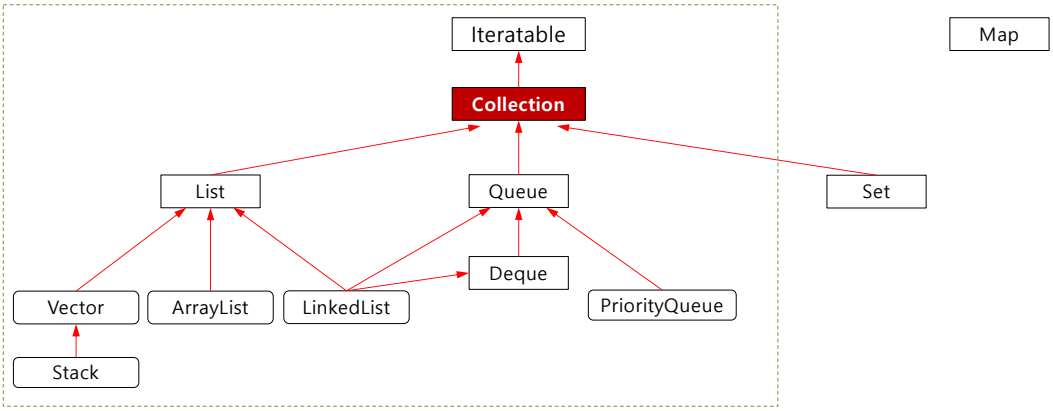
10 Collection Framework

Collection 시작하겠습니다.

Collection Framework

- Standardized design of classes that store data groups
- Consists of classes that provide an easy way to process multiple data



Covered in this course

2

Page 2

Collection Framework이라고 하는 것은
여러 가지 data group을 저장하기 위한 그런 class들을
standard화 (표준화) 해 놓은 그런 것이다 라는 거죠
여러 가지 interface들이 있고요
지금 네모로 되어 있는 것은 interface들이고
여기 동그랗게 돼 있는 것들은 class들입니다
multiple data를 저장하고 access하는 데 있어서
손쉬운 방법을 제공해 준다 라는 것이 되겠습니다
여기 보면 Collection이 사실 Iteratable이라는 interface의 Child인데요
그 다음에 Collection 밑에는 List, Queue, Set
이런 interface들이 있고
그 다음에 얘들과는 별도로 Map이라는 interface가 또 있습니다
Map은 이 Collection 안에 속하지는 않지만
Collection과 비슷한 그런 역할을 하기 때문에
같이 취급이 되구요
또 List 밑에는 Vector, ArrayList, LinkedList 이런 것들이 있고
Queue 밑에는 Deque, PriorityQueue 이런 게 있고요

그 다음에 Stack은 Vector로 implement되고 이런 식으로 돼 있습니다
이번 코스에서는 Set하고 Map은 제외하고요
요 부분, 즉 List와 Queue 부분만 다루도록 하겠습니다

Core Interfaces of Collection Framework

Interface	Features
List	Ordered set of data. Allow duplicate data. ex) Waiting List (Data with the same name can be duplicated)
	Class: ArrayList, LinkedList, Stack, Vector, ...
Set	A set of data that is out of order. Do not allow duplicate data ex) Set of four-legged animals = {dog, cat, bear, lion, ...}
	Class: HashSet, TreeSet, ...
Map	Set of data consisting of the tuple (key, value) Order not maintained, no duplicated key, value allows duplicates ex) Postal code (area, number), Local phone number (area, number)... (seoul, 02)
	Class: HashMap, TreeMap, HashTable, Properties, ...

3

Page 3

Collection Framework의 Core Interface들에 대한
서로 다른 점들에 대해서 먼저 얘기를 하면
List Interface는 ordered set of data
순서가 있다는 거죠. 순서가 있다는 것은
들어갈 때 sorting이 된다 이런 게 아니라
데이터를 집어 넣은 순서가 유지된다 이런 뜻이 되겠습니다
순서가 유지된다 그 다음에 duplicate data를 허락한다 그러니까
같은 데이터라도 여러 번 들어갈 수 있다 것이 되겠죠
waiting list 같은 건데요
이름이 같은 사람이 여러 명 있을 수 있으니까 duplicate 를 허락하고
그 다음에 waiting list에는 도착한 순서대로 기록이 돼야 되겠죠
그래서 order가 유지가 된다 라는게 되겠습니다
그래서 이 안에 class들은
ArrayList, LinkedList, Stack, Vector 이런 것들이 있겠습니다
그 다음에 이제 set 인데요
set은 우리가 여기서 자세히 다루지는 않지만
특징을 살펴보면 order가 없습니다 order가 없고

그러니까 집어 넣은 그 순서가 유지가 되지 않는다는 거죠
그러니까 이렇게 어떤 data를 Collection에
집어 넣을 수도 있고 그 다음 뺄 수도 있지만
그 순서는 유지가 안 된다
그게 몇 번째로 들어왔는지는 유지가 안 된다라는 뜻이 되겠습니다
그리고 그 duplicate 데이터를 허락하지 않는다 라는 뜻이 되겠습니다
그래서 set of 4족, 그러니까 4개의 발을 가진 animal의 set이다
그러면 dog, cat, bear, lion 이런 것들이 들어가 있겠죠
그래서 이런 것들이 뭐 duplicate 서는 안 되잖아요
그래서 그게 이제 set의 정의라고 볼 수 있겠고
class들은 HashSet, TreeSet 이런 것들이 있습니다
map은 tuple로 이루어진 그런 데이터다 라는 거죠
그래서 하나의 data item이 이렇게 쌍으로 되어 있어요
key하고 value. 그래서 순서는 유지되지 않고요 역시
그 다음에 key는 duplicate 되지 않고요
그 다음에 value는 중복을 허용한다 라는 뜻이 되겠습니다
예를 들면 postal code (우편번호) 같은 거는 area number 이렇게 돼 있는데
area는 key가 되겠고 number는 value 입니다.
그래서 서울은 우편번호가 얼마다 이런 거고요
local phone number는 area number 이런 식으로
서울 02 부산 051 뭐 이런 식으로 되는 거죠
그래서 class에 예는 HashMap, TreeMap, HashTable,
Properties 이런 것들이 있어요
그 중에 우리는 List를 중점으로 다룰 거고
그것과 유사한 Queue에 대해서 좀 알아볼 거고요

Methods in Interface Collection<E> (1/2)

Type	Method	Description
boolean	add(E e)	Ensures that this collection contains the specified element
boolean	addAll(Collection<? extends E> c)	Adds all of the elements in the specified collection to this collection
void	clear()	Removes all of the elements from this collection
boolean	contains(Object o)	Returns true if this collection contains the specified element.
boolean	containsAll(Collection<?> c)	Returns true if this collection contains all of the elements in the specified collection.
boolean	equals(Object o)	Compares the specified object with this collection for equality.
int	hashCode()	Returns the hash code value for this collection.
boolean	isEmpty()	Returns true if this collection contains no elements.
Iterator<E>	iterator()	Returns an iterator over the elements in this collection.
boolean	remove(Object o)	Removes a single instance of the specified element from this collection, if it is present (optional operation), return true when changed
boolean	removeAll(Collection<> c)	Removes all of this collection's elements that are also contained in the specified collection (optional operation), return true when changed

4

Page 4

그래서 interface Collection에는 공통적으로 가지게 되는
여러 개의 method들이 있습니다
그것들이 이제 child interface나 child class에 따라서
조금 바뀔 수도 있기 때문에
그렇지만 공통적으로 가지고 있는 것들은 이런 것들이 있다
이 Collection들은 전부 generic으로 되어 있거든요
generic interface로 되어 있기 때문에
이 Collection하고 다음에 <type>이 들어가도록 돼 있죠
그래서 이 generic의 class를 집어넣는 그게 이제 add가 되겠습니다
addAll은 여기에 Collection을 줘요
그래서 다른 Collection에 있는 모든 element를
다 한꺼번에 집어넣는 것이 되겠습니다
그 다음에 clear는 완전히 데이터를 다 삭제하는 거고요
contains는 주어진 Object o가 있으면 boolean true
아니면 false
contains all은 주어진 Collection이 다 그 안에 들어가 있으면 true
아니면 false

자 equals는 뭐 아시겠죠 Object o가 Collection 안에 속해 있으면 true
네, hashCode라는 이것도 이제 Object class에 정의가 되어 있기 때문에
사실은 모든 class에 전부 inherit가 되는
그런 일반적인 class 중에 하나입니다 toString이나 equals같이
자, 그 다음에 isEmpty는 empty면 true
아니면 false를 return하고요
iterator는 Collection에 속해있는 element들을 한바퀴 쭉
트래블스 하는데 사용될 수 있는 그런 것이고요
iterator라는 타입으로 되어 있습니다
이 iterator에 대해서는 뒷부분에서 좀 다를 것이고요
그 다음에 remove는 주어진 object를 remove하는 거고
그 다음에 removeAll은 Collection에 들어있는 모든 것들을
다 remove 하는 것이 되겠습니다

Methods in Interface Collection<E> (2/2)

Type	Method	Description
boolean	retainAll(Collection<?> c)	Retains only the elements in this collection that are contained in the specified collection. Return true if any change by this call
int	size()	Returns the number of elements in this collection.
Object[]	toArray()	Returns an array containing all of the elements in this collection.

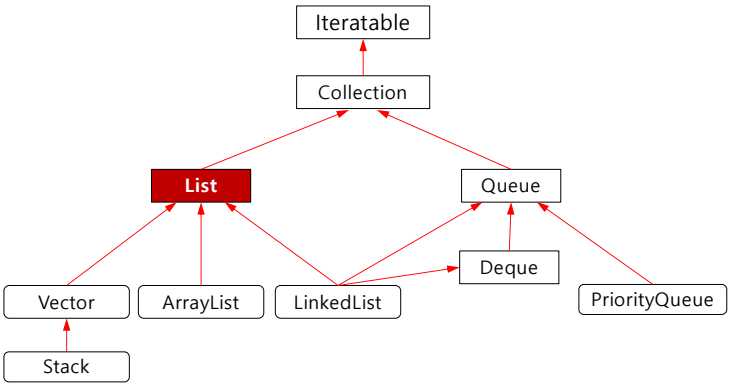
5

Page 5

그 다음에 retainAll은 주어진 Collection에도 들어있는 것들만 남기고
다른 건 다 삭제하는 거에요
말하자면 retain all 을 call 한 Collection과 주어진 Collection 간에
intersection을 구한다 이렇게 보면 되겠죠
그 다음에 size는 collection의 데이터 개수를 return하고요
toArray는 collection에 들어있는 모든 element를
일반적인 object의 array 즉 collection에 속해 있는 type의 array로
만들어주는 그런 method가 되겠습니다

Interface List<E>

- Order (O): preserving order in which data came into the list
- Duplication (O): allow duplication of the same data in the list



6

Page 6

자 그 다음에 List interface인데요
List interface는 아까 얘기했듯이 order하고 duplication을 둘 다
허용한다 그랬어요 그래서 데이터가 List에 도착한 순서대로
즉 add된 순서를 그대로 유지하고요 그 다음에 중복을 허용한다는 게 되겠죠
그래서 List는 여기 있습니다
Collection의 아래 있는 하나의 interface가 되겠습니다

Methods in Interface List<E> (1/3)

Type	Method	Description
void	**add**(int index, **E** element)	Inserts the specified element at the specified position in this list
boolean	**add**(**E** e)	Appends the specified element to the end of this list
boolean	**addAll**(int index, **Collection**<? extends **E**> c)	Inserts all of the elements in the specified collection into this list at the specified position
boolean	**addAll**(**Collection**<? extends **E**> c)	Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator
void	**clear**()	Removes all of the elements from this list
boolean	**contains**(**Object** o)	Returns true if this list contains the specified element.
boolean	**containsAll**(**Collection**<?> c)	Returns true if this list contains all of the elements of the specified collection.
static <E> **List**<E>	**copyOf**(**Collection**<? extends E> coll)	Returns an unmodifiable List containing the elements of the given Collection, in its iteration order.

7

Page 7

List의 method들도 parent인 Collection의 method들과 거의 같습니다
그래서 add들이 있고 addAll, clear, contains,
containsAll, copyOf, 그 다음에 equals,

Methods in Interface List<E> (2/3)

Type	Method	Description
boolean	**equals**(**Object** o)	Compares the specified object with this list for equality.
E	**get**(int index)	Returns the element at the specified position in this list.
int	**hashCode**()	Returns the hash code value for this list.
int	**indexOf**(**Object** o)	Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
boolean	**isEmpty**()	Returns true if this list contains no elements.
Iterator<**E**>	**iterator**()	Returns an iterator over the elements in this list in proper sequence.
int	**lastIndexOf**(**Object** o)	Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.

8

Page 8

그 다음에 get이 있죠 get은 이제 어떤 특정한 index에 들어가 있는
element를 return하게 되겠습니다
여기서 index가 가능한 것은 List이기 때문에 그렇죠
List는 순서가 유지되기 때문에
자기가 몇 번째 index를 가지고 있는지를 아는 거죠, order가 있기 때문에.
근데 뒤에 뭐 set이나 map 같은 거는 사실
순서가 유지되지 않기 때문에
이 get 메소드를 가질 수가 없겠죠
그 다음에 hashcode. 그 다음에 indexof 이거는 object
이것도 역시 List가 order를 유지하기 때문에 가능한 거고요
isEmpty(), iterator().
lastIndexOf(Object o) 는 오가 여러 개가 들어가 있을 수 있잖아요
중복이 여러 개 될 수 있는데 그 중에 가장 마지막 index,
가장 끝 쪽에 들어있는 index를 return하게 됩니다.

Methods in Interface List<E> (3/3)

Type	Method	Description
E	remove(int index)	Removes the element at the specified position in this list
boolean	remove(Object o)	Removes the first occurrence of the specified element from this list, if it is present
boolean	removeAll(Collection<?> c)	Removes from this list all of its elements that are contained in the specified collection
boolean	retainAll(Collection<?> c)	Retains only the elements in this list that are contained in the specified collection
E	set(int index, E element)	Replaces the element at the specified position in this list with the specified element
int	size()	Returns the number of elements in this list.
default void	sort(Comparator<? super E> c)	Sorts this list according to the order induced by the specified Comparator.
List<E>	subList(int fromIndex, int toIndex)	Returns a view of the portion of this list between the specified fromIndex, inclusive, and toIndex, exclusive.
Object[]	toArray()	Returns an array containing all of the elements in this list in proper sequence (from first to last element).

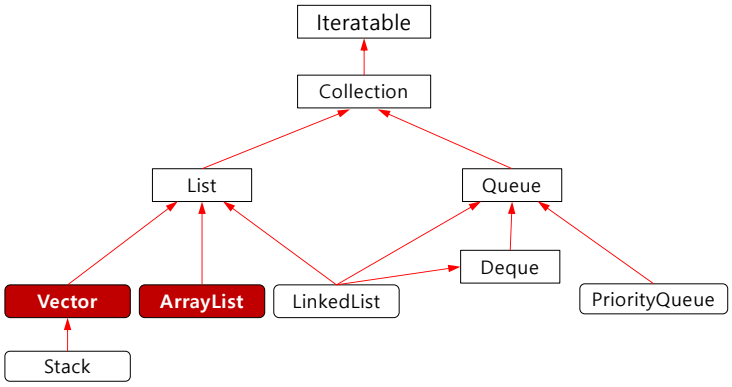
9

Page 9

그 다음에 remove 뭐 이거는 index에 있는 object를 remove 하는 거고
그 다음에 remove object를 직접 줘서 그것을 remove 하는 거고
그 다음에 removeAllCollection은
collection 안에 들어있는 걸 전부 remove 하는 거고
retainAll은 마찬가지로 collection 안에 들어있는 것만 남기고
다 삭제하는 거고요
그 다음에 set은 어떤 특정한 index의 element를 set 하는 거고요
size는 size, sort.
sort도 직접 가지고 있기 때문에 어떤 주어진 Comparator에 의해서
그 관계에 의해서 sorting이 가능하게 되겠습니다
그 다음에 sublist는 index 두 개가 주어져서
그 안에 있는 것들을 List로 만들어서 return하게 되겠고요
그 다음에 toArray는 역시 일반적인 array로 만들어서 return하는 것이 되겠습니다

ArrayList and Vector

- Order (0), Duplication (0)
- Use array as storage space
- Details of ArrayList in Lecture Note 14-1, too



10

Page 10

자 그래서 List 아래는 Vector하고 ArrayList 가 있는,
그 다음에 LinkedList가 있죠
Vector는 ArrayList 와 비슷한 건데
좀 옛날 거라서 거의 안 쓰인다 하는데
Stack 이라는 게 아직도 Vector를 기반으로 해서 implement가 되고 있습니다
그래서 ArrayList 와 벡터는 order와 duplication 다 가지고 있고
이 ArrayList와 Vector는 둘 다 array를 베이스로 하는 그런 List죠
그래서 array를 storage space로 사용한다
즉, ArrayList나 Vector 안에 있는 모든 element들은
연속된 memory공간 안에 존재를 해야 됩니다
이렇게 띄엄띄엄 있는 것이 아니라 연속된 공간 안에 있어서 찾아가기 쉽게 돼 있죠
그래서 ArrayList는 간단하게 보고 갈 텐데요

Using ArrayList (1/2)

```
import java.util.ArrayList;
import java.util.Collections;    // utility class for Collection (ex. sort())

public class ArrayListEx {

    public static void main(String[] args) {

        ArrayList<Integer> list1 = new ArrayList<Integer>(10);

        list1.add(5);
        list1.add(4);
        list1.add(2);
        list1.add(0);
        list1.add(1);
        list1.add(3);

        ArrayList<Integer> list2 = new ArrayList<Integer>(list1.subList(1, 4));

        System.out.println("list1:" + list1);  // list1:[5, 4, 2, 0, 1, 3]
        System.out.println("list2:" + list2);  // list2:[4, 2, 0]
```

11

Page 11

import java.util.ArrayList 고
여기 지금 Collections는 utility class예요
그래서 Collection이 아니라 Collections 거든요
그래서 이 utility class에는 예를 들면 sort 같은
그런 utility가 들어 있기 때문에
우리가 쉽게 사용할 수가 있습니다
지금 example에서는 ArrayList<Integer>를 type parameter로 하는
10개짜리 이 ArrayList를 먼저 만들었구요
list1은 [5 4 2 0 1 3] 이렇게 프린트가 되죠
여기 지금 println에서 list1의 toString은
이렇게 [ ] 가 되어 있는 이런 형식으로 자동으로 프린트를 해줍니다
그래서 ArrayList를 프린트할 때 이렇게 하면 된다는 거고
그 다음에 list2는 list1으로부터 subList(1, 4)를 받아가지고
list2는 [4, 2, 0] 이것만 list2에 들어가게 되죠

Using ArrayList (2/2)

```
        Collections.sort(list1);
        Collections.sort(list2);
        System.out.println("list1:" + list1);  // list1:[0,1,2,3,4,5]
        System.out.println("list2:" + list2);  // list2:[0,2,4]

        System.out.println("list1.containsall(list2):" +
                           list1.containsAll(list2)); // list1.containsall(list2):true

    }
}
```

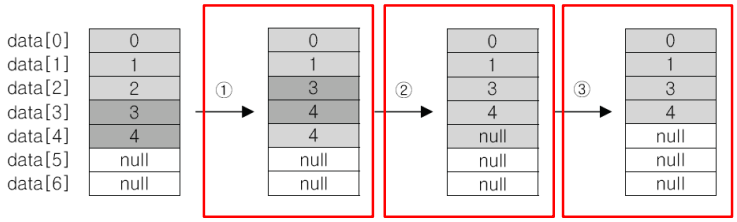
12

Page 12

그 다음에 Collections.sort(list1), Collections.sort(list2)
이렇게 0, 1, 2, 3, 4, 5로 sorting이 되구요.
Collections가 아까 utility class라고 했죠
그래서 여기에 이제 sort가 이 Collections의 static library method입니다
그래서 list1, list2를 sorting하는데 쉽게 사용할 수 있어요
이런 거 몰랐죠 이런 거 있으면 굉장히 쉽게 쓸 수 있었을텐데
그래서 list2는 0, 2, 4 역시 sorting이 됐고요
그 다음에 list1이 list2를 contains all을 하고 있느냐 하고
contains all을 실행하면 0, 2, 4는 0, 1, 2, 3, 4, 5에 완전히 포함되기 때문에
true를 리턴하게 됩니다
이런 식으로 사용하면 되구요

Removing Sequence in ArrayList

- Sequence of processing: v.remove(2);



1) The data under the element to be deleted are copied one space upward one by one, overwriting the element to be deleted
2) The last data moved is changed to be null
3) Decrease the size by one

- Note that in the worst case (the case when the first element is removed)
  - (size – 1) data should be moved ... O(n) time for n data

13

Page 13

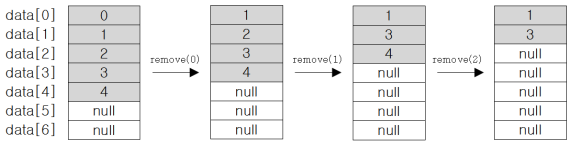
그 다음에 ArrayList에서 remove하는 그런
내부적으로 어떤 일이 일어나는지 한번 보도록 하겠습니다
v가 어떤 ArrayList라고 했을 때
거기에 v.remove(2) 즉 index 2를 remove한다
맨 처음에 이렇게 돼 있을 때, 주어진 이 target data를 delete하기 위해서는
이 데이터 뒤에 있는 것들 즉 3을 2로 copy하고 4를 3으로 copy하고
이런 식으로 되겠죠 그래서 이제 2가 3으로 됐고 3이 4가 됐고
4는 그대로 남아있게 되죠
이렇게 뭘 하나 중간에서 삭제를 하면 array기 때문에
이 중간에 비워둬서는 안되죠. 그러니까 이게 전부 이렇게 trimming을 해야 됩니다
바로 밑에서부터 하나씩 이렇게 카피를 계속 해와서
이렇게 채워야 되는 게 생기는 거죠
자 그 다음에 last 데이터가 4로 옮겨졌기 때문에 이건 null로 만드는 거죠
그 다음에 size를 하나 줄여서 최종적으로 이렇게 만들게 됩니다
그래서 최악의 경우에 worst case일 때는
어떤 때가 그 데이터를 제일 많이 옮겨야 될까요, 하나를 삭제할 때?

이 예를 보면 제일 첫 번째 element, data[0]를 삭제하면
그 밑에 있던 걸 다 옮겨야 되죠 이 꽉 차 있다고 생각하면
size - 1 개의 data size - 1개의 데이터를 옮겨야 됩니다
그래서 결국은 worst time의 time complexity는
Order of n (O(n)) 이다 이렇게 볼 수 있습니다
전체 n개 데이터가 있을 때 n-1개 데이터를 움직이게 되니까
n개 데이터를 움직이는 그런 time order를 갖게 되는 거죠
그래서 이거는 뭐 하나를 삭제하는데 n개를 다 움직이는 거니까
그렇게 좋아 보이지 않습니다 그렇지만 이건 할 수 없는 일입니다
array의 성질을 유지하기 위해서 할 수 없이 해야 되는 일이죠

Forward or Backward

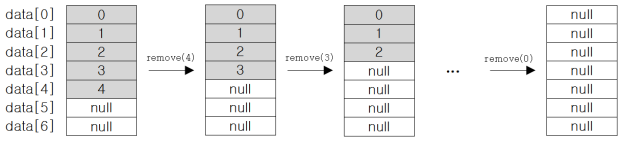
- Removing from the first object to the last (array copy occurs)

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



- Removing from the last object to the first (no array copy)

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



14

Page 14

이번엔 Array를 첫 element부터 끝 element까지
순서대로 remove하는 거를 한번 보죠
for문으로 i = 0부터 list.size() 보다 작을 때까지
i를 증가시키면서 list.remove(i) 를 했습니다
여기 i 는 이제 index가 들어가게 됩니다
지금 remove 0을 먼저 하면, 여기서 지금 0 1 2 3 4 이렇게 자리가 들어가 있죠
그래서 0번째 있는 것은 0 니까 0을 빼 버리면
얘네들을 하나씩 다 이렇게 카피를 해 줘야 되겠죠
자 1을 하나 땡기고 2를 다시 하나 땡기고
3을 땡기고 4를 땡기고 이런 식으로 1 2 3 4가 됐죠
그래서 copy를 해야 되는 상황이 온다 이제 그런 거구요
여기서 remove(1) 하면, index 1을 remove하라는 거예요
그래서 0 1 2 3 이렇게 인덱스가 되겠죠
그래서 remove(1), 1번 인덱스에는 데이터 2가 들어가 있으니까
2를 remove하면 3을 하나 땡겨주고 4를 하나 땡겨줘서
이런 식으로 카피를 해줘야 됩니다

그 다음에 마지막으로 이제 remove(2)를 하면은
0 1 2니까 4를, 요 경우에는 그냥 4를 그냥 null로 만들어 버리면 되겠네요
자 이런 식으로 앞에서 부터 뒷쪽으로, 0번부터 뒤쪽으로
이렇게 remove를 할 때는
하나씩 copy를 해줘야 되는 그런 상황이 오겠죠
근데 반대로 last object 부터 first object로
즉, i = list.size() - 1, 제일 마지막 index부터 시작해서
i를 감소시키면서 1씩, 0까지 그렇게 하나씩 remove를 하면 어떻게 될까요
자 먼저 remove(4)를 하면 마지막 걸 null로 만들고 끝. 그렇죠?
그 다음에 remove(3) 을 하면은 null로 만들고 끝
이런 식으로 하면은 마지막에 전부 null이 될 수 있겠죠
이 경우에는 이제 하나의 element를 remove 하더라도
다른 걸 땡겨 주거나 copy해 줄 필요가 없겠죠
그래서 이 Array일 경우에는 이런 식으로
앞쪽에서 부터 remove를 해 주느냐
아니면 뒤쪽에서 부터 remove를 해주느냐에 따라서
아주 작은 차이지만 계산 시간에 차이가 있을 수 있다라는 것을 보여주고 있습니다

Pros and Cons of ArrayList

- Advantages
  - Simple structure
  - Fast (direct) access time
    - The memory address can be calculated simply from the index
    - ex) list.get(index) ... address of list[index] = address of list[0] + sizeof(data) * index;
- Disadvantages
  - To change the capacity, we should create a new memory space and copy the data from old space to the new one.
    - If we create an array large enough to avoid resizing, the memory is wasted.
  - It takes a lot of time to add and delete out of sequence data.
    - To add or delete data, a lot of data must be moved.
    - However, sequential data addition (adding to the last) and data deletion sequentially (deleting from the last) are fast.

15

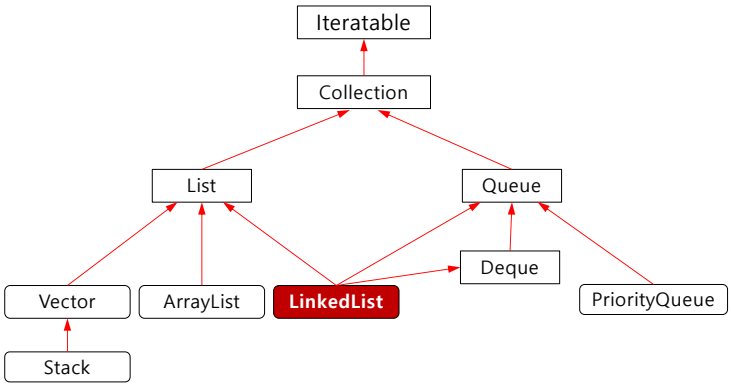
Page 15

그래서 ArrayList의 장단점을 한번 보면
장점은 굉장히 간단한 structure를 가지고 있다 라는 것이고요
그 다음에 fast direct access time을 가지고 있다
즉, 몇 번째 데이터를 찾아갈 때, 그것이 바로 한 번에 액세스가 가능해요
그거는 왜 그러냐면 이 데이터를 가지고 있는 이 Array의 특징인데
데이터를 가지고 있는 이 메모리가 전부 붙어 있기 때문에 그렇습니다
그래서 주소 계산이 쉬워지는 거죠
만약에 list.get(index) 가 주어졌다면
이 index element가 어디 있는지를 찾아갈 때는
list[index]의 address는 어떻게 돼요?
list[0]의 address에다가 sizeof(data) 곱하기 인덱스를 하면 되겠죠
그렇게 되면 쉽게 찾아갈 수 있겠죠
그래서direct access 가 굉장히 빠르다
그런데 단점이 또 있죠.
만약에 우리가 열개짜리 element를 가진 ArrayList 를 해놨는데
중간에 add, add, add 해가지고 열개 이상이 add가 되게 된다
그러면 내부적으로 어떤 일이 일어날까요?

이 ArrayList는 메모리가 전부 붙어 있어야 됩니다 그쵸 처음부터 끝까지
그러니까 지금 현재 할당된 Array를 버리고
이거보다 더 큰 Array를 새로 잡아서
거기다 다 기존의 element들을 옮겨 줘야 돼요 카피를 해야 돼요
왜냐하면 스페이스를 큰 걸 잡아야 되니까요
마치 식구가 많아지면 방이 여러 개인 큰 집을 구해야 되고
이사를 해야 되잖아요
그거하고 똑같은 거죠
그래서 그 capacity를 바꾸기 위해서는
새로운 memory space를 만들어야 되고
그 다음에 데이터를 old space에서 new one 으로 카피를 해야 된다
만약에 그래서 이 옮기는 시간이 또 걸리기 때문에
그래서 그게 싫어서 아예 처음부터 크게 메모리를 잡아 버리면
실제로 쓰는 것보다 버려지는 메모리가 훨씬 더 많을 수 있다는
얘기가 되겠습니다 waste 된다 라는 얘기가 되겠습니다
그래서 어느 쪽이나 작게 잡을 수도 없고 너무 크게 잡을 수도 없고
그래서 그런 단점이 있는 거죠
그 다음에 그 sequence 데이터를 add하거나 delete할 때
시간이 많이 걸릴 수 있다는 거죠
아까 우리가 이 전번에 슬라이드에 봤듯이
맨 첫번째에서 데이터를 삭제하거나 아니면 끼워 넣을 때
끼워 넣을 때도 마찬가지로 데이터를 다 옮겨야 되죠
그럴 때는 시간이 많이 걸릴 수 있다는 거죠
중간이나 첫 부분에, 그렇죠? 그래서 그럴 때 시간이 많이 걸릴 수 있다
그렇지만 그 맨 마지막 부분에서 데이터를 add하거나
삭제하는 것은 좀 시간이 덜 걸린다 라는 얘기가 되겠습니다
자 그래서 이러한 그 ArrayList 의 장단점
일반적인 array의 장단점과 같습니다

LinkedList

- Order (0), Duplication (0)

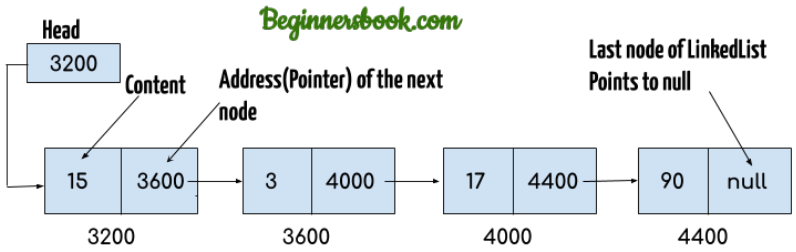


16

Page 16

그 다음에 그 ArrayList와 Vector는 같은 부류고
LinkedList는 List의 또 다른 implementation이죠
그래서 이것도 역시 List 이기 때문에
order와 duplication 유지가 되고요

Linked List



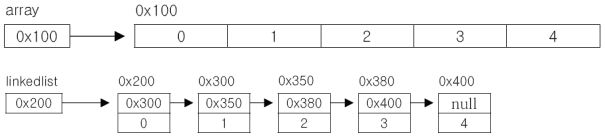
17

Page 17

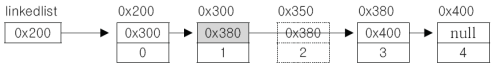
그래서 LinkedList 보면 컨텐트가 있고
그 다음에 이 뒤에는 link field가 있었죠
link field는 다음 node의 address를 가지고 있죠
그래서 이렇게 주소를 가지고 있다는 것은
바로 이렇게 얘를 이렇게 포인팅 하고 있다 라는 것과 같은 뜻이 되죠
그 다음에 그 다음 node로 가보니까 3600에 가보니까 3이 들어있고 데이터로.
그 다음에 다음 node는 4000번째에 들어있고
그래서 4000번째 가보니까 17이 들어있고 이런 식으로
그리고 맨 끝에는 이제 그 list의 끝을 나타내기 위해서
null을 가지고 있고요 link에.
그 다음에 head가 있어서 head는 맨 첫 번째 node의 주소를 가지게 있게 되죠
그래서 LinkedList 는 사실 다음 node의 주소를 가지고 있기 때문에
얘네들이 꼭 이렇게 physical 하게 붙어 있는 메모리에 있지 않아도 된다는 장점이 있습니다
왜냐하면 다음 데이터를 이렇게 실제로 데이터가 있는 곳에
위치를 다 가지고 있기 때문에 가능한 얘기죠 그게

Features of Linked List

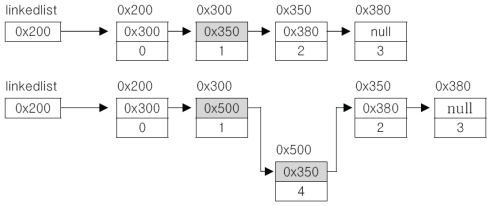
- Unlike arrays, linked lists link data that exists discontinuously.



- Delete Data: by Single change of reference



- Insert Data: One node creation + Several changes of references



18

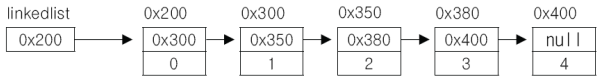
Page 18

그래서 LinkedList는 그 얘기가 바로 나오는데요 array와는 달리
LinkedList는 데이터가 discontinuous 하게 있어도 된다라는 거죠
continuous 하게 있지 않아도 된다라는 얘기죠
그래서 array일 경우에는 이렇게 100번째 위치에 시작 위치가 있다고 하면
이렇게 그다음 이 데이터들이 이렇게 쭉 계속해서 연달아서 붙어 있어야 돼요
일정한 size로. 그렇게 해야만 우리가 인덱스를 딱 지정했을 때
쉽게 찾아갈 수 있는 거잖아요
근데 LinkedList일 경우에는 이렇게 다음 node의 주소를 가지고 있기 때문에
link에
그래서 꼭 이렇게 붙어 있을 필요는 없다는 거죠
그래서 사방으로 왔다 갔다 하는 것도 가능합니다 메모리 안에서
메모리를 더 효율적으로 쓸 수 있고 또 여러 가지 장점이 있게 됩니다
만약에 중간에 이 2를 가지고 있는 이 node를 삭제하고 싶다라고 하면
그 앞에 있는 node의 link 필드를
뒤쪽 현재 그 삭제하려는 node의 link로 바꿔주면 되죠
그럼 얘를 pass해서 그냥 이렇게 하나 건너뛴 얘를 가리키게 되니까
중간에 얘는 자동적으로 삭제가 되는 겁니다 그죠

그래서 garbage collector에 의해서 모여져서
다른 곳에 사용되게 된다 라고 얘기를 했죠
자 그 다음에 insert data는 하나의 데이터를 크리에이트하고 그 다음에 reference를
몇 번 바꿔주면 된다 라고 했습니다
자 그래서 여기 지금 0, 1, 2, 3이 들어가 있는
이런 list에 4라는 거를 만들어서 1 다음에 끼워 넣고 싶어요. 여기 1과 2 사이에
그러면 여기 4라는 걸 새로 node를 만든 다음에 그 다음에
그 2의 전번 node에 있는 이 link를 새 node를 가리키게 하고
그 다음에 새 node의 link가 그 2번 node를 가리키게 하면 되겠죠
그래서 이런 식으로 하면 쉽게 Implement를 할 수 있겠습니다

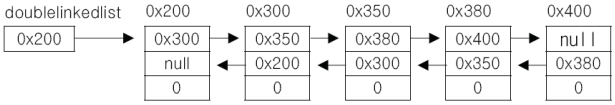
Various LinkedList Types

▶ Linked List – Hard to access



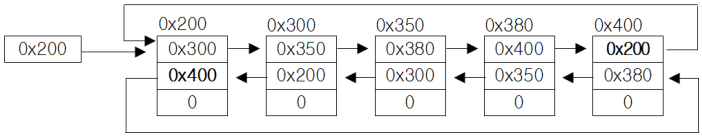

```
class Node {
    Node next;
    Object obj;
}
```

▶ Doubly linked list – Better accessibility




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

▶ Doubly circular linked list



19

Page 19

자, 그 다음에 LinkedList type에 대해서 좀 알아보겠는데요
LinkedList 일반 이게 지금 가장 단순한 Simple
LinkedList인데 이런 식으로 Node가 돼 있죠 Node의 Next,
Next 또는 우리는 Link라고 이름을 가지고 있었죠
이거는 다음 node의 주소를 가지고 있는 거고 Object obj는 데이터 필드라서
여기에 그 오브젝트를 가져도 되고 아니면
얘가 제너릭으로 어떤 타입 파라미터를 가지고 해서
타입 파라미터를 가지고 있게 해도 되죠
T 타입을 가지고 있게 해도 되죠
그런 식으로 해서 이 next 하나만 갖고 있기 때문에
link 하나만 갖고 있기 때문에
이렇게 다음 번 node만 이렇게 포인팅 하게 되어 있습니다
자 근데 Doubly Linked List는 link를 하나씩 더 가지고 있어요
즉 next 하고 previous.
next는 다음 번 node를 가리키고 있고
previous는 뭐를 가리키고 있어요 자기의 전번 걸 가지고 있는 거죠 이런 식으로

그래서 next와 previous를 다 가지고 있기 때문에
이게 훨씬 더 코딩 하기는 쉽습니다
그러니까 그 뒤쪽 node에서 앞쪽 node를 액세스 하는게 가능하죠
그래서 이런 걸 doubly linked list라고 하고요
우리는 여기에 대해서 공부는 지금 뭐 따로 안하지만 여기서
이렇게 implement할 수도 있다라는 거고요 그 다음에
doubly circular linked list는 어때요 쭉쭉쭉쭉 가서
맨 끝에 있는 게 다시 첫 번째 걸 가리키게 되는 거죠
그리고 또 previous는 쭉쭉 와서 이 previous가
첫 번째에선 previous가 더 이상 갈 데가 없기 때문에
null이었는데 여기서는 맨 끝에 걸 가리키게 하는 거죠
이렇게 하면은 next를 통해서도 한 바퀴를 쭉 돌 수 있고
그 다음에 다시 원점으로 돌아올 수 있고 previous 를 통해서도
뒤에서부터 한 바퀴 돌고 다시 맨 뒤로 돌아올 수 있고요
그렇죠? 그런 장점이 있죠
이런 걸 이렇게 계속 연결되어 있는 이런 형태를
Circular Linked List라고 합니다

Example: LinkedList - 1

```java
import java.util.LinkedList;

public class CollectionLinkedList1 {
    public static void main(String args[]) {

        LinkedList<String> ll = new LinkedList<String>();

        ll.add("A");        // [A]
        ll.add("B");        // [A, B]
        ll.addLast("C");    // [A, B, C]
        ll.addFirst("D");   // [D, A, B, C]
        ll.add(2, "E");     // [D, A, E, B, C]
        System.out.println(ll);

        ll.remove("B");     // [D, A, E, C]
        ll.remove(3);       // [D, A, E]
        ll.removeFirst();   // [A, E]
        ll.removeLast();    // [A]
        System.out.println(ll);
    }
}
```

20

Page 20

Linked List<String> 이렇게 하면 이제 String 타입을 가지게 되는데
그 타입 파라미터로 new 해서 하나 만들었구요
그 다음에 처음에 A B
C 자 A B 를 add 하면은 A 가 들어가고 그 다음에 list
add B 를 하면은 A 뒤에 B 가 들어가게 되죠 그 다음에 add
rest C 하면은 add rest 도 마찬가지로 add 랑 마찬가지예요
그래서 맨 끝에 집어 넣는 거죠 A B C add first 하게 되면
제일 먼저 앞쪽에 집어넣게 되죠 DABC 그 다음에 add 하게 되면은
2번에 2번 인덱스에 2를 넣으라 했으니까 0, 1, 2
여기 들어가게 되니까 얘는 A하고 B 사이에 여기 2를 끼워 넣게 되죠
그래서 이런 식으로 add를 하게 되고
remove는 직접 B를 찾아서
여기 있는 B를 찾아서 remove하는 방법 그 다음에
인덱스에서 remove하는 방법 0, 1, 2, 3, 0, 1, 2,
3 이었으니까 3번에 C가 있었죠 그리고 C가 날아가는 거고요
remove first는 D를 날리는 거고 remove
last는 맨 끝에 있는 걸 날리는 거고

이런 식으로 사용을 하면 되겠습니다

Example: LinkedList - 2

```
import java.util.*;

public class GFG {

    public static void main(String args[])
    {
        LinkedList<String> ll = new LinkedList<>();

        ll.add("Geeks");     // [Geeks]
        ll.add("Heeks");     // [Geeks, Heeks]
        ll.add(1, "Jeeks");  // [Geeks, Jeeks, Heeks]

        System.out.println("Initial LinkedList " + ll);

        ll.set(1, "For");    // [Geeks, For, Heeks]

        System.out.println("Updated LinkedList " + ll);
    }
}
```

21

Page 21

두 번째 예에서는 스트링에
이것도 역시 스트링, 여기 지금 스트링 빠졌는데요 이건 에러는 안 나는데
여기 넣는 게 좋겠습니다 <String> 이라고
그 ll 에다가 Geeks 라는 걸 넣으면 [Geeks],
그 다음에 Heeks 를 넣으면 [Geeks, Heeks]
그 다음에 1번에다 Jeeks 를 넣으라 그러면
0, 1이니까 Heeks가 뒤쪽으로 가야 되겠죠
그래서 [Geeks, Jeeks, Heeks] 이렇게 되고요
그 다음에 ll.set(1, for) 하면 set은 뭐냐면
이거 새 node를 만드는 것이 아니라 데이터를 바꾸는 겁니다
그래서 1번 index의 Jeeks를 for로 바꾸는 거죠
[Geeks, for, Heeks] 이렇게 세팅을 하게 됩니다

Example: LinkedList - 3

```
import java.util.*;

public class GFG {

    public static void main(String args[])
    {
        LinkedList<String> ll = new LinkedList<>();

        ll.add("Geeks");
        ll.add("Geeks");
        ll.add(1, "For");

        for (int i = 0; i < ll.size(); i++) {
            System.out.print(ll.get(i) + " "); //Geeks For Geeks
        }

        System.out.println();

        for (String str : ll)
            System.out.print(str + " ");      // Geeks For Geeks
    }
}
```

22

Page 22

자, 그 다음 3번 3번 example, 이거는 Geeks, Geeks를 add 하고
중복을 허락하기 때문에 둘 다 들어가죠
그 다음에 1번에다가 for 를 넣었죠
네, 그렇게 되면은 Geeks, Geeks 있다가 1번에다 for 를 넣으니까
Geeks 두 개 사이에 for 가 들어가죠
그래서 프린트를 하면은 [Geeks for Geeks] 가 프린트 되고요
자 요거는 size까지 따라가면서 ll.get(i) 을 해서 프린트를 하게 한 건데
뭐 ll을 직접 println으로 바로 쏴 줘도 되지만
여기서는 지금 size까지 이렇게 한 바퀴를 돌 때
인덱스를 통해서 이렇게 액세스 하는 것도 가능하다는 걸
보여주기 위해서 이렇게 한 거구요
그 다음에 요거는 똑같이 for문을 쓸 때도 str : ll
이렇게 하면은 ll 안에 들어있는 모든 스트링에 대해서
이렇게 프린트하는 것도 가능하다는, 이제 for each loop라고 했죠
이렇게 쓰는 것도 가능하다는 얘기가 되겠습니다

(More) Methods in class LinkedList

Type	Method	Description
void	addFirst(E e)	Inserts the specified element at the beginning of this list.
void	addLast(E e)	Appends the specified element to the end of this list.
void	clear()	Removes all of the elements from this list.
Object	clone()	Returns a shallow copy of this LinkedList.
E	element()	Retrieves, but does not remove, the head (first element) of this list.
E	getFirst()	Returns the first element in this list.
E	getLast()	Returns the last element in this list.
int	lastIndexOf(Object o)	Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
boolean	offer(E e)	Adds the specified element as the tail (last element) of this list.
boolean	offerFirst(E e)	Inserts the specified element at the front of this list.
boolean	offerLast(E e)	Inserts the specified element at the end of this list.
E	peek()	Retrieves, but does not remove, the head (first element) of this list.
E	peekFirst()	Retrieves, but does not remove, the first element of this list, or returns null if this list is empty.
E	peekLast()	Retrieves, but does not remove, the last element of this list, or returns null if this list is empty.
E	poll()	Retrieves and removes the head (first element) of this list.

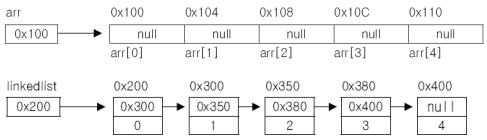
23

Page 23

자 그래서 LinkedList 를 봤는데 이게 이제 앞에 List
interface 에 있는 method에다가 더 추가된 그런 것들입니다
그래서 addFirst, addRest, clear, clone,
element, element는 뭐냐면, 그 첫 번째 element
LinkedList의 첫 번째 first element를 가져옵니다
return하지만 remove 하진 않는다 라는 거고요
사실 이 element는 peek 하고 똑같습니다 하는 일이.
그 다음에 get first는 첫 번째 element를 return하는 get first하고도 똑같은 거죠
그 다음에 get last는 맨 마지막 element를 return하게 되었고요
그 다음에 lastIndexOf(Object o) 는
Object o가 존재하는 가장 마지막 인덱스
만약에 없을 경우에는 - 1 return하게 되겠구요.
offer는 tail에다가 집어넣는 겁니다
addRest하고 같은 일을 하는데 여기서 boolean을 return하게 되죠
그 다음에 offer가 있고 offerFirst가 있고 offerLast가 있고
그래서 offer라는 건 이제 insert하고 같은 의미죠

add하고 같은 의미고요
다음에 peek, peekFirst, peekLast는 마찬가지로 읽어오기만 하는데
remove 하지는 않습니다
poll 은 뭐냐면 head에서 하나를 읽어오고
그것을 return하고 그것을 remove를 하게 돼요
그래서 여기서 중요한 것은
offer 그 다음에 peek, poll 뭐 이런 것들이 되겠습니다

ArrayList vs. LinkedList



Operation	Worst case time (ArrayList)	Worst case time (LinkedList)
Finding an element	n – 1	n – 1
Add an element at tail	C	n – 1
Add an element at head	n – 1	C
Find and Remove an element	n – 1	(n – 1) + C
Remove an element at tail	C	(n – 1) + C
Remove an element at head	n – 1	C
Access i'th element	C	n – 1

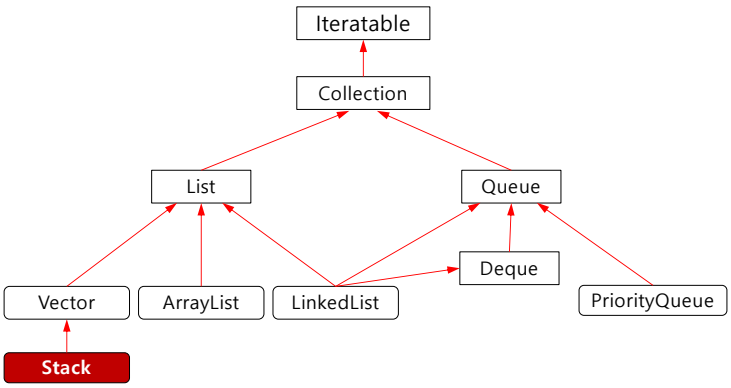
24

Page 24

ArrayList와 LinkedList의 차이에 대해서
어떤 수행 시간의 차이에 대해서 좀 보여주고 있는데요
여기서 지금 이 위에는 지금 ArrayList고요
그 다음에 밑에는 LinkedList입니다 자, 그래서
하나씩 element가 있는지 없는지 볼 때
Worst Case Time은 Array나 아니면 LinkedList에서
둘 다 n-1개씩 걸립니다
즉, 어떤 element가 이 안에 있는지를 살펴보기 위해서
sorting 안 된 상태라고 보면요 전부 다 끝까지 다 봐야 되죠
그래서 계속해서 움직여서 n-1개는 다 봐야 된다라는 얘기가 되겠고요
그 다음에 tail에다가 element 하나 추가할 때 ArrayLst는
어때요 size만 여기다 넣어주고 size만 변경시키면 된다 그랬으니까
어떤 constant 뭐 1이라고 볼 수도 있고
1, 2 이런 거는 다 constant로 보니까요
constant C라고 볼 수 있고
worst case의 linked list는 어떻게 돼요?
맨 끝에 하나 추가할 때는 일단 얘네들을 다 거쳐서

가서 tail까지 가야 되죠 tail까지 가는 시간만 n-1 입니다
그래서 n-1를 먼저 다 액세스하고 그 다음에 여기서 뭘 하나 더 추가해야 되죠
자, 그 다음에 add an element at the head
head에다 추가할 때 이때는 느낌이 오지만
LinkedList 가 훨씬 유리합니다
LinkedList 는 새 node를 만들고
link만 두 개 딱 바꿔주면 되거든요 근데 ArrayList 일 때는 어때요
아까 얘기한 대로 자 arr[0] 를 여기다가
여기다 하나 추가를 하려면 얘네들을 다 이렇게 한 칸씩 다 옮겨야 되죠
그래서 move 하는게 몇 번 걸리냐면 n-1번 최악의 경우에
그래서 이때는 이제 LinkedList가 훨씬 Constant Time으로 유리합니다
그 다음에 Element를 찾고 Remove할 때
이것도 역시 찾는 시간이 걸리기 때문에 n-1개, LinkedList도 n-1번
그 다음에 Tail에서 Remove를 할 때
Tail에서 Remove할 때 무조건
그 ArrayList는 Tail에서 하는 거는 쉬워요 그쵸? 네, 한두 번 가지고 끝나고
얘는 일단 tail에서 뭘 하는 거는 일단 찾아가야 되기 때문에
일단 n-1은 기본으로 걸립니다 LinkedList 는 그렇게 되고요
그 다음에 head에서 뭘 할 때는 무조건 LinkedList 가 더 유리하죠
그래서 이쪽은 link만 두 번 바꿔 주면 되고 얘는 일단 remove 하니까
얘네들을 다 또 옮겨서 앞쪽으로 옮겨야 되죠 그래서 최악의 n-1이 걸리고
I 번째 element access 할 때 이거는 기본적으로 array의 장점입니다
그래서 한 번에 찾아가죠 그런데 LinkedList 일 경우에는 어떻게 돼요?
최악의 경우 I가 끝번일 때는 n-1개가 걸리는 거죠
그래서 이런 식으로 어떤 데이터의 특성에 따라서
내가 어떤 operation 을 많이 하느냐에 따라서
ArrayList 를 선택할 거냐 아니면 LinkedList 를 선택할 거냐 이거를
우리가 결정을 해야 되겠죠
그래서 application을 딱 보고
여기서 주로 하는 operation이 어떤 operation이다
중간에 있는 거를 많이 insert했다가 다시 delete하고
이런 일들이 많이 일어난다 라고 하면은
무조건 LinkedList 를 해야 되겠죠
왜냐하면 ArrayList 를 사용하면
예를 들면 천만 명 인구 중간에서 한 명을 삭제한다
굉장히 힘든 거죠 array에서는
중간에서 삭제하면 계속해서 copy를 해야 되기 때문에
그래서 어떤 data structure 를 사용할지에 대해서
잘 우리가 생각을 해봐야 된다라는 얘기가 되겠습니다

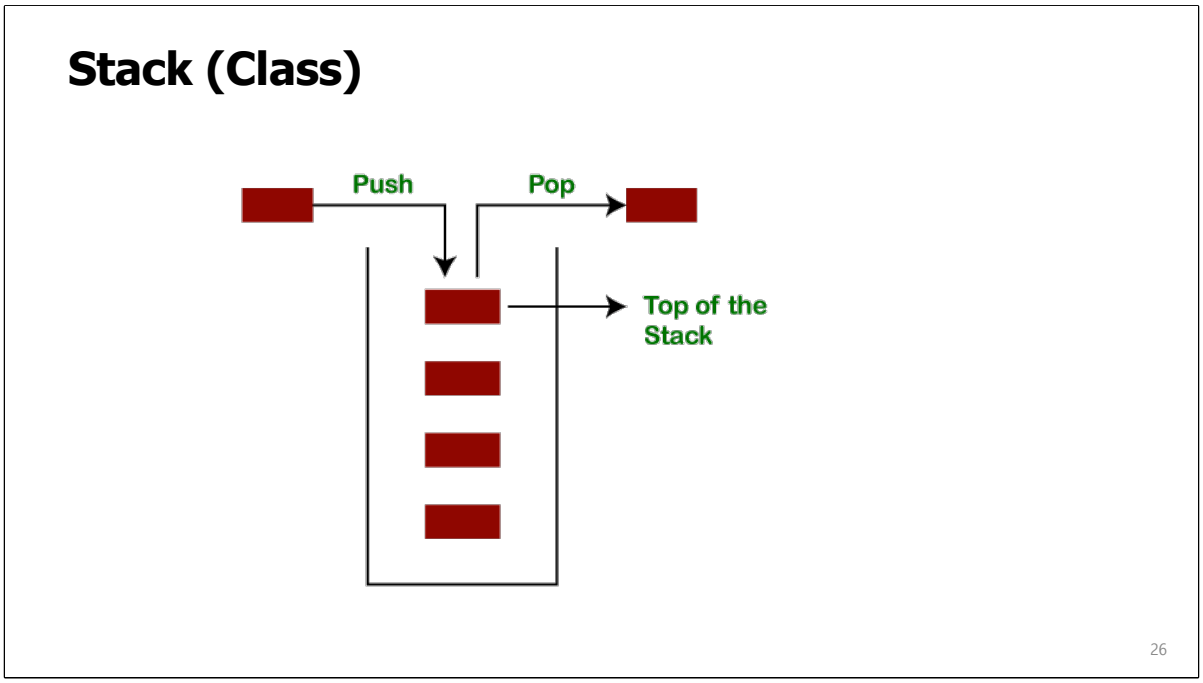
Stack (Class)



25

Page 25

Stack에 대해서 강의하겠습니다
Stack은 계속해서 얘기하지만
Collection에서 List interface 밑에 속하는 class 구요
Stack은 Vector, 그러니까 기본적으로 array를 바탕으로 해서
implement되는 그런 것이 되겠습니다
Stack에 대해서 강의하겠습니다
Stack은 계속해서 얘기하지만
Collection에서 List interface 밑에 속하는 class 구요
Stack은 Vector, 그러니까 기본적으로 array를 바탕으로 해서
implement되는 그런 것이 되겠습니다

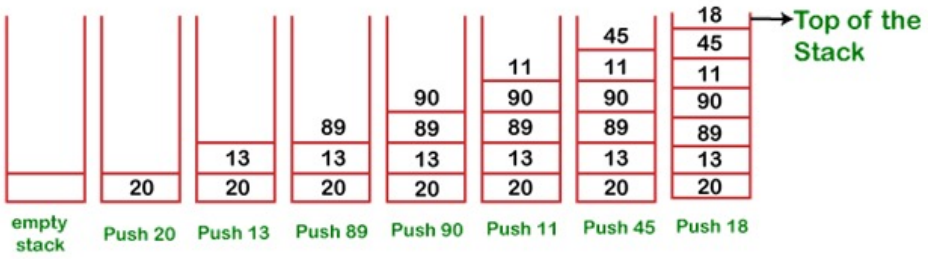


Page 26

Stack은 많이 우리가 들어봤지만
한쪽으로만 push하고 한쪽에서만 pop을 하는
그런 data structure가 되겠습니다
그래서 push를 하게 되면 항상 Stack의 top에 올라가 있게 되고요
top of the Stack이죠
pop을 할 때는 순서대로 제일 위에 있는 것
top부터 pop을 하게 됩니다
그렇기 때문에 이 Stack은 항상
Last In, First Out, 그래서 LIFO 라고 부르죠
그래서 제일 나중에 도착한 것이
가장 처음으로 pop된다 라는 얘기가 되겠습니다

Stack (Class): Push

- Push 20, 13, 89, 90, 11, 45, 18



Push operation

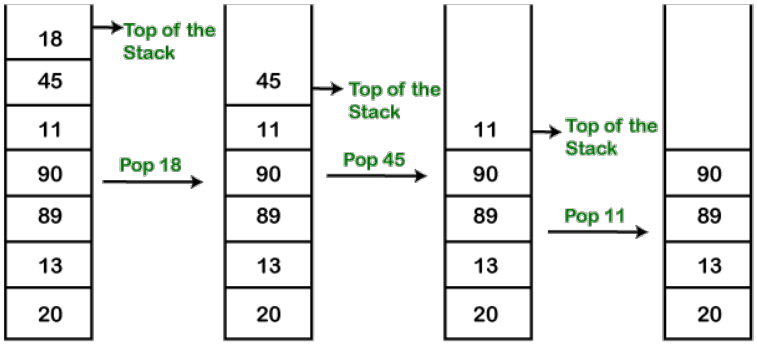
27

Page 27

그래서 Stack을 push하는 그 그림을 예를 보면
empty일 때 push 20을 하고 그럼 top이 20이 됐고
그 다음에 13을 하게 되면 20위에 13이 올라가고
네 이런 식으로 22에 13 89 push가 됐고
90 11 45 18 이런 식으로 push가 되겠습니다
항상 여기가 이제 제일 위에 있는 걸 top이라고 얘기하고요

Stack (Class): Pop

- Pop 18, 45, 11 from the stack



Pop operation

28

Page 28

pop operation 어떻게 돼요?
항상 top에서부터 빠져나가죠
그래서 pop 18 하게 되면 18이 먼저 나가고
나가게 되면 top이 이제 45가 됐고
pop하게 되면 45가 나가게 되고 그 다음 11이 나가게 돼서
이런 식으로 제일 위에부터 나가게 그렇게 되겠습니다

(More) Methods in class Stack

- Stack is implemented on Vector class, so it has all the methods of Vector class
- More methods in Stack:

Method	Type	Description
empty()	boolean	The method checks the stack is empty or not.
push(E item)	E	The method pushes (insert) an element onto the top of the stack.
pop()	E	The method removes an element from the top of the stack and returns the removed element
peek()	E	The method looks at the top element of the stack without removing it
search(Object o)	int	The method searches the specified object and returns the position of the object.

29

Page 29

Stack class는 이제 기본적으로
Vector 즉 array를 기본으로 해서 implement가 되지만
다른 그런 method를 가지고 있어요
그래서 empty() 는 Stack이 empty면은 true를 return하는 거고
push() 하고 pop() 이 다 있습니다
push() 는 Stack의 top에다가 push하는 거구요
pop() 은 뭐냐면 일단 기본적으로 remove를 합니다
top에 있는 거를 remove를 하면서
그 remove 된 top에 있는 것을 return을 하게 되죠
그 다음에 peek() 은 top에 있는 것을 return을 하긴 하지만
remove 하진 않습니다 그대로 남겨두는 거죠
그냥 살펴만 보는 게 peek이 되겠고요
search(Object o) 는 뭐냐면 어떤 주어진 Object o가 Stack 안에 있는지 보고
만약에 있으면 index position을 return하게 되고
없으면 -1을 return하게 되는 거죠

Example: Stack - 1

```java
import java.util.Stack;

public class StackEmptyMethodExample {
    public static void main(String[] args) {
        //creating an instance of Stack class
        Stack<Integer> stk= new Stack<Integer>();

        // checking stack is empty or not
        System.out.println("Is the stack empty? " + stk.empty());

        // pushing elements into stack
        stk.push(78);
        stk.push(113);

        //prints elements of the stack
        System.out.println("Elements in Stack: " + stk);
        System.out.println("Is the stack empty? " + stk.empty());
    }
}
```

```
Is the stack empty? true
Elements in Stack: [78, 113]
Is the stack empty? false
```

30

Page 30

그 다음에 example을 좀 보도록 하겠습니다
Stack을 지금 Integer를 type parameter로 해서 만들었죠
그래서 Is the Stack empty? 하고 물어보면
처음에는 아무것도 push를 안 했으니까 true가 나오겠죠
그 다음에 Stack에다가 push 78, 113 해서
그 다음에 println은 Elements in Stack은 78, 113이 되겠고
그 다음에 empty냐고 물어보고
Stack empty를 또 print하면 false가 되겠죠 이번에는

Example: Stack - 2

```
import java.util.*;

public class StackPushPopExample {

    public static void main(String args[]) {

        Stack <Integer> stk = new Stack<Integer>();
        System.out.println("stack: " + stk);

        stk.push(20);
        stk.push(13);
        stk.push(89);
        stk.push(90);
        System.out.println("stack: " + stk);

        System.out.println(stk.pop() + " popped");
        System.out.println(stk.pop() + " popped");
        System.out.println("stack: " + stk);
    }
}
```

```
stack: []
stack: [20, 13, 89, 90]
90 popped
89 popped
stack: [20, 13]
```

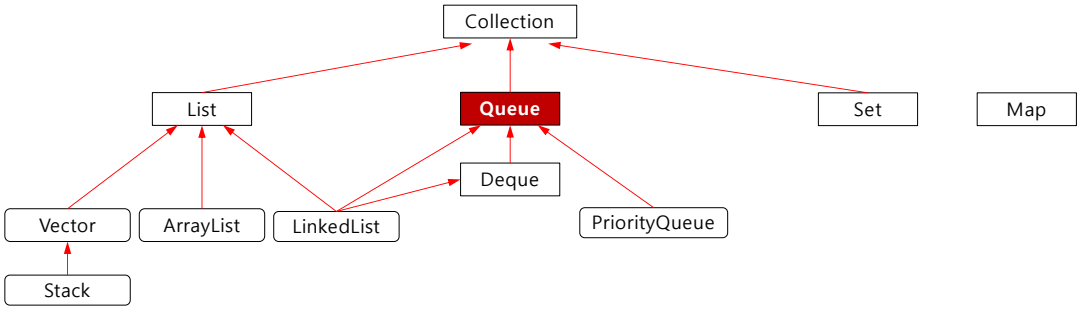
31

Page 31

자 그 다음에 두 번째 example에서는
Integer를 type parameter로 해서 만들었죠
그래서 20, 13, 89, 90 처음엔 Stack이 비어 있었고 [ ]
그 다음에 두 번째 print 할 때는
[20, 13, 89, 90] 이렇게 print가 되구요
딱 보니까 Stack이 array를 기본으로 해서 implement 된다고 했는데
20이 지금 제일 첫 번째 element로 들어가 있죠
그래서 항상 Stack에는 push 할 때는 이런 식으로 print가 되게 된다 라는 거죠
top이 제일 먼저 앞에 나와 있고
자 그 다음에 Stack의 pop을 하게 되면 이 return하는 것이
현재 top에 있는 것을 return하게 되죠
그래서 90이 return되고요
여기서는 20, 13, 89, 90 이니까 top에 지금 90이 올라가 있죠
90은 맨 끝에 있는 게 맞겠죠
여러분들 array에서 어느 쪽에 add를 하고
어느 쪽에서 remove를 하는 게 더 편했습니까

array에선 항상 맨 끝에다가 추가를 하고 삭제를 하는 것이 훨씬 효율적이죠
그렇기 때문에 Stack top은 항상 맨 끝이 돼야 되겠죠 당연히
그래서 90이 pop 되고 나면 0, 20, 13, 89만 남았을 거고 여기서는
그 다음에 두 번째 pop을 하면 또 return을 89를 삭제하면서 개를 return하게 되죠
그래서 89가 return되니까 맨 나중에 Stack은 20, 14만 남게 됩니다
이런 식으로 안에서 어떤 일이 벌어지고 있는지를 좀 생각해 가면서
그렇게 coding을 하면 좋겠습니다

Interface Queue<E>



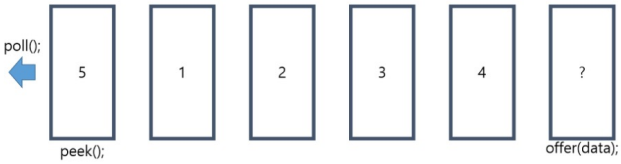
32

Page 32

자 그 다음에 Queue interface 인데요
Queue는 Collection 아래에 있기도 한데
이 Queue는 또 LinkedList의 또 parent이기도 해요
이 Queue interface가 그 다음에 뒤에 나올
PriorityQueue를 또 child로 가지고 있고요

Interface Queue

- First In First Out (FIFO)
- Add data at one end (offer)
- Remove data at the other end (poll)
- Observe data without removal (peek)



- LinkedList is used in many cases for implementing Queue

Operation	Worst case time (ArrayList)	Worst case time (LinkedList)
Add an element at tail (offer)	C	n – 1
Remove an element at head (poll)	n – 1	C

33

Page 33

자 그래서 interface Queue는
Stack이랑은 다르게 First In First Out을 제공합니다
즉 제일 먼저 들어온 것이 제일 먼저 빠져나간다
그러니까 add 될 때는 이 처음에 들어온 거 뒤에
add가 이렇게 되는데 얘가 빠져나갈 때는
제일 처음부터 순서대로 나가게 되는 거죠
fair 하죠 그 다음에 data를 끝 쪽에
항상 그 뒤쪽으로 add하게 되고요 줄을 서 있기 때문에
그래서 그걸 offer라고 하고요 offer(data) 라는 method를 사용하고
그 다음에 remove data 할 때는 poll이라고 합니다
얘를 remove하면서 return하게 되고요
그 다음에 observe만 하는 거
즉 뭐가 들어있는지 보고 remove하지 않는 건 peek이죠
그래서 이 poll, peek 그 다음에 offer
이 세 개가 중요한 그런 operation이 되겠습니다
LinkedList 가 많은 경우에 Queue를 Implement하는 데
사용된다는 얘기가 되겠습니다

그래서 사실 ArrayList를 쓸지 LinkedList 를 쓸지
Queue를 Implement하는 데 있어서 좀 고민이 있을 수 있어요
왜냐하면 이 Queue는 들어오는 건 한쪽 끝에서 들어오고
나가는 건 다 반대쪽 끝에서 나가기 때문에
ArrayList와 LinkedList가 둘 다 장단점을 상반되게 가지고 있어요
예를 들면 tail에다가 add 할 때 ArrayList는 금방 할 수 있죠
근데 LinkedList는 일단 tail로 따라가야 되니까 n-1번을 봐야 되고
근데 이거는 이제 사실 tail에 그 포인터를 하나 가지고 있으면 해결되는 거라서
LinkedList는 이제 그런 식으로 해결을 하고 있다 이렇게 보면 될 거고요
그 다음에 element를 head에서 삭제할 때 poll 할 경우에는
ArrayList는 필연적으로 n-1번 즉 이 data를 다 옮겨야 되죠 array이기 때문에
그런데 LinkedList는 금방 해결을 할 수 있습니다
그리고 아까 tail에 그 포인터 하나를 가지고 있으면
그거 가지고 쉽게 해결된다는 얘기를 했는데
그렇기 때문에 LinkedList가 쓰이는 경우가 많습니다

Methods in Interface Queue<E>

Type	Method	Description
boolean	add(E e)	Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions, returning true upon success and throwing an IllegalStateException if no space is currently available.
E	element()	Retrieves, but does not remove, the head of this queue.
boolean	offer(E e)	Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions.
E	peek()	Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.
E	poll()	Retrieves and removes the head of this queue, or returns null if this queue is empty.
E	remove()	Retrieves and removes the head of this queue.

34

Page 34

Queue interface의 method로 중요한 것은 아까 세 개를 말했죠
offer 요거는 그 뒤쪽에 Queue에다가 하나씩 add하는 거다 라고 했고
그 다음 peek은 제일 앞에 거를 return하지만
remove하지는 않는다 라고 했고
poll은 어때요 제일 앞에 거를 return하면서 걔를
Queue에서 remove해 버리죠
그 중요한 것들을 좀 알고 있어야 되구요

Queue by LinkedList

```
import java.util.LinkedList;
import java.util.Queue;

public class QueueLinkedList {
   public static void main(String[] args) {
      Queue<Integer> queue = new LinkedList<Integer>();  // creating queue
      queue.offer(5);  queue.offer(1);  queue.offer(2);  queue.offer(3);  queue.offer(4); // pushing 5 data
      System.out.println(queue);
      System.out.print("poll: " + queue.poll() + " "); System.out.println(queue); //poll
      System.out.print("poll: " + queue.poll() + " "); System.out.println(queue); //poll
      System.out.print("poll: " + queue.poll() + " "); System.out.println(queue); //poll
      System.out.print("peek: " + queue.peek() + " "); System.out.println(queue); //peek
      System.out.print("peek: " + queue.peek() + " "); System.out.println(queue); //peek
   }
}
```

```
[5, 1, 2, 3, 4]
poll: 5 [1, 2, 3, 4]
poll: 1 [2, 3, 4]
poll: 2 [3, 4]
peek: 3 [3, 4]
peek: 3 [3, 4]
```

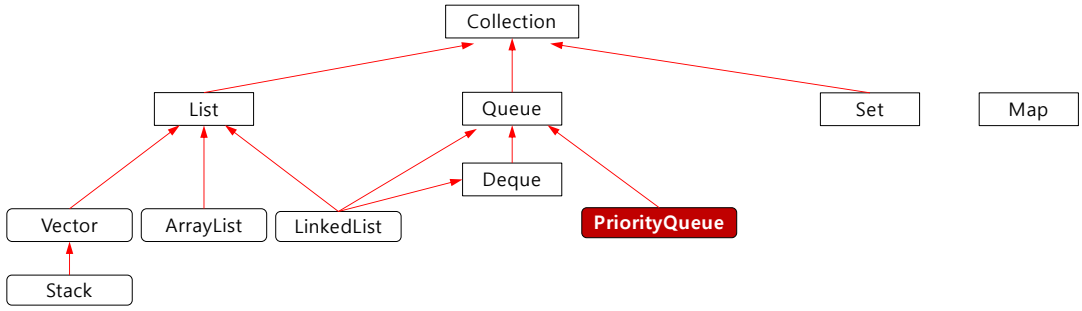
35

Page 35

그래서 LinkedList로 구현된 Queue에 대한 예제인데요
여기서 보면은 Queue<Integer>, LinkedList<Integer> 이렇게 돼 있죠?
신기한 게 여기 Queue 이쪽은 앞에는 Queue인데
뒤에는 LinkedList로 돼 있어요
그건 왜 그러냐면 Queue가 interface이기 때문에 그래요
Queue는 자기가 이렇게 실제로 instancing을 하지 못하죠
interface이기 때문에. 그래서 Queue를 instancing 할 때는
이렇게 실제 자기를 implement하고 있는 하나의 concrete class로
instancing을 해야 됩니다
이 경우에는 LinkedList, 아까 얘기한 대로 LinkedList를 사용해서
Integer type parameter를 사용해서 Queue를 instancing을 한 거고요
그 다음에 queue.offer(5), 그 다음에 1, 2, 3, 4,
offer를 하면 이건 push 하는 거죠
push라고 하지는 않죠, Queue에서는
Queue가 있을 때 처음부터 이렇게 하나 뒤쪽으로
계속 매달려서 5, 1, 2, 3, 4가 되니까
print를 하게 되면 [5, 1, 2, 3, 4] 이렇게 print가 되죠

그 다음에 Queue에서 poll을 세번 했어요 poll을 세번 하면서
여기에서 poll() 된 거를 Queue.poll() 된 거를
그 return value를 계속 여기서 print를 했거든요
그리고 전체 Queue를 print하고 뒤쪽에 그렇게 되면은
poll 한 개 처음에 5가 poll이 되죠?
빠져나가고 그 다음에 나머지 큐는 [1, 2, 3, 4] 남게 되고요
두 번째 poll을 하면 그 다음 제일 앞에 있는 거 1이 빠져나가고 [2, 3, 4]
그 다음에는 맨 앞에 있는 건 2가 빠져나가고 그 다음에 [3, 4] 이런 식으로 그
죠?
그 다음에 peek을 할 때는 어때요?
peek을 할 때는 제일 앞에 있는 게 return이 되긴 하지만 remove 되진 않죠
그래서 [3, 4] 이런 식으로 되게 됩니다
이제 poll하고 peek의 차이를 아시겠죠

class PriorityQueue



36

Page 36

그 다음에 priorityQueue 인데요 priorityQueue는
Queue에서 Queue를 implement 하는 그런 class입니다
Concrete class이고 굉장히 특이한 특징을 가지고 있습니다

PriorityQueue

- (Minimum) Priority Queue: default
  - Minimum data always at the head (NOTE: not completely sorted)
  - After removing head (remove(), poll()), minimum of remaining data at the head
  - ex) PriorityQueue<Integer> pq = **new** PriorityQueue<Integer>();

- (Maximum) Priority Queue
  - Maximum data always at the head (NOTE: not completely sorted)
  - ex) PriorityQueue<Integer> pqr = **new** PriorityQueue<Integer>(Collections.*reverseOrder*());

37

Page 37

그래서 PriorityQueue 중에 기본은 뭐냐면 minimum PriorityQueue입니다
그래서 minimum priority가 default인데요
이거는 data를 막 계속 넣을 때 얘가 무슨 짓을 하냐면
지금까지 들어온 것 중에 minimum이 항상 head에 있도록 합니다
제일 앞에 있도록
그런데 그 특이한 것은 완전히 data들이 다 sorting되어 있는 건 아니에요
sorting되어 있는 건 아닌데 제일 minimum만 제일 앞에 나와 있습니다
그래서 어떤 식으로 그렇게 하는 건지는 우리가 heep이라는
그런 트리 structure를 사용하게 되는데 그것에 대해서는 여러분들
data structure 시간에 배우실 거고 그래서 여기서는 일단
내부에서 어떤 일이 일어나게 되지만 어떤 성질을 갖게 되냐면
PriorityQueue 는 항상 head에 minimum을 가지게 된다라고 보시면 되겠습니다
그래서 head를 remove하고 나면 역시 하나가 minimum이 빠져나가잖아요
minimum을
그런데 그 다음에는 그 나머지 중에서 또 minimum이
가장 head에 와 있게 됩니다

항상 그렇게 돼 있어요
그래서 처음에 instancing 할 때는 PriorityQueue Integer
이런 식으로 주게 되고요
그 다음에 pq = new PriorityQueue<Integer> 이런식으로
일반적으로 그냥 instancing 하는 식으로
type parameter를 줘서 instancing을 하면 되겠습니다
그런데 이 PriorityQueue를 maxumum으로 바꿀 수도 있어요
maxumum은 뭐냐면 Min이 아니라 Maximum data가
항상 head에 있도록 하는 거죠
그렇지만 역시 그 sorting되어 있지는 않습니다
그래서 얘는 예를 들면 PriorityQueue<Integer> 하고
new PriorityQueue<Integer> 하고 여기
Collections.reverseOrder() 이 Collections
아까 나왔었죠 Utility method를 가지고 있는 그런 class라고 했죠?
java.util 밑에 있고요
그래서 Collections 밑에 있는 그 reverseOrder() 라는 것이
이제 이게 Comparator라는 건데 이거를 주게 되면은
이 PriorityQueue 가 Mean Queue가 아니라 Max Queue로 바뀌어 버립니다
그래서 maxumum이 항상 head에 와 있도록 돼 있어요
자 그래서 이런 걸로 굉장히 그 유용한 일들을 많이 할 수 있는데

Methods in class PriorityQueue<E>

Modifier and Type	Method	Description
boolean	add(E e)	Inserts the specified element into this priority queue.
void	clear()	Removes all of the elements from this priority queue.
boolean	contains(Object o)	Returns true if this queue contains the specified element.
Iterator<E>	iterator()	Returns an iterator over the elements in this queue.
boolean	offer(E e)	Inserts the specified element into this priority queue.
boolean	remove(Object o)	Removes a single instance of the specified element from this queue, if it is present.
Object[]	toArray()	Returns an array containing all of the elements in this queue.

38

Page 38

여기서 그 PriorityQueue의 역시 method를 보면
add, clear, contains, iterator, offer
offer는 PriorityQueue에 data 하나를 추가하는 것이 되겠고요
remove는 오브젝트를 head에서 하나 remove하면서
걔를 return하며 remove하는 거고요
그 다음에 Queue에 기본적으로 peek하고 poll 이런 걸 다 가지고 있죠
그 peek이나 poll 같은 거를 다 사용할 수 있습니다
왜냐하면 PriorityQueue는 interface Queue를
implement하고 있기 때문에 그렇죠

PriorityQueue

```
import java.util.*;

public class PriorityQueueTest {
	public static void main(String args[]){
		PriorityQueue<Integer> queue=new PriorityQueue<Integer>();
		queue.add(4);
		System.out.println("Adding 4: " + queue);
		queue.add(7);
		System.out.println("Adding 7: " + queue);
		queue.add(2);
		System.out.println("Adding 2: " + queue);
		queue.add(5);
		System.out.println("Adding 5: " + queue);
		queue.add(9);
		System.out.println("Adding 9: " + queue);
		System.out.print("Removing " + queue.peek() + ": ");
		queue.remove();
		System.out.println(queue);
		System.out.println("Removing " + queue.poll() + ": " + queue);
	}
}
```

```
Adding 4: [4]

Adding 7: [4, 7]

Adding 2: [2, 7, 4]

Adding 5: [2, 5, 4, 7]

Adding 9: [2, 5, 4, 7, 9]

Removing 2: [4, 5, 9, 7]

Removing 4: [5, 7, 9]
```

39

Page 39

그래서 이런 PriorityQueue가 있을 때 또 얘를 한번 볼까요?
PriorityQueue 여기도 지금 priority Queue를 하나 만들었는데
mean Queue죠 여기 default로 아무것도 안 줬기 때문에 4를 add 하고
print 하면 [4] 이렇게 되고요 그 다음에 7을 add 하고
print 하면 [4, 7] 이 됩니다
그래서 4가 더 지금 더 작기 때문에
이 경우에 앞에 4가 항상 head에 있어요
그 다음에 2를 add 하면 이번에는 2가 이 head가 되죠
왜냐하면 2가 4보다 더 작거든요
그런데 희한한 것은 얘가 들어갈 때
[2, 4, 7] 이 되느냐 그건 아니에요
그렇게 되면 이 heep을 유지하기가 힘들어집니다
그 정도로만 그 얘기를 할게요
그래서 이 전체가 sorting되어 있는 건 아니다
하지만 minimum이 제일 앞으로 나온다
그 다음에 5를 집어 넣으면 [2, 5, 4, 7]
왜냐하면 5는 2보다 작지 않기 때문에 뒤쪽으로 가게 되죠

그렇다고 4보다 작기 때문에 얘가 뒤로 가느냐 그것도 아니라는 얘기죠
그 다음에 9를 집어 넣게 되면 [2 5 4 7 9] 9는 또 맨 끝으로 가게 되는데
그렇다고 해서 뭐 이게 지금 sorting이 돼 있느냐
그것도 아니에요 자 그 다음에 Queue를 peek를 하면 peek를 하면
peek는 지금 제일 head에 있는 걸 하나 가져와서 print하게 되고
여기까지는 아직 pop을 안 한 거죠 poll을 안 한 건데
여기서 remove를 해버렸어요 그러니까 removing 2 하고
여기서 peek이 return한 2를 print하는데 정작 2가
remove되는 건 뭐 때문이냐? 여기 remove 때문에
remove 때문에 print되고 그 다음에 Queue를 여기서 print했더니
[4, 5, 9, 7] 이 된 거죠 그 다음에 여기서는 그냥 간단히 for를 써서
Queue에서 for를 해버리고 Queue를 붙여서 print하면
이런 식으로 4가 return 되면서도 얘가 remove가 동시에 되죠
그래서 [5, 7, 9] 만 남게 됩니다
근데 보니까 뭐 597이 579가 되고 그래서 제일 앞으로 나오는 게
4가 앞으로 나오고 5가 앞으로 나오고 해서 minimum을 유지하는 건
맞는데 이 뒤에 순서는 꼭 sorting되어 있는 건 아니다

Interface Iterator

- Interface used to access (in order if exists) the data stored in the collection
- Generating an iterator:
  - ex) // let al be any collection (list, queue, ...)

```
Iterator itr = al.iterator();
```

- Methods in Iterator:

메서드	설 명
boolean hasNext( )	읽어 올 요소가 남아있는지 확인한다. 있으면 true, 없으면 false를 반환한다.
Object next( )	다음 요소를 읽어 온다. next( )를 호출하기 전에 hasNext( )를 호출해서 읽어 올 요소가 있는지 확인하는 것이 안전하다.
void remove( )	next( )로 읽어 온 요소를 삭제한다. next( )를 호출한 다음에 remove( )를 호출해야한다.(선택적 기능)

40

Page 40

자 그 다음에 iterator 라는 interface에 대해서 얘기를 하려고 합니다
뭐냐면, order가 있는, 그러니까 오더가 있을 때,
그 Collection의 data들에 access를 순서대로 가능하도록 하는 것입니다.
그래서 iterator를 generate 할 때는 이런 식으로 하는 거죠
al 이 어떤 컬렉션이라고 하면
그 al 에서 iterator라는 method를 call 하면
바로 iterator를 새로 하나 만들어 줍니다
그러고 나선 이 iterator가 call 할 수 있는 method는
hasNext, next, remove 이런 것들이 있습니다
hasNext는 읽어올 요소가 남았는지 확인하고
있으면 true, 없으면 false를 return합니다
그 다음에 next는 다음 요소를 읽어오고요
next를 호출하기 전에 hasNext를 호출해서
읽어올 요소가 있는지 확인하는 것이 안전하다 그랬습니다
그 다음에 remove는 next로 읽어온 요소를 삭제하는데
next 를 호출한 다음에 remove 를 호출해야 됩니다
반드시 앞에 next 를 먼저 call 해야 된다 라는 얘기가 되겠습니다

Example: Using Iterator

```
import java.util.ArrayList;
import java.util.Iterator;

public class IteratorTest {

   public static void main(String args[]) {
      ArrayList al = new ArrayList();
      al.add("C");
      al.add("A");
      al.add("E");
      al.add("B");
      al.add("D");
      al.add("F");

      System.out.print("Original contents of al: ");
      Iterator itr = al.iterator();

      while(itr.hasNext()) {
         Object element = itr.next();
         System.out.print(element + " ");
      }
   }
}
```

```
Original contents of al: C A E B D F
```

41

Page 41

example을 하나 보면요
iterator도 역시 java.util 밑에 있고
arrayList를 쓰기 위해서 java.util arrayList를 import했습니다
그리고 나서 arrayList를 하나 만들었죠 arrayList를 만들어서 new 해가지고
add를 C, A, E, B, D, F 이렇게 넣었습니다
Original Content of al 은 지금 iterator를 사용해서
한 바퀴 돌면서 print를 하려고 그래요
iterator를 새로 만들었는데 al 이 iterator를 call 해서
itr 이라는 iterator를 하나 만들었고요
itr의 hasNext는 불리한을 return하는데
뒤에 것이 있는지 없는지를 return한다 그랬죠.
hasNext가, 즉 Next가 있을 경우에 while 문으로 진입해서
element는 itr.next() 를 읽어오는 거죠
hasNext를 먼저 테스트해보고 그 다음에 next를 사용해라
그 다음에 System.out.println
print 해가지고 element를 print하게 이렇게 했습니다
이렇게 하면은 오리지널 콘텐츠 오브 al 은 들어간 순서대로

C A E B D F 이렇게 print가 되게 됩니다
사실은 for문을 쓰고 get을 쓰는 거나 별 차이는 없습니다
그런데 이제 이 iterator가 유용한 것은 특히 그 뒤쪽에 오더가 없는
그런 Collection에 가면 어떻게 읽어야 되는지
전부 다 한 번씩 traverse를 해야 되는데 오더가 없기 때문에 좀 힘들어지거든
요
그럴 때 이제 아이터레이터를 유용하게 사용할 수 있습니다

ListIterator

- Improves the accessibility of Iterator (unidirectional → bidirectional)

메서드	설 명
boolean hasNext( )	읽어 올 다음 요소가 남아있는지 확인한다. 있으면 true, 없으면 false를 반환한다.
boolean hasPrevious( )	읽어 올 이전 요소가 남아있는지 확인한다. 있으면 true, 없으면 false를 반환한다.
Object next( )	다음 요소를 읽어 온다. next( )를 호출하기 전에 hasNext( )를 호출해서 읽어 올 요소가 있는지 확인하는 것이 안전하다.
Object previous( )	이전 요소를 읽어 온다. previous( )를 호출하기 전에 hasPrevious( )를 호출해서 읽어 올 요소가 있는지 확인하는 것이 안전하다.
int nextIndex( )	다음 요소의 index를 반환한다.
int previousIndex( )	이전 요소의 index를 반환한다.
void add(Object o)	컬렉션에 새로운 객체(o)를 추가한다.(선택적 기능)
void remove( )	next( ) 또는 previous( )로 읽어 온 요소를 삭제한다. 반드시 next( )나 previous( )를 먼저 호출한 다음에 이 메서드를 호출해야한다.(선택적 기능)
void set(Object o)	next( ) 또는 previous( )로 읽어 온 요소를 지정된 객체(o)로 변경한다. 반드시 next( )나 previous( )를 먼저 호출한 다음에 이 메서드를 호출해야한다.(선택적 기능)

42

Page 42

그리고 ListIterator 는 bidirectional 로 바꿀 수가 있어요
그래서 next, previous 그러니까 뒤쪽으로 가는 거
앞쪽으로 가는 거 이거 다 가능해지고요
그 다음에 next index, previous index
그 다음에 add, remove, set
새로운 걸 추가한다거나 remove한다거나 아니면 변경한다거나
그럴 때 사용할 수 있습니다

Example: Using ListIterator

```
import java.util.*;

public class ListIteratorTest {
   public static void main(String args[]) {
      ArrayList al = new ArrayList();
      al.add("C");
      al.add("A");
      al.add("E");

      ListIterator litr = al.listIterator();
      while(litr.hasNext()) {
         Object element = litr.next();
         litr.set(element + "+");
      }

      System.out.print("Modified List backwards: ");
      while(litr.hasPrevious()) {
         Object element = litr.previous();
         System.out.print(element + " ");
      }
      System.out.println();
   }
}
```

```
Modified List backwards: E+ A+ C+
```

43

Page 43

ListIterator 를 쓰는 것을 한번 볼게요 자
ArrayList al 역시 그 ArrayList를 만들었습니다 C, A, E 를 집어 넣었죠
처음에 ListIterator 를 하나 만들어서요 next 방향으로 진행하면서
그 앞에서 뒤로 진행하는 거죠 C, A, E 순서대로 하나씩 진행하면서
element를 next로 읽어 오게 되고요 그럼 얘를 현재 element에다가
플러스를 더 추가한 스트링으로 세팅을 하는 거죠
현재 C, A, E 가 있으니까 얘를 E+, A+, C+ 로 변경하게 됩니다
지금 순서가 반대로 나오고 있죠
그건 왜 그러냐 지금 litr이라는 List iterator가 지금 끝까지 다 갔어요
hasNext가 null 일 때까지 false일 때까지 끝까지 다 가고 나서
그 다음에 반대 방향으로 돌아오려고 하는 겁니다
litr의 has previous를 불러가지고 previous 쪽으로 계속 돌아와서
첫번 element까지 litr previous로 계속 읽어오고
element를 print하게 하는 거죠
이렇게 하면 여기서 순서는 C, A, E 가 될텐데
거꾸로 돌아오기 때문에 E, A, C 가 되고
거기에 아까 set을 했기 때문에

+ 가 더해져서 E+, A+, C+ 가 print 되게 되겠죠
ListIterator를 정방향, 역방향으로 왔다 갔다 하는데
사용할 수 있다라는 얘기가 되겠습니다

java.util.Arrays Class (1/3)

- Offer the static methods to manipulate arrays easily

```
import java.util.Arrays;
...

double[] values = {1.0, 1.1, 1.2};
int[][] arr = {{1,2,3,4,5},{5,4,3,2,1}};

System.out.println(values.toString());              // [D@46a49e6...
System.out.println(Arrays.toString(values));    // [1.0, 1.1, 1.2]
System.out.println(Arrays.deepToString(arr)); // [[1,2,3,4,5],[5,4,3,2,1]]

int[] arr1 = { 1, 2 };
int[] arr2 = { 1, 2 };
int[][] arr3 = {{1,2,3,4,5},{5,4,3,2,1}};

System.out.println(Arrays.equals(arr1, arr2));        // true
System.out.println(Arrays.deepEquals(arr, arr3));  // true
```

44

Page 44

마지막으로 우리가 Arrays라는 유틸리티가 있어요 유틸리티 class가 있는데
이게 이제 Arrays 뒤에 s가 붙어 있죠
이 Arrays가 굉장히 유용한 유틸리티입니다
그래서 여러분들이 아마 진작 알았으면 이 Arrays를 사용하려고 했을 텐데
그래서 보면 double type의 어레인데 { 1.0, 1.1, 1.2} 이렇게 들어가 있고
이거는 2D array arr이네요 그래서 이런 식으로 들어가 있습니다
이 values라는 array에 toString을 직접 하게 되면 이런 식으로 나오게 돼요
toString을 가지고는 있는데, 왜 가지고 있냐면
이 values가 arra가 되면 그 순간 오브젝트가 되거든요
그러니까 Object 의 toString을 이용하게 되는 거죠
그런데 우리가 이런 프리미티브 type이나 어떤 array 이든 간에
그 array type을 Object로 사용할 수는 있지만
그 Object class 자체의 method를 오버라이딩 할 수는 없잖아요
그래서 이 toString이 이상한 거 이렇게 print가 되게 되는데
이걸 바꿀 수가 없습니다
이럴 때는 Arrays를 사용하면요
Arrays에 toString이라는 이런 스태틱 유틸리티 method가 있습니다

그래서 여기에다가 values라는 array를 넣어주게 되면
이렇게 아까 우리가 Collections 에서 사용하던 그 print, println처럼
똑같이 이렇게 print가 되게 됩니다
그 다음에 Arrays.deepTostring 은
2 dimension 이상의 multi-dimensional array를
이렇게 print해주는데 사용합니다
equals인데요 이거는 역시 두 개의 array를 비교할 때
우리가 두 개의 array 비교할 때 이걸 equals 같은 것을 그냥
우리가 직접 짠다고 하면은 for 문 두 번 돌려서 이렇게 하는
그런 boolean method를 짜야 되겠죠? 근데
그것도 다 제공이 되고 있습니다 그래서 두 개의 array가 같을 때
이 arr1, arr2 가 내용이 같다는 거죠
내용이 같으면 이제 true를 return하게 되는 거고요
그 다음에 deepEquals 는 여기 앞에 있는 arr 과 arr3 가
두 개가 내용이 똑같기 때문에 이것도 true로 하게 됩니다
즉 deepEquals() 는 multi-dimensional array의
equality를 테스트하는데 사용하게 됩니다

java.util.Arrays Class (2/3)

- Copy: copyOf(), copyOfRange()

```
int[] arr = {0,1,2,3,4};
int[] arr2 = Arrays.copyOf(arr, arr.length);      // [0,1,2,3,4]
int[] arr3 = Arrays.copyOf(arr, 3);               // [0,1,2]
int[] arr4 = Arrays.copyOf(arr, 7);               // [0,1,2,3,4,0,0]
int[] arr5 = Arrays.copyOfRange(arr, 2, 4);       // [2,3]
int[] arr6 = Arrays.copyOfRange(arr, 0, 7);       // [0,1,2,3,4,0,0]
```

- Fill: fill(), setAll()

```
int[] arr = new int[5];
Arrays.fill(arr, 9);                                          // [9,9,9,9,9]
Arrays.setAll(arr, () -> (int) (Math.random() * 5) + 1);      // [1,5,2,1,1]
```

45

Page 45

자 그 다음에 copyOf 하고 copyOfRange 가 있는데
copyOf 는 original array에 어느 영역에 들어가는 거를 만들어 주는 거죠
그래서 기본적으로 arr이 0, 1, 2, 3, 4일 때 arr length까지 length만큼의 크기를
그러니까 이걸 전부 다 들어가게 되겠죠
그래서 arr2는 0, 1, 2, 3, 4가 그대로 copy 가 되게 되고요
copyOf(arr, 3) 하면은 element 3개만 앞에서부터 0, 1, 2만 copy되고요
그 다음에 7은 이게 지금 5개밖에 없으니까 2개가 모자라잖아요
모자라는 것은 0으로 채워서 카피가 돼서 arr4로 만들어주게 됩니다
copyOfRange는 2부터 시작을 해서 4까지
0, 1, 2, 3, 4니까 시작 index endindex,
그래서 endindex-1까지 copy를 하게 되는 거죠
이것도 역시 이 index가 초과를 할 경우에
그럴 경우엔 0으로 채워서 이렇게 만들어 주게 됩니다
fill하고 setAll인데, fill은 어떤 같은 element로
다 arr을 fill하고 싶을 때는 이렇게 99999 이렇게 fill 되는 거고요
setAll은 이 뒤에 들어갈 value를 이런 식으로 지정을 할 수가 있어요

그래서 Math.random() * 5 + 1 이런 식으로
이렇게 하면 이게 1부터 6 사이에 랜덤 넘버가 채워지게 되겠죠

java.util.Arrays Class (3/3)

- Convert to List: asList(Object... a)

```
ArrayList<Integer> list = new ArrayList<Integer>(Arrays.asList(1, 2, 3, 4, 5));
```

- Sort and Search: sort(), binarySearch()

```
int[] arr = {3, 2, 0, 1, 4};

Arrays.sort(arr);                          // sorting the array arr
System.out.println(Arrays.toString(arr));  // [0, 1, 2, 3, 4]
int index = Arrays.binarySearch(arr, 2);   // index = 2
```

46

Page 46

자, 그 다음에 asList라는 게 있는데 이거는 List로 만들어주는 것이 되겠습니다
그래서 여기에 array element들을 쭉 넣어요 이런 식으로
그래서 쭉 넣어주고 이렇게 되면 이걸 List로 변환해서 return을 해 주는 거죠
그래서 이게 뭐에 편리하냐면 이 Collection들의 그 constructor에 보면
List를 뭉테기로 받아서 걔를 data로 한 List를 만드는
ArrayList나 아니면 뭐 LinkedList나 이런 걸 다 가지고 있습니다
그래서 data를 실제로 이렇게 주어서 만드는 방법이 없나요 하고
전부 add add add 하나씩 add를 해야 되나요
이렇게 고민을 하고 있을 수 있습니다
여기서 보면 이제 이런 이런 식으로 그 data를 쉽게 주어서
이렇게 List 로 만들어서 넘겨주게
Collection 으로 만들어서 넘겨 주게 되기 때문에
이렇게 Initialize 할 때 쉽게 사용할 수 있습니다
그 다음에 Sort하고 Binary Search도 다 구현이 돼 있어요
Arrays에 Sort하면 기본적으로 Array를 Sorting 하는 거고
그 다음에 [3 2 0 1 4] 는 [0 1 2 3 4] 로 이렇게 print 되는 거고
Binary Search는 뭐예요?

arr에 2가 있으면 Index를 return하고 아니면 -1을 return하게 돼 있습니다
여기까지 하도록 하겠습니다