

Iterator Pattern

Prof. Jonathan Lee (李允中)
Department of CSIE
National Taiwan University

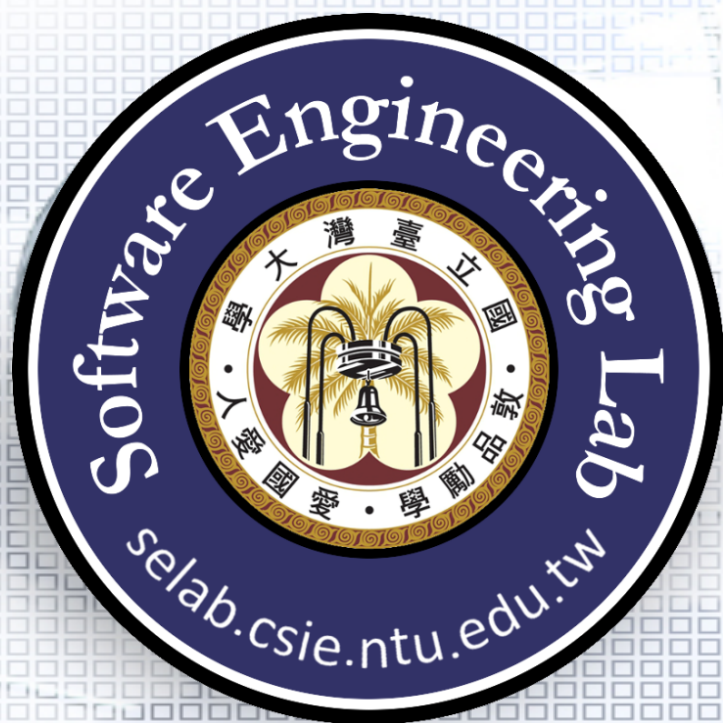


Design Aspect of Iterator

How an aggregate's
elements are accessed,
traversed



- 3



Print Out Items in Different Data Structures (Iterator)

Prof. Jonathan Lee (李允中)

Department of Computer Science and
Information Engineering
National Taiwan University



Requirements Statements

- ❑ A List data structure is implemented with a String array which can contain a series of String objects.
- ❑ We can access List by calling the `get()` method with an index, and know how many Strings inside the List with a public attribute: `length`.
- ❑ Furthermore, another data structure called SkipList which consists of a series of SkipNodes.
- ❑ Each SkipNode can be accessed by invoking the `getNode()` method in SkipList with an index. And we have the idea about the size of SkipList with its `size()` method.
- ❑ Now we have to traverse both List and SkipList to print out those object items in the two different data structures.



Requirements Statement₁

- A List data structure is implemented with a String array which can contain a series of String objects.

List
-data: String[*]



Requirements Statement₂

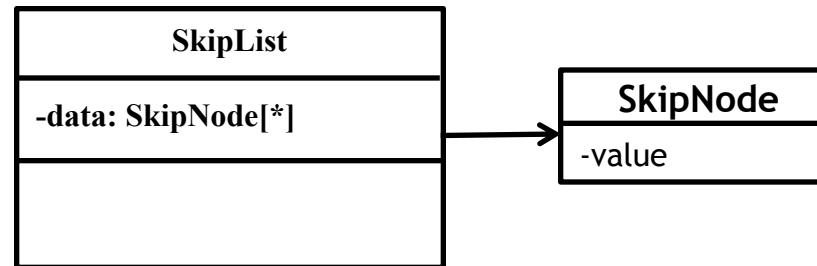
- We can access List by calling the `get()` method with an index, and know how many Strings inside the List with a public attribute: `length`.

List
-data: String[*] +length
+get(index)



Requirements Statement₃

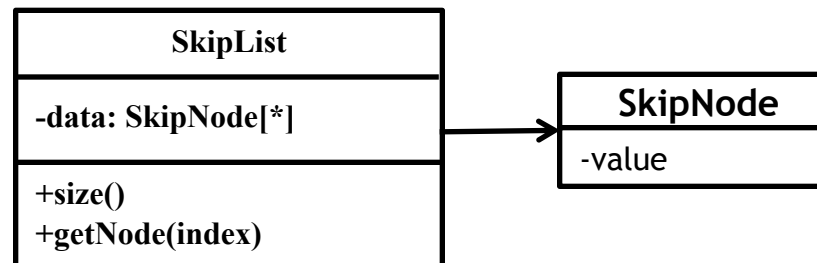
- Furthermore, another data structure called SkipList which consists of a series of SkipNodes.





Requirements Statement₄

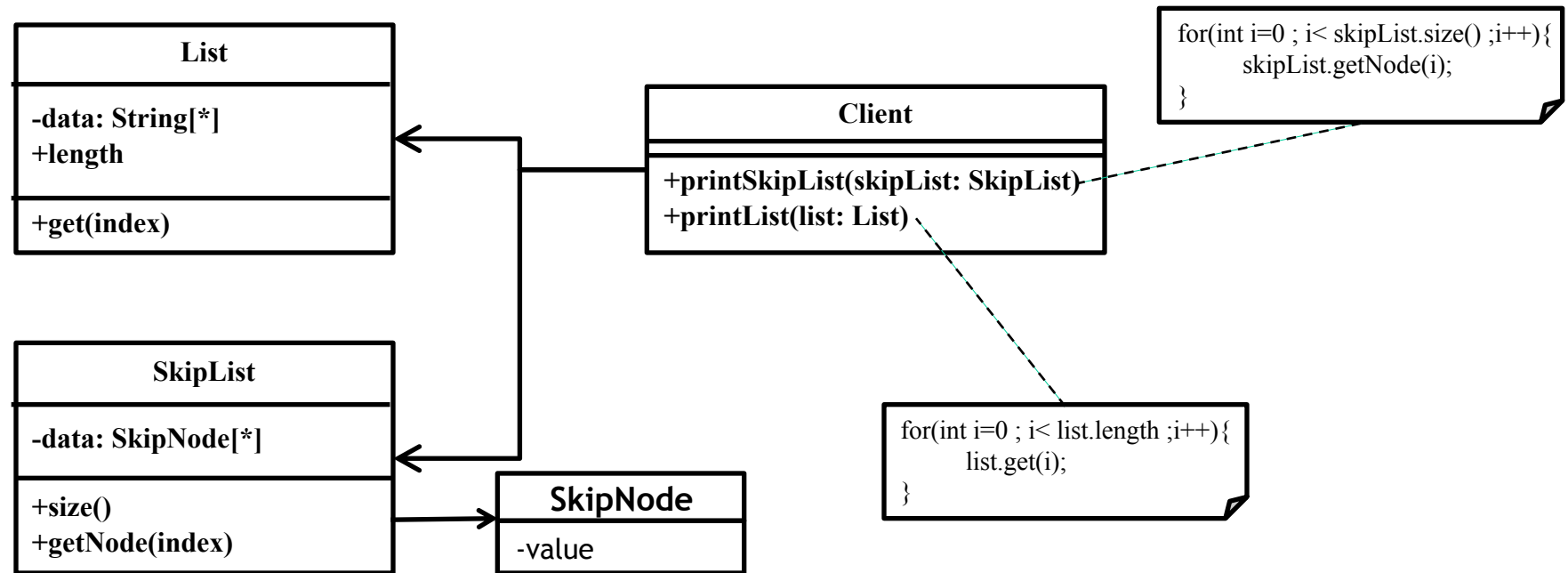
- ❑ Each SkipNode can be accessed by invoking the getNode() method in SkipList with an index. And we have the idea about the size of SkipList with its size() method.





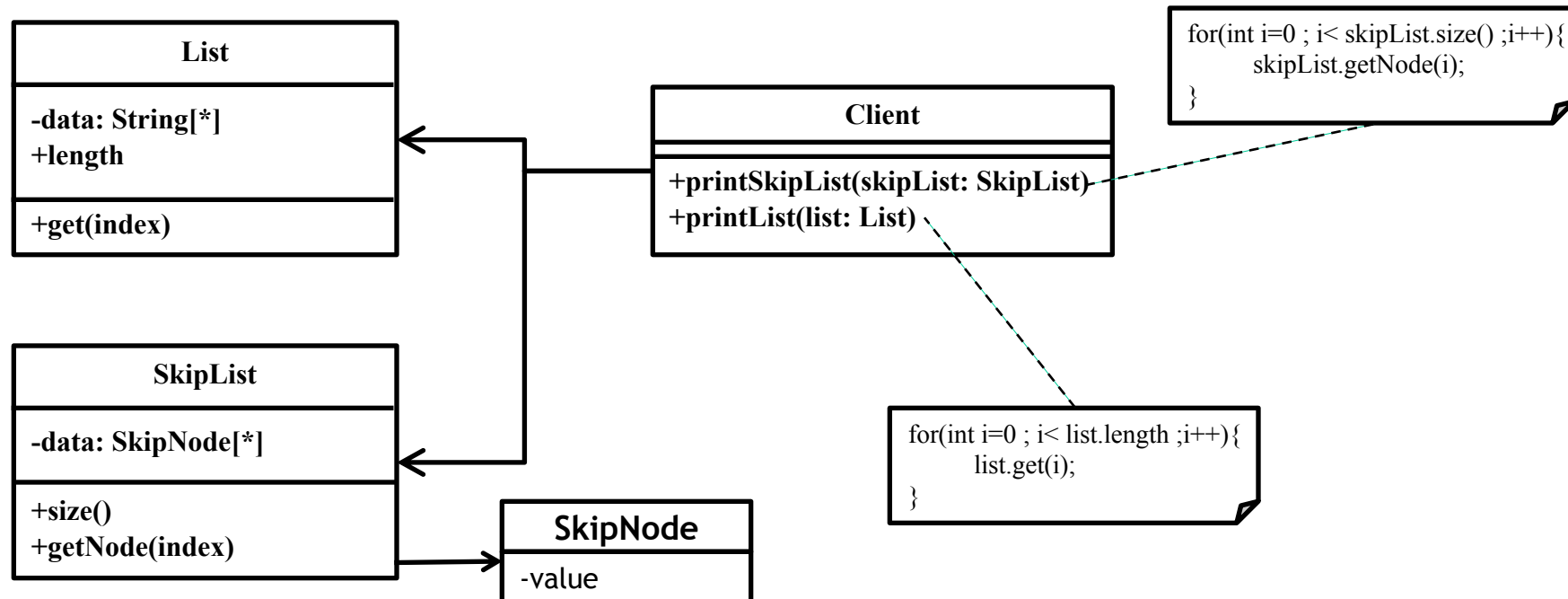
Requirements Statement₅

- Now we have to traverse both List and SkipList to print out those object items in the two different data structures.



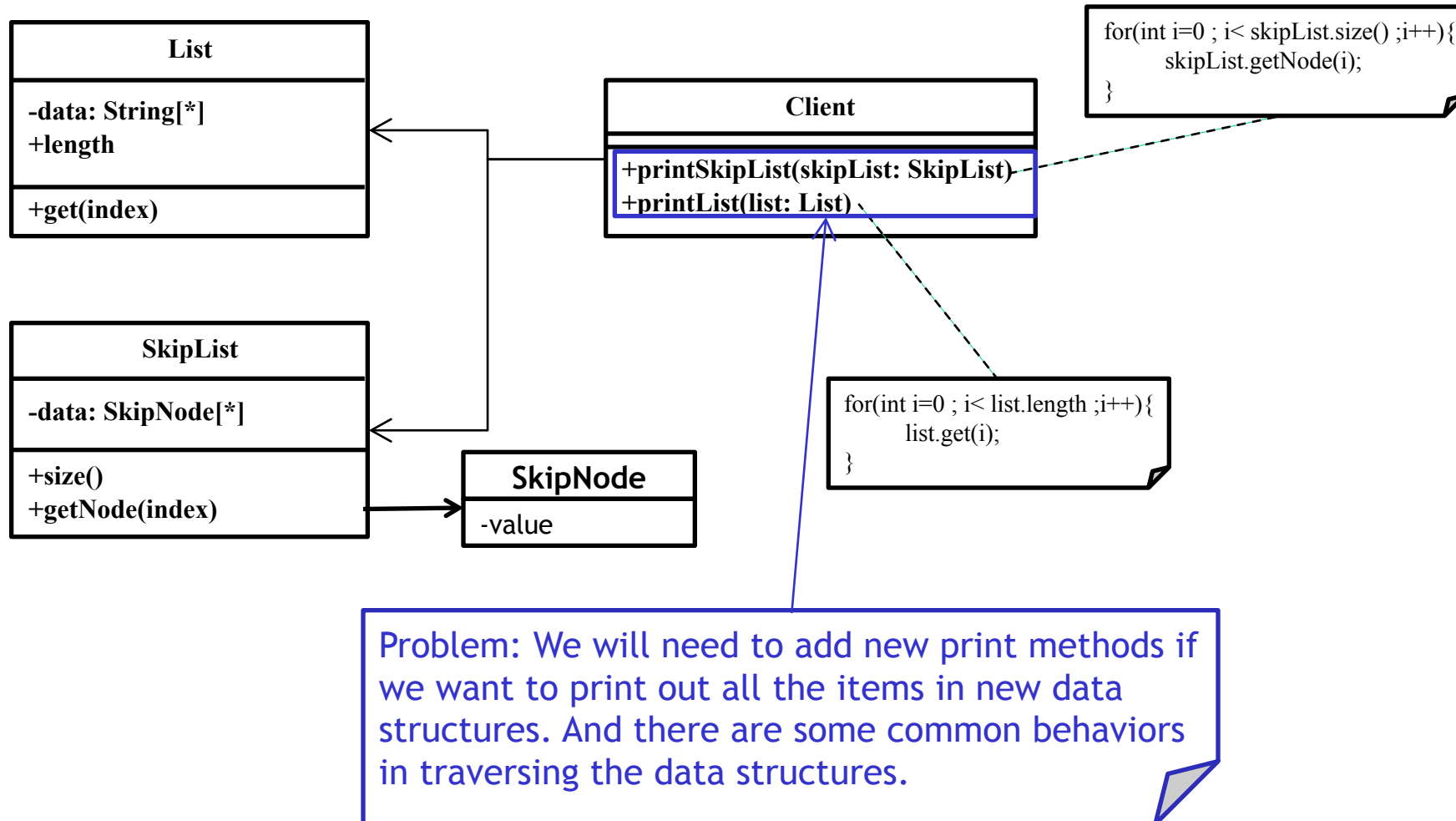


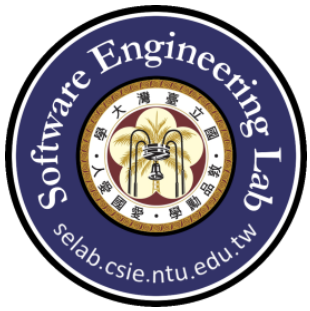
Initial Design



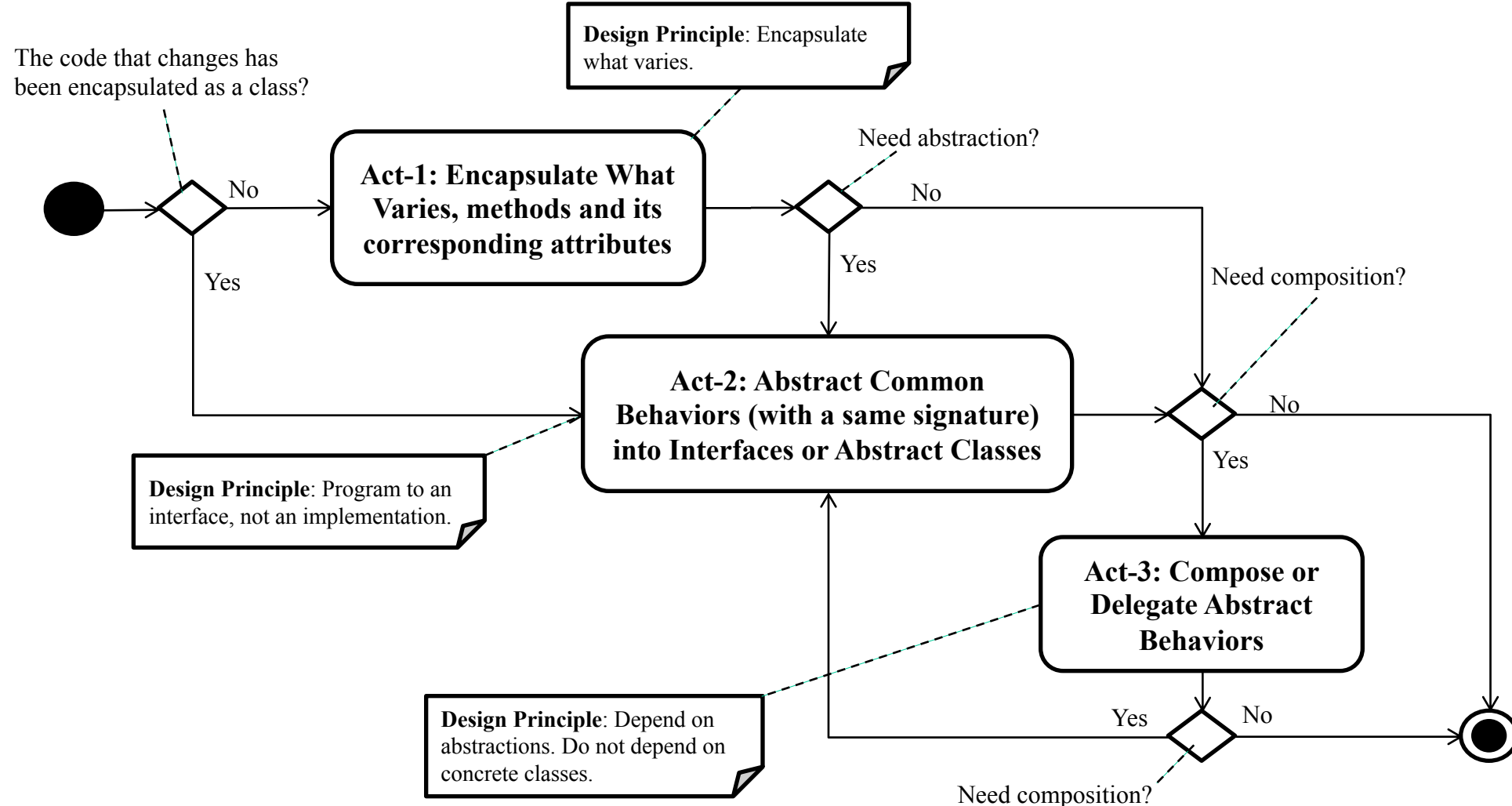


Problems with Initial Design



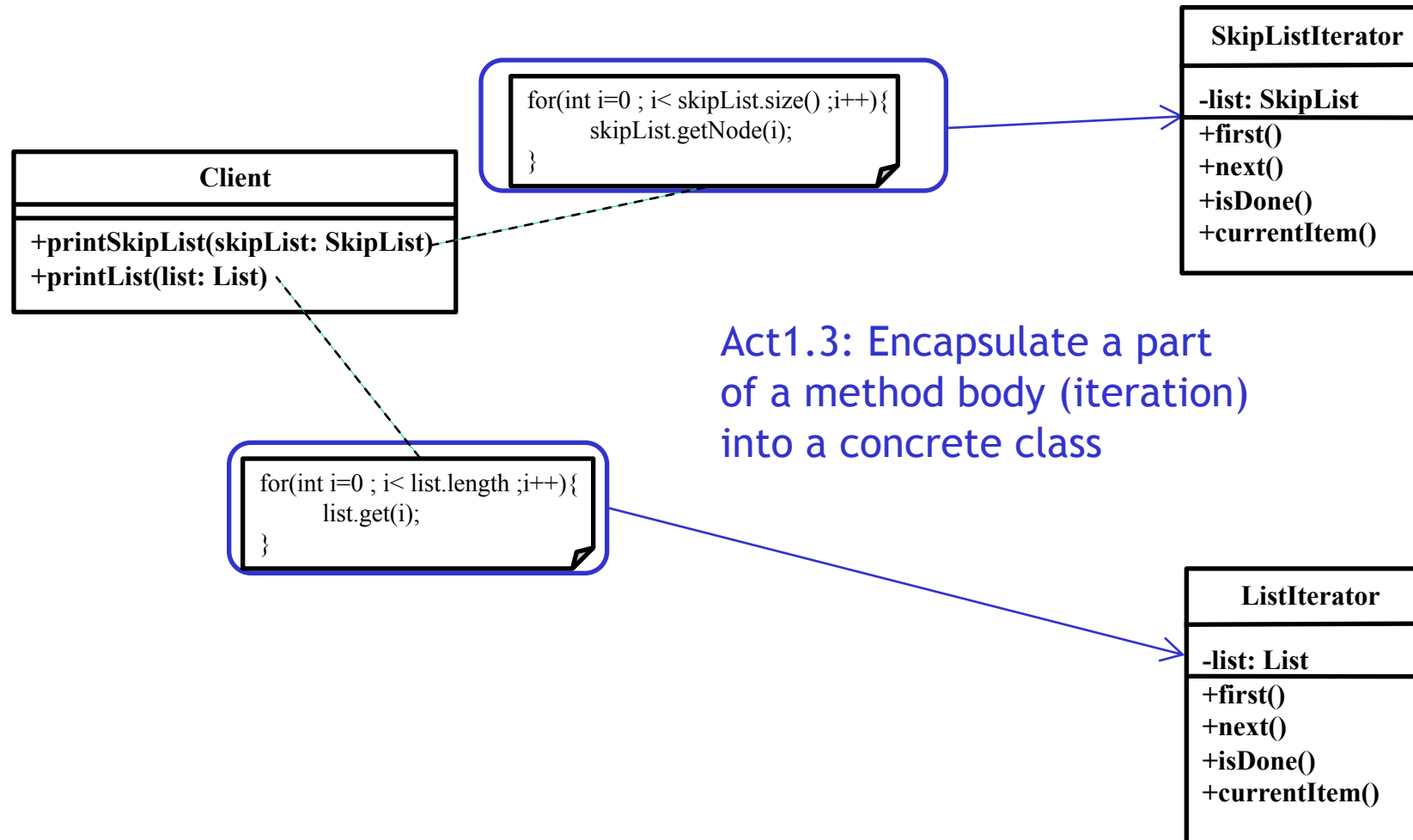


Design Process for Change



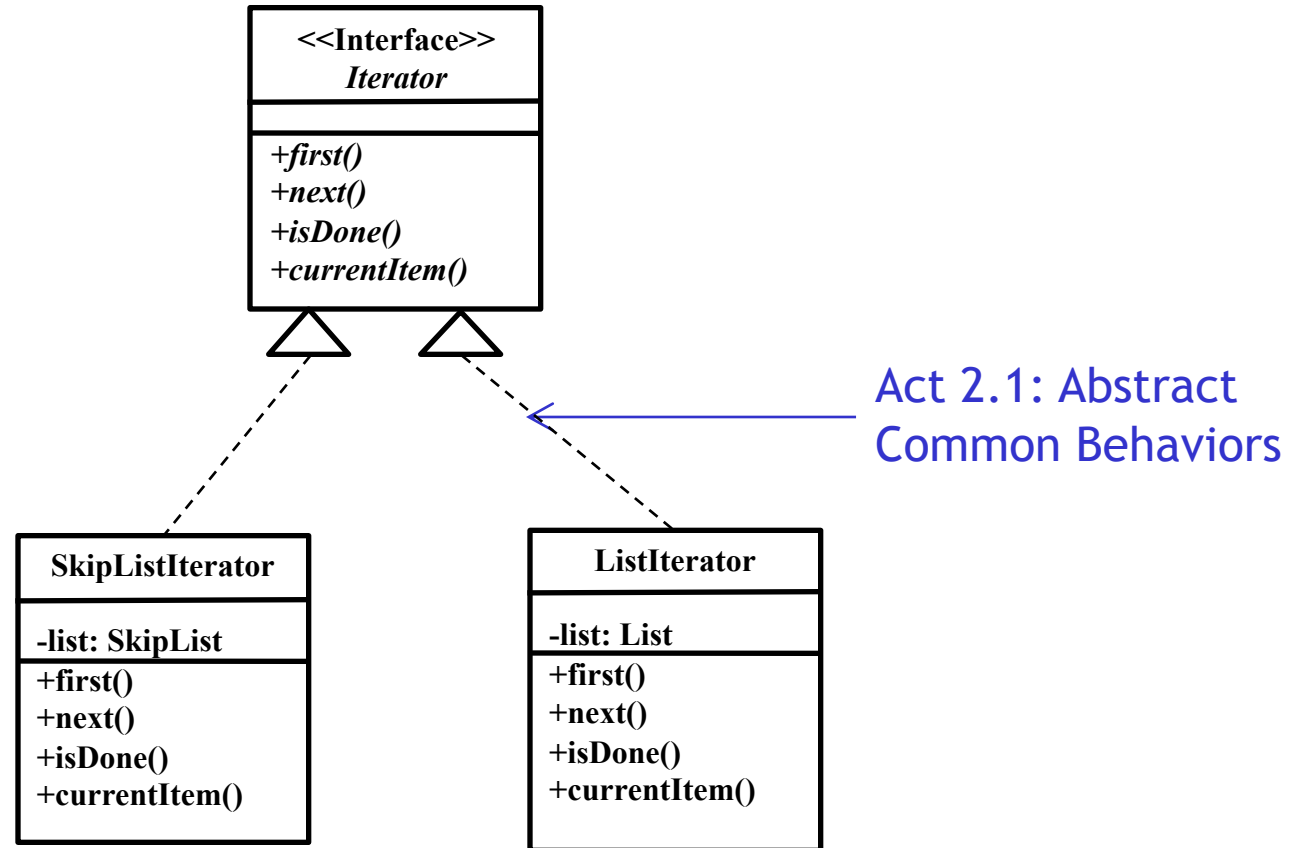


Act-1: Encapsulate What Varies



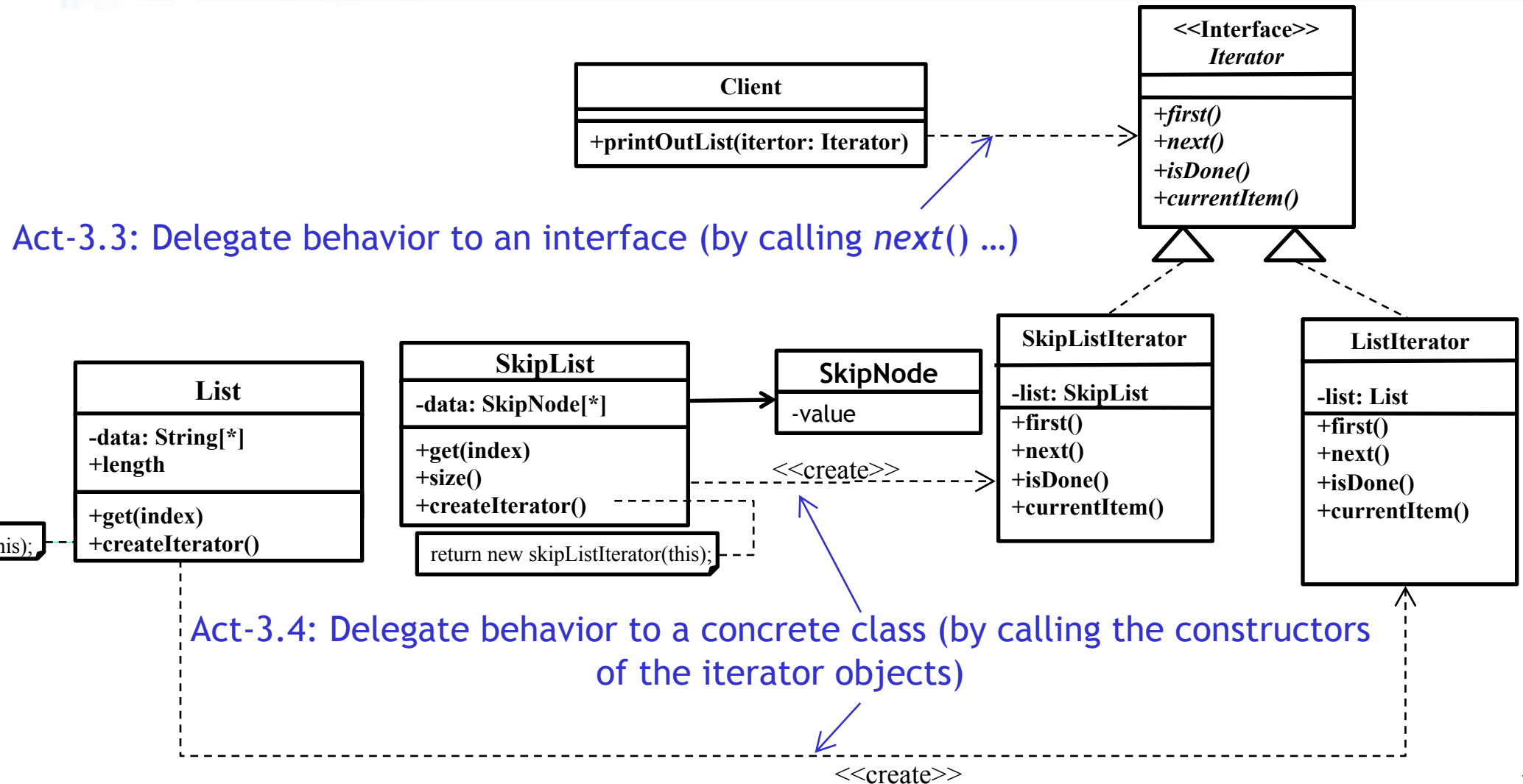


Act-2: Abstract Common Behaviors



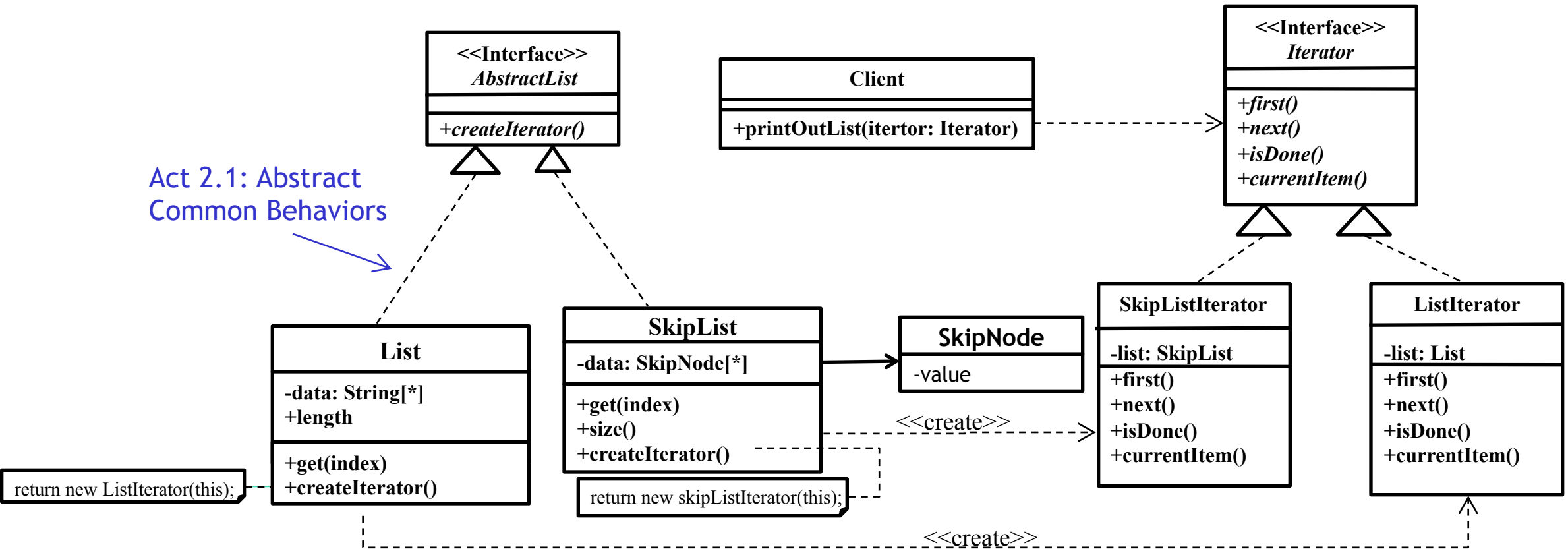


Act-3: Compose Abstract Interfaces/Abstract Classes or Delegate Behaviors



Act-2: Abstract Common Behaviors

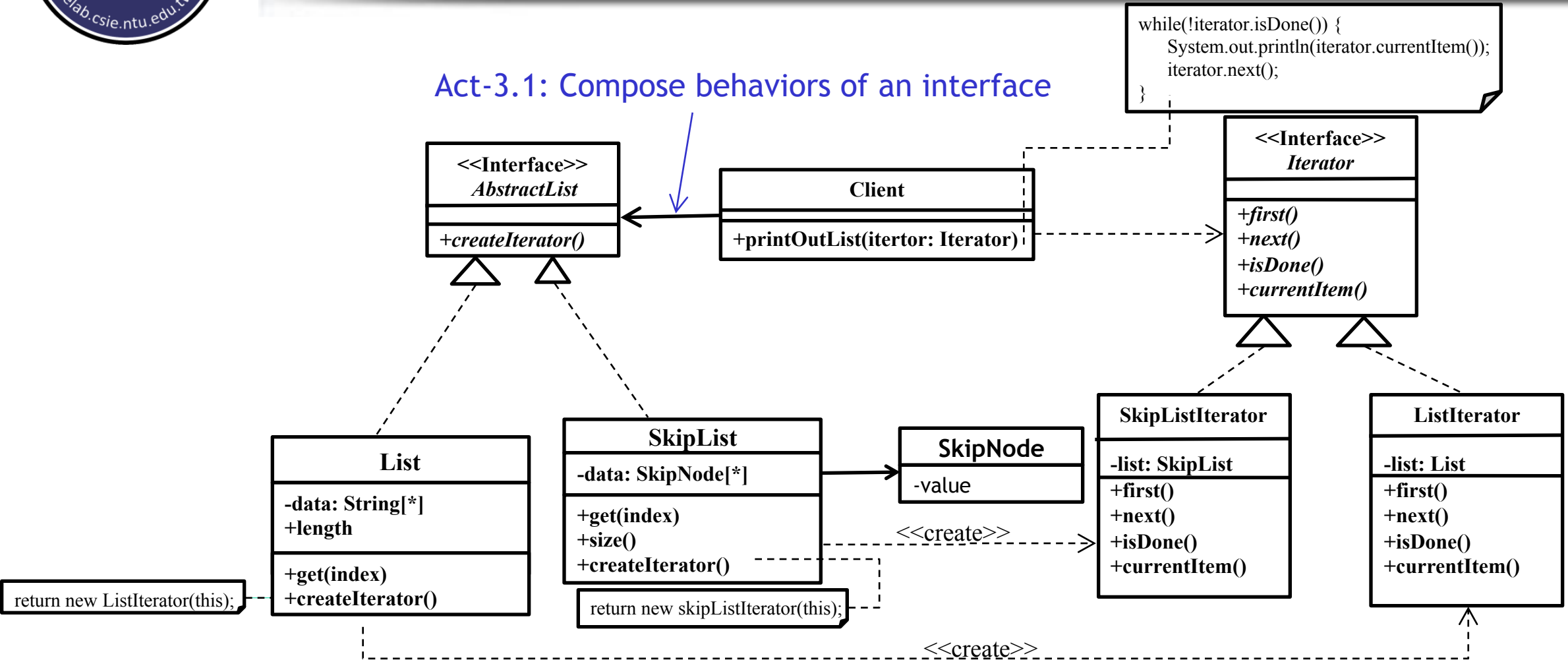
Act 2.1: Abstract
Common Behaviors



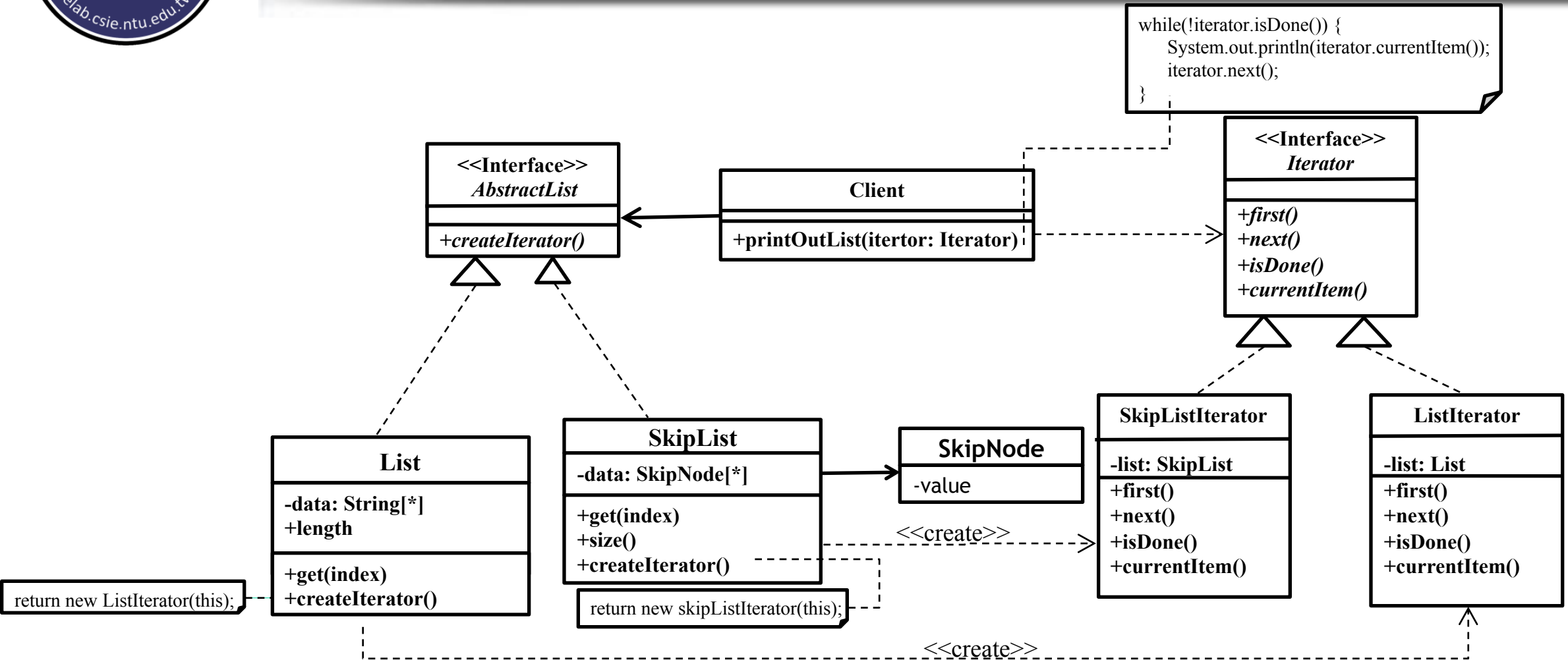


Act-3: Compose Abstract Interfaces/Abstract Classes or Delegate Behaviors

Act-3.1: Compose behaviors of an interface



Refactored Design after Design Process





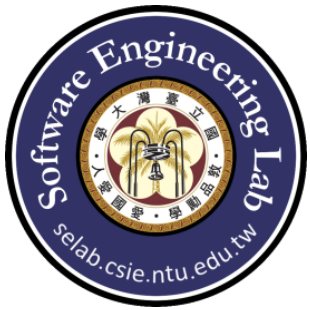
Client

```
public class Client {  
    public void printOutList(Iterator iterator){  
        while(!iterator.isEmpty()){  
            System.out.println(iterator.currentItem());  
            iterator.next();  
        }  
    }  
}
```



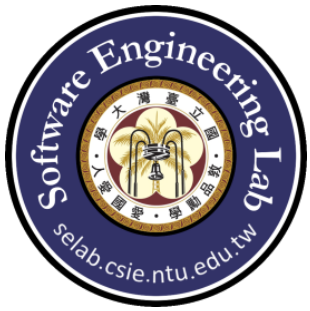
AbstractList

```
public interface AbstractList {  
    public Iterator createIterator();  
}
```



List

```
public class List implements AbstractList{  
    public int length = 0;  
    private String[] datas = new String[10000];  
  
    @Override  
    public Iterator createIterator() { return new ListIterator( list: this); }  
  
    public String get(int index){  
        if(index >= length){  
            return null;  
        }  
        else {  
            return datas[index];  
        }  
    }  
  
    public void add(String data){  
        datas[length] = data;  
        length ++;  
  
        if(length == datas.length){  
            datas = Arrays.copyOf(datas, newLength: length * 2);  
        }  
    }  
}
```

SkipList

```
public class SkipList implements AbstractList{
    private java.util.List<SkipNode> skipNodes = new ArrayList<>();

    @Override
    public Iterator createIterator() { return new SkipListIterator( skipList: this); }

    public SkipNode getNode(int index) { return skipNodes.get(index); }

    public int size(){
        return skipNodes.size();
    }

    public void add(SkipNode skipNode) { skipNodes.add(skipNode); }
```



SkipNode

```
public class SkipNode {  
    private String value;  
  
    public SkipNode(String value) {  
        this.value = value;  
    }  
}
```



Iterator

```
public interface Iterator {  
    public Object first();  
    public Object next();  
    public boolean isEmpty();  
    public Object currentItem();  
}
```



ListIterator

```
public class ListIterator implements Iterator{
    private List list;
    private int curIndex = 0;
    public ListIterator(List list) { this.list = list; }

    @Override
    public String first() {
        if(list.length > 0){
            return list.get(0);
        }
        return null;
    }

    @Override
    public String next() {
        String curNode = currentItem();
        curIndex++;
        return curNode;
    }

    @Override
    public boolean isEmpty() { return curIndex >= list.length; }

    @Override
    public String currentItem() {
        if(!isEmpty()){
            return list.get(curIndex);
        }
        else
            return null;
    }
}
```



SkipIterator

```
public class SkipListIterator implements Iterator{
    private SkipList skipList;
    private int curIndex = 0;

    public SkipListIterator(SkipList skipList) { this.skipList = skipList; }

    @Override
    public SkipNode first() {
        if(skipList.size() > 0){
            return skipList.getNode(index: 0);
        }
        return null;
    }

    @Override
    public SkipNode next() {
        SkipNode curNode = currentItem();
        curIndex++;
        return curNode;
    }

    @Override
    public boolean isEmpty() { return curIndex >= skipList.size(); }

    @Override
    public SkipNode currentItem() {
        if(!isEmpty()){
            return skipList.getNode(curIndex);
        }
        else
            return null;
    }
}
```



Input / Output

Input:

```
Create [DataStructure_name] [DataStructure]

Add [DataStructure_name] [Content]

Length [DataStructure_name]

Size [DataStructure_name]

Get [DataStructure_name] [index]

GetNode [DataStructure_name] [index]

PrintOutList [DataStructure_name]

...
```

Output:

```
//if [DataStructure] is List

    //input: Length [DataStructure_name]: print how many Strings

    [String_num]

    //input: Size [DataStructure_name]

    List do not have method size

    // input: Get [DataStructure_name] [index]: print content at [index]

    [Content_index]
```



Test cases

- ☐ TestCase 1: List (Include Invalid Command Size and GetNode)
- ☐ TestCase 2: SkipList (Include Invalid Command Length and Get)
- ☐ TestCase 3: Complex



Test case1

◀ ▶	Sample1.in	◀ ▶	Sample1.out
1	Create abc List	1	2
2	Add abc a	2	List do not have method size
3	Add abc b	3	a
4	Length abc	4	List do not have method getNode
5	Size abc	5	a
6	Add abc c	6	b
7	Get abc 0	7	c
8	GetNode abc 0		
9	PrintOutList abc		



Test case2

Sample2.in	Sample2.out
1 Create abc SkipList	1 SkipList can not access length
2 Add abc a	2 2
3 Add abc b	3 SkipList do not have method get
4 Length abc	4 SkipNode:a
5 Size abc	5 SkipNode:a
6 Add abc c	6 SkipNode:b
7 Get abc 0	7 SkipNode:c
8 GetNode abc 0	
9 PrintOutList abc	



Test case3

Sample3.in	Sample3.out
1 Create abc SkipList	1 SkipList can not access length
2 Add abc a	2 2
3 Add abc b	3 SkipList do not have method get
4 Length abc	4 SkipNode:a
5 Size abc	5 SkipNode:a
6 Add abc c	6 SkipNode:b
7 Get abc 0	7 SkipNode:c
8 GetNode abc 0	8 2
9 PrintOutList abc	9 List do not have method size
10 Create list List	10 a
11 Add list a	11 List do not have method getNode
12 Add list b	12 a
13 Length list	13 b
14 Size list	14 c
15 Add list c	15 a
16 Get list 0	16 b
17 GetNode list 0	17 c
18 PrintOutList list	18 d
19 Add list d	19 List do not have method getNode
20 PrintOutList list	20 d
21 GetNode list 3	
22 Get list 3	



- 33

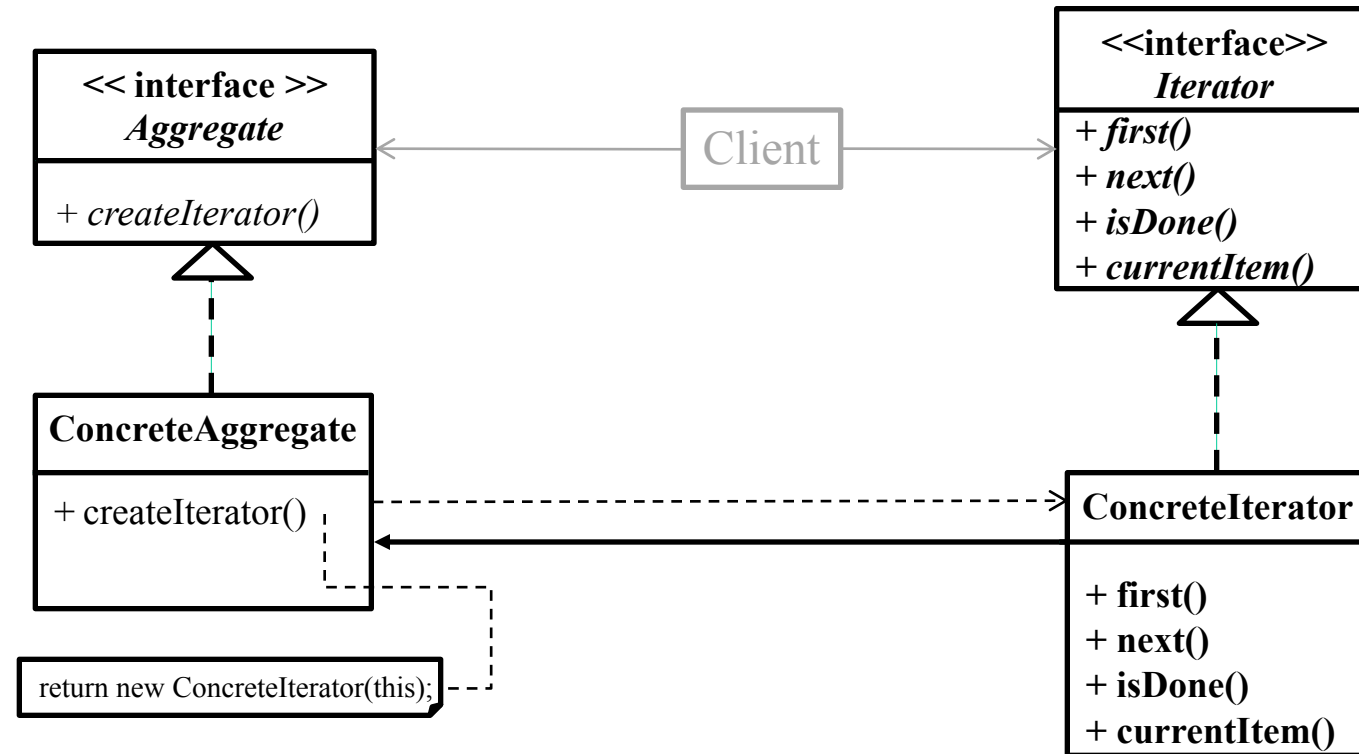


Intent

- ❑ Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.



Iterator Pattern Structure₁



Iterator Pattern Structure₂

1. Client creates a ConcreteAggregate.

2. Client invokes createIterator() to get a ConcreteIterator.

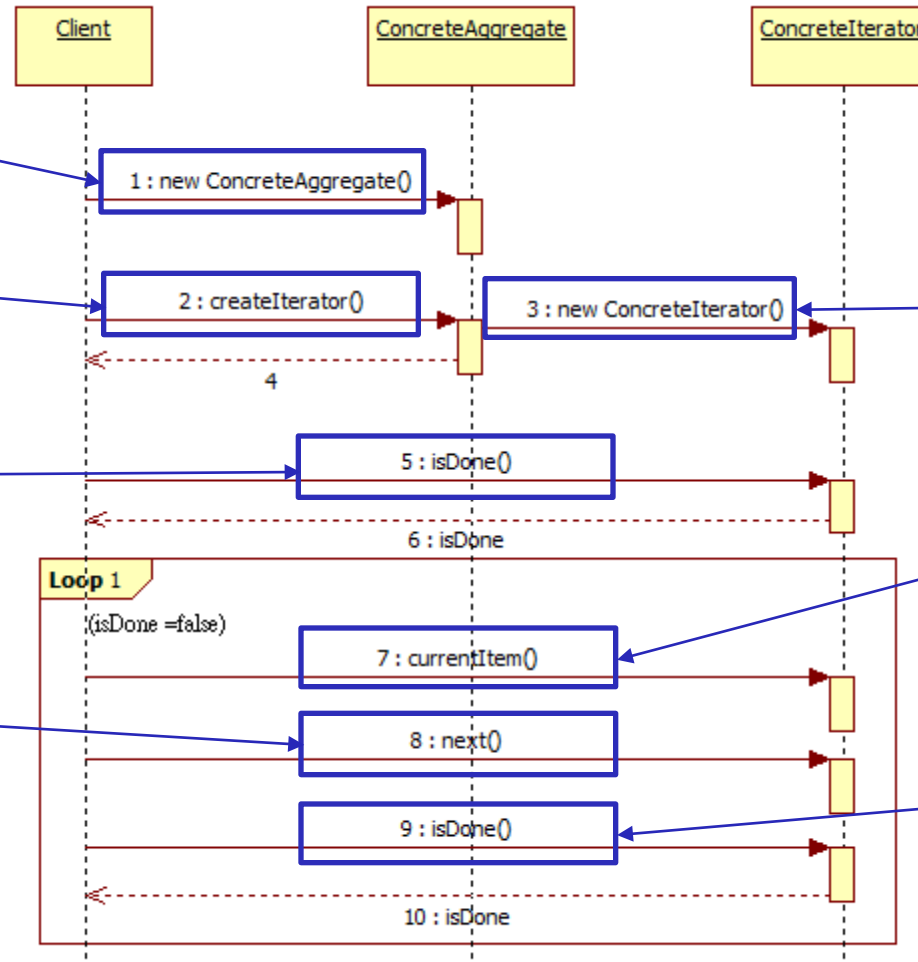
4. Client invokes isDone() to test whether we've advanced beyond the last element

6. Client invokes next() to advance the current element to the next element.

3. ConcreteAggregate creates a ConcreteIterator.

5. Client invokes currentItem() to return the current element in the list.

7. Client invokes isDone() again to test whether we've advanced beyond the last element





Iterator Pattern Structure₃

	Instantiation	Use	Termination
Client	Other class except classes in the Iterator Pattern	Other class except classes in the Iterator Pattern	Other class except classes in the Iterator Pattern
Aggregate	X	Client class uses this interface to get a ConcreteIterator through polymorphism	X
Concrete Aggregate	Other class or the client class	Client class uses this class to get a ConcreteIterator through Aggregate	Other class or the client class
Iterator	X	Client class uses ConcreteIterator through this interface	X
Concrete Iterator	ConcreteAggregate	Client class use this class to access the elements of an aggregate object sequentially	Other class or the client class

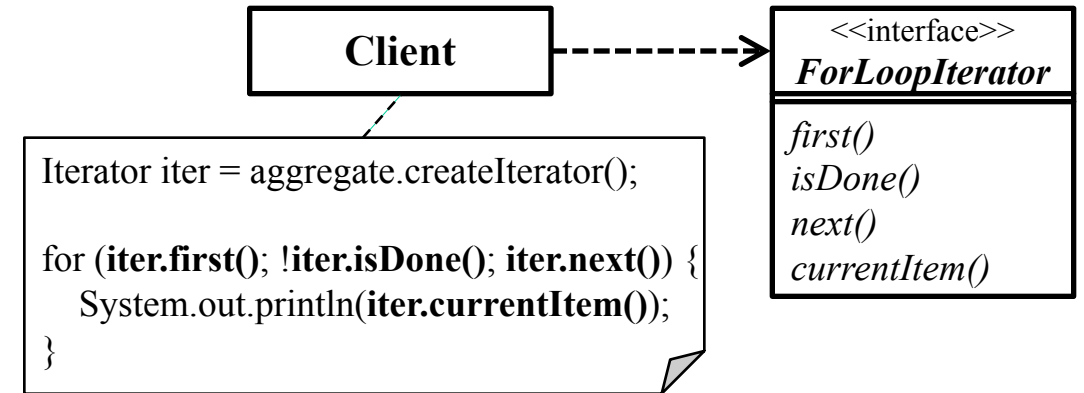


Two kinds of Iterator

❑ Two kinds of Iterators for different iterations

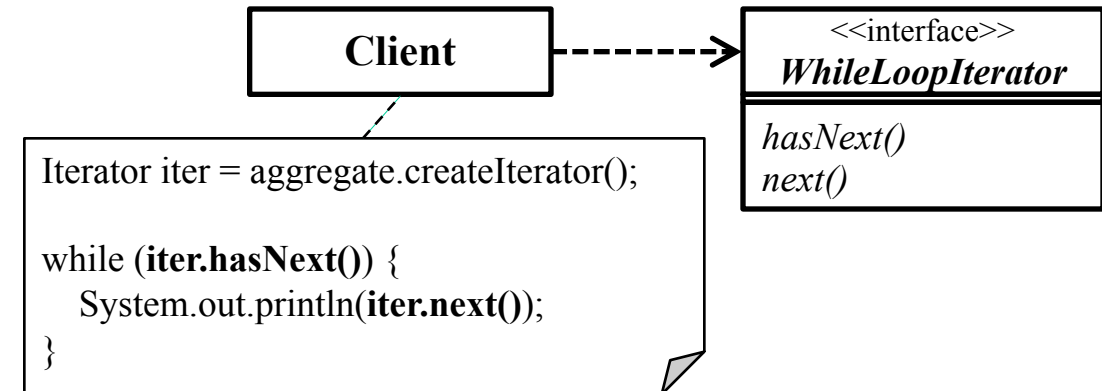
➤ For-Loop

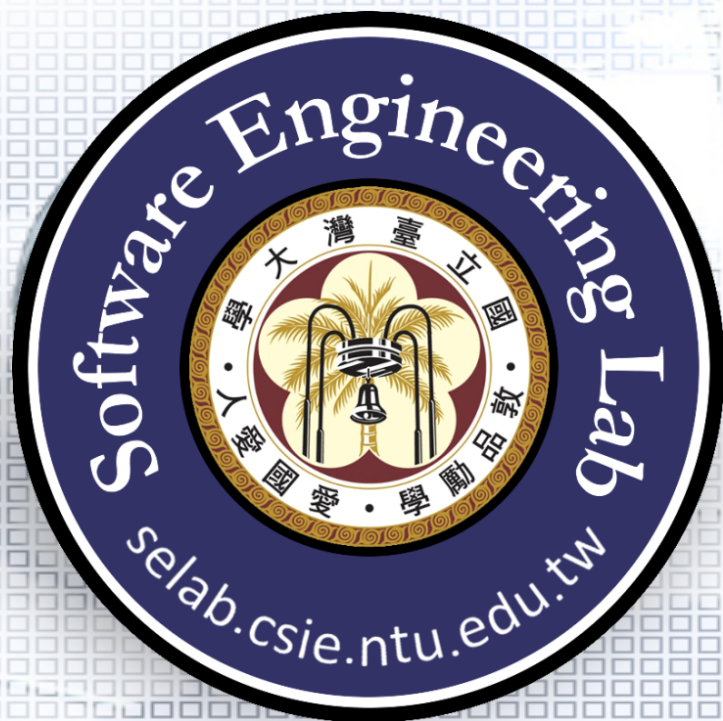
- **first()**: rewind to the first element in the aggregate.
- **isDone()**: check if all elements are traversed.
- **next()**: advance to the next element.
- **currentItem()**: return the current element.



➤ While-Loop

- **hasNext()**: check if there is any element that has not been traversed.
- **next()**: advance to the next element and return it.





Merge Two Menus

Prof. Jonathan Lee (李允中)

Department of Computer Science and
Information Engineering
National Taiwan University



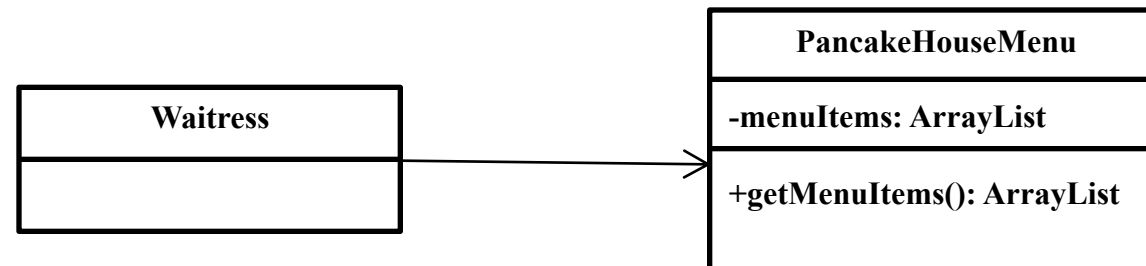
Requirements Statement

- ☐ A waitress of Pancake House keeps a breakfast menu which uses an ArrayList to hold its menu items.
- ☐ And a waitress of Diner keeps a lunch menu which uses an array to hold its menu items.
- ☐ Now, these two restaurants are merged and intend to provide services in one place, so a waitress should keep both menus in hands.
- ☐ The waitress would like to print two different menu representations at a time.



Requirements Statement

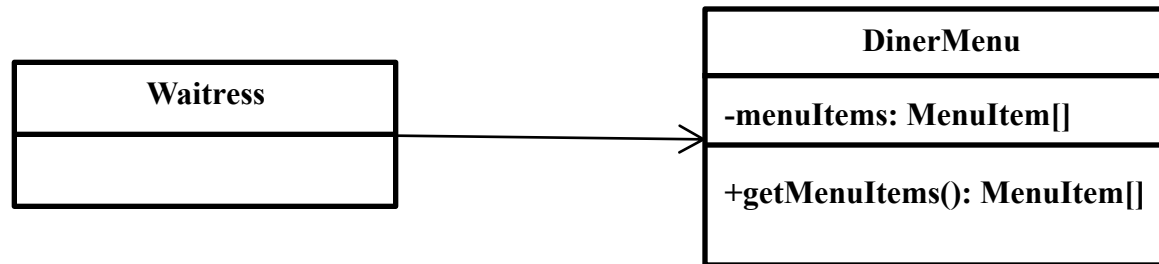
- ❑ A waitress of Pancake House keeps a breakfast menu which uses an ArrayList to hold its menu items.





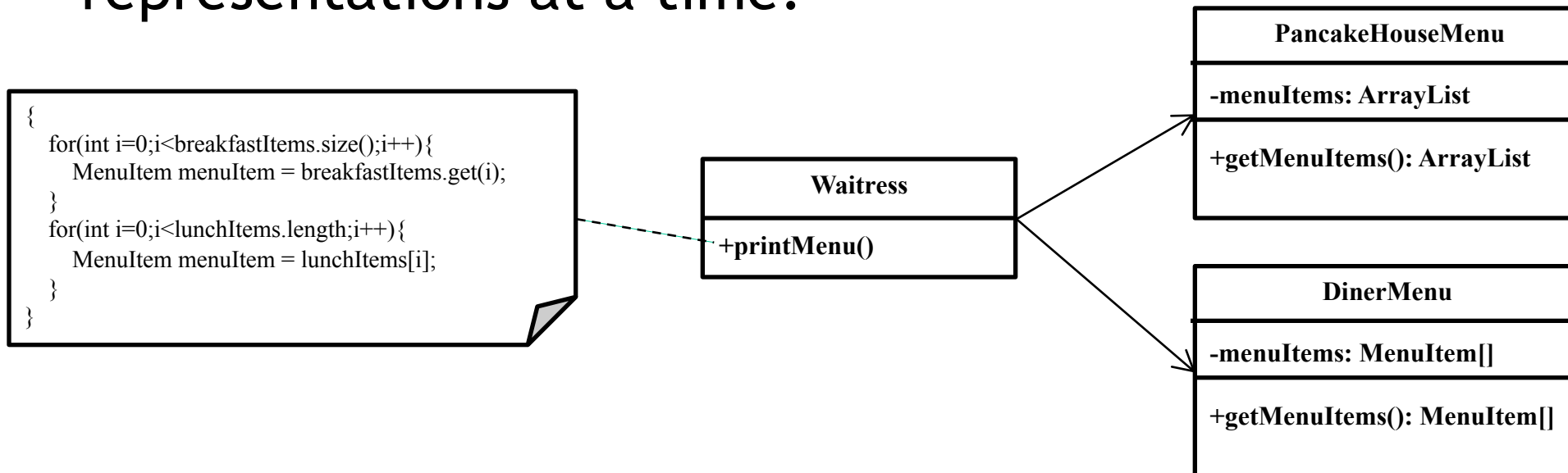
Requirements Statement

- ❑ And a waitress of Diner keeps a lunch menu which uses an array to hold its menu items.



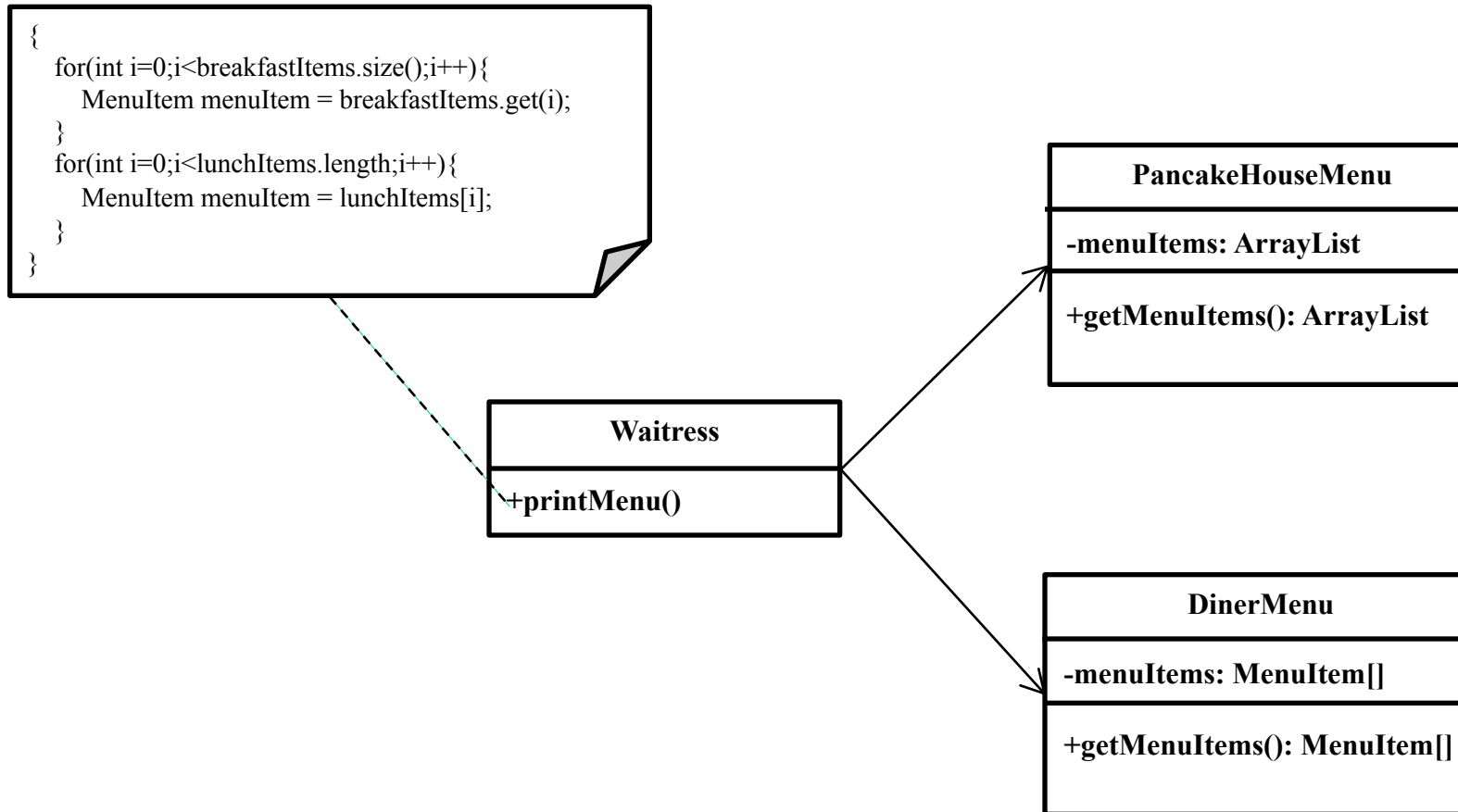
Requirements Statement

- ❑ Now, these two restaurants are merged and intend to provide service in one place, so a waitress should keep both menus in hands.
- ❑ The waitress would like to print two different menu representations at a time.

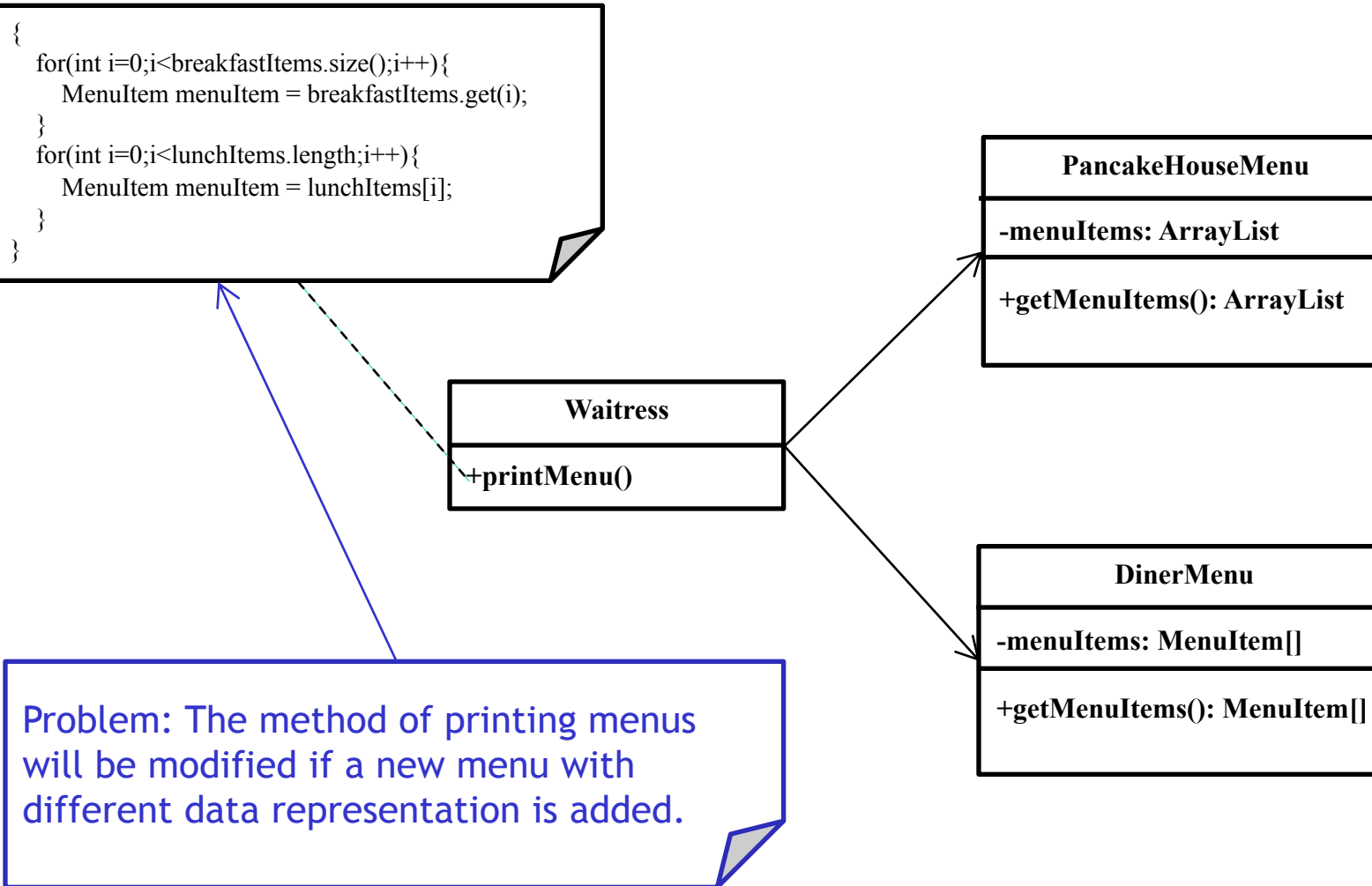




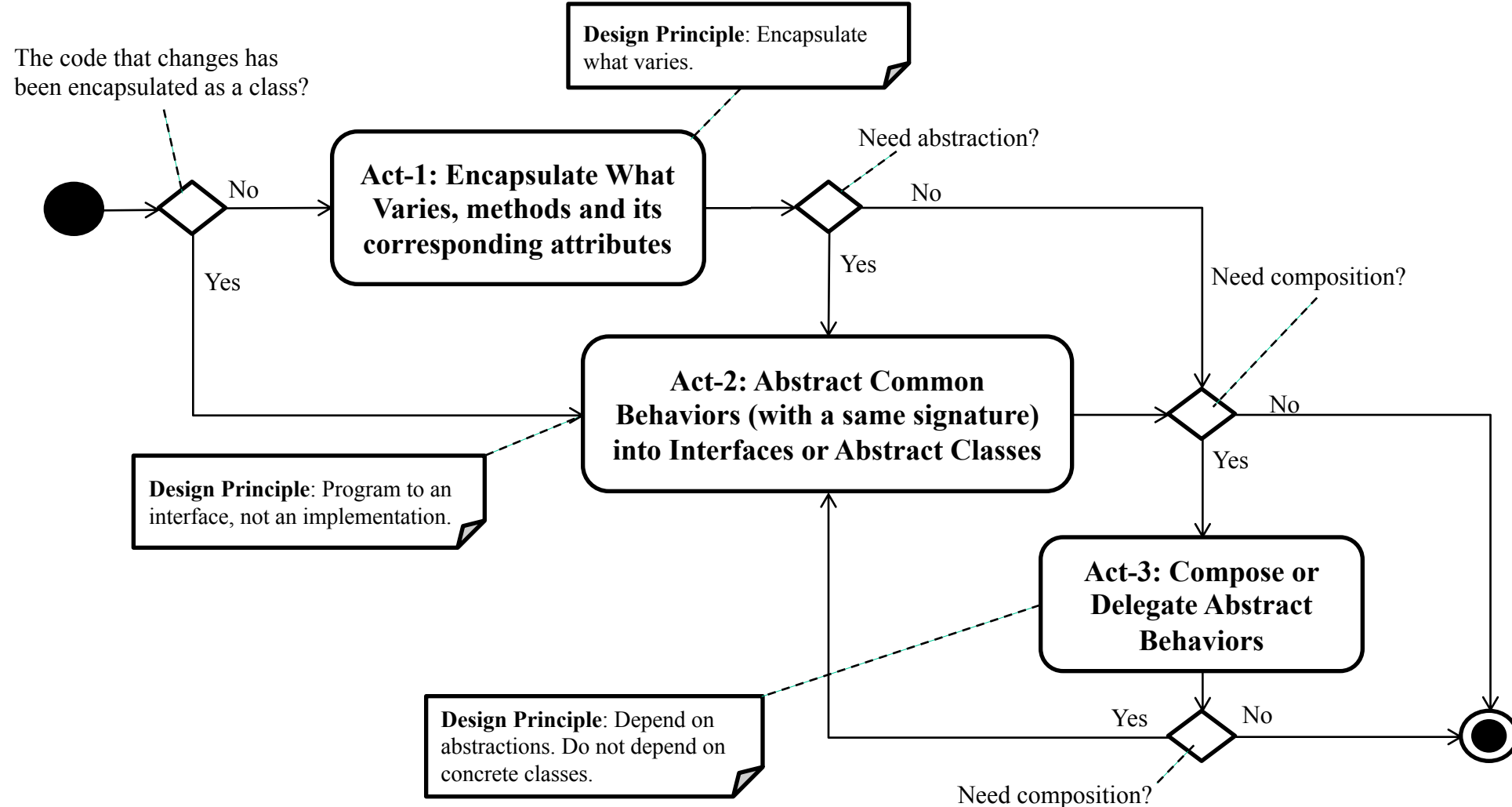
Initial Design - Class Diagram



Problems with Initial Design

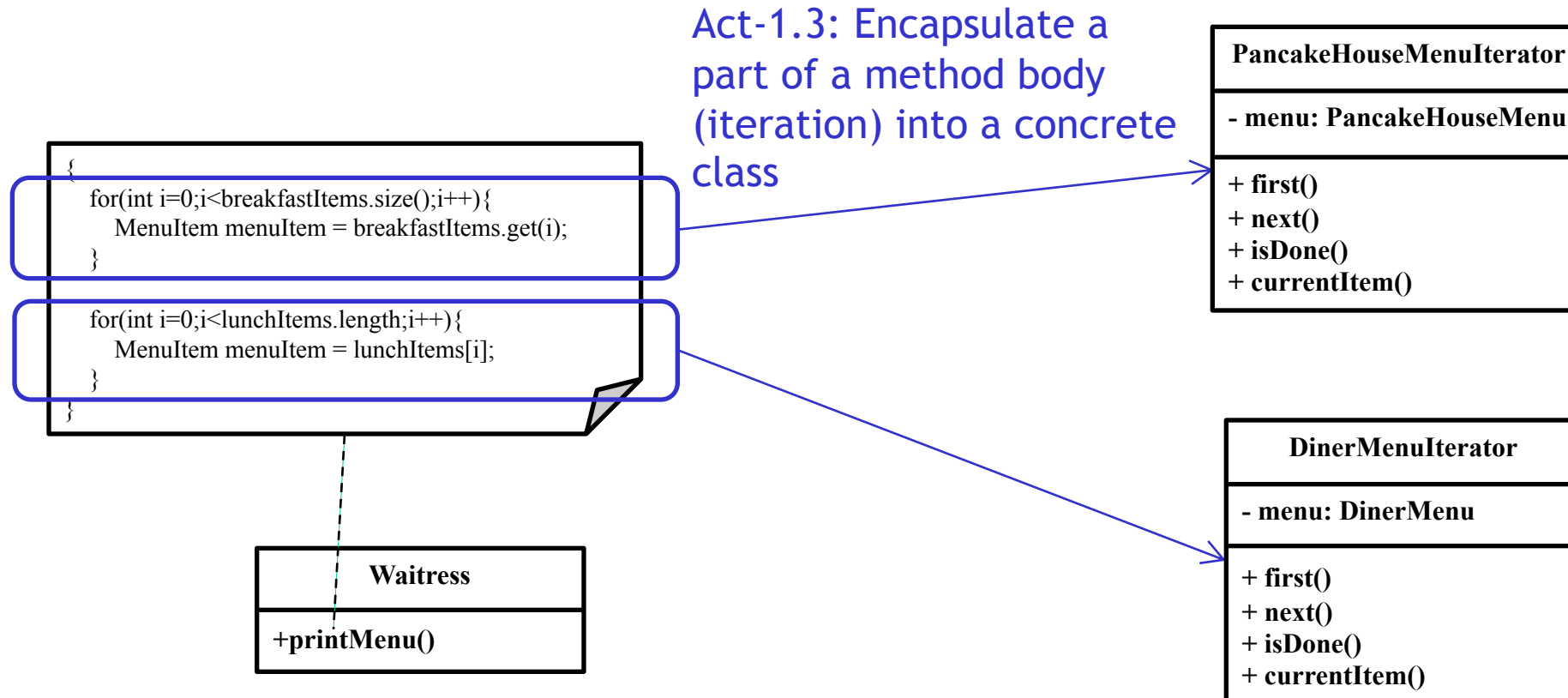


Design Process for Change

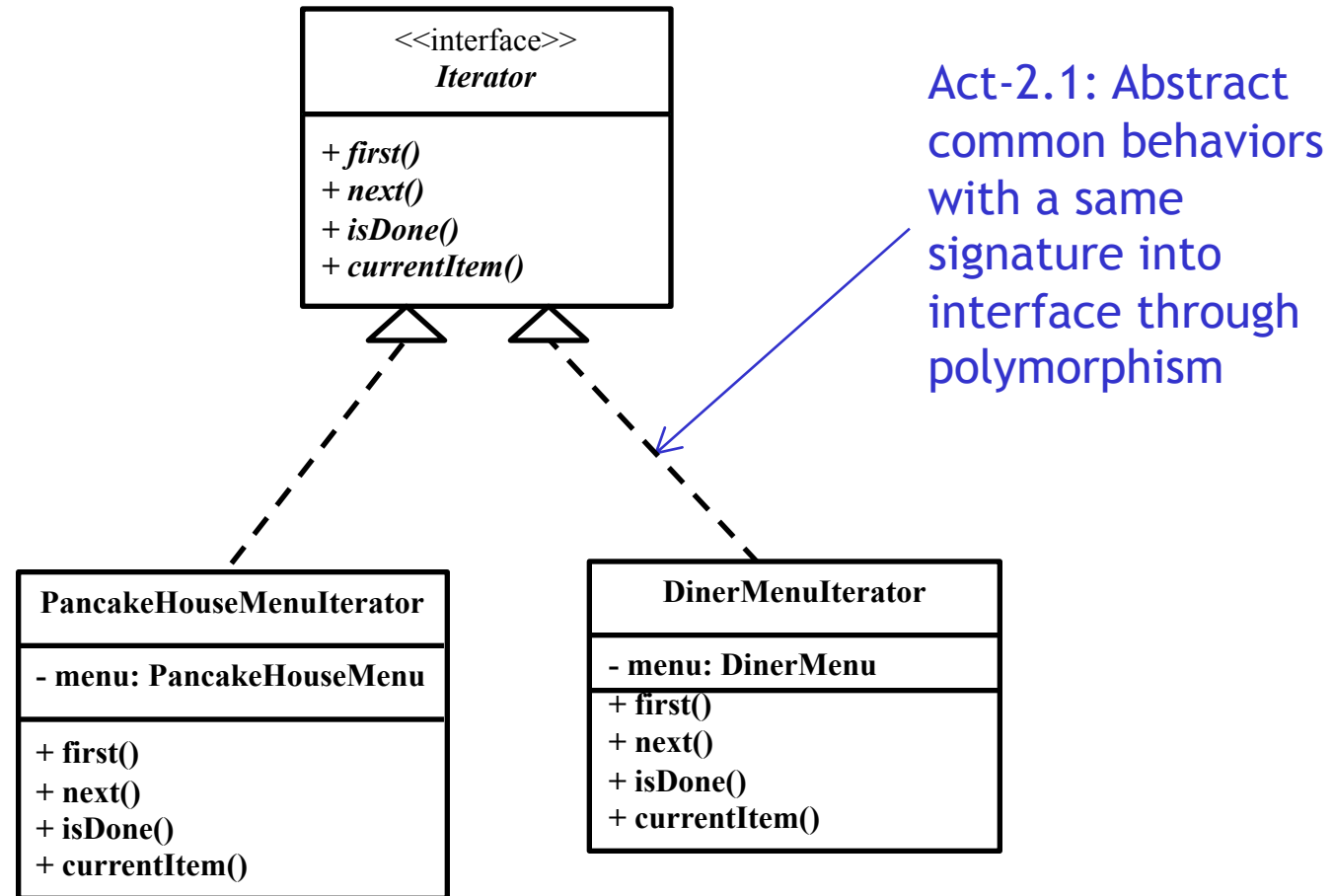




Act-1.3: Encapsulate What Varies



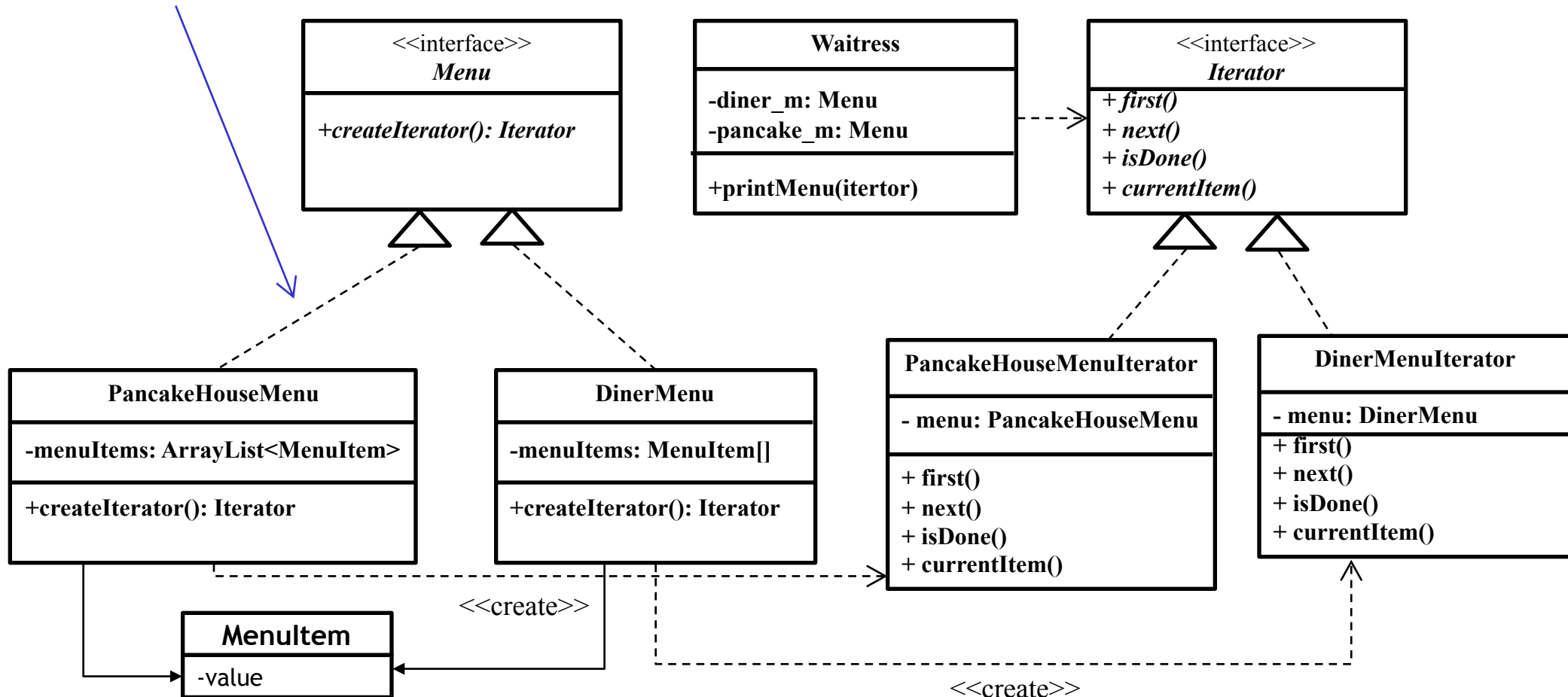
Act-2.1: Abstract Common Behaviors





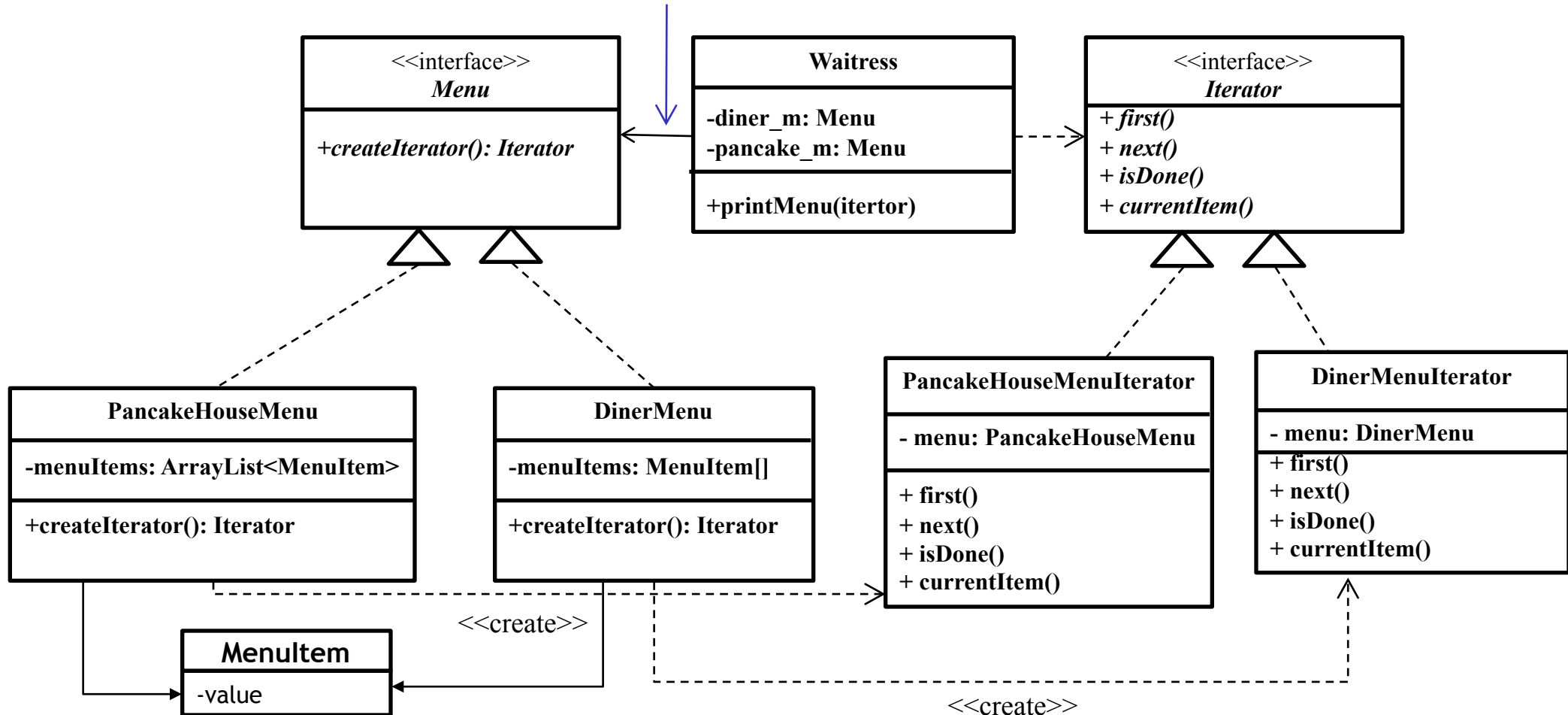
Act-2: Abstract Common Behaviors

Act-2.1: Abstract common behaviors with a same signature into interface through polymorphism

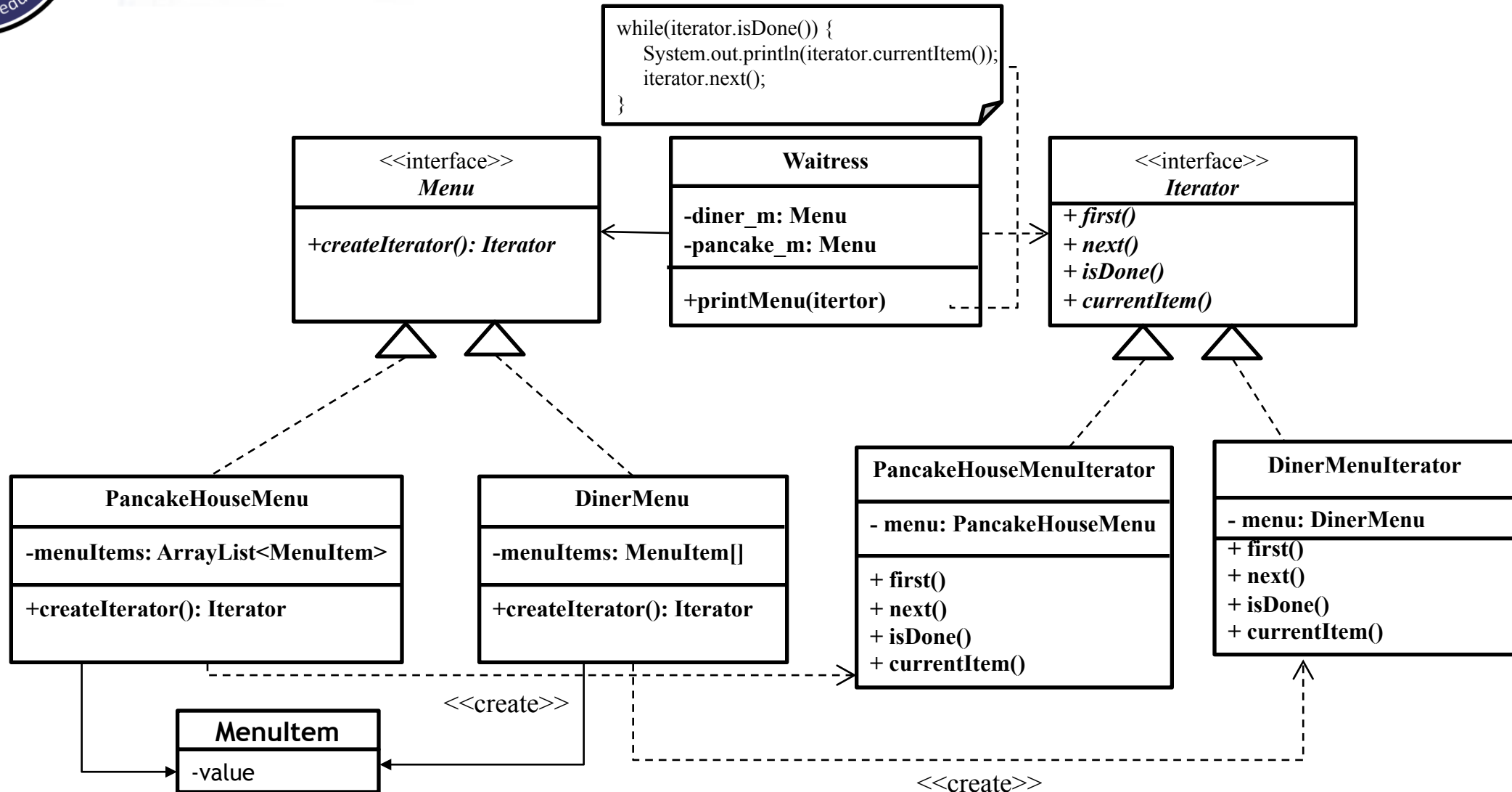


Act-3: Compose Abstract Interfaces/Abstract Classes

Act-3.1: Compose behaviors of an interface (through Waitress's attributes)



Refactored Design after Design Process





Waitress

```
public class Waitress {  
    private PancakeHouseMenu pancakeHouseMenu;  
    private DinerMenu dinerMenu;  
  
    public Waitress(PancakeHouseMenu pancakeHouseMenu, DinerMenu dinerMenu) {  
        this.pancakeHouseMenu = pancakeHouseMenu;  
        this.dinerMenu = dinerMenu;  
    }  
  
    public void printMenu(){  
        System.out.println("PancakeHouseMenu:");  
        printMenu(pancakeHouseMenu.createIterator());  
        System.out.println("DinerMenu:");  
        printMenu(dinerMenu.createIterator());  
    }  
  
    public void printMenu(Iterator iterator){  
        while(!iterator.isDone()){  
            System.out.println(iterator.currentItem());  
            iterator.next();  
        }  
    }  
}
```



Menu

```
public interface Menu {  
    public Iterator createIterator();  
    public MenuItem get(int index);  
    public void add(MenuItem menuItem);  
    public int size();  
}
```



PancakeHouseMenu

```
public class PancakeHouseMenu implements Menu{
    private ArrayList<MenuItem> menuItems = new ArrayList<>();

    @Override
    public Iterator createIterator(){ return new PancakeHouseMenuIterator( menu: this); }

    public MenuItem get(int index){ return menuItems.get(index); }

    public void add(MenuItem menuItem){ menuItems.add(menuItem); }

    public int size(){ return menuItems.size(); }
}
```



DinerMenu

```
public class DinerMenu implements Menu{
    private int length = 0;
    private MenuItem[] menuItems = new MenuItem[100];

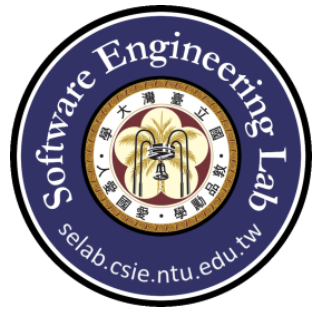
    @Override
    public Iterator createIterator(){ return new DinerMenuIterator( menu: this); }

    public MenuItem get(int index){
        if(index >= length ){
            return null;
        }
        else {
            return menuItems[index];
        }
    }

    public void add(MenuItem menuItem){
        menuItems[length] = menuItem;
        length ++;

        if(length == menuItems.length){
            menuItems = Arrays.copyOf(menuItems, newLength: length * 2);
        }
    }

    public int size(){ return length; }
}
```



MenuItem

```
public class MenuItem {  
    private String value;  
  
    public MenuItem(String value){  
        this.value = value;  
    }  
  
    @Override  
    public String toString(){  
        return "MenuItem:" + value;  
    }  
}
```



Iterator

```
public interface Iterator {  
    public MenuItem first();  
    public MenuItem next();  
    public boolean isDone();  
    public MenuItem currentItem();  
}
```



DinerMenuIterator

```
public class DinerMenuIterator implements Iterator{
    private DinerMenu menu;
    private int curIndex = 0;

    public DinerMenuIterator(DinerMenu menu) { this.menu = menu; }

    @Override
    public MenuItem first() {
        if(menu.size() > 0){
            return menu.get(0);
        }
        return null;
    }

    @Override
    public MenuItem next() {
        MenuItem curNode = currentItem();
        curIndex++;
        return curNode;
    }

    @Override
    public boolean isDone() { return curIndex >= menu.size(); }

    @Override
    public MenuItem currentItem() {
        if(!isDone()){
            return menu.get(curIndex);
        }
        else
            return null;
    }
}
```



PancakeHouseMenuIterator

```
public class PancakeHouseMenuIterator implements Iterator{
    private PancakeHouseMenu menu;
    private int curIndex = 0;
    public PancakeHouseMenuIterator(PancakeHouseMenu menu) { this.menu = menu; }

    @Override
    public MenuItem first() {
        if(menu.size() > 0){
            return menu.get(0);
        }
        return null;
    }

    @Override
    public MenuItem next() {
        MenuItem curNode = currentItem();
        curIndex++;
        return curNode;
    }

    @Override
    public boolean isDone() { return curIndex >= menu.size(); }

    @Override
    public MenuItem currentItem() {
        if(!isDone()){
            return menu.get(curIndex);
        }
        else
            return null;
    }
}
```




Input / Output

Input:

```
/*  
  
The order of PancakeHouse and Diner could be different from  
following example.  
  
[menu_item] should be a string.  
  
*/  
  
PancakeHous  
  
[menu_item]  
  
...  
  
Diner  
  
[menu_item]  
  
...
```

Output:

```
/*  
  
The order of PancakeHouse and Diner should be the same as  
following example.  
  
MenuItem:[menu_item] should be shown with sequential order from  
input.  
  
*/  
  
PancakeHouseMenu:  
  
MenuItem:[menu_item]  
  
...  
  
DinerMenu:  
  
MenuItem:[menu_item]  
  
...
```



Test case

```
Sample0.in
1 Diner|
2 beef0
3 chicken0
4 pork0
5 beef1
6 chicken1
7 pork1
8 beef2
9 chicken2
10 pork2
11 beef3
12 chicken3
13 pork3
14 beef4
15 chicken4
16 pork4
17 beef5
18 chicken5
19 pork5
20 beef6
21 chicken6
22 pork6
23 beef7
24 chicken7
25 pork7
26 beef8
27 chicken8
28 pork8
29 beef9
30 chicken9
31 pork9
32 beef10
33 chicken10
34 pork10
35 beef11
36 chicken11
37 pork11
38 beef12
39 PancakeHouse
40 toast
41 toast10
42 egg10
43 hamburger10
44 toast11
45 egg11
46 hamburger11
47 toast12
48 egg12
49 hamburger12
50 toast13
51 egg13
52 hamburger13
53 toast14
54 egg14
55 hamburger14
56 toast15
57 egg15
58 hamburger15
59 toast16
60 egg16
61 hamburger16
62 toast17
63 egg17
64 hamburger17
```

```
Sample0.out
1 PancakeHouseMenu:
2 MenuItem:toast
3 MenuItem:toast10
4 MenuItem:egg10
5 MenuItem:hamburger10
6 MenuItem:toast11
7 MenuItem:egg11
8 MenuItem:hamburger11
9 MenuItem:toast12
10 MenuItem:egg12
11 MenuItem:hamburger12
12 MenuItem:toast13
13 MenuItem:egg13
14 MenuItem:hamburger13
15 MenuItem:toast14
16 MenuItem:egg14
17 MenuItem:hamburger14
18 MenuItem:toast15
19 MenuItem:egg15
20 MenuItem:hamburger15
21 MenuItem:toast16
22 MenuItem:egg16
23 MenuItem:hamburger16
24 MenuItem:toast17
25 MenuItem:egg17
26 MenuItem:hamburger17
27 DinerMenu:
28 MenuItem:beef0
29 MenuItem:chicken0
30 MenuItem:pork0
31 MenuItem:beef1
32 MenuItem:chicken1
33 MenuItem:pork1
34 MenuItem:beef2
35 MenuItem:chicken2
36 MenuItem:pork2
37 MenuItem:beef3
38 MenuItem:chicken3
39 MenuItem:pork3
40 MenuItem:beef4
41 MenuItem:chicken4
42 MenuItem:pork4
43 MenuItem:beef5
44 MenuItem:chicken5
45 MenuItem:pork5
46 MenuItem:beef6
47 MenuItem:chicken6
48 MenuItem:pork6
49 MenuItem:beef7
50 MenuItem:chicken7
51 MenuItem:pork7
52 MenuItem:beef8
53 MenuItem:chicken8
54 MenuItem:pork8
55 MenuItem:beef9
56 MenuItem:chicken9
57 MenuItem:pork9
58 MenuItem:beef10
59 MenuItem:chicken10
60 MenuItem:pork10
61 MenuItem:beef11
62 MenuItem:chicken11
63 MenuItem:pork11
64 MenuItem:beef12
```