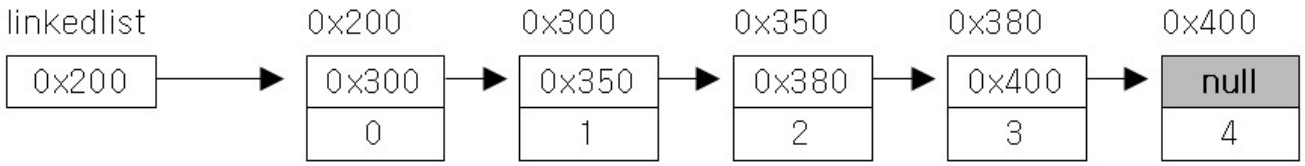
CSE3040 Java Language

Lecture 17: Collection Framework (2)

Dept. of Computer Engineering,
Sogang University

3. LinkedList

- LinkedList
  - A structure where data is "linked" together.

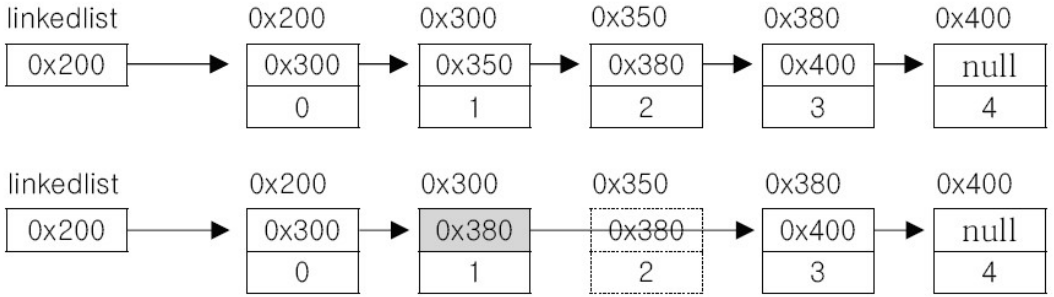


  - An element in a linked list has data, and a link to the next element.

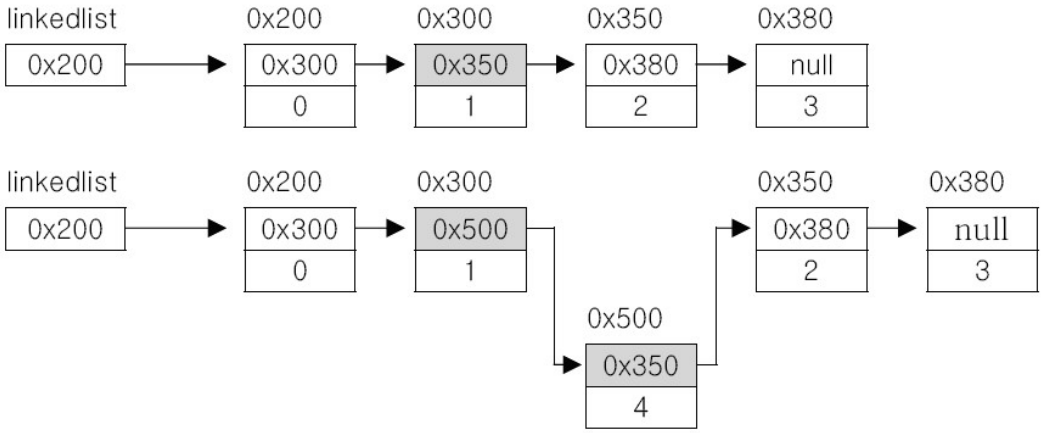
```
class Node {
    Node next;    // a link to the next element
    Object obj;   // data
}
```

Addition and deletion with linked lists

- Deleting an element from the linked list
  - Can be done with updating a reference.

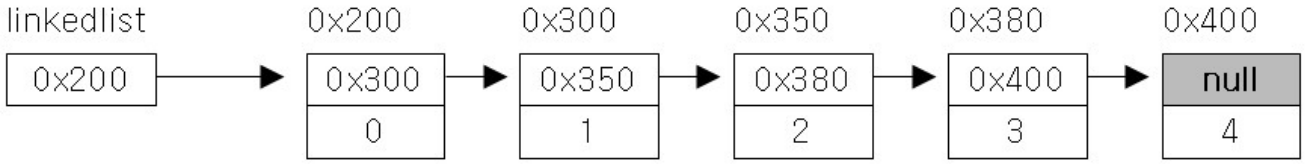


- Adding an element to the linked list
  - Can be done with an instance generation and two reference updates.

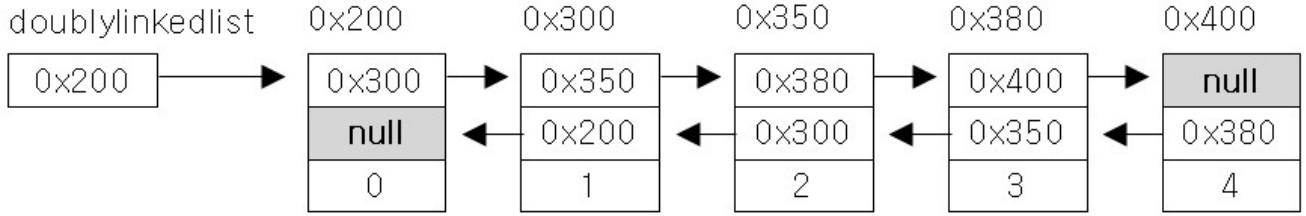


Types of linked lists

- (Basic) linked list

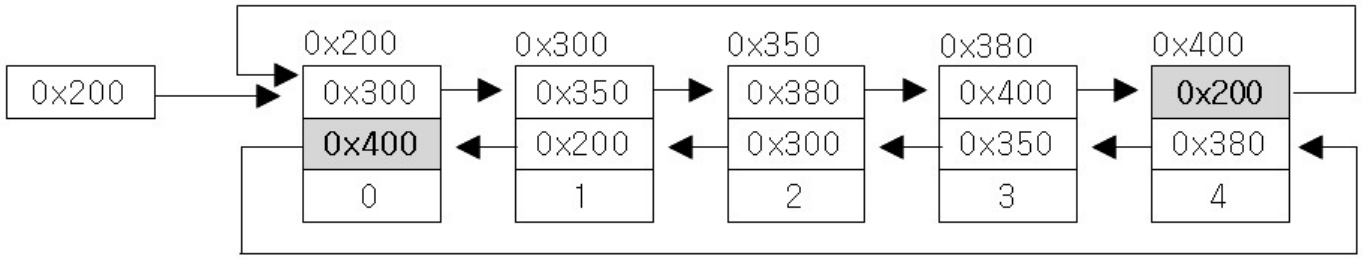


- Doubly linked list: better accessibility




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

- Doubly circular linked list



Example 1: ArrayList vs. LinkedList

- A program to compare performance of an ArrayList and a LinkedList

```
public class Lecture {
    public static long add1(List<String> list) {
        long start = System.currentTimeMillis();
        for(int i=0; i<1000000; i++) list.add(i+"");
        long end = System.currentTimeMillis();
        return end - start;
    }
    public static long add2(List<String> list) {
        long start = System.currentTimeMillis();
        for(int i=0; i<10000; i++) list.add(500, "X");
        long end = System.currentTimeMillis();
        return end - start;
    }
    public static long remove1(List<String> list) {
        long start = System.currentTimeMillis();
        for(int i=list.size()-1; i>=0; i--) list.remove(i);
        long end = System.currentTimeMillis();
        return end - start;
    }
    public static long remove2(List<String> list) {
        long start = System.currentTimeMillis();
        for(int i=0; i<10000; i++) list.remove(i);
        long end = System.currentTimeMillis();
        return end - start;
    }
```

Example 1: ArrayList vs. LinkedList

- A program to compare performance of an ArrayList and a LinkedList (cont.)
  - Which one is better for sequential and non-sequential operations? Why?

```
    public static void main(String args[]) {
        ArrayList<String> al = new ArrayList<>(2000000);
        LinkedList<String> ll = new LinkedList<>();

        System.out.println("=== sequential add ===");
        System.out.println("ArraList  : " + add1(al));
        System.out.println("LinkedList: " + add1(ll));
        System.out.println();
        System.out.println("=== non-sequential add ===");
        System.out.println("ArraList  : " + add2(al));
        System.out.println("LinkedList: " + add2(ll));
        System.out.println();
        System.out.println("=== non-sequential delete ===");
        System.out.println("ArraList  : " + remove2(al));
        System.out.println("LinkedList: " + remove2(ll));
        System.out.println();
        System.out.println("=== sequential delete ===");
        System.out.println("ArraList  : " + remove1(al));
        System.out.println("LinkedList: " + remove1(ll));
    }
}
```

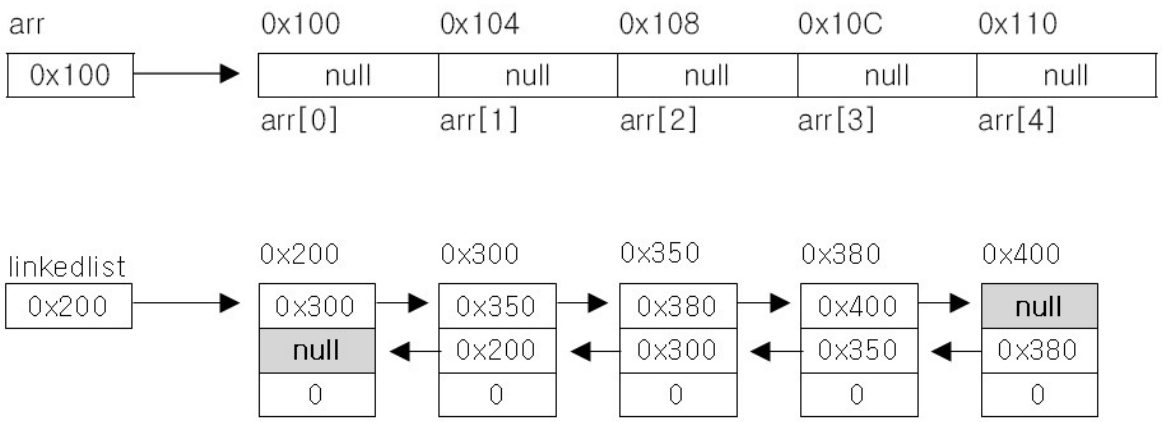
Example 2: ArrayList vs. LinkedList

- A program to compare performance of an ArrayList and a LinkedList
  - Which one is better for data access? Why?

```
public class Lecture {
    public static void main(String args[]) {
        ArrayList<String> al = new ArrayList<>(1000000);
        LinkedList<String> ll = new LinkedList<>();
        add(al);
        add(ll);
        System.out.println("=== access time ===");
        System.out.println("ArrayList : " + access(al));
        System.out.println("LinkedList: " + access(ll));
    }
    public static void add(List<String> list) {
        for(int i=0; i<100000; i++) list.add(i+"");
    }
    public static long access(List<String> list) {
        long start = System.currentTimeMillis();
        for(int i=0; i<10000; i++) list.get(i);
        long end = System.currentTimeMillis();
        return end - start;
    }
}
```

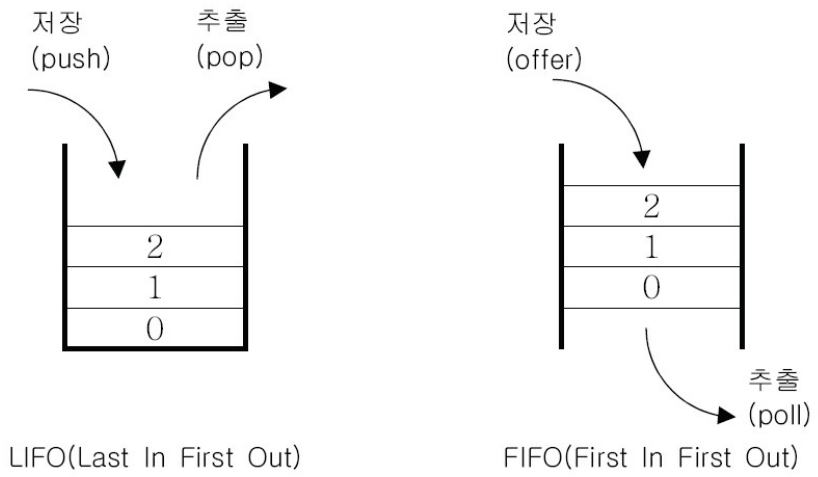
Example 2: ArrayList vs. LinkedList

- Accessing data
  - ArrayList: memory can be calculated from the starting address and the index
  - LinkedList: must sequentially follow the links.



4. Stack and Queue

- Stack: a data structure which implements LIFO (Last-In First-Out) policy.
  - Usage: arithmetic expression evaluation, undo/redo, back/forward (web browser)
  - Defined in Java as a class (class Stack<E>)
- Queue: a data structure which implements FIFO (First-In First-Out) policy.
  - Usage: printer queue, CPU scheduling, I/O buffers
  - Defined in Java as an interface (interface Queue<E>)
  - Class LinkedList implements Queue



Methods

- Methods defined in class Stack<E>

Method	Description
boolean empty()	Tests if this stack is empty.
E peek()	Looks at the object at the top of this stack without removing it from the stack.
E pop()	Removes the object at the top of this stack and returns that object as the value of this function.
E push(E item)	Pushes an item onto the top of this stack.
int search(Object o)	Returns the 1-based position where an object is on this stack.

- Methods defined in interface Queue<E>

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.

Example: Stack vs. Queue

- Shows how elements are removed from the data structures.

```
public class Lecture {
    public static void main(String[] args) {
        Stack<String> st = new Stack<String>();
        Queue<String> q = new LinkedList<String>();

        st.push("0");
        st.push("1");
        st.push("2");

        q.offer("0");
        q.offer("1");
        q.offer("2");

        System.out.println("=== Stack ===");
        while(!st.empty()) System.out.println(st.pop());

        System.out.println("=== Queue ===");
        while(!q.isEmpty()) System.out.println(q.poll());
    }
}
```

Example: Implementing Stack

- Here we implement MyStack, a class that implements a stack.
- We implement the following methods
  - push, pop, peek, empty, search
- MyStack.java

```
import java.util.*;

public class MyStack<E> extends Vector<E> {
    public E push(E item) {
        addElement(item);   // defined in class Vector
        return item;
    }

    public E pop() {
        E e = peek();
        removeElementAt(size() - 1);
        return e;
    }

    public boolean empty() {
        return size() == 0;
    }
```

Example: Implementing Stack

- MyStack.java (cont.)

```
    public E peek() {
        int len = size();
        if( len == 0 ) throw new EmptyStackException();
        return elementAt(len - 1);
    }

    public int search(Object o) {
        int i = lastIndexOf(o);
        if(i >= 0) { return size() - i; }
        return -1;
    }
}
```

Example: Using Stack

- The program uses Stack to emulate back/forward buttons in a web browser.

```
public class Lecture {
    public static Stack<String> back = new Stack<>();
    public static Stack<String> forward = new Stack<>();

    public static void main(String[] args) {
        goURL("1. Google");
        goURL("2. Facebook");
        goURL("3. Naver");
        goURL("4. Daum");
        printStatus();

        goBack();
        System.out.println("=== After pushing 'back' ===");
        printStatus();
        goBack();
        System.out.println("=== After pushing 'back' ===");
        printStatus();
        goForward();
        System.out.println("=== After pushing 'forward' ===");
        printStatus();
        goURL("www.sogang.ac.kr");
        System.out.println("=== After moving to a new URL ===");
        printStatus();
    }
```

Example: Using Stack

- The program uses Stack to emulate back/forward buttons in a web browser. (cont.)

```
    public static void printStatus() {
        System.out.println("back:"+back);
        System.out.println("forward:"+forward);
        System.out.println("current page is " + back.peek() + ".");
        System.out.println();
    }

    public static void goURL(String url) {
        back.push(url);
        if(!forward.empty()) forward.clear();
    }

    public static void goForward() {
        if(!forward.empty()) back.push(forward.pop());
    }

    public static void goBack() {
        if(!back.empty()) forward.push(back.pop());
    }
}
```

Example: Using Queue

- This program uses a queue to implement command history.

```
public class Lecture {
    static Queue<String> q = new LinkedList<String>();
    static final int MAX_SIZE = 5;

    public static void main(String[] args) {
        System.out.println("input 'help' to display help message.");
        while(true) {
            System.out.print(">>");
            try {
                Scanner s = new Scanner(System.in);
                String input = s.nextLine().trim();
                if("".equals(input)) continue;
                if(input.equalsIgnoreCase("q")) {
                    System.exit(0);
                } else if(input.equalsIgnoreCase("help")) {
                    System.out.println(" help – displays help message.");
                    System.out.println(" q or Q – exit the program.");
                    System.out.println(" history – shows recent commands.");
                }
```

Example: Using Queue

- This program uses a queue to implement command history. (cont.)
  - Can you properly address the warning?

```
                else if(input.equalsIgnoreCase("history")) {
                    int i=0;
                    save(input);
                    LinkedList<String> tmp = (LinkedList<String>)q;
                    ListIterator<String> it = tmp.listIterator();
                    while(it.hasNext()) System.out.println(++i+"."+it.next());
                } else {
                    save(input);
                    System.out.println(input);
                }
            } catch(Exception e) { System.out.println("input error."); }
        }
    }
    public static void save(String input) {
        if(!"".equals(input)) q.offer(input);
        if(q.size() > MAX_SIZE) q.remove();
    }
}
```

Priority Queue

- Priority Queue
  - A type of queue where elements are removed based on the priority, not the order of input.
    - The element with the highest priority is removed first.
  - The elements are organized as a structure called "heap".
    - The heap is implemented using an array.

```
public class Lecture {
    public static void main(String[] args) {
        Queue<Integer> pq = new PriorityQueue<Integer>();
        pq.offer(3);
        pq.offer(1);
        pq.offer(5);
        pq.offer(2);
        pq.offer(4);
        System.out.println(pq);

        Integer i = null;
        while((i = pq.poll()) != null) System.out.println(i);
    }
}
```

Programming Lab #17

17-01. ArrayList vs. LinkedList: add & remove

- Write the following code and understand the results.

```
public class Ex17_01 {
    public static long add1(List<String> list) {
        long start = System.currentTimeMillis();
        for(int i=0; i<1000000; i++) list.add(i+"");
        long end = System.currentTimeMillis();
        return end - start;
    }
    public static long add2(List<String> list) {
        long start = System.currentTimeMillis();
        for(int i=0; i<10000; i++) list.add(500, "X");
        long end = System.currentTimeMillis();
        return end - start;
    }
    public static long remove1(List<String> list) {
        long start = System.currentTimeMillis();
        for(int i=list.size()-1; i>=0; i--) list.remove(i);
        long end = System.currentTimeMillis();
        return end - start;
    }
    public static long remove2(List<String> list) {
        long start = System.currentTimeMillis();
        for(int i=0; i<10000; i++) list.remove(i);
        long end = System.currentTimeMillis();
        return end - start;
    }
```

17-01. ArrayList vs. LinkedList: add & remove

- Why does the running time vary between ArrayList and LinkedList?

```
    public static void main(String args[]) {
        ArrayList<String> al = new ArrayList<>(2000000);
        LinkedList<String> ll = new LinkedList<>();

        System.out.println("=== sequential add ===");
        System.out.println("ArraList  : " + add1(al));
        System.out.println("LinkedList: " + add1(ll));
        System.out.println();
        System.out.println("=== non-sequential add ===");
        System.out.println("ArraList  : " + add2(al));
        System.out.println("LinkedList: " + add2(ll));
        System.out.println();
        System.out.println("=== non-sequential delete ===");
        System.out.println("ArraList  : " + remove2(al));
        System.out.println("LinkedList: " + remove2(ll));
        System.out.println();
        System.out.println("=== sequential delete ===");
        System.out.println("ArraList  : " + remove1(al));
        System.out.println("LinkedList: " + remove1(ll));
    }
}
```

17-02. ArrayList vs. LinkedList: data access

- Write the following code and understand the results.
- Why does the running time vary between ArrayList and LinkedList?

```
public class Ex17_02 {
    public static void main(String args[]) {
        ArrayList<String> al = new ArrayList<>(1000000);
        LinkedList<String> ll = new LinkedList<>();
        add(al);
        add(ll);
        System.out.println("=== access time ===");
        System.out.println("ArrayList : " + access(al));
        System.out.println("LinkedList: " + access(ll));
    }
    public static void add(List<String> list) {
        for(int i=0; i<100000; i++) list.add(i+"");
    }
    public static long access(List<String> list) {
        long start = System.currentTimeMillis();
        for(int i=0; i<10000; i++) list.get(i);
        long end = System.currentTimeMillis();
        return end - start;
    }
}
```

17-03. Stack vs. Queue

- Write the following code and understand the results.

```
public class Ex17_03 {
    public static void main(String[] args) {
        Stack<String> st = new Stack<String>();
        Queue<String> q = new LinkedList<String>();

        st.push("0");
        st.push("1");
        st.push("2");

        q.offer("0");
        q.offer("1");
        q.offer("2");

        System.out.println("=== Stack ===");
        while(!st.empty()) System.out.println(st.pop());

        System.out.println("=== Queue ===");
        while(!q.isEmpty()) System.out.println(q.poll());
    }
}
```

17-04. Using Stack

- Write the following code and understand the results.

```
public class Ex17_04 {
    public static Stack<String> back = new Stack<>();
    public static Stack<String> forward = new Stack<>();

    public static void main(String[] args) {
        goURL("1. Google");
        goURL("2. Facebook");
        goURL("3. Naver");
        goURL("4. Daum");
        printStatus();

        goBack();
        System.out.println("=== After pushing 'back' ===");
        printStatus();
        goBack();
        System.out.println("=== After pushing 'back' ===");
        printStatus();
        goForward();
        System.out.println("=== After pushing 'forward' ===");
        printStatus();
        goURL("www.sogang.ac.kr");
        System.out.println("=== After moving to a new URL ===");
        printStatus();
    }
```

17-04. Using Stack

```
public static void printStatus() {
        System.out.println("back:"+back);
        System.out.println("forward:"+forward);
        System.out.println("current page is " + back.peek() + ".");
        System.out.println();
    }

    public static void goURL(String url) {
        back.push(url);
        if(!forward.empty()) forward.clear();
    }

    public static void goForward() {
        if(!forward.empty()) back.push(forward.pop());
    }

    public static void goBack() {
        if(!back.empty()) forward.push(back.pop());
    }
}
```

17-05. Using Queue

- Write the following code and understand the results.
- Modify the code so that the warning disappears.

```
public class Ex17_05 {
    static Queue<String> q = new LinkedList<String>();
    static final int MAX_SIZE = 5;

    public static void main(String[] args) {
        System.out.println("input 'help' to display help message.");
        while(true) {
            System.out.print(">>");
            try {
                Scanner s = new Scanner(System.in);
                String input = s.nextLine().trim();
                if("".equals(input)) continue;
                if(input.equalsIgnoreCase("q")) {
                    System.exit(0);
                } else if(input.equalsIgnoreCase("help")) {
                    System.out.println(" help – displays help message.");
                    System.out.println(" q or Q – exit the program.");
                    System.out.println(" history – shows recent commands.");
                }
```

17-05. Using Queue

```
                else if(input.equalsIgnoreCase("history")) {
                    int i=0;
                    save(input);
                    LinkedList<String> tmp = (LinkedList<String>)q;
                    ListIterator<String> it = tmp.listIterator();
                    while(it.hasNext()) System.out.println(++i+"."+it.next());
                } else {
                    save(input);
                    System.out.println(input);
                }
            } catch(Exception e) { System.out.println("input error."); }
        }
    }
    public static void save(String input) {
        if(!"".equals(input)) q.offer(input);
        if(q.size() > MAX_SIZE) q.remove();
    }
}
```

17-06. Priority Queue

- Write the following code and understand how a priority queue works.

```
public class Ex17_06 {
    public static void main(String[] args) {
        Queue<Integer> pq = new PriorityQueue<Integer>();
        pq.offer(3);
        pq.offer(1);
        pq.offer(5);
        pq.offer(2);
        pq.offer(4);
        System.out.println(pq);

        Integer i = null;
        while((i = pq.poll()) != null) System.out.println(i);
    }
}
```

End of Class

Instructor office: AS818A

Email: jso1@sogang.ac.kr