제20장
컬렉션 프레임워크-Part1

1. 컬렉션 프레임워크란?

❖ 컬렉션 프레임워크(Collection Framework)

- 컬렉션
  - 사전적 의미로 요소(객체)를 수집해서 저장하는 것
- 배열의 문제점
  - 저장할 수 있는 객체 수가 배열을 생성할 때 결정(정적)
    → 불특정 다수의 객체를 저장하기에는 문제점이 많다.
  - 객체 삭제했을 때 해당 인덱스가 비게 됨
    → 날알이 덤성 덤성 빠진 옥수수와 같은 배열이 된다.
    → 객체를 저장하려면 어디가 비어있는지 확인 필요하다.

배열


```
//길이가 10인 배열 생성
Product[] arr = new Product[10]

//객체 추가
arr[0] = new Product( "Model1" );
arr[1] = new Product( "Model1" );

//객체 검색
Product model1 = arr[0];
Product model2 = arr[1];

//객체 삭제
arr[0] = null;
arr[1] = null;
```

삭제가 된 곳 즉 null을 찾아서 다시 저장을 해야하는 코드가 필요하다. 객체가 10개가 아니라 만개 ~ 수만 개라면 엄청난 시간이 소요될 것이다. 이게 배열의 단점이다. 정적인 것도 당연히 문제이고.

❖ 컬렉션 프레임워크(Collection Framework)

- 객체들을 효율적으로 추가, 삭제, 검색할 수 있도록 제공되는 컬렉션 라이브러리
- 프레임웍(틀, 작업) = 라이브러리(기능) + 프로그래밍 방식
- 표준화, 정형화된 프로그램 방식, java.util 패키지에 포함, JDK 1.2부터 적용
- 인터페이스를 통해서 정형화된 방법으로 다양한 컬렉션 클래스 이용

1. 컬렉션 프레임워크란?

❖ 컬렉션 프레임워크의 주요 인터페이스

- **List(ex. 대기자 명단)**
  - 배열과 유사하게 인덱스로 관리

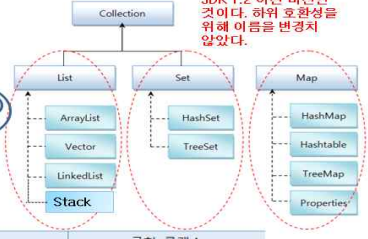
0	1	3	...	n-1
객체1	객체2	객체1		객체n

- **Set**
  - 집합과 유사(ex. 양의 정수집합 소수의 집합)

- **Map**
  - 키와 값의 쌍으로 관리(엔트리맵) ex.)아이디-패스워드, 우편번호 등

K1	K2	K3	...	Kn
객체1	객체2	객체1		객체n

인터페이스의 이름이 첼미사로 없는 것들은 JDK 1.2 이전 버전인 것이다. 하위 호환성을 위해 이름을 변경치 않았다.



인터페이스 분류		특징	구현 클래스
Collection	List 계열	- 순서를 유지하고 저장 - 중복 저장 가능	ArrayList, Vector, LinkedList, Stack
	Set 계열	- 순서를 유지하지 않고 저장 - 중복 저장 안됨	HashSet, TreeSet
Map 계열		- 키와 값의 쌍으로 저장 - 키는 중복 저장 안됨 - 순서를 유지 안함	HashMap, Hashtable, TreeMap, Properties

**** 위의 표의 내용은 컬렉션프레임웍에서 상당히 중요한 개념이라서 자바 개발자라면 외울 정도로 학습해야 한다.

1-1. Collection 인터페이스의 메서드

* Collection인터페이스의 주요 메서드(List, Set의 조상)

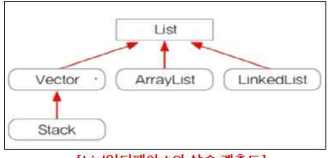
메서드	설 명
boolean add(Object o) boolean addAll(Collection c)	지정된 객체(o) 또는 Collection(c) 의 객체들을 Collection에 추가한다.
void clear( )	Collection의 모든 객체를 삭제한다.
boolean contains(Object o) boolean containsAll(Collection c)	지정된 객체(o) 또는 Collection의 객체들이 Collection에 포함되어 있는지 확인한다.
boolean equals(Object o)	동일한 Collection인지 비교한다.
int hashCode( )	Collection의 hash code를 반환한다.
boolean isEmpty( )	Collection이 비어있는지 확인한다.
Iterator iterator( )	Collection의 Iterator를 얻어서 반환한다.
boolean remove(Object o)	지정된 객체를 삭제한다.
boolean removeAll(Collection c)	지정된 Collection에 포함된 객체들을 삭제한다.
boolean retainAll(Collection c)	지정된 Collection에 포함된 객체만을 남기고 다른 객체들은 Collection 에서 삭제한다. 이 작업으로 인해 Collection에 변화가 있으면 true를 그렇지 않으면 false를 반환한다.
int size( )	Collection에 저장된 객체의 개수를 반환한다.
Object[ ] toArray( )	Collection에 저장된 객체를 객체배열(Object[ ])로 반환한다.
Object[ ] toArray(Object[ ] a)	지정된 배열에 Collection의 객체를 저장해서 반환한다.

Collection인터페이스는 공통적인 추상메서드를 제공함으로써 정형화(통일화)된 프로그램을 할 수 있도록 하는데 목적이 있다. 또한, 매개변수가 Collection이라는 뜻은 당연히 이제 알 것이다. 바로 List, Set인터페이스를 구현한 클래스들은 다 매개변수로 받겠다 라는 의미인 것이다.

2. List 컬렉션

❖ List인터페이스를 구현한 컬렉션의 특징(순서유지, 중복허용)

- 특징
  - 인덱스로 관리
  - 중복해서 객체 저장 가능
- 구현 클래스
  - ArrayList
  - Vector, Stack
    (Stack은 Vector의 자손)
  - LinkedList



[List인터페이스의 상속 계층도]

만약, 1번인덱스에 객체가 저장되어있는데, 다시 삽입을 하게 되면, 원래 객체는 2번 인덱스로 밀려난다.
그냥 추가를 하게 되면 마지막에 추가가 됨을 알자.

2. List 컬렉션

❖ **List인터페이스의 주요 메서드**

메서드	설 명
void add(int index, Object element) boolean addAll(int index, Collection c)	지정된 위치(index)에 객체(element) 또는 컬렉션에 포함된 객체들을 추가한다.
Object get(int index)	지정된 위치(index)에 있는 객체를 반환한다.
int indexOf(Object o)	지정된 객체의 위치(index)를 반환한다. (List의 첫 번째 요소부터 순방향으로 찾는다.)
int lastIndexOf(Object o)	지정된 객체의 위치(index)를 반환한다. (List의 마지막 요소부터 역방향으로 찾는다.)
ListIterator listIterator( ) ListIterator listIterator(int index)	List의 객체에 접근할 수 있는 ListIterator를 반환한다.
Object remove(int index)	지정된 위치(index)에 있는 객체를 삭제하고 삭제된 객체를 반환한다.
Object set(int index, Object element)	지정된 위치(index)에 객체(element)를 저장한다
void sort(Comparator c)	지정된 비교자(comparator)로 List를 정렬한다.
List subList(int fromIndex, int toIndex)	지정된 범위(fromIndex부터 toIndex)에 있는 객체를 반환한다.

Collection인터페이스에 비해 매개값인 index가 추가됨을 주목하자.

2. List 컬렉션

● 객체 추가, 찾기, 삭제

```
List<String> list = ...;    //List에 String만 저장하겠다
list.add("홍길동");          //맨끝에 추가
list.add(1, "신은혁");       //지정한 인덱스에 추가(삽입)
String str = list.get(1);   //인덱스로 객체 찾기
list.remove(0);             //인덱스로 객체 삭제
list.remove("신은혁");       //객체 삭제
```

0	1	2	...	n-1
홍길동	신은혁	null	null	객체n

0	1	2	...	n-1
신은혁	null	null	null	객체n

0	1	2	...	n-1
null	null	null	null	객체n

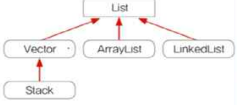
● 전체 객체를 대상으로 반복해서 얻기

```
List<String> list = ...;

for(int i=0; i<list.size(); i++)        ← 저장된 총 객체 수만큼 루핑(배열과 똑같다)
{
    String str = list.get(i);
}
```

```
for(String str : list)        ← 저장된 총 객체 수만큼 루핑(advanced for loop)
{
    ......;
}
```

2. List 컬렉션

❖ ArrayList

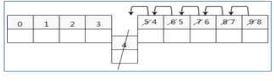


상속 계층도
Vector(구버전) = ArrayList(신버전)

<E>는 제네릭 타입으로 Element의 약자이다.(어떤타입이라도 상관없다)

List<String> list = ArrayList<String>(30);
30은 ArrayList 내부에 생성되는 배열의 기본크기

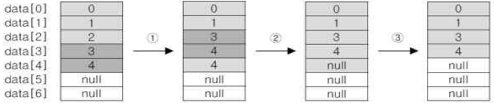
- **저장 용량(capacity)**
  - 초기 용량 : 10 (따로 지정 가능)
  - 저장 용량을 초과한 객체들이 들어오면 자동적으로 늘어난다.(기본적 10개가 또 증가됨)
- **특징**
  - 데이터 저장 공간을 배열을 사용한다.(배열기반)
  - 동기화 처리가 되어 있지 않다.(싱글스레드)
  - Vector는 동기화 처리 되어 있다.(멀티스레드)
- **객체 제거**
  - 바로 뒤 인덱스부터 마지막 인덱스까지 모두 앞으로 1씩 당겨진다.



2. List 컬렉션

● ArrayList에 저장된 객체의 삭제과정(1/2)

- ArrayList에 저장된 세 번째 데이터(data[2])를 삭제하는 과정. list.remove(2);를 호출



① 삭제할 데이터 아래의 데이터를 한 칸씩 위로 복사해서 삭제할 데이터를 덮어쓴다.(배열복사 발생)

```
System.arraycopy(data, 3, data, 2, 2)
```

data[3]에서 data[2]로 2개의 데이터를 복사하라는 의미이다.

② 데이터가 모두 한 칸씩 이동했으므로 마지막 데이터는 null로 변경한다.

```
data[size-1] = null;
```

③ 데이터가 삭제되어 데이터의 개수가 줄었으므로 size의 값을 감소시킨다.

```
size--;
```

②과 ③을 합쳐서 data[--size] = null; 을 하면 된다.

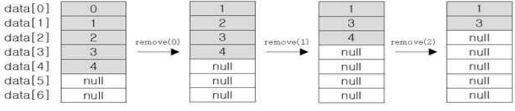
※ 마지막 데이터를 삭제하는 경우, ①의 과정(배열의 복사)은 필요 없다.(size값만 변경하면 된다)

2. List 컬렉션

● ArrayList에 저장된 객체의 삭제과정(2/2)

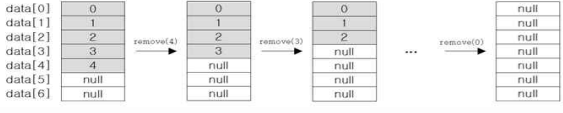
① ArrayList에 저장된 첫 번째 객체부터 삭제하는 경우(배열 복사 발생)

```
for(int i=0;i<list.size();i++)
    list.remove(i);
```



② ArrayList에 저장된 마지막 객체부터 삭제하는 경우(배열 복사 발생 안함) – 중요!! (clear()이용해도 된다)

```
for(int i=list.size()-1;i>=0;i--)
    list.remove(i);
```



2. List 컬렉션

❖ Vector

```
List<E> list = new Vector<E>();
```

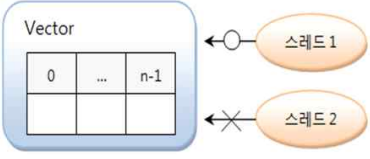
- 특징
  - Vector는 스레드 동기화(synchronization)가 되어 있기 때문에,
    멀티 스레드가 사용 가능하므로 복수의 스레드가 Vector에 접근해 객체를 추가, 삭제 하더라도 스레드에 안전(thread safe)하다.

ArrayList(비동기화)

```
public boolean add(...)
{...}
```

Vector(동기화)

```
public synchronized boolean add(...)
{...}
```



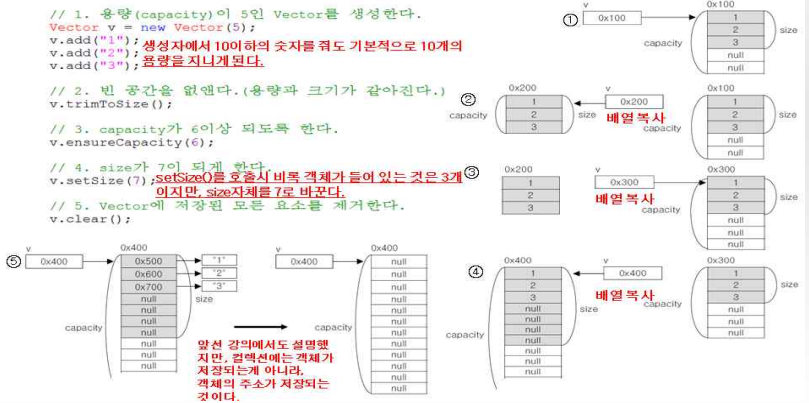
2. List 컬렉션

● Vector의 크기(size-객체수)와 용량(capacity-배열길이)

```
// 1. 용량(capacity)이 5인 Vector를 생성한다.
Vector v = new Vector(5);
v.add("1");
v.add("2");
v.add("3");

// 2. 빈 공간을 없앤다.(용량과 크기가 같아진다.)
v.trimToSize();

// 3. capacity가 6이상 되도록 한다.
v.ensureCapacity(6);

// 4. size가 7이 되게 한다.
v.setSize(7);
// 5. Vector에 저장된 모든 요소를 제거한다.
v.clear();
```

생성자에서 10이하의 숫자를 줘도 기본적으로 10개의 용량을 지니게 된다.

setSize()를 호출시 비록 객체가 들어 있는 것은 3개이지만, size자체를 7로 바꾼다.



말씀 강의에서도 설명했지만, 컬렉션에는 객체가 저장되는게 아니라, 객체의 주소가 저장되는 것이다.

2. List 컬렉션

● Vector를 직접 구현하기

① 객체를 저장할 객체배열(objArr)과 크기(size)를 저장할 변수를 선언

② 생성자 MyVector(int capacity)와 기본 생성자 MyVector()를 선언

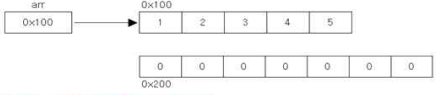
③ 메서드 구현

- int size() – 컬렉션의 크기(size, 객체배열에 저장된 객체의 개수)를 반환
- int capacity() – 컬렉션의 용량(capacity, 객체배열의 길이)을 반환
- boolean isEmpty() – 컬렉션이 비어있는지 확인
- void clear() – 컬렉션의 객체를 모두 제거(객체배열의 모든 요소를 null)
- Object get(int index) – 컬렉션에서 지정된 index에 저장된 객체를 반환
- int indexOf(Object obj) – 지정된 객체의 index를 반환(못찾으면 -1)
- void setCapacity(int capacity) – 컬렉션의 용량을 변경
- void ensureCapacity(int minCapacity) – 컬렉션의 용량을 확보
- Object remove (int index) – 컬렉션에서 객체를 삭제
- boolean add(Object obj) – 컬렉션에 객체를 추가

2. List 컬렉션

● ArrayList의 장점과 단점

▶ 장점 : 배열은 구조가 간단하고 데이터를 읽는 데 걸리는 시간 (접근 시간, access time)이 짧다.



▶ 단점 1 : 크기를 변경할 수 없다.

- 크기를 변경해야 하는 경우 새로운 배열을 생성 후 데이터를 복사 해야 함.
- 크기 변경을 피하기 위해 충분히 큰 배열을 생성하면, 메모리가 낭비됨.

▶ 단점 2 : 비순차적인 데이터의 추가, 삭제에 시간이 많이 걸린다.

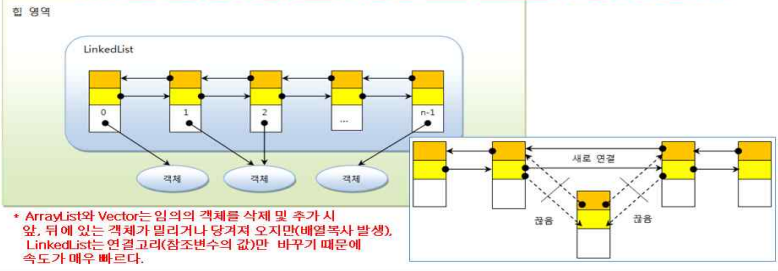
- 데이터를 추가하거나 삭제하기 위해, 다른 데이터를 옮겨야 함.(배열복사 O)
- 그러나 순차적인 데이터추가(끝에 추가)와 삭제(끝부터 삭제)는 빠름(배열복사 X)

2. List 컬렉션

❖ LinkedList

```
List<E> list = new LinkedList<E>();
```

- 특징
  - 인접 참조를 링크해서 체인처럼 관리
  - 특정 인덱스에서 객체를 제거하거나 추가하게 되면 바로 앞뒤의 링크만 변경
  - 빈번한 객체 삭제와 삽입이 일어나는 곳에서는 ArrayList보다 좋은 성능을 발휘한다.

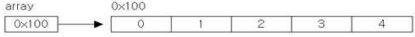


* ArrayList와 Vector는 임의의 객체를 삭제 및 추가 시 앞, 뒤에 있는 객체가 밀리거나 당겨져 오지만(배열복사 발생), LinkedList는 연결고리(참조변수의 값)만 바꾸기 때문에 속도가 매우 빠르다.

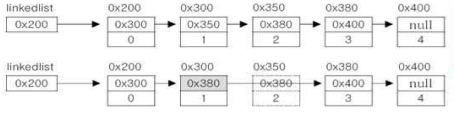
2. List 컬렉션

● LinkedList - 배열의 단점을 보완

- 배열과 달리 링크드 리스트는 불연속적으로 존재하는 데이터를 연결(link)



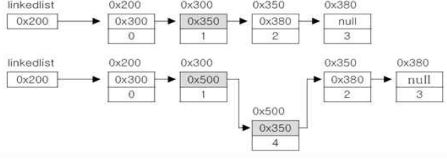
▶ 데이터의 삭제 : 단 한 번의 참조변경만으로 가능




```
class Node {
    Node next;
    Object obj;
}
```

1. Node : 다음 Node의 주소
2. Object : 저장할 데이터

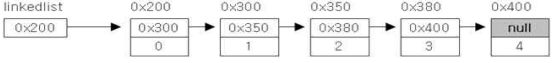
▶ 데이터의 추가 : 한번의 Node객체생성과 두 번의 참조변경만으로 가능



2. List 컬렉션

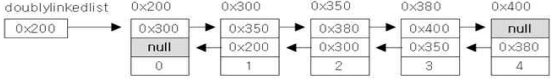
● LinkedList – 이중 연결 리스트

▶ 링크드 리스트(linked list) – 연결리스트. 데이터 접근성이 나쁨



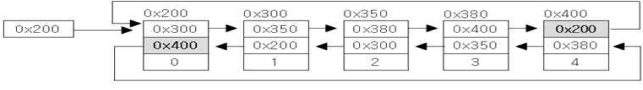

```
class Node {
    Node next;
    Object obj;
}
```

▶ 더블리 링크드 리스트(doubly linked list) – 이중 연결리스트, 접근성 향상
(LinkedList는 더블리 링크드리스트로 구현 되어 있음)




```
class Node {
    Node next;
    Node previous;
    Object obj;
}
```

▶ 더블리 써큘러 링크드 리스트(doubly circular linked list) – 이중 원형 연결리스트
(원래는 이것으로 구현 되어 있었지만, 효용성이 떨어져 더블리로 구현됨)



2. List 컬렉션

● ArrayList vs. LinkedList – 성능 비교

① 순차적으로 데이터를 추가/삭제 – ArrayList가 빠름

② 비순차적으로 데이터를 추가/삭제 – LinkedList가 빠름

③ 접근시간(access time) – ArrayList가 빠름

```
= 순차적으로 추가하기 =
ArrayList :406
LinkedList :606

= 순차적으로 삭제하기 =
ArrayList :11
LinkedList :46

= 중간에 추가하기 =
ArrayList :7382
LinkedList :31

= 중간에서 삭제하기 =
ArrayList :6694
LinkedList :380

= 접근시간테스트 =
ArrayList :1
LinkedList :432
```

n번째 데이터의 주소 = 배열의 주소 + (n-1) * 데이터 타입의 크기

컬렉션	읽기(접근시간)	추가 / 삭제	비 고
ArrayList	빠르다	느리다	순차적인 추가삭제는 빠름. 비효율적인 메모리사용
LinkedList	느리다	빠르다	데이터가 많을수록 접근성이 떨어짐

[상기 표는 필히 알아둘 개념]

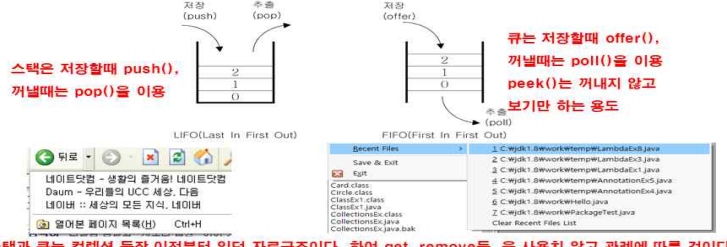
2. List 컬렉션

● 스택과 큐(Stack & Queue)

► 스택(Stack) : LIFO구조. 마지막에 저장된 것을 제일 먼저 꺼내게 된다.
ex) 수식계산, 수식괄호검사, undo/redo, 뒤로/앞으로(웹브라우져)

► 큐(Queue) : FIFO구조. 제일 먼저 저장한 것을 제일 먼저 꺼내게 된다.
ex) 최근 사용문서, 인쇄작업대기목록, 버퍼(buffer)

► 스택, 큐, 배열, 링크드리스트 등 이런 것들은 모두 자료구조(Data Structure)이다.

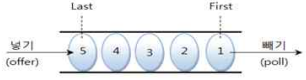


** 스택과 큐는 컬렉션 등장 이전부터 있던 자료구조이다. 하여 get, remove등 을 사용치 않고 관례에 따른 것이다.

2. List 컬렉션

❖ Queue 인터페이스

```
Queue queue = new LinkedList();
```



큐(FIFO)

- 특징
  - 선입선출(FIFO: First In First Out)
  - 응용 예: 작업 큐(스레드풀), 메시지 큐, 파이프, 수도호스..
  - 구현 클래스: LinkedList(폴 객체), ArrayDeque(사용 안함)

- 주요 메소드

리턴타입	메소드	설명
boolean	offer(E e)	주어진 객체를 넣는다.
E	peek()	객체 하나를 가져온다. 객체를 큐에서 제거하지 않는다.
E	poll()	객체 하나를 가져온다. 객체를 큐에서 제거한다.

감사합니다.