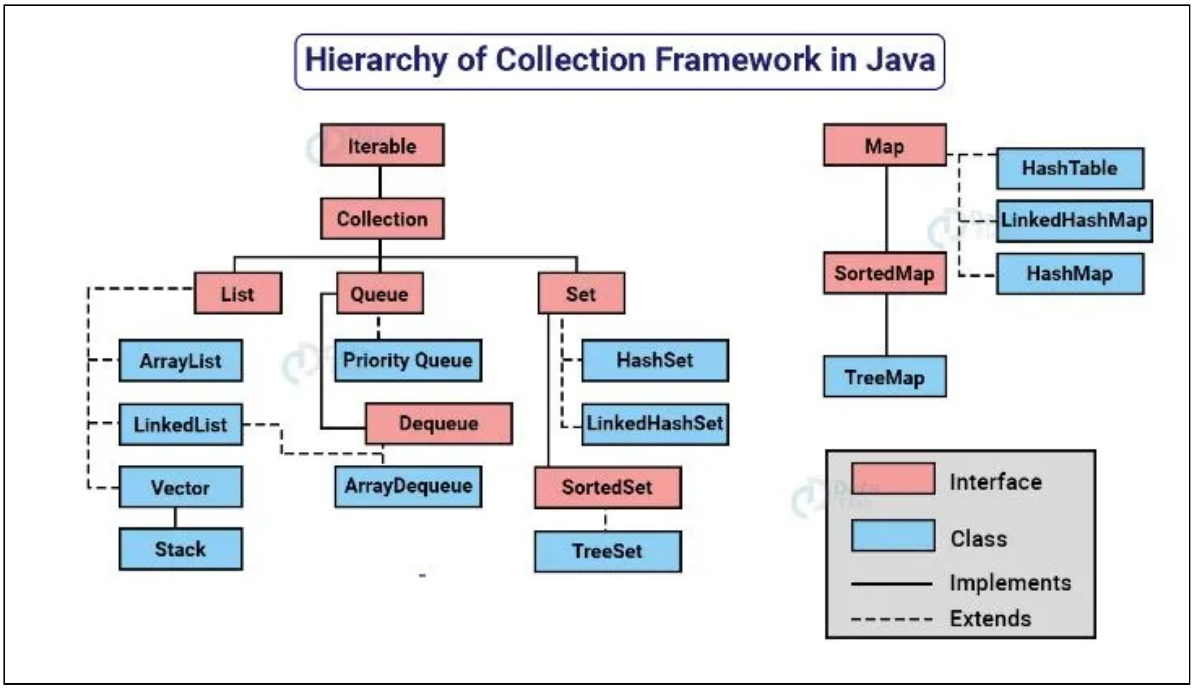
[TODO] 콜렉션

- 컬렉션 프레임워크(collection framework)란 **다수의 데이터를 쉽고 효과적으로 처리할 수 있는 표준화된 방법**을 제공하는 클래스 및 인터페이스의 집합
  - 데이터를 저장하는 자료 구조와 데이터를 처리하는 알고리즘을 구조화하여 클래스로 구현해 놓은 것
- 컬렉션 프레임워크 클래스는 제네릭을 사용하므로 **객체만 저장 가능**하며 원시형 값들을 저장하기 위해서 대응하는 Wrapper 클래스를 저장
- 컬렉션 프레임워크에서는 데이터를 저장하는 자료 구조에 따라 다음과 같은 핵심이 되는 주요 인터페이스를 정의
  - List 인터페이스
  - Set 인터페이스
  - Map 인터페이스

각 인터페이스의 특징

인터페이스	설명	구현 클래스
List	순서가 있는 데이터의 집합으로, 데이터의 중복을 허용	Vector, ArrayList, LinkedList, Stack, Queue
Set	**순서가 없는** 데이터의 집합으로, 데이터의 **중복을 허용하지 않음**	HashSet, TreeSet
Map	**키와 값의 한 쌍**으로 이루어지는 데이터의 집합으로, 순서가 없으며 키의 중복을 허용하지 않음	HashMap, TreeMap, Hashtable, Properties

콜렉션 프레임워크 상속/구현 구조



- Collection 인터페이스
  - List와 Set 인터페이스의 공통된 부분을 Collection 인터페이스에서 정의하고, 두 인터페이스는 그것을 상속 받음
    - 단, Map 인터페이스는 데이터 저장 방식이 달라서 Collection 인터페이스와 무관하게 작동함
  - 따라서 Collection 인터페이스는 컬렉션을 다루는데 가장 기본적인 동작들을 정의하고, 그것을 메소드로 제공함

Collection 인터페이스 메서드

메소드	설명
boolean add(E e)	해당 컬렉션(collection)에 전달된 요소를 추가
void clear()	해당 컬렉션의 모든 요소를 제거
boolean contains(Object o)	해당 컬렉션이 전달된 객체를 포함하고 있는지를 확인
boolean equals(Object o)	해당 컬렉션과 전달된 객체가 같은지를 확인
boolean isEmpty()	해당 컬렉션이 비어있는지를 확인
Iterator<E> iterator()	해당 컬렉션의 반복자(iterator)를 반환
boolean remove(Object o)	해당 컬렉션에서 전달된 객체를 제거
int size()	해당 컬렉션의 요소의 총 개수를 반환
Object[] toArray()	해당 컬렉션의 모든 요소를 Object 타입의 배열로 반환

List 컬렉션 클래스

- List 인터페이스를 구현한 모든 List 컬렉션 클래스는 다음과 같은 특징을 가집니다.

- 요소의 **저장 순서가 유지**됨
- 같은 요소의 **중복 저장을 허용**함

List 인터페이스 메서드

메소드	설명
boolean add(E e)	해당 리스트(list)에 전달된 요소를 추가
void add(int index, E e)	해당 리스트의 특정 위치에 전달된 요소를 추가
void clear()	해당 리스트의 모든 요소를 제거
boolean contains(Object o)	해당 리스트가 전달된 객체를 포함하고 있는지를 확인
boolean equals(Object o)	해당 리스트와 전달된 객체가 같은지를 확인
E get(int index)	해당 리스트의 특정 위치에 존재하는 요소를 반환
boolean isEmpty()	해당 리스트가 비어있는지를 확인
Iterator<E> iterator()	해당 리스트의 반복자(iterator)를 반환
boolean remove(Object o)	해당 리스트에서 전달된 객체를 제거
boolean remove(int index)	해당 리스트의 특정 위치에 존재하는 요소를 제거
E set(int index, E e)	해당 리스트의 특정 위치에 존재하는 요소를 전달받은 객체로 대체
int size()	해당 리스트의 요소의 총 개수를 반환
Object[] toArray()	해당 리스트의 모든 요소를 Object 타입의 배열로 반환

- 대표적인 List 컬렉션 구현 클래스
  - **ArrayList**
  - LinkedList
  - Vector
  - Stack

Map 컬렉션 클래스

- Map 인터페이스는 Collection 인터페이스와는 다른 저장 방식을 가짐
  - Map 인터페이스를 구현한 Map 컬렉션 클래스들은 **키와 값을 하나의 쌍으로 저장하는 방식(key-value 방식)을 사용**합니다.
    - 여기서 **키(key)란 실질적인 값(value)을 찾기 위한 이름의 역할**을 합니다.
- Map 인터페이스를 구현한 모든 Map 컬렉션 클래스는 다음과 같은 특징을 가집니다.
  - 요소의 저장 순서를 유지하지 않음
  - 키의 중복을 허용하지 않음
    - 키가 중복되는지 여부는 hashCode 및 equals 메서드를 통해서 검사
- 대표적인 Map 컬렉션 구현 클래스

    - **HashMap**
    - Hashtable
    - TreeMap
- Map 인터페이스 공통 메서드

Set 컬렉션 클래스

- Set 인터페이스를 구현한 모든 Set 컬렉션 클래스는 다음과 같은 특징을 가집니다.
    - 요소의 저장 순서를 유지하지 않음
    - 같은 요소의 **중복 저장을 허용하지 않음**
        - 요소가 중복되는지 여부는 hashCode 및 equals 메서드를 통해서 검사
- 대표적인 Set 컬렉션 구현 클래스
    - **HashSet**
    - TreeSet
- Set 인터페이스 공통 메서드

Set 인터페이스 메서드

메소드	설명
boolean add(E e)	해당 집합(set)에 전달된 요소를 추가
void clear()	해당 집합의 모든 요소를 제거
boolean contains(Object o)	해당 집합이 전달된 객체를 포함하고 있는지를 확인
boolean equals(Object o)	해당 집합과 전달된 객체가 같은지를 확인
boolean isEmpty()	해당 집합이 비어있는지를 확인
Iterator<E> iterator()	해당 집합의 반복자(iterator)를 반환
boolean remove(Object o)	해당 집합에서 전달된 객체를 제거
int size()	해당 집합의 요소의 총 개수를 반환
Object[] toArray()	해당 집합의 모든 요소를 Object 타입의 배열로 반환

Iterable, Iterator 인터페이스

- Collection 인터페이스는 Iterable 인터페이스를 상속
- Iterable 인터페이스를 구현하는 클래스는 Iterator 인터페이스 타입값을 반환하는 **iterator 추상 메서드를 구현**해야 함
- 자바의 컬렉션 프레임워크는 **컬렉션에 저장된 요소를 읽어오는 방법을 Iterator 인터페이스로 표준화**

Iterator 인터페이스 메서드

메소드	설명
boolean hasNext()	해당 이터레이션(iteration)이 다음 요소를 가지고 있으면 true를 반환하고, 더 이상 다음 요소를 가지고 있지 않으면 false를 반환함.
E next()	이터레이션(iteration)의 다음 요소를 반환함.
default void remove()	해당 반복자로 반환되는 마지막 요소를 현재 컬렉션에서 제거함. (선택적 기능)

Collections 유틸 클래스

- Collection 인터페이스가 아니고 Collection**s** 클래스임
- 클래스 메서드(=static 메서드)로 구성되어 있는 클래스이며 메서드는 **콜렉션 객체를 대상으로 연산을 수행하고 새로운 콜렉션을 반환**함

```
When teaching the concept of the Collection Framework in Java, it's essential to cover the following topics to provide a comprehensive understanding:

1. Introduction to the Collection Framework:
   - The purpose of the Collection Framework
   - Advantages of using the Collection Framework over arrays and other data structures
   - Overview of the main interfaces and classes in the Collection Framework

2. Basic interfaces and their implementations:
   - `Collection` interface: The root interface for all collection types.
   - `List` interface: An ordered collection (sequence) that allows duplicate elements. Examples: `ArrayList`, `LinkedList`.
   - `Set` interface: A collection that does not allow duplicate elements. Examples: `HashSet`, `LinkedHashSet`, `TreeSet`.
   - `Map` interface: A collection of key-value pairs where keys are unique. Examples: `HashMap`, `LinkedHashMap`, `TreeMap`.

3. Iterator and Iterable interfaces:
   - The `Iterator` interface and its usage in traversing collections.
```

   - The `Iterable` interface and its relationship with the enhanced for loop (for-each loop).

4. Collections utility class:
   - The `Collections` class and its various utility methods for sorting, searching, reversing, and other common operations on collections.

5. The Queue and Deque interfaces:
   - `Queue` interface: A collection designed for holding elements prior to processing. Examples: `LinkedList`, `PriorityQueue`.
   - `Deque` interface: A double-ended queue that allows elements to be added or removed at both ends. Example: `ArrayDeque`.

6. The SortedSet and SortedMap interfaces:
   - `SortedSet` interface: A `Set` that maintains its elements in sorted order. Example: `TreeSet`.
   - `SortedMap` interface: A `Map` that maintains its keys in sorted order. Example: `TreeMap`.

7. Concurrent collection classes:
   - Overview of thread-safe collections in the `java.util.concurrent` package.
   - Examples: `ConcurrentHashMap`, `CopyOnWriteArrayList`, `CopyOnWriteArraySet`.

8. Generics and the Collection Framework:
   - The importance of using generics with collections for type safety.
   - Basic usage of generics with the Collection Framework interfaces and classes.

9. Comparison and Comparator interfaces:
   - The `Comparable` interface and its usage for natural ordering of elements.
   - The `Comparator` interface for custom ordering of elements.

10. Best practices and performance considerations:
    - Choosing the right collection type for specific use cases.
    - Performance considerations, such as time and space complexity, when working with different collection classes.
    - Understanding the importance of equals and hashCode methods when using collections.

By covering these topics, you will provide your students with a solid understanding of the Java Collection Framework and its various components, preparing them to work effectively with collections in their projects.