Data Structure

(Java programming)

Chapter 11.

Contents

- **Dictionary**
- **Practice**

- ***Dictionary***

• Dictionary

Dictionary

- Like a map, a ***dictionary*** models a searchable collection of (*key*, *value*) entries
- Unlike a map, multiple entries with the same key are allowed
  - Cf., **`put`** of map vs. **`insert`** of dictionary
- The main operations of a dictionary are also searching, inserting and deleting entries
- Applications
  - English dictionary (same words)
  - Phone book (same names)
  - Etc.

• Dictionary

Dictionary Queue ADT

- `entry` **`find`**`(`***k***`)`
  - If the dictionary has an entry with key ***k***, returns it, else, returns null
- `list` **`findAll`**`(`***k***`)`
  - Returns a collection (list) of all entries with key ***k***
- `entry` **`insert`**`(`***k***`,` ***v***`)`
  - Inserts and returns the entry (***k***, ***v***)
  - Just inserts, does **NOT** replace the old entry
- `entry` **`remove`**`(`***e***`)`
  - Removes and returns the entry ***e*** from the dictionary, if fails, returns null
  - Cf., `entry` **`remove`**`(`***k***`)`: find an entry with key ***k*** and remove it
- `list` **`entries`**`()`
  - Returns a list of all the entries in the dictionary
- Common operations: **`size`**`()`, **`isEmpty`**`()`

• Dictionary

Dictionary Example

Operation	*Output*	*Dictionary*
insert(5,*A*)	(5,*A*)	(5,*A*)
insert(7,*B*)	(7,*B*)	(5,*A*), (7,*B*)
insert(2,*C*)	(2,*C*)	(5,*A*), (7,*B*), (2,*C*)
insert(8,*D*)	(8,*D*)	(5,*A*), (7,*B*), (2,*C*), (8,*D*)
insert(2,*E*)	(2,*E*)	(5,*A*), (7,*B*), (2,*C*), (8,*D*), (2,*E*)
find(7)	(7,*B*)	(5,*A*), (**7**,*B*), (2,*C*), (8,*D*), (2,*E*)
find(4)	**null**	(5,*A*), (7,*B*), (2,*C*), (8,*D*), (2,*E*)
find(2)	(2,*C*)	(5,*A*), (7,*B*), (**2**,*C*), (8,*D*), (2,*E*)
findAll(2)	{(2,*C*), (2,*E*)}	(5,*A*), (7,*B*), (**2**,*C*), (8,*D*), (**2**,*E*)
size()	5	(5,*A*), (7,*B*), (2,*C*), (8,*D*), (2,*E*)
remove(find(5))	(5,*A*)	(7,*B*), (2,*C*), (8,*D*), (2,*E*)
find(5)	**null**	(7,*B*), (2,*C*), (8,*D*), (2,*E*)

• Dictionary

Dictionary Operations

Algorithm findAll(k):
Input: A key k
Output: An iterable collection of entries with key equal to k
Create an initially-empty list L
for each entry e in D.entries() **do**
if e.getKey() $= k$ **then**
L.addLast(e)
return L {the elements in L are the selected entries}

• Dictionary

Dictionary Operations

Algorithm insert(k, v):
  Input: A key k and value v
  Output: The entry (k, v) added to D
  Create a new entry $e = (k, v)$
  S.addLast(e) {S is unordered}
  return e

Algorithm remove(e):
  Input: An entry e
  Output: The removed entry e
  {We don't assume here that e stores its location in S}
  for each node p in S.nodes() **do**
    if p.element() $= e$ **then**
      S.remove(p)
      return e
  return null {there is no entry e in D}

- ***Practice***

- **Practice**

LinkedList를 이용한 Dictionary 구현

- Dictionary Class
- DictionaryNode Class
- Main Class

- **Practice**

DictionaryNode class

- key와 value를 멤버변수로 가짐(String)

```
public class DictionaryNode {

    private DictionaryNode prev;
    private DictionaryNode next;
    private String key;
    private String value;
```

- getter, setter 구현

- **Practice**

Dictionary class

- LinkedList를 이용

- DictionaryNode header, trailer 가짐

```
public class Dictionary {

    private DictionaryNode header;
    private DictionaryNode trailer;
```

- **Practice**

Dictionary class

- insert : key와 value를 입력으로 받음
  Dictionary에 순차적으로 입력
  (key값 중복되게 입력 가능)

```
public void insert(int key, String value){

}
```

- **Practice**

Dictionary class

- remove : key를 입력으로 받아 해당 key에 해당되는 값 모두 삭제

- find : key를 입력으로 받아 해당 key에 해당되는 첫번째 값의 value return

- findAll : key를 입력으로 받아 해당 key를 가지는 value값 모두 출력

- **Practice**

Dictionary class

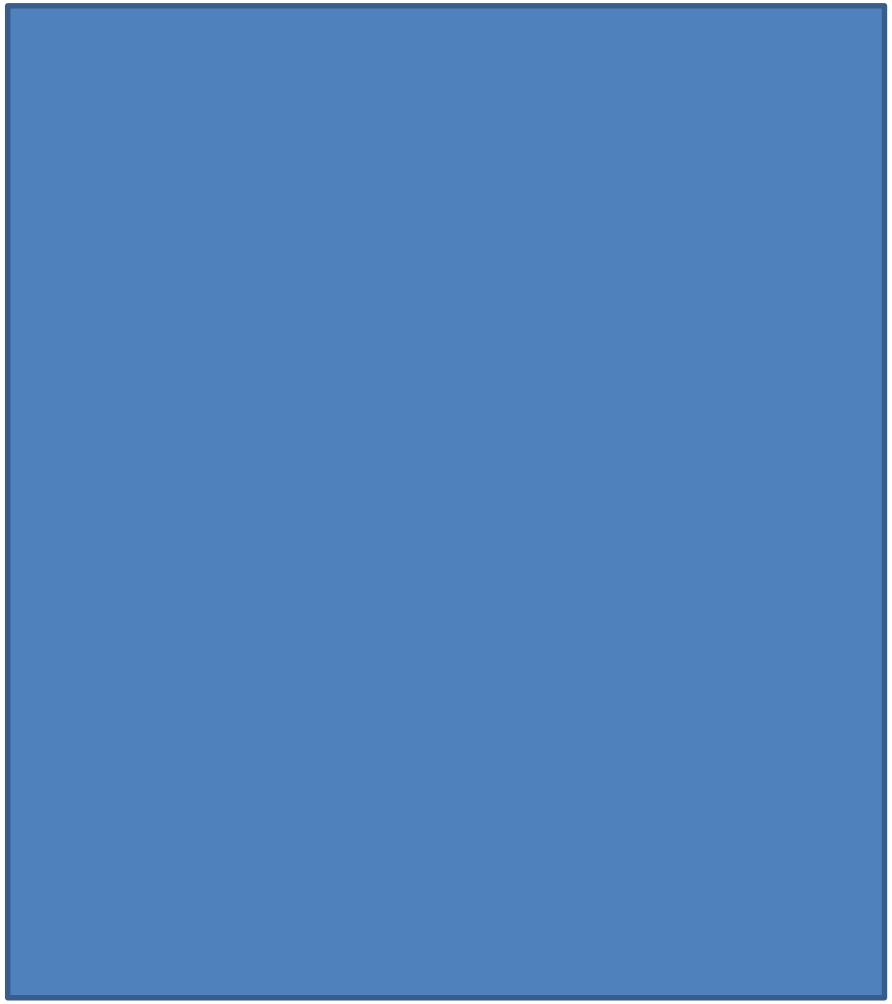
```
public void remove(int key){
```

```
}
```

- **Practice**

Dictionary class

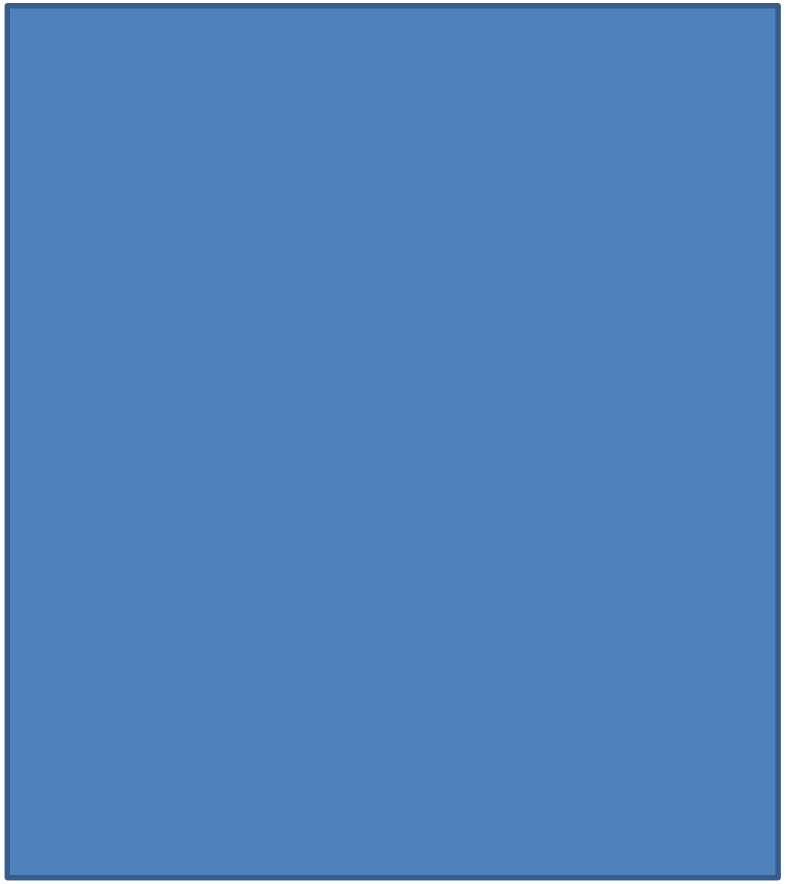
```
public String find(int key){
```



```
}
```

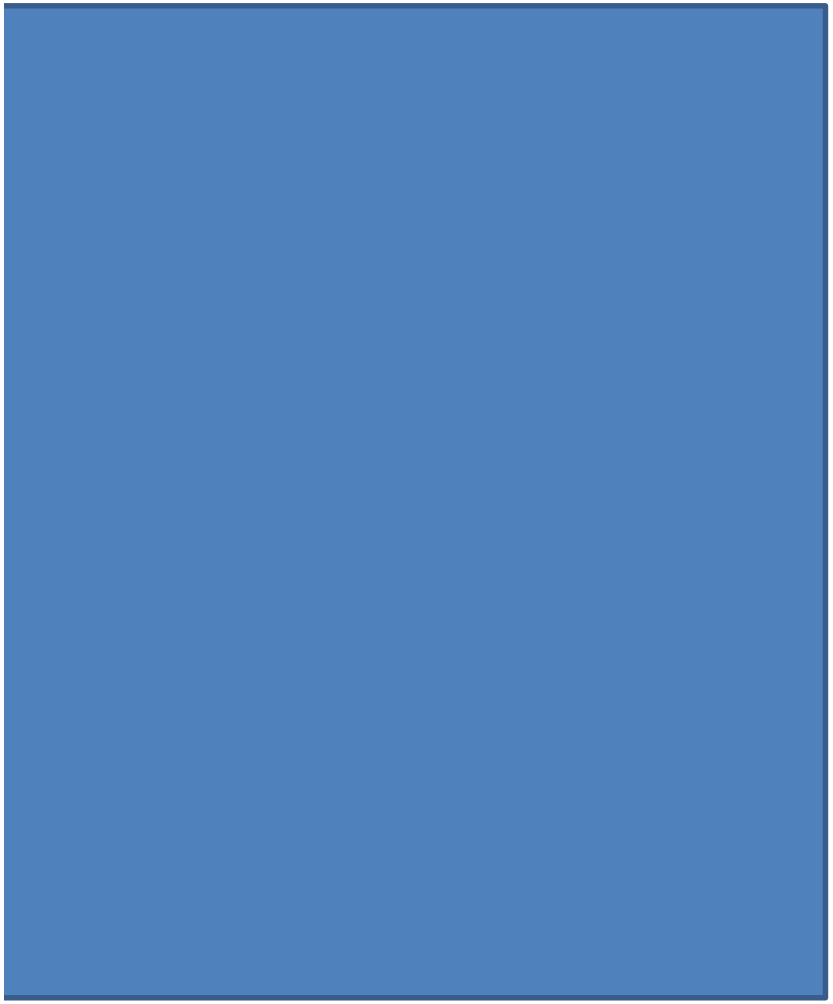
- # Practice

```
public void findAll(int key){
```



```
}
```

```
public String find(int key){
```



```
}
```

- # Practice

```
Dictionary dic = new Dictionary();

dic.insert("apple", "sweet");
dic.insert("banana", "sweet");
dic.insert("apple", "fresh");
dic.insert("bear", "pretty");
dic.insert("bear", "smart");

dic.print();

System.out.println(dic.find("bear"));

dic.findAll("apple");
dic.findAll("bear");
```

```
key : apple, value : sweet
key : banana, value : sweet
key : apple, value : fresh
key : bear, value : pretty
key : bear, value : smart
pretty
[sweet] [fresh]
[pretty] [smart]
```

• Submission

e-mail : 2017kudatastructure@gmail.com

Attach methods :

1. Create zip file (java project folder)
2. Modify the file names :
   ID_name_datastructure[Classcode].dat
   ex) 201173378_이명재_datastructure_chapter11[2403-1].dat
3. e-mail title :
   datastructure_name_chapter
   ex) datastructure_최수용_chapter11[2404-1]

Classcode

2403-1 **[A-1반]**	2403-2 **[A-2반]**	
2404-1 **[B-1반]**	2404-2 **[B-2반]**	2405 **[C반]**