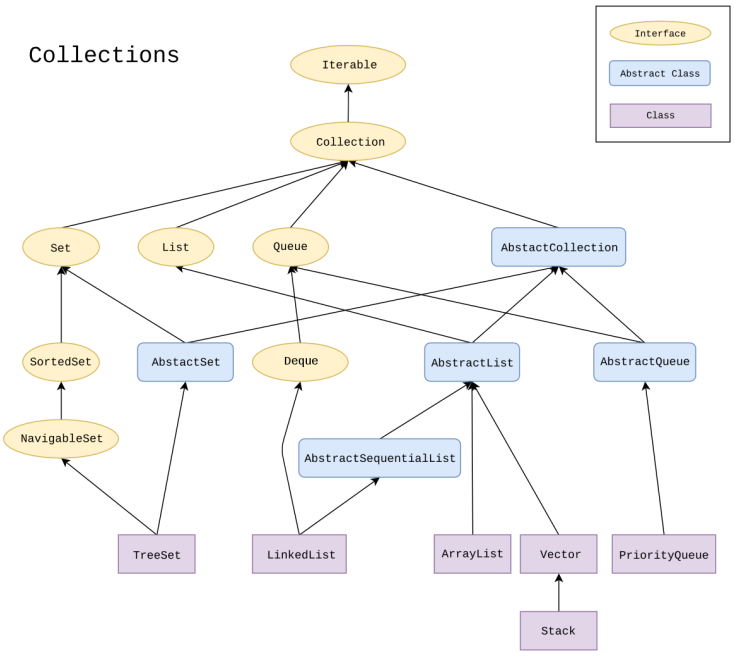
◈ **Collections**
- 컬렉션과 관련된 method를 제공
- java.util.Collection은 인터페이스 이고, java.util.Collections는 클래스이다.
- 기존 method에 아래의 키워드를 조합해서 특징을 부여할 수 있다.

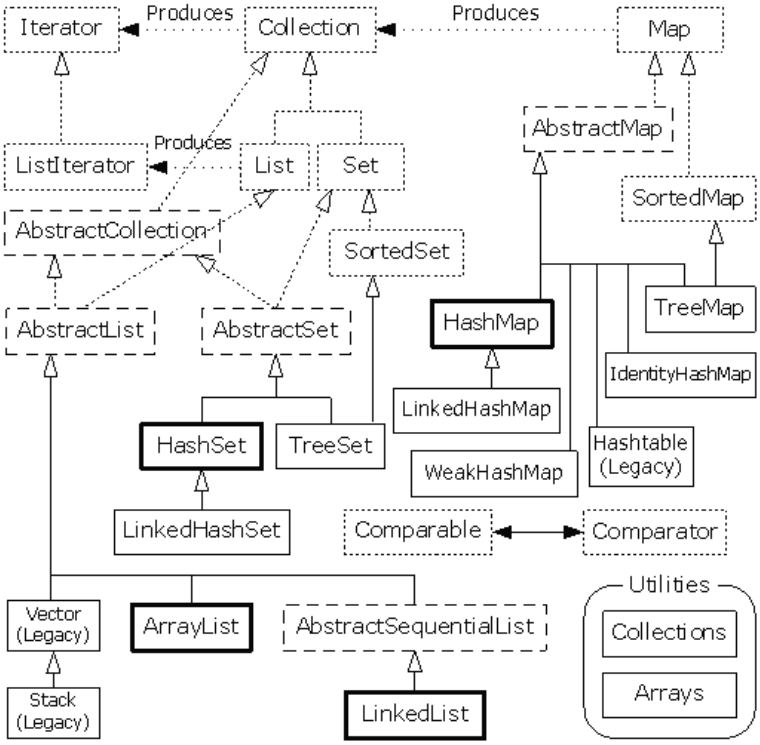
기능	키워드	설명
컬렉션의 동기화	synchoronizedXXX	- 데이터의 일관성을 유지하기 위해 공유되는 객체에 synchoronized가 필요함 - Single thread에서는 불필요
변경 불가 컬렉션 만들기	unmodifiableXXX	- read only로 만들어서 컬렉션 보호 - Single thread에서는 불필요
Singleton 컬렉션 만들기	singletonXXX	- 단 하나의 객체만을 저장하는 컬렉션을 만들 때
한 종류의 객체만 저장하는 컬렉션 만들기	checkedXXX	- 한 종류(String, int, float 등)의 데이터 타입만 parameter로 받아서 저장 가능

◈ **Singleton pattern**
- 객체의 인스턴스가 오직 한개만 생성되는 pattern
- constructor는 외부에서 호출하지 못하게 private으로 선언한다.
- 최초 한번의 new 연산자를 통해서 고정된 메모리 영역을 사용하기 때문에 추후 해당 객체에 접근할 때 메모리 낭비를 방지할 수 있다. 이미 생성된 인스턴스를 활용하기 때문에 속도 측면에서도 이점이 있다. 그리고 다른 클래스 간에 데이터 공유가 쉽다. global로 사용되는 인스턴스이기 때문에 다른 클래스의 인스턴스들이 접근하여 사용할 수 있다. 하지만, 여러 클래스의 인스턴스에서 singleton 인스턴스의 데이터에 동시 접근하게 되면 sync 이슈가 있을 수 있다. synchronized 키워드 사용으로 해결한다.
- 단점으로는, 테스트 하기가 어렵다. singleton 인스턴스는 자원을 공유하고 있기 때문에 테스트가 결정적으로 격리된 환경에서 수행되려면 매번 인스턴스의 상태를 초기화 시켜주어야 한다. 또한 child 클래스를 만들 수 없고, 내부 상태를 변경하기 어려워서 유연성이 떨어진다.



▲ 그림11-27 컬렉션 클래스간의 관계





Interface	Hash Table	Resizable Array	Balanced Tree	Linked List	Hash Table + Linked List
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Deque		ArrayDeque		LinkedList	
Map	HashMap		TreeMap		LinkedHashMap