Collections

Lab 7

TA : Hyuna Seo, Kichang Yang, Minkyung Kim, Jaeyong Kim

SEOUL NATIONAL UNIVERSITY

Overview

- Lecture Recap
  - Collections
    - List
    - HashMap
- Problem 1. Simple Diary Application(1)

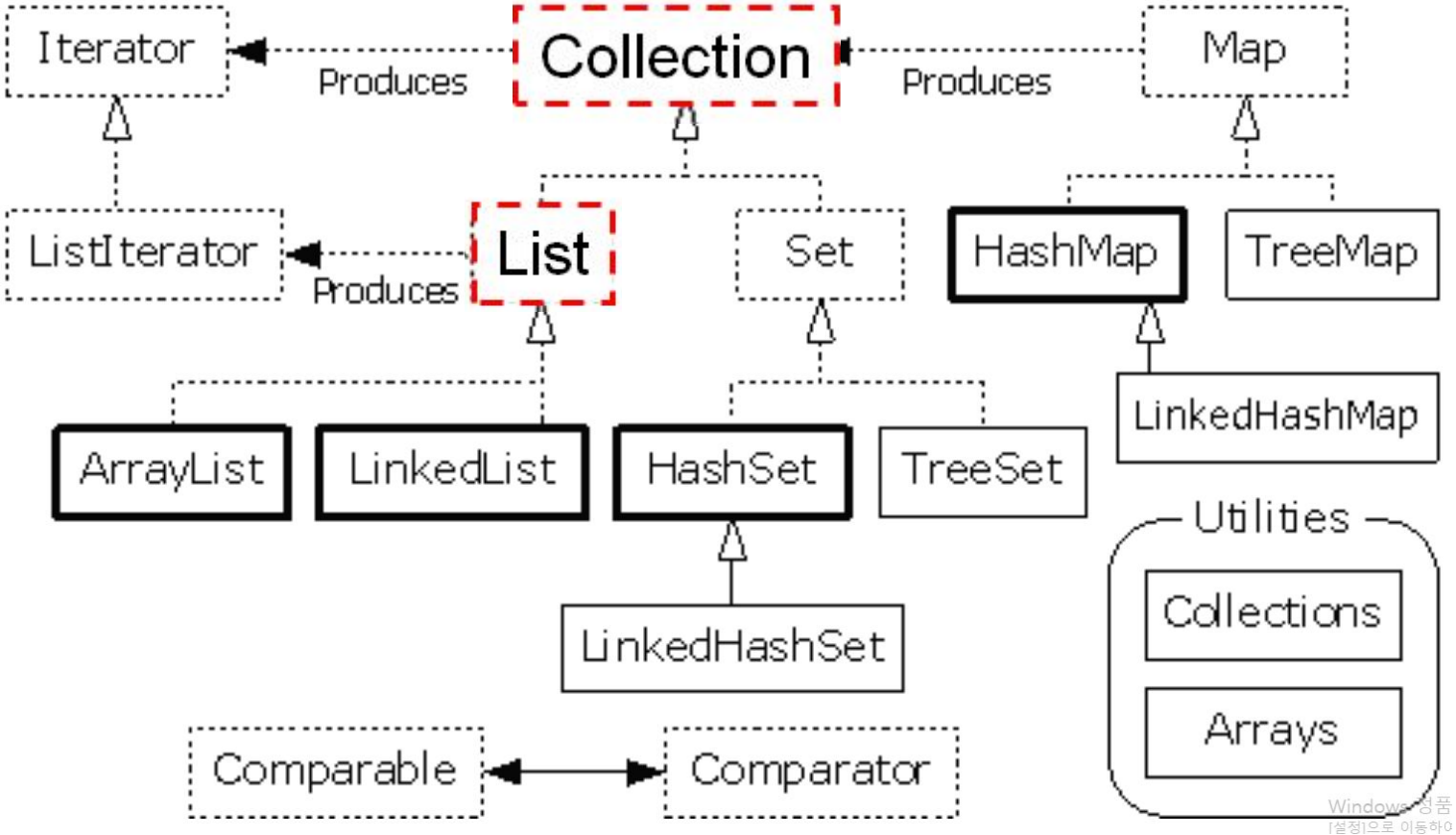
Recap: `Collection` Interface

- The `Collection` interface is the root interface of other collections (e.g., `List` and `Set`).

- Different collections have different characteristics.
  - Ordered vs. unordered
  - Duplicate elements allowed vs. unique elements only.

List Interface

- List interface extends Collection interface.



Windows 정품
[설정]으로 이동하여

`List` Interface

- An “ordered” collection (a.k.a. a sequence).
- The user has the precise control over where in the list each element is stored.
- The user can access elements in the list by their integer index (position in the list).
- Lists allow duplicate elements. (Allow `e1` and `e2` such that `e1.equals(e2)`.).
- `List` is defined in java.util.List.

`List` Interface Methods (1/2)

- `boolean add(E e)`
  - Appends the specified element to the end of this list.
- `void add(int index, E element)`
  - Inserts the specified element at the specified position in this list.
- `E get(int index)`
  - Returns the element at the specified position in this list.
- `E remove(int index)`
  - Removes an element from 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.

`List` Interface Methods (2/2)

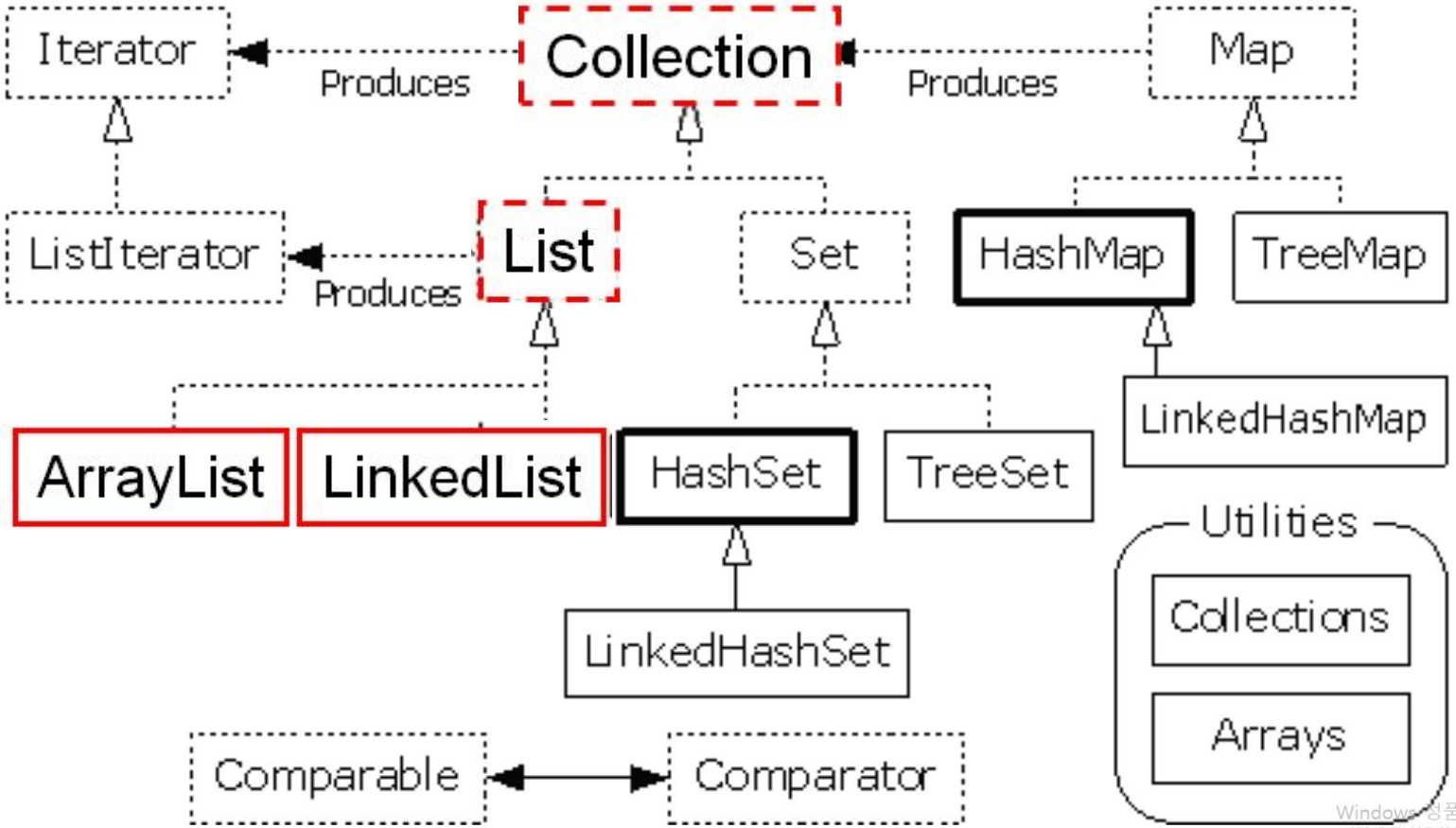
- `List<E> subList(int fromIndex, int toIndex)`
  - Returns a view of the portion of this list between the specified from Index, inclusive, and toIndex, exclusive.
- `int indexOf(Object o)`
  - Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.

List Interface Example

```
List<String> list = new ArrayList<>();
list.add("Apple"); // ["Apple"]
list.add("Banana"); // ["Apple", "Banana"]
list.add("Carrot"); // ["Apple", "Banana", "Carrot"]
list.remove(1); // ["Apple", "Carrot"]
list.size(); // 2
list.get(1); // "Carrot"
list.contains("Banana"); // false
...
```

`List` Implementations

- `ArrayList` and `LinkedList` classes implement `List` interface.



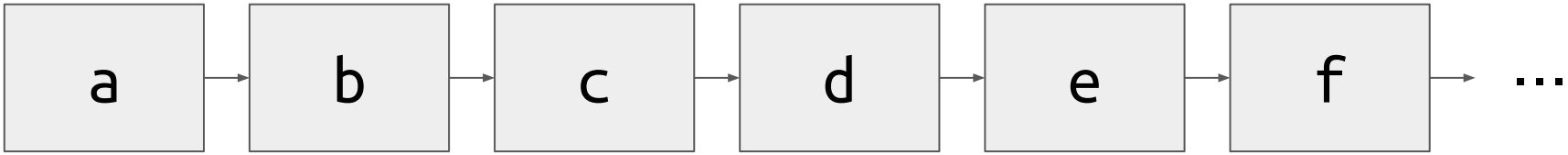
`LinkedList` Methods

- LinkedList provides a set of methods.
  - `void addFirst(Object o)`, `void addLast(Object o)`
  - `Object getFirst()`, `Object getLast()`
  - `Object removeFirst()`, `Object removeLast()`
- Other methods in the `List` interface are provided too. But, be careful when you use them.
- **`ListIterator`** helps traverse the list and add/remove an item at a position.
  - `add()` from `ListIterator` to add at a position.
  - `remove()` from `ListIterator` to remove at a position.

Looping Through `LinkedList`

```
for(int i = 0; i < 100; i++) {
  l.get(i);
}
```

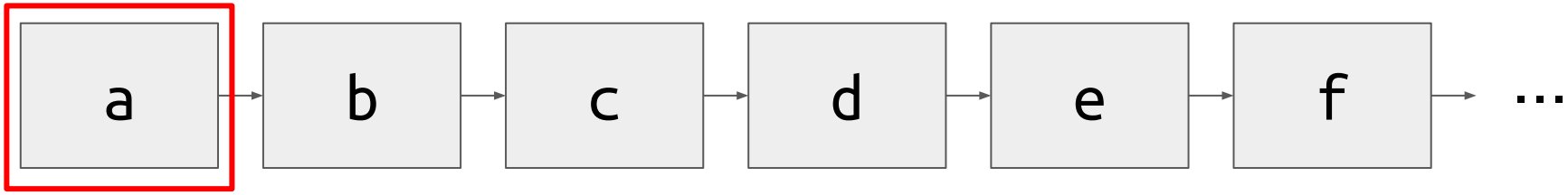
`l.get(0)` →



Looping Through `LinkedList`

```
for(int i = 0; i < 100; i++) {
  l.get(i);
}
```

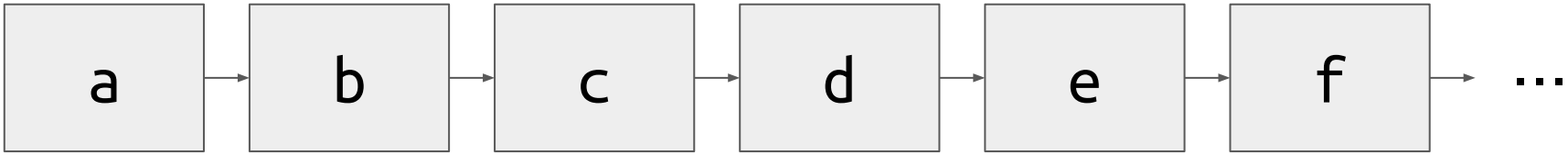
`l.get(0)` →



Looping Through `LinkedList`

```
for(int i = 0; i < 100; i++) {
  l.get(i);
}
```

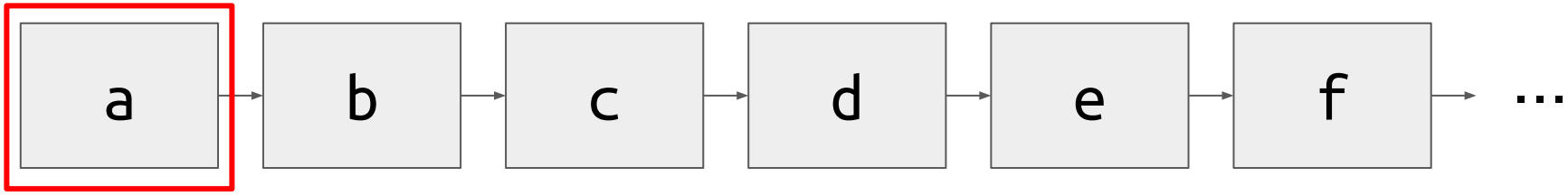
`l.get(1)` →



Looping Through `LinkedList`

```
for(int i = 0; i < 100; i++) {
  l.get(i);
}
```

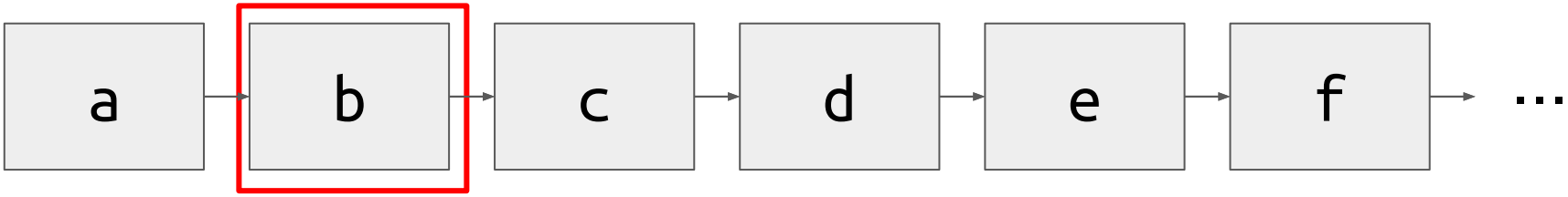
`l.get(1)` →



Looping Through `LinkedList`

```
for(int i = 0; i < 100; i++) {
  l.get(i);
}
```

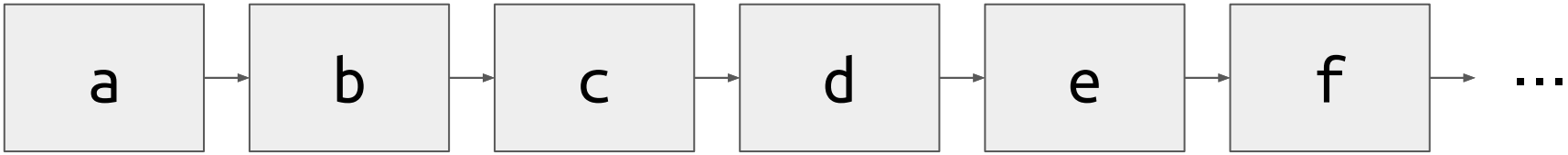
`l.get(1)` → b



Looping Through `LinkedList`

```
for(int i = 0; i < 100; i++) {
  l.get(i);
}
```

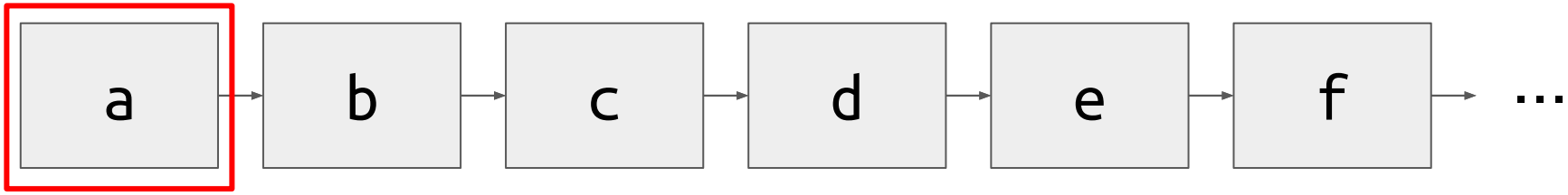
`l.get(2)` →



Looping Through `LinkedList`

```
for(int i = 0; i < 100; i++) {
  l.get(i);
}
```

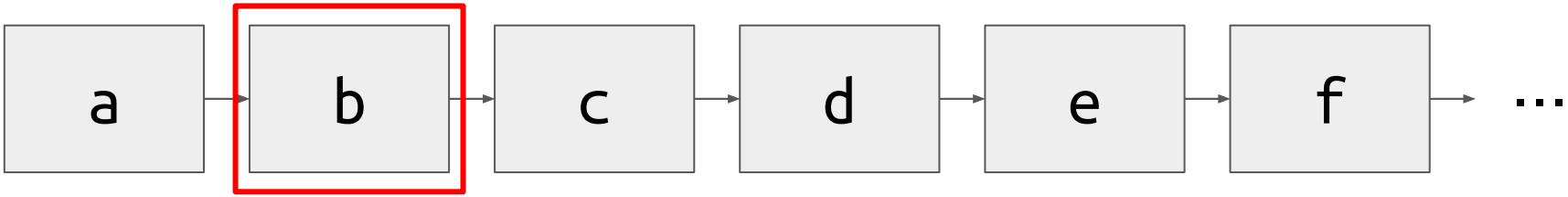
`l.get(2)` →



Looping Through `LinkedList`

```
for(int i = 0; i < 100; i++) {
  l.get(i);
}
```

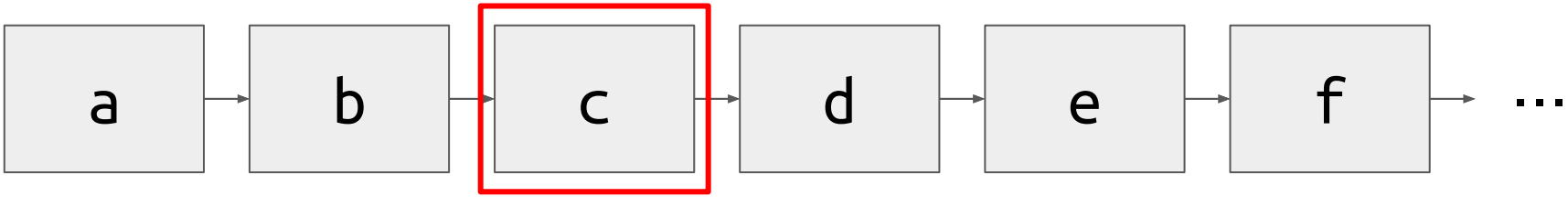
`l.get(2)` →



Looping Through `LinkedList`

```
for(int i = 0; i < 100; i++) {
  l.get(i);
}
```

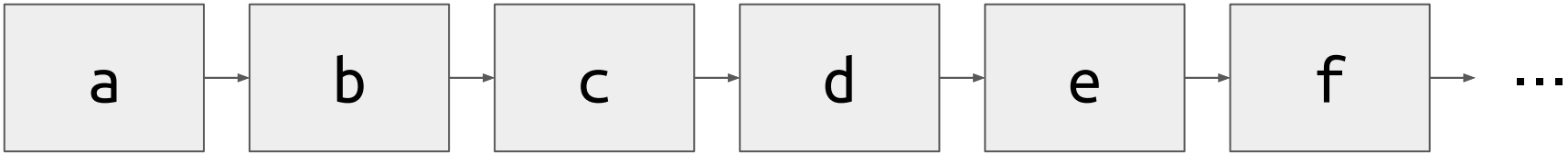
`l.get(2)` → `c`



Looping Through `LinkedList`

```
for(int i = 0; i < 100; i++) {
  l.get(i);
}
```

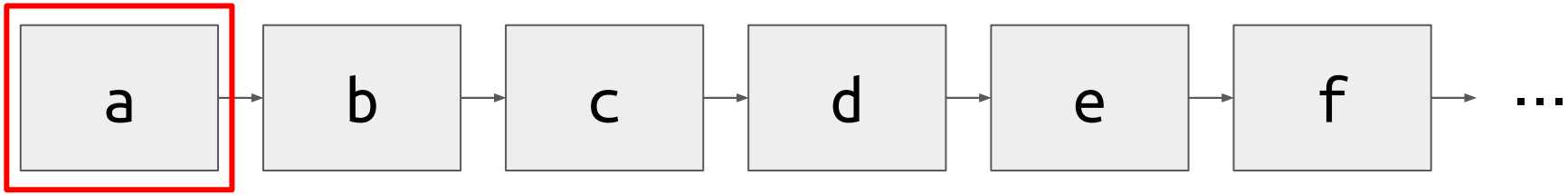
`l.get(3)` →



Looping Through `LinkedList`

```
for(int i = 0; i < 100; i++) {
  l.get(i);
}
```

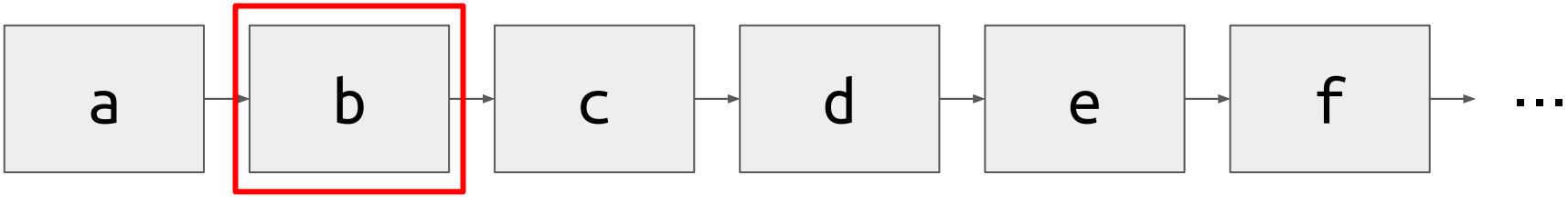
`l.get(3)` →



Looping Through `LinkedList`

```
for(int i = 0; i < 100; i++) {
  l.get(i);
}
```

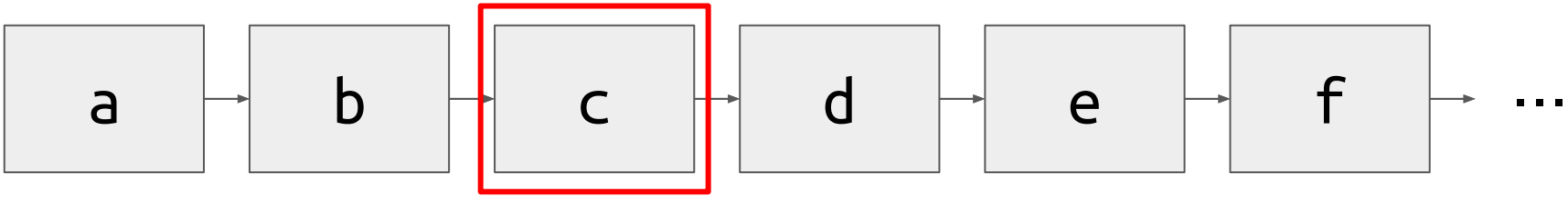
`l.get(3)` →



Looping Through `LinkedList`

```
for(int i = 0; i < 100; i++) {
  l.get(i);
}
```

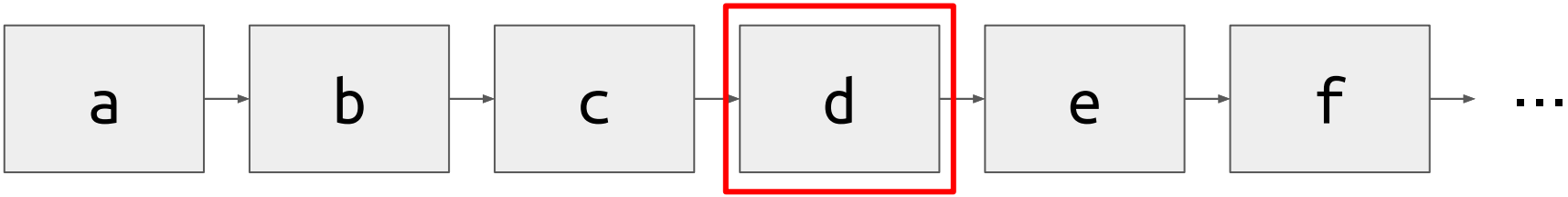
`l.get(3)` →



Looping Through `LinkedList`

```
for(int i = 0; i < 100; i++) {
  l.get(i);
}
```

`l.get(3)` → `d`



Too much overhead!

Iterator Example

main Method

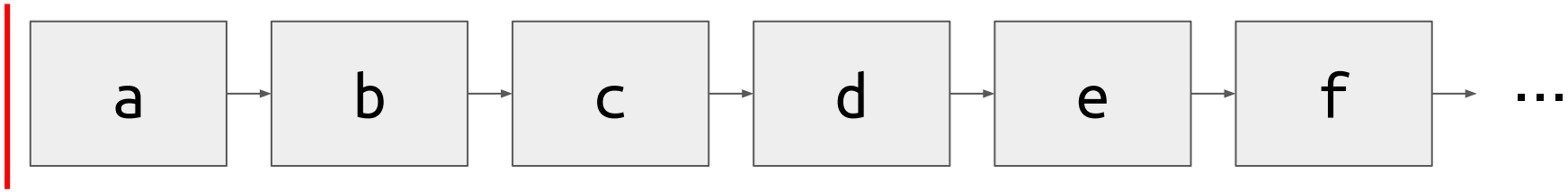
```
LinkedList<Integer> list =
    new LinkedList<>(Arrays.asList(1, 2, 3, 4, 5));
Iterator<Integer> iterator = list.iterator();
while (iterator.hasNext()) {
    int i = iterator.next();
    if (i == 2 || i == 4) { iterator.remove(); }
}
for (int e : list) { System.out.print(e + " "); }
System.out.println();
```

Output

```
1 3 5
```

Efficient Looping with `Iterator`

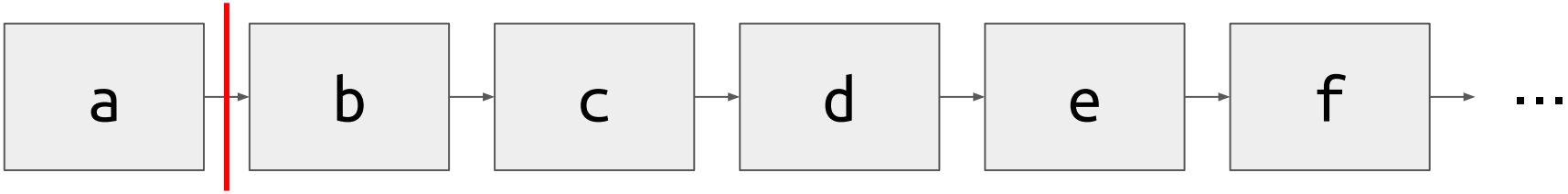
```
while(iterator.hasNext()) {
  iterator.next()
}
```



Efficient Looping with `Iterator`

```
while(iterator.hasNext()) {
  iterator.next()
}
```

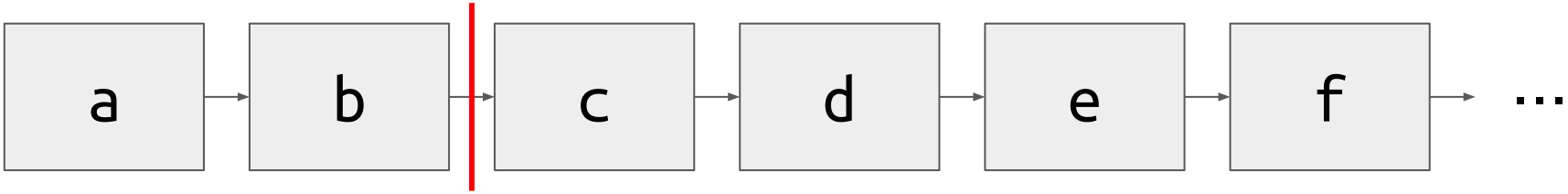
`iterator.next()` → a



Efficient Looping with `Iterator`

```
while(iterator.hasNext()) {
  iterator.next()
}
```

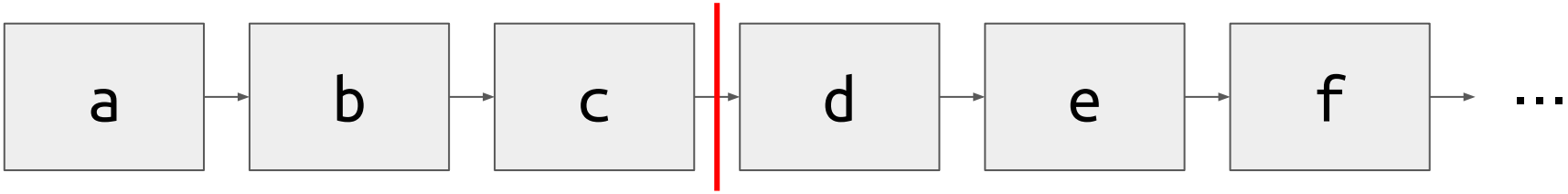
`iterator.next()` → b



Efficient Looping with `Iterator`

```
while(iterator.hasNext()) {
  iterator.next()
}
```

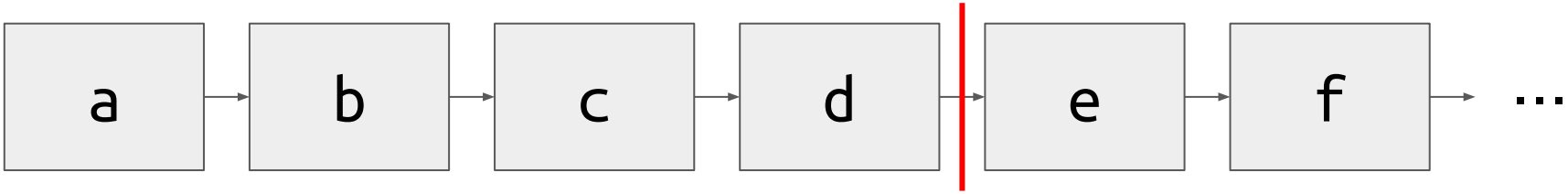
`iterator.next() → c`



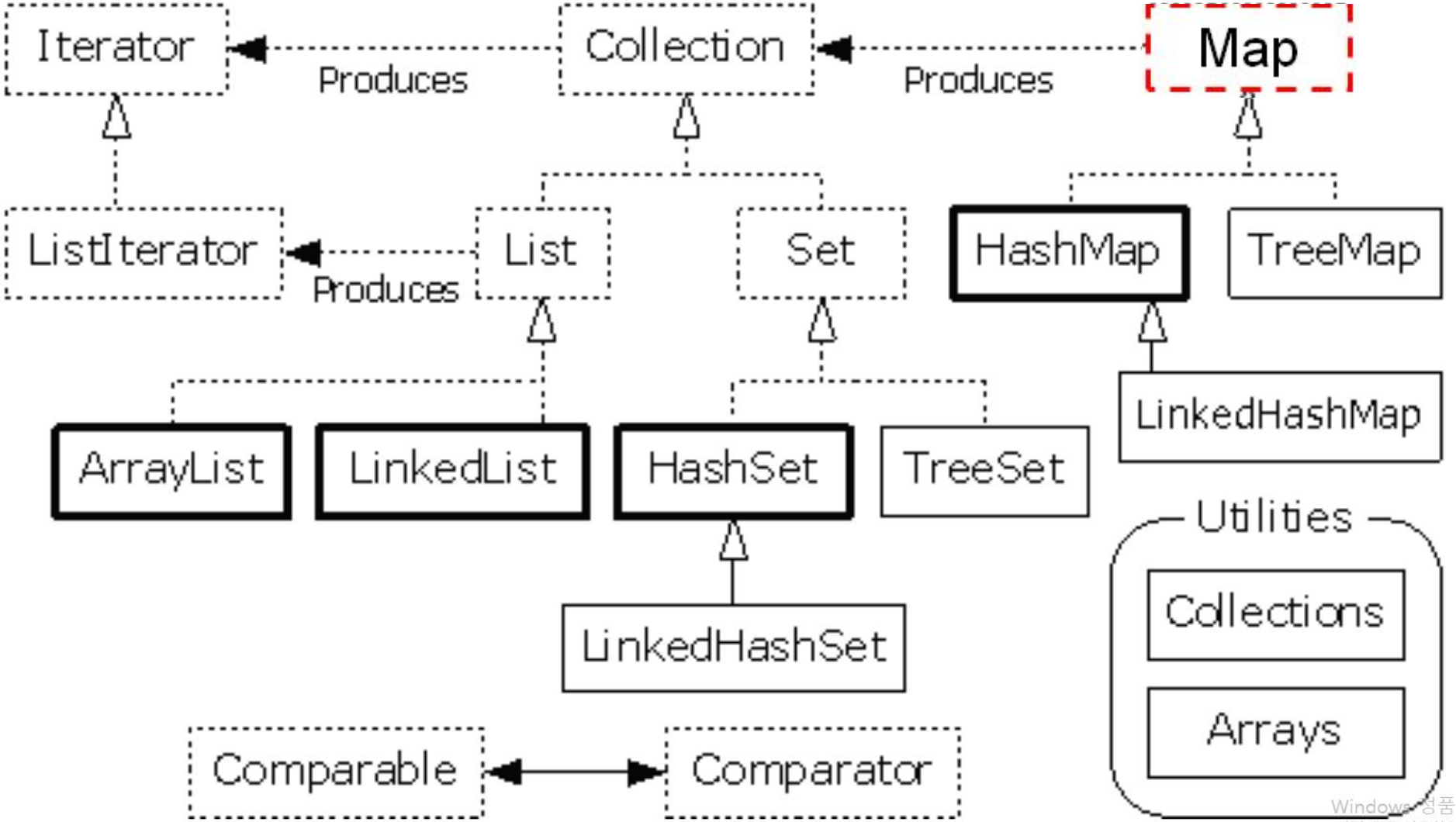
Efficient Looping with `Iterator`

```
while(it.hasNext()) {
  it.next()
}
```

`iterator.next()` → `d`



Map Interface



Windows 정품
[설정]으로 이동하여

`Map` Interface

- It stores the mapping between keys and values.
- It provides a way to retrieve the corresponding value using a key.
- Keys are unique.
  - A key only appears once in the Map.
  - A key can map to only one value.
- Values do not have to be unique.
- **Takes two generic types for keys and values `Map<K, V>`.**

`Map` Interface Methods (1/2)

- `put(K key, V value)`
  - Associates the specified value with the specified key in this map.
- `V get(Object key)`
  - Returns the value to which the specified key is mapped, or null if there is no mapped value for the key.
- `boolean containsKey(Object key)`
  - Returns true if this map contains a mapping for the specified key.
- `boolean containsValue(Object value)`
  - Returns true if this map maps one or more keys to the

`Map` Interface Methods (2/2)

- `boolean isEmpty()`
  - Returns true if this map contains no key-value mappings.
- `Set<K> keySet()`
  - Returns a Set view of the keys contained in this map.
- `int size()`
  - Returns the number of key-value mappings in this map.

`HashMap` Class

- It uses the concept of hashing similar to `HashSet`.
- `HashMap` is the most efficient `Map` implementation in terms of retrieving data.
- `HashMap` makes no guarantees as to the order of the map. In particular, it does not guarantee that the order will remain constant over time.

HashMap Example

```
static HashMap<String, Integer> countFrequency(
  String[] names) {
    HashMap<String, Integer> frequency =
                 new HashMap<String, Integer>();
    for(String name : names) {
        Integer currentCount = frequency.get(name);
        if(currentCount == null) {
            currentCount = 0; // auto-boxing
        }
        frequency.put(name, ++currentCount);
    }
    return frequency;
}
```

HashMap Example (cont'd)

```
public static void main(String[] args) {
    System.out.println(
        countFrequency(new String[]{
            "Momo", "Momo", "Koko", "Noa", "Momo", "Koko"
        }).toString());
}
```

```
{Momo=3, Noa=1, Koko=2}
```

Output

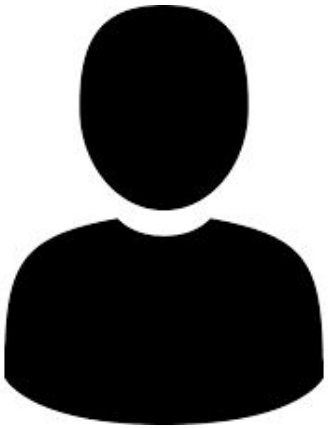
`HashMap` has a good `toString()` method!

`HashMap` doesn't guarantee order!

Objectives

- To properly use Java Collections

Problem 1 - Simple Diary Application (1)



Write/List/Read/Delete/Search

DiaryUI

Output

Implement here!

Diary

Write/List/Read/Delete /Search Entries

DiaryEntry

id, created time, title, content

+ Other classes if you need

Descriptions of Diary Application

- Implement a diary application using **collections**.
- A user should be able to **write/list/read/delete/search diary entries**.
- DiaryUI class is <u>already implemented</u>. It contains methods to get user commands/inputs, print messages, and handle some errors.
- Implement **createEntry, listEntries, readEntry, deleteEntry and searchEntry** methods of Diary class.
- You don't have to care about exceptions not shown in this slide.

Commands of Diary Application

- When you run the skeleton, *DiaryUI* class prints out as shown below.
- There are five commands what you have to implement; **create, list, read, delete, search, sorted list**
- If a command is entered following the given format, the corresponding result must be obtained.

```
Type a command
   create: Create a diary entry
   list <condition1(optional)> <condition2(optional)>
                                        : List diary entries
   read <id>: Read a diary entry with <id>
   delete <id>: Delete a diary entry with <id>
   search <keyword>: List diary entries whose titles or
   contents contain <keyword>
Command:
```

5 mins

Create List & HashMap

- A created diary entry should be stored in a **List**.
- Also there should be a **hashmap** to be used for search command and the created diary entry also has to be stored in the hashmap.

- **HINT**
  - **List:** LinkedList
  - **HashMap:** Key of map is the unique ID, Value is title and content of the diary entry.

10 mins

Command 1 - Create Entries

- Create a diary entry with **title** and **content**.
- Each entry should have its own **unique id** when created and the created time.
- A created diary entry should be stored in a **List**.
- Assume that title input contains **only** alphanumeric characters and spaces(‘ ’).

```
Command: create
    title: First Entry
    content: Dear Diary, Life is beautiful.
    The entry is saved.
```

Use methods in class DiaryUI

5 mins

Command 2 - Basic List Entries

- Print the list of the diary entries(id, created time, title) you created before.
- The listed entries should be sorted in **created time**, by **ascending** order. Print nothing if the list is empty.

```
Type a command
    (...)
Command: list
    id: 1, created at: 2020/10/21 11:47:28, title: First Entry
    id: 2, created at: 2020/10/21 11:48:30, title: Self Reflection
    id: 3, created at: 2020/10/21 11:55:30, title: Third Entry
```

Tab

Command 2 - List Entries

- List command that we implemented in the prior stage is the simplest one which prints the DiaryEntry in ascending order for the ID.

```
Type a command
    (...)
Command: list
    id: 1, created at: 2020/10/21 11:47:28, title: First Entry
    id: 2, created at: 2020/10/21 11:48:30, title: Self Reflection
    id: 3, created at: 2020/10/21 11:55:30, title: Third Entry
```

- Now, let's implement additional List command which receives extra criteria for sorting as an argument.

Command 2 - List Entries (Single Condition)

- **list title**: print the list of the diary entries in **ascending** order for the **title**
- Print the list of the diary entries(id, created time, title) you created before.
- Use `Comparator` to compare and determine which entry's title comes first.

Command 2 - List Entries (Single Condition)

```
Type a command
    (...)
Command: list
    id: 1, created at: 2020/10/21 11:47:28, title: First Entry
    id: 2, created at: 2020/10/21 11:48:30, title: Self Reflection
    id: 3, created at: 2020/10/21 11:55:30, title: Third Entry
```

```
Type a command
    (...)
Command: list title
    List of entries sorted by the title.
    id: 1, created at: 2020/10/21 11:47:28, title: First Entry
    id: 2, created at: 2020/10/21 11:48:30, title: Self Reflection
    id: 3, created at: 2020/10/21 11:55:30, title: Third Entry
```

Command 2 - List Entries (Single Condition)

```
Type a command
    (...)
Command: list
    id: 1, created at: 2020/10/21 11:47:28, title: Zimbabwe
    id: 2, created at: 2020/10/21 11:48:30, title: Apple store
    id: 3, created at: 2020/10/21 11:55:30, title: Love CP Lab
```

```
Type a command
    (...)
Command: list title
    List of entries sorted by the title.
    id: 2, created at: 2020/10/21 11:48:30, title: Apple store
    id: 3, created at: 2020/10/21 11:55:30, title: Love CP Lab
    id: 1, created at: 2020/10/21 11:47:28, title: Zimbabwe
```

Command 2 - List Entries (Multiple Conditions)

- **list title length**: print the list of the diary entries in **ascending** order for the **title**, and if the title is the same, then print in **descending** order for the **content length**
- Print the list of the diary entries(id, created time, title, content length) you created before.
- Use Comparator to compare and determine which entry's title comes first.
- Determine the content length by the number of letters in the content.
- Assume that there is no case where two entries have the same title and length.

Command 2 - List Entries (Multiple Conditions)

```
Type a command
    (...)
Command: list
    id: 1, created at: 2020/10/21 11:47:28, title: First Entry
    id: 2, created at: 2020/10/21 11:48:30, title: First Entry
    id: 3, created at: 2020/10/21 11:55:30, title: First Entry
```

```
Type a command
    (...)
Command: list title length
    id: 2, created at: 2020/10/21 11:48:30, title: First Entry, length: 30
    id: 1, created at: 2020/10/21 11:47:28, title: First Entry, length: 20
    id: 3, created at: 2020/10/21 11:55:30, title: First Entry, length: 10
```

5 mins

Command 3 - Read Entry

- **read <id>** : read a diary entry with <id>
- Print the diary entry(id, created time, title, content) selected by **the <id> input**.

```
Type a command
    (...)
Command: read 1
    id: 1
    created at: 2020/10/21 11:47:28
    title: First Entry
    content: Dear Diary, Life is beautiful.
```

Use method in class Diary

5 mins

Command 3 - Read Entry

- If there is no entry that has the input id, print an **error message.**

```
Type a command
    (...)
Command: read 5
    There is no entry with id 5.
```

5 mins

Command 4 - Delete Entries

- **delete <id>** : delete a diary entry with <id>

```
Type a command
    (...)
Command: delete 1
    Entry 1 is removed.
```

```
Type a command
    (...)
Command: list
    id: 2, created at: 2020/10/21 11:48:30, title: Self Reflection
    id: 3, created at: 2020/10/21 11:55:30, title: Third Entry
```

5 mins

Command 4 - Delete Entries

- If there is no entry that has the input id, print an **error message.**

```
Type a command
    (...)
Command: delete 5
    There is no entry with id 5.
```

```
Type a command
    (...)
Command: list
    id: 2, created at: 2020/10/21 11:48:30, title: Self Reflection
    id: 3, created at: 2020/10/21 11:55:30, title: Third Entry
```

10 mins

Command 5 - Search Entry

- **search <keyword>** : search the diary entry with <keyword>
- The user should be able to search entries which contain a given **keyword** exactly in their <u>titles or contents</u>. The search should be **case-sensitive**.
  - ex) title: First Entry

    keyword: First ⇒ First Entry (O)
    keyword: first ⇒ First Entry (X)
    keyword: entr ⇒ First <u>Entr</u>y (X)

- Use split("\\s") method to split keywords.
- Print the diary entries(id, created time, title, content) searched by the keyword. The entries don't need to be sorted. Just check whether all the results are printed. Print a **blank line** between the entries.
- If there is no entry that contains keyword in the title or content, print an **error message**.

10 mins

Command 5 - Search Entry

```
Type a command
   (…)
Command: search Entry
   id: 1
   created at: 2020/10/21 11:47:28
   title: First Entry
   content: I want to become a great engineer!

   id: 3
   created at: 2020/10/21 11:55:30
   title: Third Entry
   content: I want to become a great engineer!
```

10 mins

Command 5 - Search Entry

```
Type a command
   (…)
Command: search Third
   id: 3
   created at: 2020/10/21 11:55:30
   title: Third Entry
   content: I want to become a great engineer!
```

```
Type a command
   (…)
Command: search Thi
   There is no entry that contains “Thi”.
```

Submission

- Compress your final `src` directory into a zip file.
  - After unzipping, the ‘src’ directory must appear.
- Rename your zip file as 20XX-XXXXX_{name}.zip - for example, 2021-12345_SeoHyuna.zip
- Upload it to eTL - Lab 7 assignment.

Thank You!!!

Q&A Time!