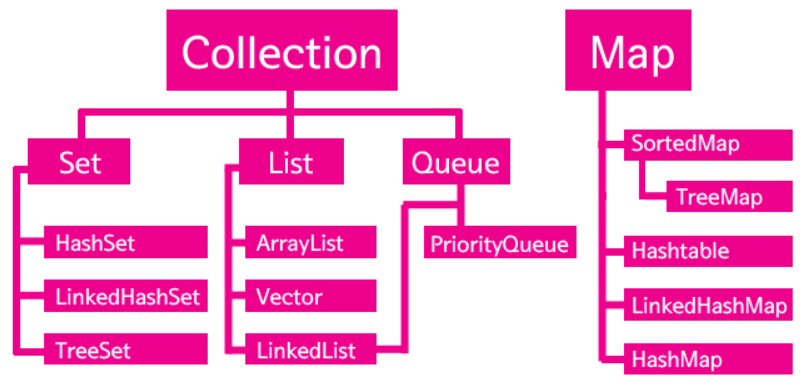
**Collection Framework (자료구조)**

**I. Collection Framework 분류**

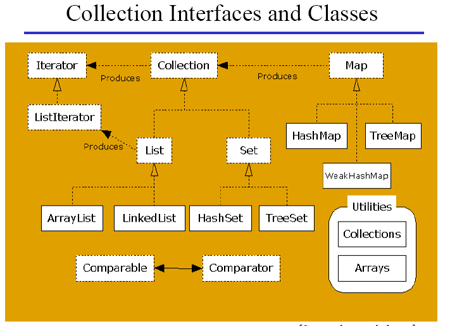
(1) Collection & Map



[Collections Framework=Container]

**※** Map 인터페이스는 Collection 인터페이스를 직접 확장한 것은 아니지만 Map도 컬렉션 프레임워크에

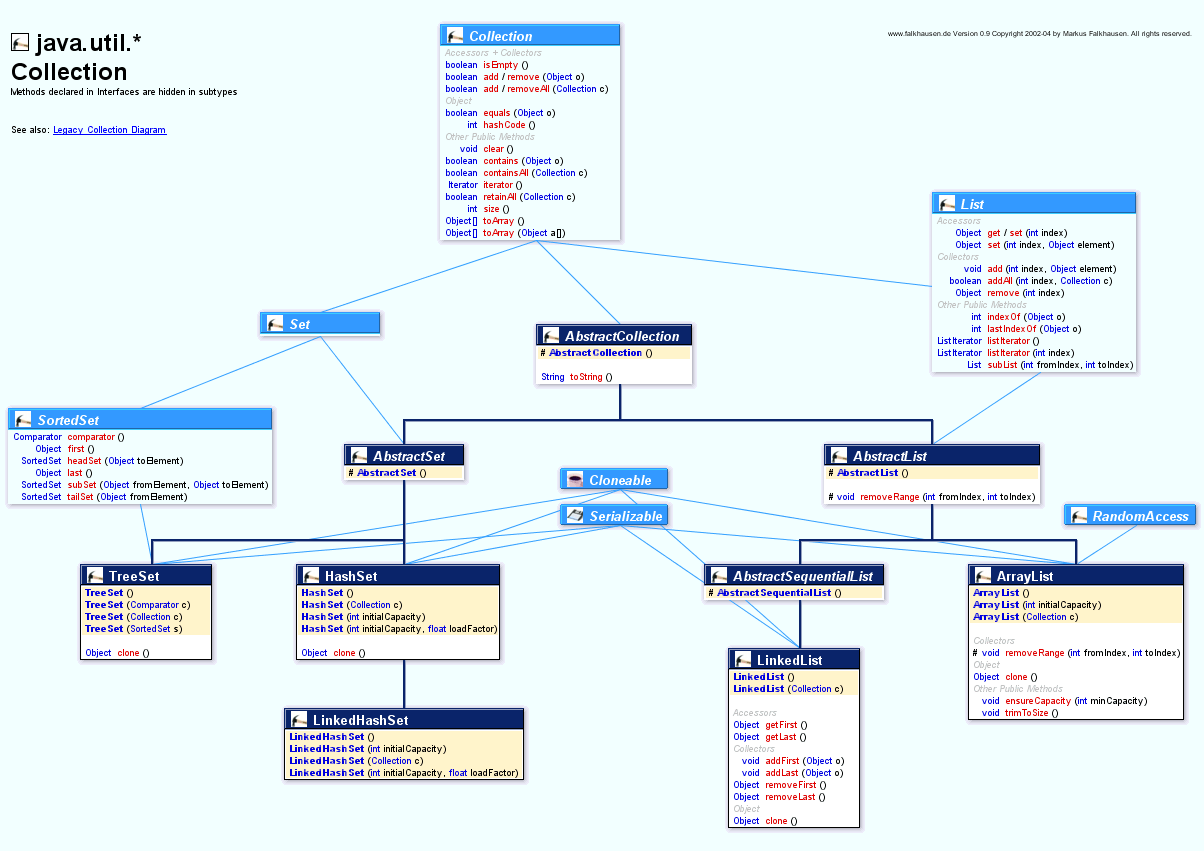
포함되는 것으로 간주한다. 따라서 Map도 그냥 컬렉션이라고 부른다.

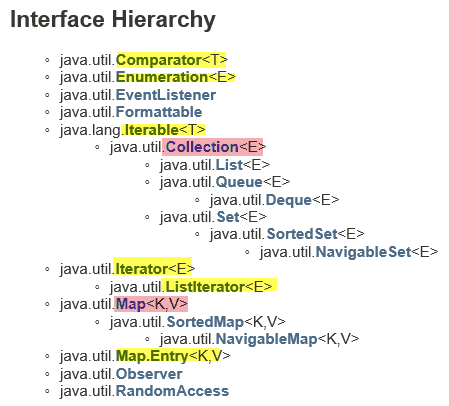


|  |  |
| --- | --- |
| 인터페이스 | 클래스 |
| java.util.Collection  동적자료저장공간 | java.util.Collections  컬랙션의 sort, search |

|  |  |  |
| --- | --- | --- |
| array | ArrayList | Arrays |
| 배열 | Java.util.**ArrayList** Class | Java.util.**Arrays** Class |
| 정적저장공간 | 동적저장공간(Collection 요소) | 컬렉션에 있는 자료 검색 및 정렬 |

**(2) Collection Interface & Class API**





※ Interface 는 고수준의 추상 클래스라서 몸뚱이를 가지고 있는 일반 메소드를 포함할 수가 없다. 다만 메소드 원형만 가지는 추상 메소드 또는 상수만 멤버로 포함할 수가 있다. 그래서 몸뚱이가 없는 추상 메소드는 자손클래스를 하나 만들어서 몸뚱이를 구현(Implementation)해 줘야 한다.



**II. Collection Framework**

**1.** **Collection framework ?**:  Class를 제어하기 위한 인터페이스, 인터페이스를 구현한 Class로 구성

컬렉션 프레임워크(만들고자하는 구조물의 기본골격)는 컬렉션 클래스를 쉽게 이해하고 사용하도록 다형성을 이용해서 컬렉션 객체의 메소드를 형태적으로 동일하게 유지함으로써 사용법을 일관되게 만든 디자인 패턴(가이드라인) 이다. 즉, 자바 프레임 워크 = 라이브러리(클래스) + 디자인 패턴(가이드라인)

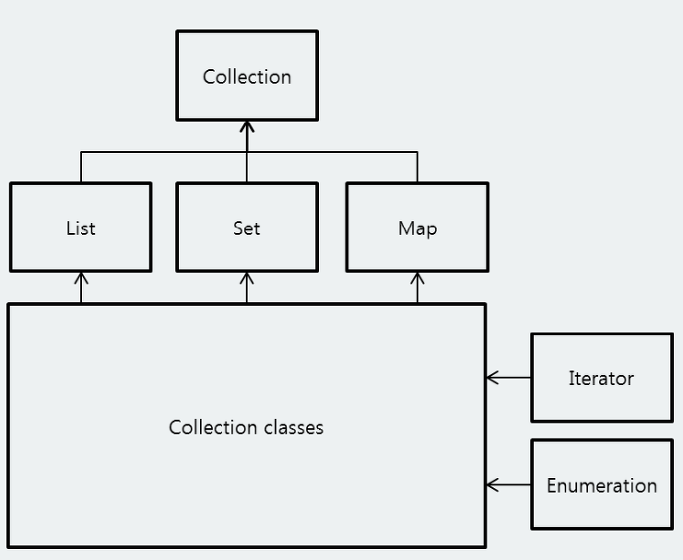
|  |  |
| --- | --- |
| Colletion | Framework |
| 다수의 데이터, 즉 데이터 그룹 | 표준화, 정형화된 체계적인 프로그래밍 방식 |

﻿

|  |  |
| --- | --- |
| JDK | 자료구조 저장공간 변경사 |
|  | -Vector, Hashtable, Properties등 각자 방식으로 데이터 저장 (collection 개념X)  -Vector, Hashtable등은 자체적으로 동기화 처리: multi thread환경이 아닌 경우 속도 저하 |
| JDK1.2 | **Collection framework** 시작  -데이터 군을 저장하는 클래스들을 표준화한 설계  -다수의 데이터를 쉽게 처리할수 있는 방법을 제공하는 클래스들로 구성  -ArrayList, HashMap 는 동기화 자체처리 않음 : 필요한 경우만 **java.util.Collections** 동기화 메서드를 이용해서 동기화 처리 |
| JDK1.4 | LinkedHashSet |
| JDK1.5 | ① Iterable인터페이스 추가 : iterator( ) 중복제거용  ② **auto-boxing 과 generic**  ③ EnumSet, EnumMap |
| JDK1.6 |  |
| JDK1.7 |  |

|  |  |
| --- | --- |
| Collection & Map class | Array class |
| 자료구조적인 측면에서의 Data 저장공간 | 단순 용량적인 측면에서의 Datat 저장공간 |
| 동적메모리 할당 | 초기에 배열 크기가 정해지면 변경불가 |
| 객체만을 저장대상으로 한다. | ① 기본 DataType의 변수 : int, long,,,  ②일반적인 클래스의 객체 : string[ ], Image[ ] |
| 배열의 발전된 형태 | => ArrayList |
| Java.util 패키지에 있는 Interface  ⇒ Method의 프로토타입만 존재 |  |
| collection : 집합적인 저장공간  Map : 검색적인 저장공간 |  |

**2. 컬렉션 클래스**

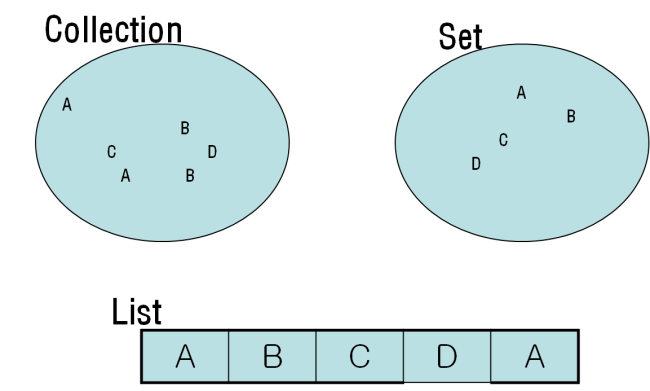
****

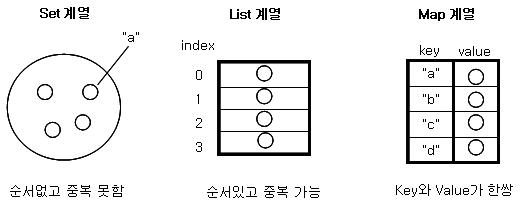
Interface

Collection classes

Data의 관리를 위해서 예전에는 배열 혹은 구조체(C언어)을 사용하였지만 자바에서는 발전된 형태인 배열을 사용하는데 이것을 **컬렉션 클래스(Interface Collection+ Interface Map으로 구현된 클래스)라 한다.**

컬렉션은 배열의 크기를 동적으로 늘릴 수 없다는 단점을 보완해서, **동적으로 필요할 때 메모리를 확장할 수 있게 만들었다.** 이러한 저장공간에 여러가지 자료구조적인 기능이 가미된 저장공간으로서 객체의 삽입, 삭제, 수정, 검색의 기능을 가지고 있다.



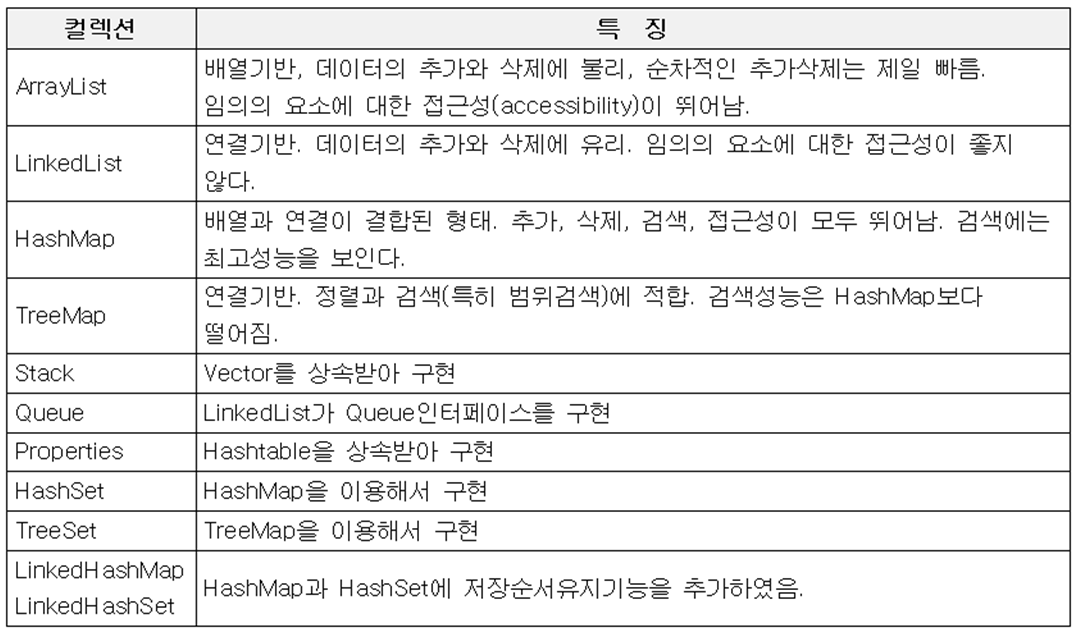


※ Collection 하위 Class들의 비교

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Collection  분류 | 중첩  가능 | 순서  정렬 | 주요  Class |  |
| Collection | Set | X | X | HashSet | HashMap 보다는 느리지만 Set인터페이스를 구현한 클래스중 가장 빠름 |
| LinkedHashSet | 삽입순서에 따라 배열 요소에 접근 TreeSet 보다 빠르다 |
| TreeSet | 가장느림, 키 순서에 따른 접근 가능 |
| Map | O | 키,값 | IdentityHashMap | 동일성 개념을 사용할 수 있는 경우 HashMap보다 더 빠른 고성능 대응을 제공 |
| HashMap | 가장 빠른 범용 Map |
| HashTable | HashMap보다 느리나 동기화된 HashMap 보다는 빠르다 |
| LinkedHashMap | TreeMap보다는 빠르지만 위의 다른것들보다는 느리다 |
| TreeMap | 가장 느림, 키순서에 따른 접근 가능 |
| List | O | O | ArrayList | 가장 빠르다. |
| Vector | ArrayList 보다 느리지만 동기화된 ArrayList보다 빠르다 |
| Stack | Vector와 동일 속도, LIFO 큐 기능을 제공 |
| LinkedList | 다른 List 클래스 보다 느리다 그러나 몇가지 형태의 큐에 있어서는 다른 클래스 보다 빠르다. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 인터페이스 | 구현 | 클래스(1.6) | 순서 | 중복 | 사용예 |
| Collection | 값 모아 관리 |  | ⅹ |  |  |
| List | 순서 | ArrayList, LinkedList  Stack, Vector | ○ | ○ | 대기자명단 |
| Set | 집합 | HashSet, TreeSet  EnumSet, LinkedHashSet | ⅹ | ⅹ | 양의정수집합/소수집합 |
| Map | 검색 | HashMap, TreeMap  Hashtable, **Properties**  EnumMap, LinkedHashMap  WeakHashMap | ⅹ | 키(ⅹ), 값(○) | 우편/지역/전화번호 |

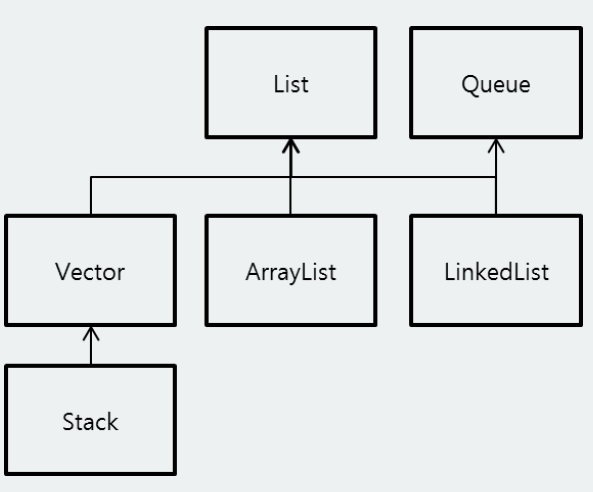
|  |  |
| --- | --- |
| **Package** | **Definition** |
| java.util.Collection | 순서 없는 단순한 요소들의 집합 |
| java.util.List | 순차적 나열, 순서지정이 가능한 요소들의 집합 (순서) |
| java.util.Set | 중복을 허용하지 않는 요소들의 집합 (집합) |
| java.util.Map | Key와 Key에 대응하는 값으로 이루어진 구조 (검색) |
| java.util.SortedSet | 값들이 정렬된 Set |
| java.util.SortedMap | key가 정렬된 Map |
| java.util.Iterator | Collection 검색 |
| Java.util.Enumeration | Collection 검색 (Iterator의 구버전) |
| Java.util.ListIterator |  |



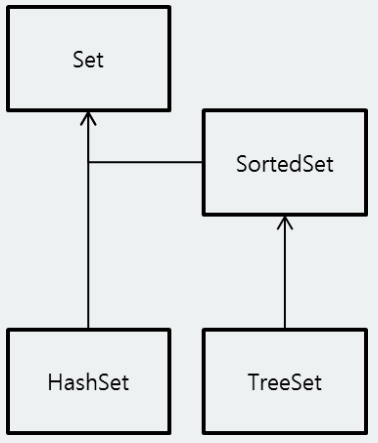
**(1) Interface Collection**

**Collection 인터페이스로부터 구현된 클래스들은** 기본적인 기능을 바탕으로 조금씩 확장되어 다양한 기능이 첨가된 클래스들을 제공된다. 보통은 집합적인 데이터를 관리하기 위해서 사용된다. **Collection 인터페이스를 구현한 클래스에서는 각각의 기능의 기능에 맞게 오버라이딩되어 있다..**

**① List** : 순서가 유지되는 형태로 객체를 모아둘떄 사용



**② Set** : 중복되지 않은 객체를 모아두기 위해 사용

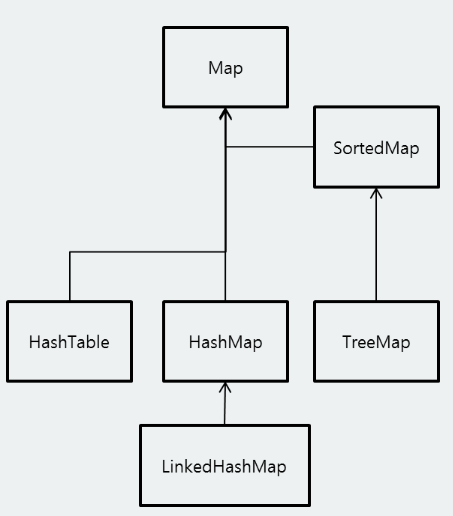


(**2) Interface Map**

Map 인터페이스로 구현된 클래스는 Collection과 달리 Map은 검색적인 개념을 담고 있는 인터페이스이다. 검색을 위해서 각각의 데이터에 키를 함께 보유하게 된다. **Map 인터페이스는 검색적인 저장공간을 지원하기 위해서 사용하는 경우가 많다.**

키 없이 단순히 데이터만을 넣어 두고자 할 때는 Collection 계열을 사용하면 된다. 키를 사용해서 데이터를 빠르게 검색하기 위해서는 Map 계열을 사용하면 된다.

**③Map** : 키와 값을 하나의 쌍으로 묶어서 저장



**(3) Iterator, ListIterator, Enumeration Interface**

Collection Class에 저장된 구성 요소들을 순차적으로 접근하기 위한 인터페이스

Iterator의 구버젼이 Enumerattion이다. Iterator은 단방향이지만, ListIterator은 양방향 접근 기능이다.

**(4) Interface comparator**

: 객체를 정렬하는데 필요한 메서드 제공하는 인터페이스

Comparable : 기본 정렬기준을 구현하는 사용.

Comparator : 기본 정렬기준외에 다른 기준으로 정렬하고자 할때 사용.

**3. Utilities Classs**

컬렉션의 sort, search : Collections, Arrays 클래스

정렬 클래스 : Comparable

**4. Synchronization (동기화)**

: Multi-thread환경에서 하나의 객체를 여러 Thread가 동시에 데이터에 접근할수 있으므로 Data Consistency

(일관성) 유지를 위해서 동기화 필요.

|  |  |  |  |
| --- | --- | --- | --- |
| Vector, Hashtable,,, | JDK1.2이전 | 자체적 동기화 처리 |  |
| ArrayList, HashMap | JDK1.2이후 | 필요시 java.util.Collections | 권장 |

|  |  |
| --- | --- |
| synchronized | non-synchronized |
|  | java.util.TreeMap |
| java.util.Hashtable java.util.concurrent.ConcurrentHashMap | java.util.HashMap |
|  | java.util.LinkedHashMap |
|  | java.util.IdentityHashMap |
|  | java.util.EnumMap |

**5. Collection API 의 선택**

아래의 특징에서 동작속도와 기능, 동기화여부등을 고려해서 클래스를 사용하면 된다.

**[ LIST ]  
ArrayList**     동기화 X    가장빠름, null 허용  
**LinkedList**   동기화 X    느림  
**Vector**         동기화 O    ArrayList 보다 느림  
**Stack**           동기화 O    Vector와 동일한 속도, LIFO 가능

**[ SET ]**  
**HashSet**     동기화 X    가장 빠른 집합, HashMap보다 느림  
**TreeSet**      동기화 X    HashSet 보다 느림, 키가 정렬됨  
  
**[ MAP ]  
HashMap**    동기화 X    가장 빠름, null 값 허용  
**Hashtable**   동기화 O    HashMap보다 느림, 동기화, null 값 허용안함  
**TreeMap**     동기화 X    Hashtable과 HashMap보다 느림, 키가 정렬됨  
  
개발하다보면 Set은 거의 쓸일이 없는것 같다. Java로 개발하는 Array 타입의 프로그래밍에서 중복가능여부를 따지는 경우는 거의 없다. 결국 List 와 Map 중 원하는 것을 골라 쓰면된다. key 로 인한 저장 검색을 원한다면 Map을 쓰면되고,나머지 경우는 List 를 사용하면 된다.  
그렇다면, 동기화는 어디에 사용될까? 동기화는 운영체제에서 배운 크리티컬섹션과 같은 개념이다. 공유하는 자원에 대해서 하나의 쓰레드만이 해당 공유자원에접근할수 있도록 제한해주고, 나머지 쓰레드는 공유자원이 해제될때까지 기다린다. 동기화가 지원되는 Collection 객체를사용하면 굳히 메소드나 변수에 Synchronized 를 처리를 하지 않아도 된다.  
  
**List 에서의 선택 - ArrayList ? Vector ?**  
ArrayList 와 Vector 의 차이점은 동기화를 지원하느냐의 여부다. 동기화가 필요한 곳에는 Vector를 사용하면 되고, 대부분의 경우는 ArrayList를 사용하면 된다. 사용자가 동기화 처리한  ArrayList 보다는 Vector 가 빠르다.  
  
**Map 에서의 선택 - HashMap ? Hashtable ?**  
HashMap과 Hashtable 도 ArrayList 와 Vector와 유사하다. 동기화가 필요한 곳에는 Hashtable 을 사용하고,대부분의 경우는 HashMap을 사용하면 된다. 사용자가 동기화 처리한 HashMap 보다는 Hashtable이 빠르다.