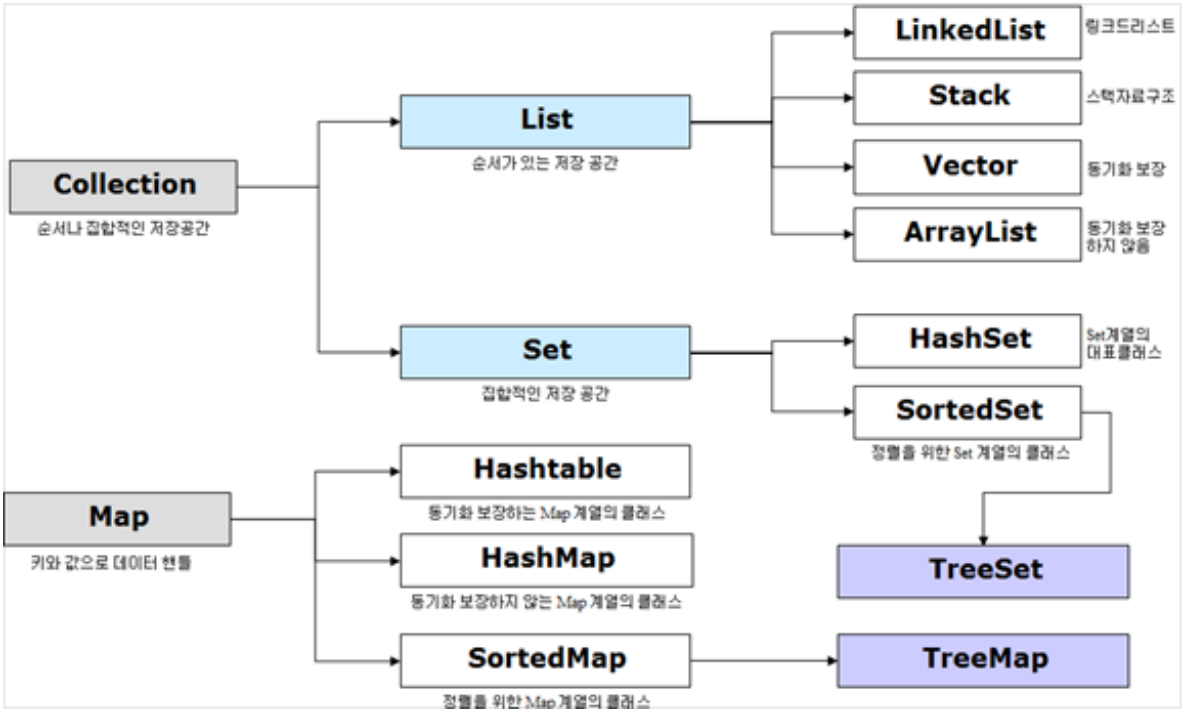
Java Collection Framework

자바 컬렉션 프레임워크란 Java에서 데이터를 저장하는 자료구조들을 한 곳에 모아 편리하게 관리하고 사용하기 위해 제공하는 것입니다. (ex_ List, Set, Map)



List 인터페이스와 Set 인터페이스를 설명하기 이전에 알고 넘어가야 할 점은, 컬렉션은 기본 데이터형이 아닌, 참조 데이터 형만 저장이 가능하다는 것입니다. 따라서 Collection에서의 데이터는 Object 타입의 객체로서 저장이 되는 것인데, 그렇다면 여기서 기본 데이터형은 어떻게 저장하고 관리할 수 있을까?

기본 데이터 형은 Wrapper 클래스를 이용하여 Boxing 시켜주거나, (Integer num = new Integer(5); 와 같은 코드로 구현 가능합니다.

📌List

1. 동일한 데이터의 중복을 허용한다
2. 데이터 저장 순서가 유지된다.
3. 힙(heap) 영역 내에서 List는 객체를 일렬로 늘어놓은 구조를 하고 있다
4. 객체를 인덱스로 관리하기 때문에 객체를 저장하면 자동으로 인덱스가 부여되고 인덱스로 객체를 검색, 삭제할 수 있다. 이 때 List 컬렉션은 객체 자체를 저장하여 인덱스를 부여하는 게 아니라, 해당하는 인덱스에 객체의 주소값을 참조하여 저장한다.
5. List 컬렉션에서 공통적으로 사용가능한 추가, 검색, 삭제 메소드를 갖고있다.

ArrayList

ArrayList는 List 인터페이스의 구현 클래스로, 여기서의 객체는 인덱스로 관리됩니다.
ArrayList에 객체를 추가하면 객체가 인덱스로 관리되는 것입니다.
이 점은 일반 배열과 별 다를바 없지만, 자바에서 배열은 초기화 시 그 크기가 고정되어야 하고 사용 중에는 크기를 변경할 수 없다는 점에서 ArrayList는 가치가 있습니다.

위에서 언급했던 저장소의 용량 확장에는 ArrayList<E>의 단점이라고 부를수 있는 문제점이 있습니다.
바로, 저장소의 용량을 늘리는 과정에서 많은 시간이 소요 된다는 점 입니다.
저장 공간 부족으로 ArrayList의 용량을 늘리게 되는 경우, 기존의 ArrayList에 추가하는 것이 아닌, 확장된 크기의 ArrayList를 새로 생성하고, 그 새로 생성된 ArrayList에 기존의 ArrayList의 값들을 복사해주는 과정을 거칩니다.
그리고 기존의 ArrayList는 gc에 의해 메모리에서 제거됩니다.
따라서 ArrayList에서 용량을 늘린다는 것은 새로운 배열 인스턴스의 생성과 기존 데이터의 복사가 필요한 번거로운 작업이 되는 것입니다.

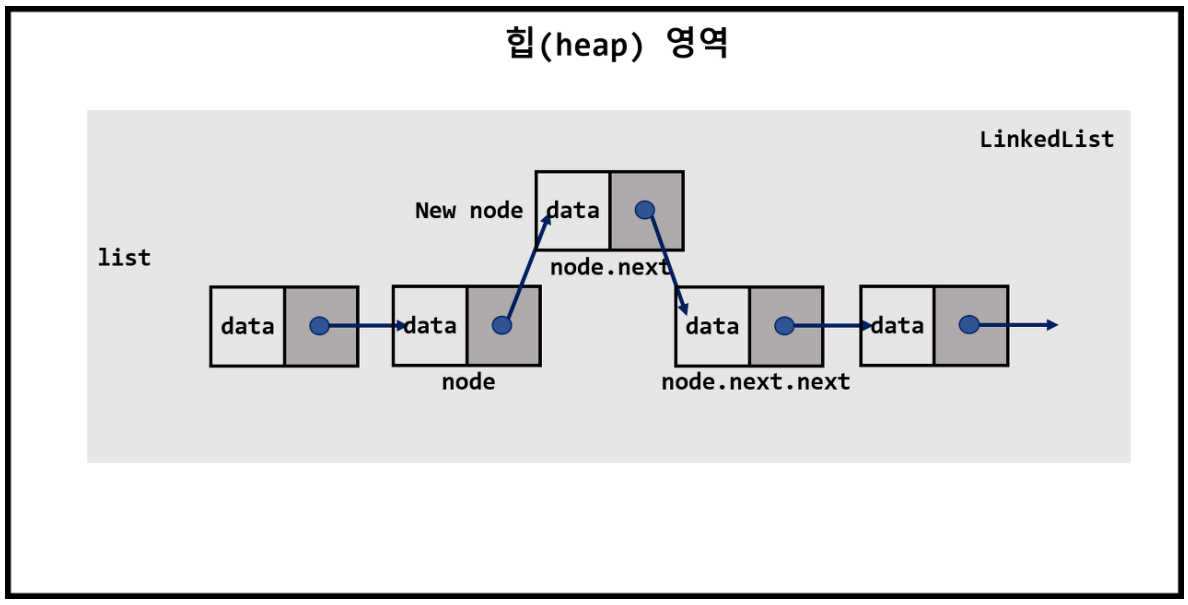
Vector

Vector는 ArrayList와 동일한 내부 구조를 가지고 있습니다
Vector를 생서앟기 위해서는 지정할 객체 타입을 타입 파라미터로 표기하고 기본 생성자를 호출하면 됩니다.
"List<E> list = new Vector<E>();"

ArrayList와 다르게, Vector는 동기화된 메소드로 구성되어 있기 때문에 멀티 스레드가 동시에 이 메소드들을 실행할 수 없고, 하나의 스레드가 실행을 완료해야만 다른 스레드가 실행 할 수 있습니다.
따라서 멀티 스레드 환경에서 안전하게 객체를 추가, 삭제할 수 있습니다.
객체를 추가하고 삭제하고 가져오는 메소드는 ArrayList 코드와 같기 때문에 Vector 예제를 따로 올리지 않겠습니다.

LinkedList

LinkedList는 List 구현 클래스이므로 ArrayList와 사용하는 메소드는 같지만 내부 구조는 완전 다릅니다.
ArrayList는 내부 배열 객체를 저장하서 인덱스로 관리하지만, LinkedList는 양방향 포인터 구조로, 각각마다 인접하는 참조를 링크해서 체인처럼 관리합니다.



따라서, LinkedList는 특정 인덱스의 객체를 제거하거나 삽입하면, 앞 뒤 링크만 변경되고 나머지 링크는 변경되지 않습니다.
그러므로 중간에 삽입/삭제가 빈번하게 일어날 수록 LinkedList를 쓰는것이 효율적입니다.
반대로, 순차적인 삽입/삭제가 빈번하게 일어나면 ArrayList를 사용하는 것이 효율적입니다.

LinkedList를 생성할 때, 처음에는 어떠한 링크도 만들어지지 않기 때문에 내부적으로 비어있습니다.
아래와 같은 코드로 생성할 수 있습니다.
"List<E> list = new LinkedList<E>();"

자바에서 LinkedList 클래스는 스택과 큐를 구현하는 데에 자주 쓰입니다.
그 중 큐는 자바에서 일반적으로 두 가지 방법으로 구현됩니다.
배열을 사용하거나 구현하거나, LinkedList나 ArrayList 클래스를 사용합니다.

스택에 대한 코드도 LinkedList클래스를 활용하여 구현할 수 있지만 큐와 사용하는 메소드만 다를 뿐 전반적인 흐름은 같습니다.
Stack의 자료구조를 위해서는 push(), pop(), peek() 메소드를 활용하면 쉽게 구현이 가능합니다.

Set 인터페이스

1. 데이터의 저장 순서를 유지하지 않습니다.
2. 같은 데이터의 중복 저장을 허용하지 않습니다. 따라서, null도 하나의 null만

저장 가능합니다.

3. Set 컬렉션은 List 컬렉션처럼 인덱스로 객체를 검색해서 가져오는 메소드가 없습니다. 대신 전체 객체를 대상으로 한 번 씩 다 가져오는 반복자, Iterator를 제공합니다.

```
Set<String> setExample = new...;
Iterator<String> iterator = setExample.iterator();
```

이 코드를 구현하여 Iterator 객체를 통해 사용할 수 있습니다.

```
Set<String> setExample = new...;
Iterator<String> iterator = setExample.iterator();

while(iterator.hasNext()){
    String getin = iterator.next();
}
```

보통 위와 같은 방식으로, Iterator 인터페이스에 선언된 hasNext()와 next() 메소드를 사용하여 구현합니다.

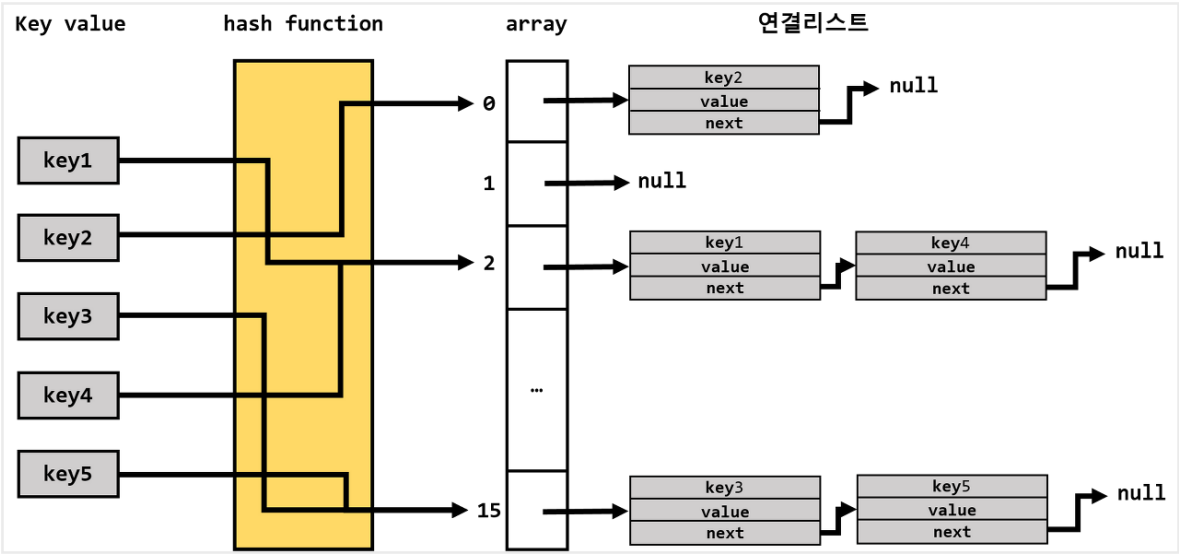
Set 인터페이스를 구현한 주요 클래스는 3개 있습니다.

Class	특징
HashSet	순서가 필요없는 데이터를 hash table에 저장, Set 중에 가장 성능이 좋다.
TreeSet	저장된 데이터의 값에 따라 정렬됨, red-black tree타입으로 값이 저장됨. HashSet 보다 성능이 느림
LinkedHashSet	연결된 목록 타입으로 구현된 hash table에 데이터 저장. 저장된 순서에 따라 값이 정렬되나 셋 중 가장 느림

3개의 클래스의 성능 차이는 클래스 때문인데, 셋 중 HashSet 이 특히 큰 dataset에서, 별도의 정렬작업이 없기 때문에 가장 빠릅니다.
또한 JDK 1.2부터 제공된 HashSet 클래스는 해시 알고리즘을 사용하였기 때문에 매우 빠른 검색 속도를 가집니다.

HashSet

해시 알고리즘이란 해시 함수(hash function)를 사용하여 데이터를 해시 테이블에 저장하고, 다시 그것을 검색하는 알고리즘입니다.



자바에서 해시 알고리즘을 이용한 자료구조는 위의 그림과 같이 배열과 연결리스트로 구현됩니다.
저장할 key값과 value를 넣으면 해시함수는 int index = key.hashCode() % capacity 연산으로 배열의 인덱스를 구하여 해당 인덱스에 저장된 연결 리스트에 데이터를 저장하게 됩니다.
예를 들어, key = 16이라면, hashCode() 메소드가 해당하는 int값을 그대로 반환하며, 16크기의 배열이 존재하므로 이 key의 인덱스는 16%16 = 0 이 됩니다.
따라서 첫번째 요소에 연결된 연결리스트에서 검색을 시작합니다.

HashSet 기본 생성자
Set<E> set = new HashSet<E>();

HashSet에서는 순서 없이, 동일한 객체의 중복 저장 없이 저장을 수행한다는 점을 언급합니다.
따라서 add() 메소드를 사용하여 해당 HashSet에 이미 존재하고 있는 요소를 추가하려면, 해당하는 요소를 바로 저장하지 않고 내부적으로 객체의 hashCode() 메소드와 equals() 메소드를 호출하고 검사합니다.
이 때 사용하는 hashCode() 와 equals() 코드는 자신이 정의한 클래스 인스턴스에 대해 프로그래머가 직접 오버라이딩하여 구현할 수 있는데, 그 흐름을 이해하기 위해 먼저 String 클래스에서 오버라이딩된 두 메소드의 정의를 짚어 보겠습니다.

문자열을 HashSet에 저장할 경우, 같은 문자열을 갖는 String 객체는 동등한

객체로, 다른 문자열을 갖는 String 객체는 다른 객체로 간주됩니다.
그 이유는 String 클래스가 hashCode() 와 equals()메소드를 오버라이딩하여, 같은 문자열일 경우 hashCode()의 리턴값을 같게, equals()의 리턴값을 true로 나오도록 구현해놓았기 때문입니다.

Map 인터페이스

Map 컬렉션에는 키(key)와 값(value)으로 구성된 Entry 객체를 저장하는 구조를 가지고 있습니다.
여기서 키와 값은 모두 객체입니다. 값은 중복 저장이 가능하지만, 키는 중복저장이 불가능합니다. Set과 마찬가지로, Map 컬렉션에서는 키 값의 중복 저장이 허용되지 않지만, 만약 중복저장 시 먼저 저장된 값은 저장되지 않은 상태가 됩니다. 즉, 기존 값은 없어지고 새로운 값으로 대체되는 것입니다.

HashMap

HashMap은 Map 인터페이스 구현을 위해 해시테이블을 사용한 클래스입니다.
또한 중복을 허용하지 않고 순서를 보장하지 않습니다.
중요한 점은, HashTable과 다르게 HashMap은 키와 값으로 null값이 허용된다는 것입니다.

HashMap의 생성자
Map<K,V> map = new HashMap<K,V>();

코드 예제를 보면서 공부하는것은 참고한 블로그를 확인해주세요
https://joooootopia.tistory.com/13