CS2040 Lecture Note #3A:
List ADT & Linked Lists

Abstraction of a list

Lecture Note #3A: List ADT & Linked Lists

- **Objectives:**
  - Able to define a List ADT, a LinkedList ADT
  - Able to implement a List ADT with array
  - Able to implement a LinkedList ADT with linked list
  - Able to use Java API LinkedList class
  - In general, able to define and implement own ADTs

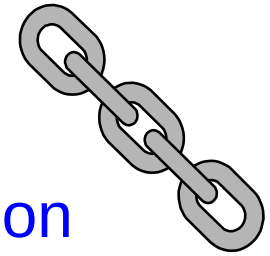
Outline

1. Use of a List (Motivation)
   - List ADT
2. List ADT Implementation via Array
   - Add and remove with an array
   - Time and space efficiency
3. List ADT Implementation via Linked Lists
   - Linked list approach
   - ListNode class: forming a linked list with ListNode
4. ADT of a Linked List
   - Various types of linked lists
   - BasicLinkedList, ExtendedLinkedList, TailedLinkedList, CircularLinkedList
   - DListNode class for DoublyLinkedList
5. Java class: LinkedList

1 Use of a List

Motivation

1. Use of a List (Motivation)



- List is one of the most basic types of data collection
  - For example, list of groceries, list of modules, list of friends, etc.
  - In general, we keep items of the same type (class) in one list
- Typical Operations on a data collection
  - Add data
  - Remove data
  - Query data
  - The details of the operation vary from application to application. The overall theme is the management of data

1. ADT of a list (1/3)

- A list ADT is a dynamic linear data structure
  - A collection of data items, accessible one after another starting from the beginning (head) of the list
- Examples of List ADT operations:
  - Create an empty list
  - Determine whether a list is empty
  - Determine number of items in the list
  - Add an item at a given position
  - Remove an item at a position
  - Remove all items
  - Read an item from the list at a position
- The next slide on the basic list interface does not have all the above operations… we will slowly build up these operations in list beyond the basic list.

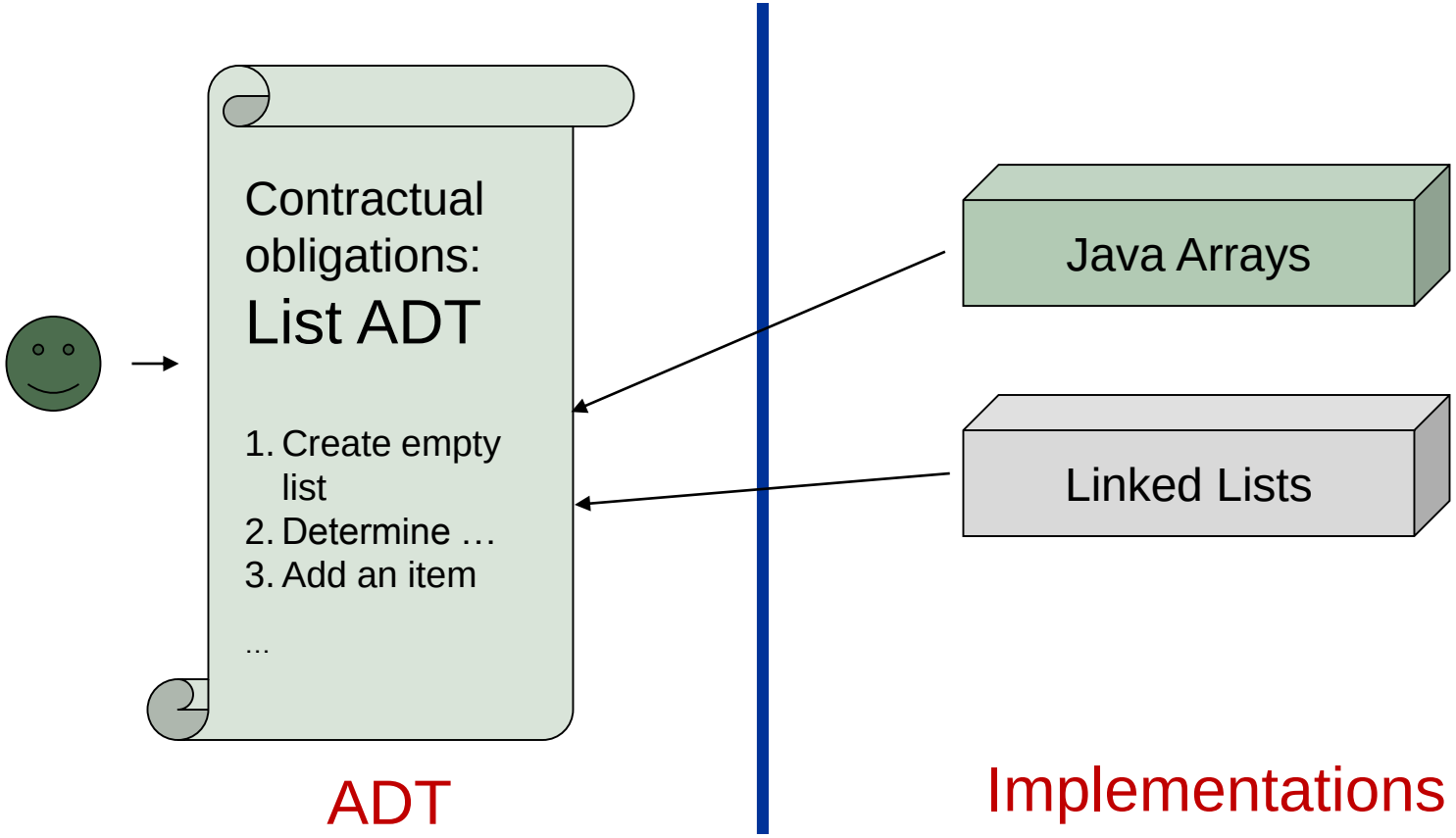
1. ADT of a list (2/3)

ListInterface.java

```
import java.util.*;

public interface ListInterface <E> {

  public boolean isEmpty();
  public int     size();
  public E       getFirst() throws NoSuchElementException;
  public boolean contains(E item);
  public void    addFirst(E item);
  public E       removeFirst() throws NoSuchElementException;

  public void    print();
}
```

1. ADT of a list (3/3)

- We will examine 2 implementations of list ADT

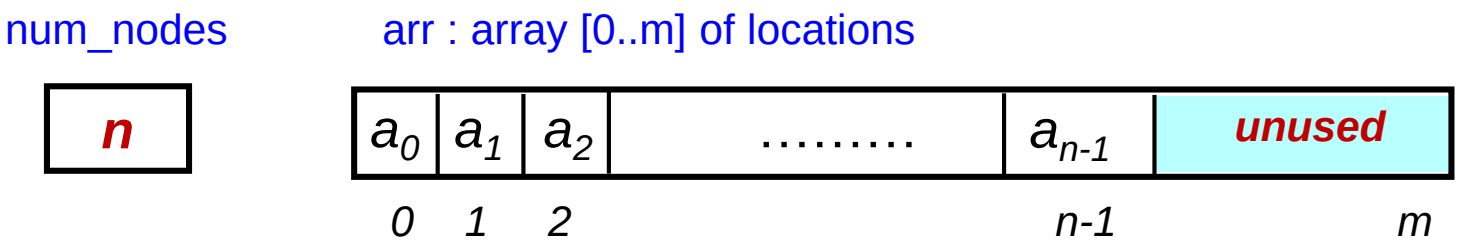


2 List Implementation via Array

Fixed-size list

2. List Implementation: Array (1/7)

- This is a straight-forward approach
  - Use Java array of a sequence of n elements



2. List Implementation: Array (2/7)

MyList.java

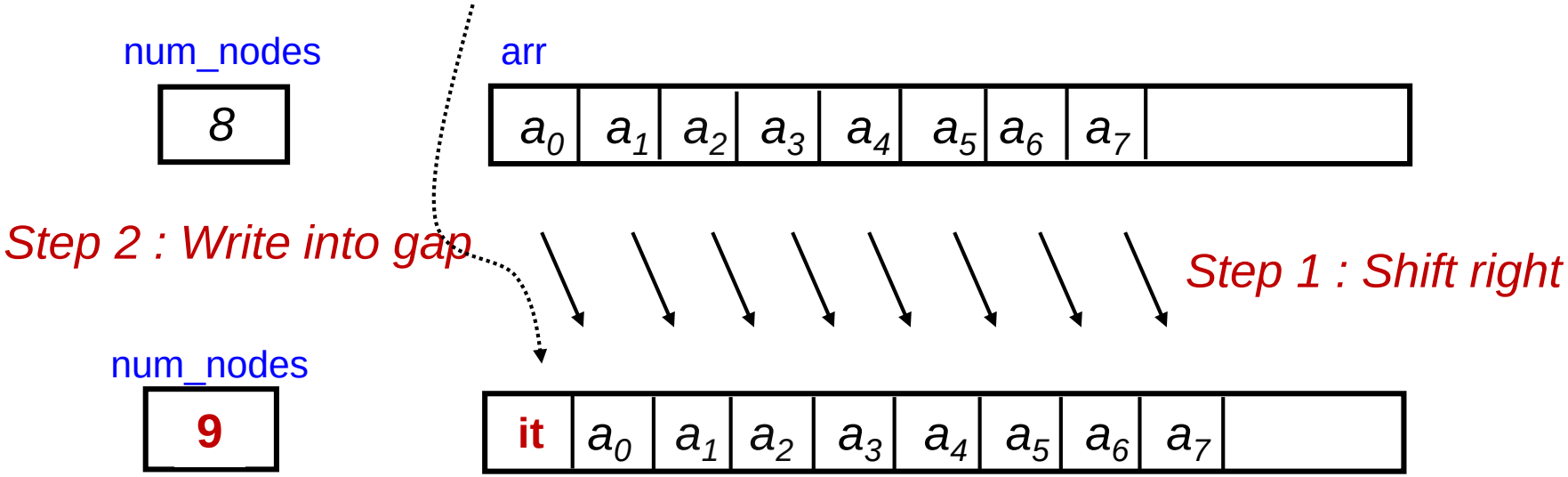
```
import java.util.*;

class MyList <E> implements ListInterface <E> {
  private static final int MAXSIZE = 1000;
  private int num_nodes = 0;
  private E[] arr = (E[]) new Object[MAXSIZE];

  public boolean isEmpty() { return num_nodes==0; }
  public int size()        { return num_nodes; }

  public E getFirst() throws NoSuchElementException {
    if (num_nodes == 0)
      throw new NoSuchElementException("can't get from an empty list");
    else return arr[0];
  }

  public boolean contains(E item) {
    for (int i = 0; i < num_nodes; i++)
      if (arr[i].equals(item)) return true;

    return false;
  }
```

Code continued in slide 14

2. List Implementation: Array (3/7)

- For insertion, need to shift "right" to create room

Example: addFirst("it")

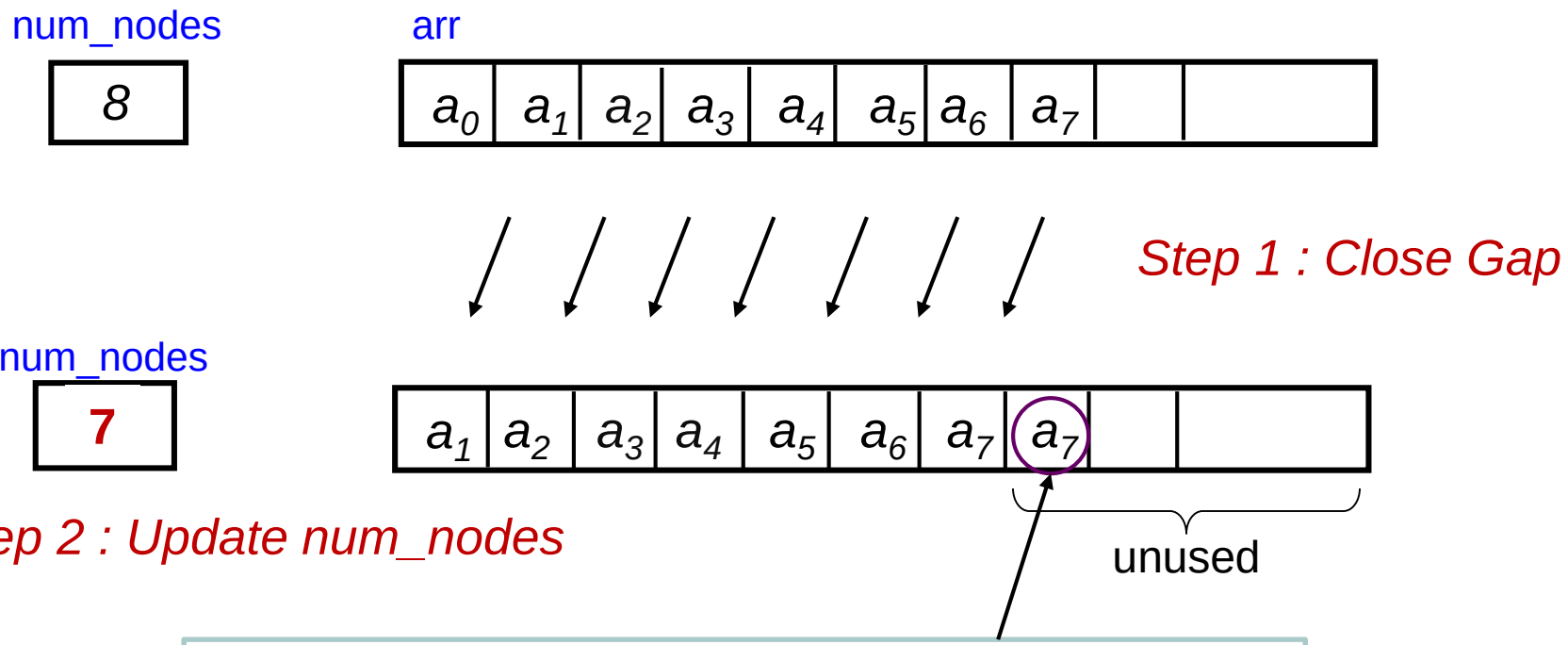


Step 3 : Update num_nodes

2. List Implementation: Array (4/7)

- For deletion, need to shift "left" to close gap

Example: removeFirst()



Need to maintain *num_nodes* so that program would not access beyond the valid data.

2. List Implementation: Array (5/7)

```
public void addFirst(E item) throws IndexOutOfBoundsException {
  if (num_nodes == MAXSIZE)
    throw new IndexOutOfBoundsException("insufficient space for add");
  for (int i = num_nodes-1; i >= 0; i--)
    arr[i+1] = arr[i];

  arr[0] = item;
  num_nodes++; // update num_nodes
}

public E removeFirst() throws NoSuchElementException {
  if (num_nodes == 0)
    throw new NoSuchElementException("can't remove from an empty list");
  else {
    E tmp = arr[0];
    for (int i = 0; i<num_nodes-1; i++)
      arr[i] = arr[i+1];
    num_nodes--; // update num_nodes
    return tmp;
  }
}
```

`print()` method not shown here. Refer to program.

MyList.java

2. List Implementation: Array (6/7)

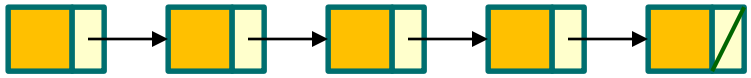
- Question: Time Efficiency?
  - Retrieval: getFirst()
    - Always fast with 1 read operation
  - Insertion: addFirst(E item)
    - Shifting of all n items – bad!
  - Insertion: add(int index, E item)
    - Inserting into the specified position (not shown in MyList.java)
      - Best case: No shifting of items (add to the last place)
      - Worst case: Shifting of all items (add to the first place)
  - Deletion: removeFirst(E item)
    - Shifting of all n items – bad too!
  - Deletion: remove(int index)
    - Delete the item at the specified position (not shown in MyList.java)
      - Best case: No shifting of items (delete the last item)
      - Worst case: Shifting of all items (delete the first item)

2. List Implementation: Array (7/7)

- Question : Space Efficiency?
  - Size of array collection limited by MAXSIZE
  - Problems
    - We don't always know the maximum size ahead of time
    - If MAXSIZE is too liberal, unused space is wasted
    - If MAXSIZE is too conservative, easy to run out of space
- Idea: make MAXSIZE a variable, and create/copy to a larger array whenever the array runs out of space
  - No more limits on Size
  - But copying overhead is still a problem
- When to use such a list?
  - For a fixed-size list, an array is good enough!
  - For a variable-size list, where dynamic operations such as insertion/deletion are common, an array is a poor choice; better alternative – **Linked List**

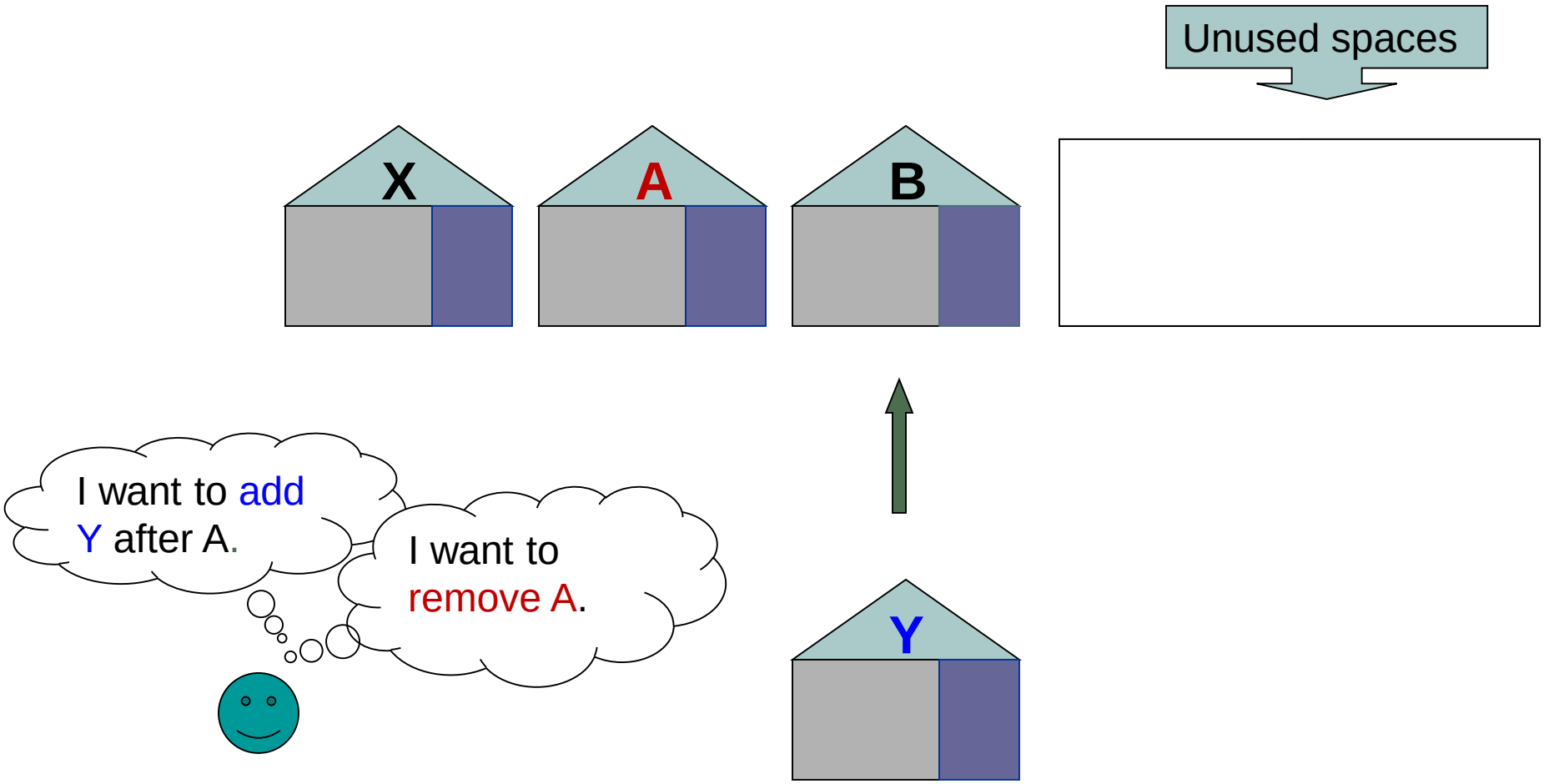
3 List Implementation via Linked List

Variable-size list



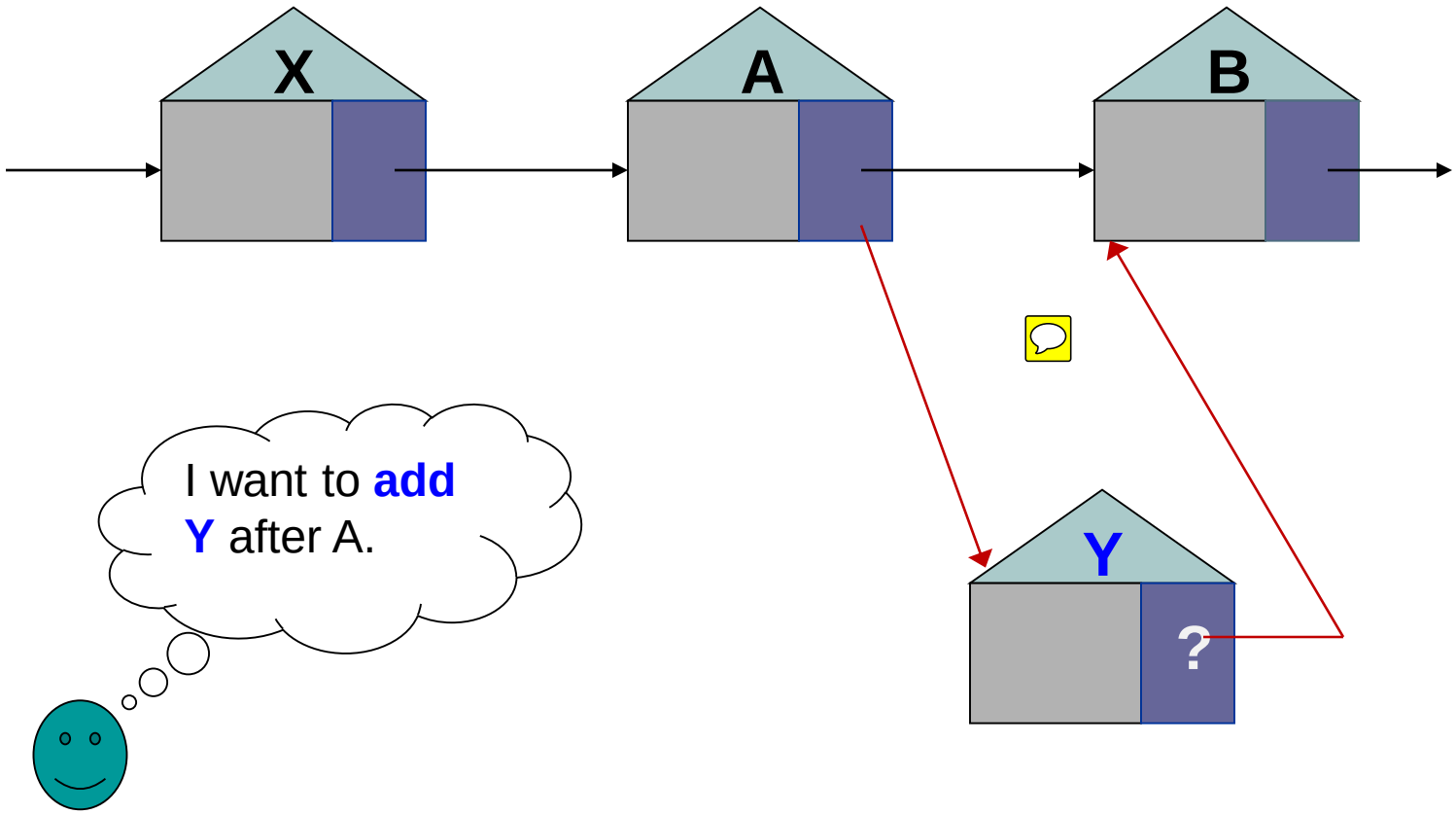
3. List Implementation: Linked List (1/3)

- Recap when using an array...



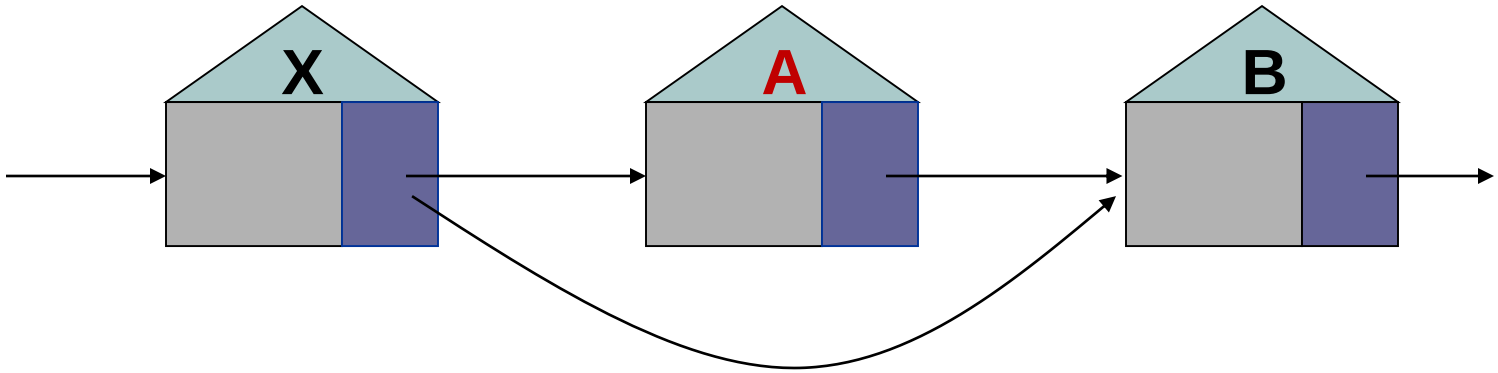
3. List Implementation: Linked List (2/3)

- Now, we see the (add) action with linked list…



3. List Implementation: Linked List (3/3)

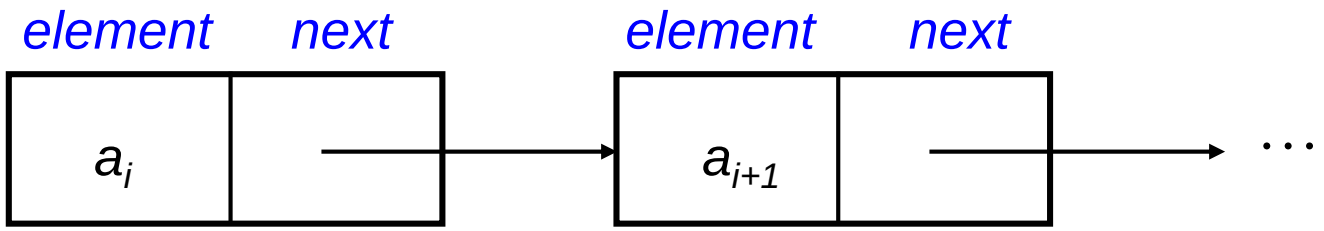
- Now, we see the (remove) action with linked list…



Node A becomes a *garbage*. To be removed during garbage collection.

3.1 Linked List Approach (1/4)

- Idea
  - Allow items to be non-contiguous in memory
  - Order the items by associating each with its neighbour(s)



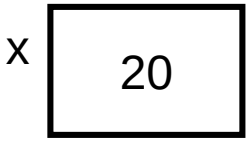
This is one item of the collection…

… and this one comes after it.

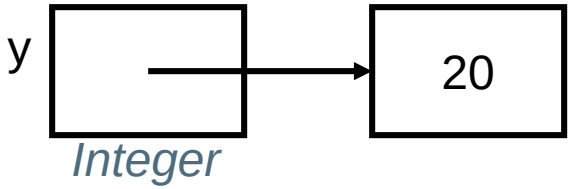
3.1 Linked List Approach (2/4)

- Recap: Object References (1/2)
  - Note the difference between primitive data types and reference data types

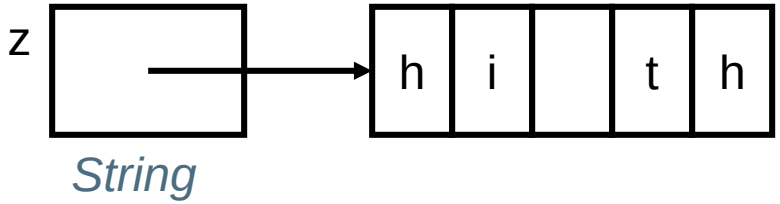
```
int x = 20;
```




```
Integer y = new Integer(20);
```




```
String z = new String("hi th");
```

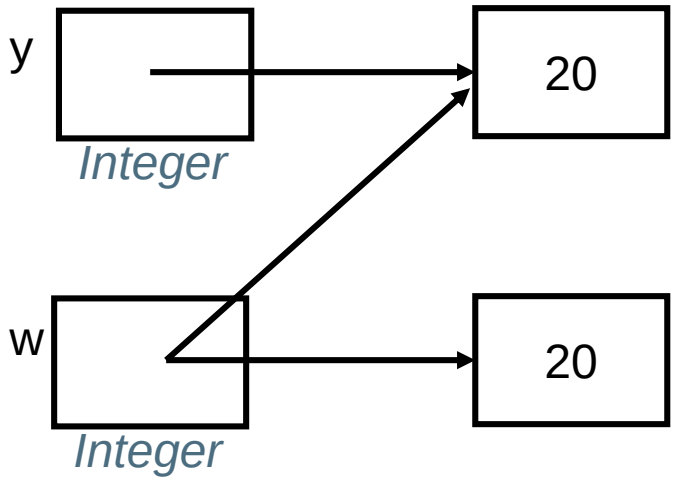


- An object of a class comes into existence when applying the new operator.
- A reference variable only contains a reference or pointer to an object.

3.1 Linked List Approach (3/4)

- Recap: Object References (2/2)
  - Look at it in more details:

```
Integer y = new Integer(20);
Integer w;
w = new Integer(20);
if (w == y)
  System.out.println("1. w == y");
w = y;
if (w == y)
    System.out.println("2. w == y");
```



- Output:

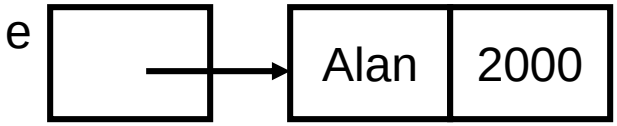
3.1 Linked List Approach (4/4)

- Quiz: Which is the representation of e?

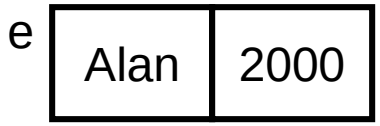
```
class Employee {
    private String name;
    private int salary;
    // etc.
}
```

Employee e = new Employee("Alan", 2000);

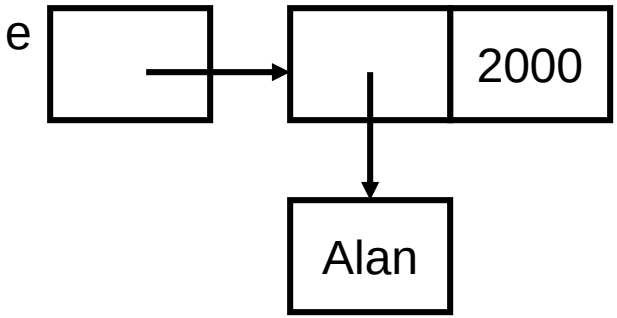
(A)



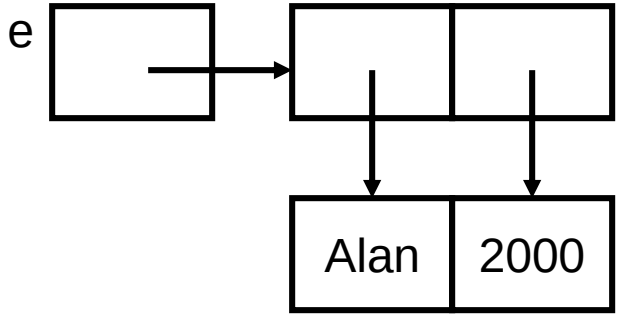
(B)



(C)

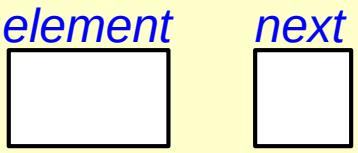


(D)



3.2 ListNode

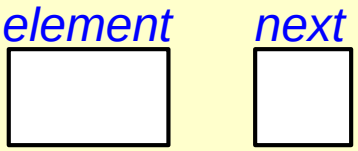
```
class ListNode0 {
  protected Object element;
  protected ListNode0 next;

  /* constructors */
  public ListNode0(Object item) { element = item; next = null; }
  public ListNode0(Object item, ListNode0 n) { element=item; next=n;}

  /* get the next ListNode */
  public ListNode0 getNext() {
    return this.next;
  }

  /* get the element of the ListNode */
  public Object getElement() {
    return this.element;
  }
}
```



ListNode0.java

3.2 ListNode (using generic)

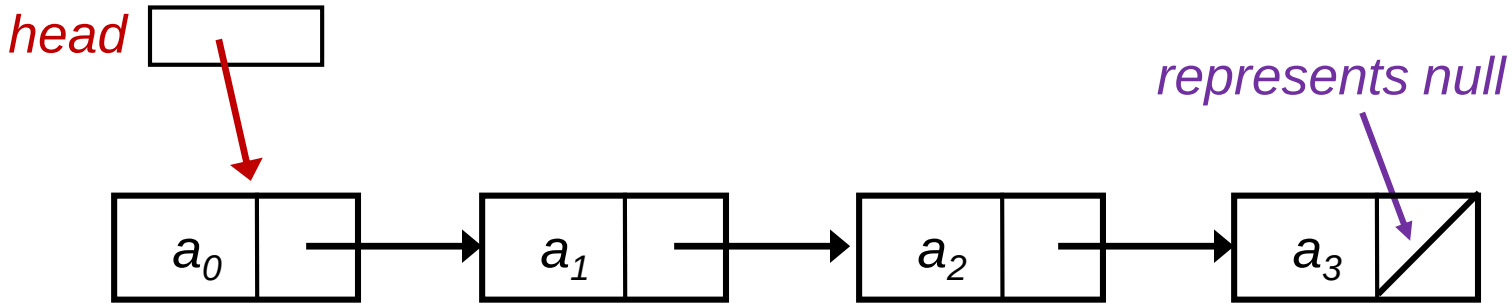
```
class ListNode <E> {
  protected E element;
  protected ListNode <E> next;

  /* constructors */
  public ListNode(E item) { element = item; next = null; }
  public ListNode(E item, ListNode <E> n) { element = item; next=n;}

  /* get the next ListNode */
  public ListNode <E> getNext() {
    return this.next;
  }

  /* get the element of the ListNode */
  public E getElement() {
    return this.element;
  }
}
```



ListNode.java

3.3 Forming Linked List (1/2)

- For a sequence of 4 items < a_0, a_1, a_2, a_3 >



We need a *head* to indicate where the first node is.
From the *head* we can get to the rest.

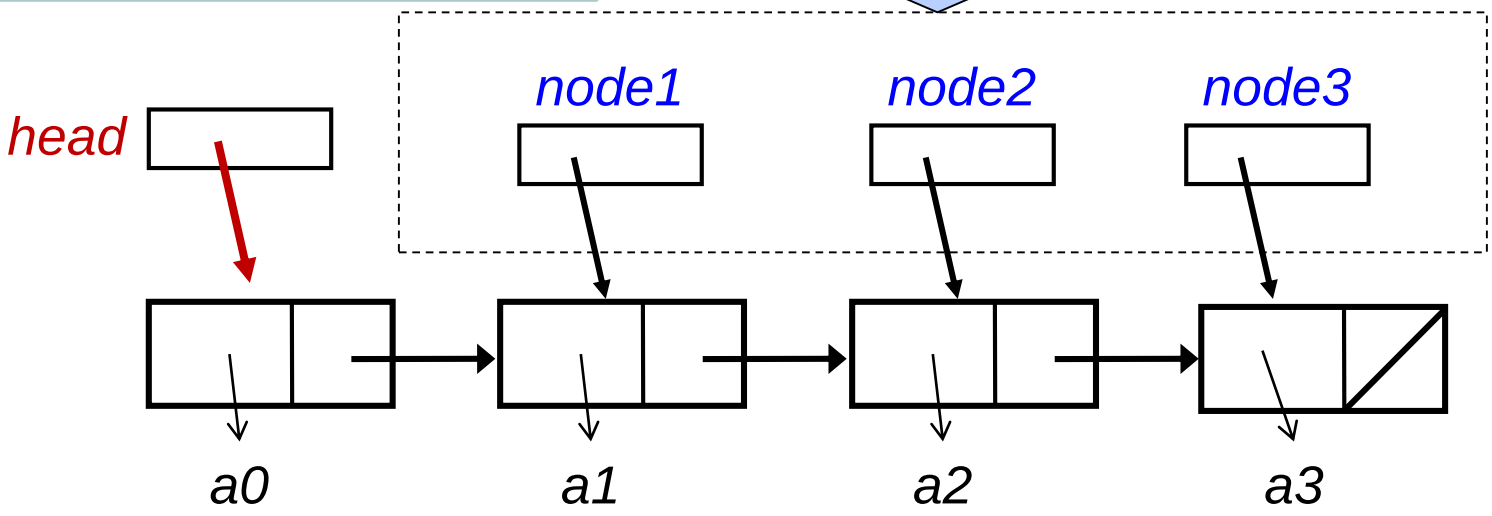
3.3 Forming Linked List (2/2)

- For a sequence of 4 items < a_0, a_1, a_2, a_3 >

```
ListNode <String> node3 = new ListNode <String>("a3",null);
ListNode <String> node2 = new ListNode <String>("a2",node3);
ListNode <String> node1 = new ListNode <String>("a1",node2);
ListNode <String> head =  new ListNode <String>("a0",node1);
```

Can the code be rewritten without using these objects node1, node2, node3?

No longer needed after list is built.

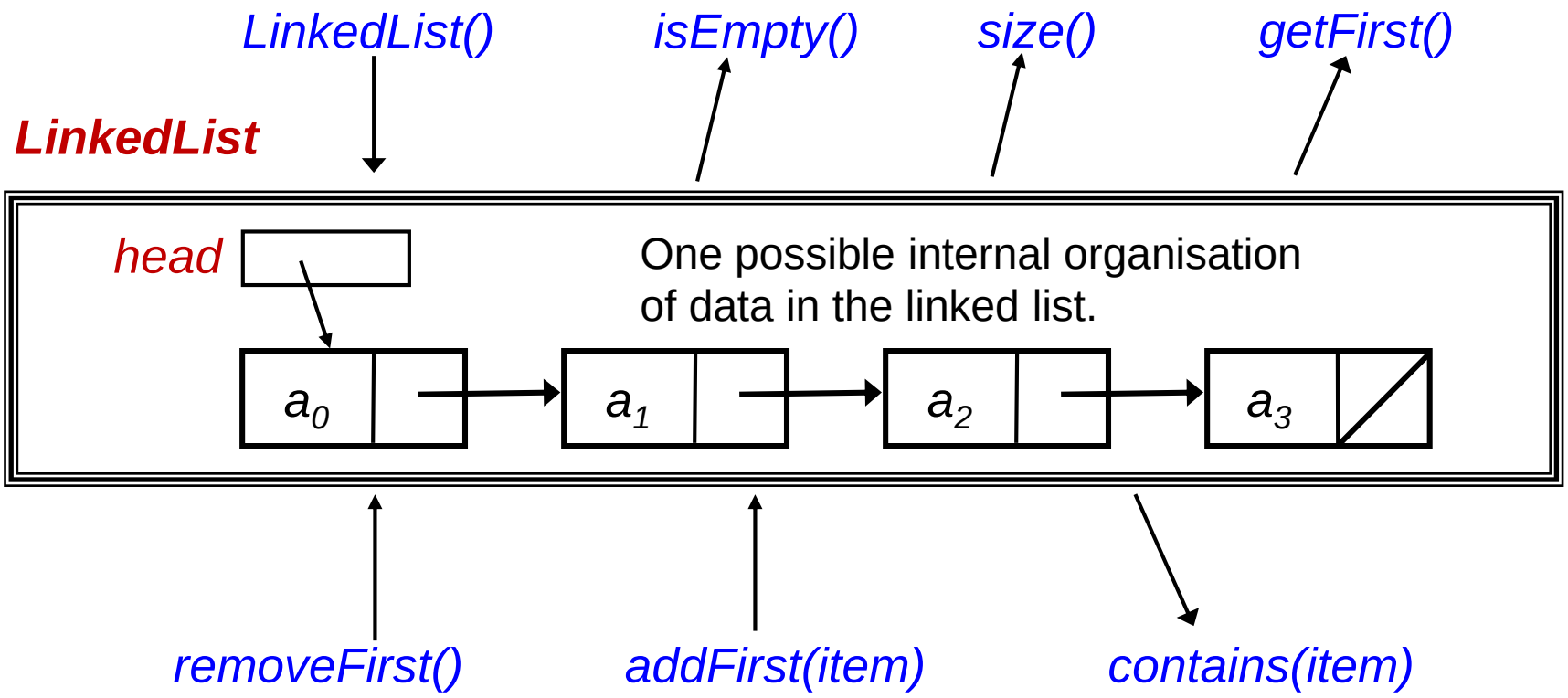


4 Linked List ADT

Exploring linked list and its variants

4. ADT of a Linked List (1/3)

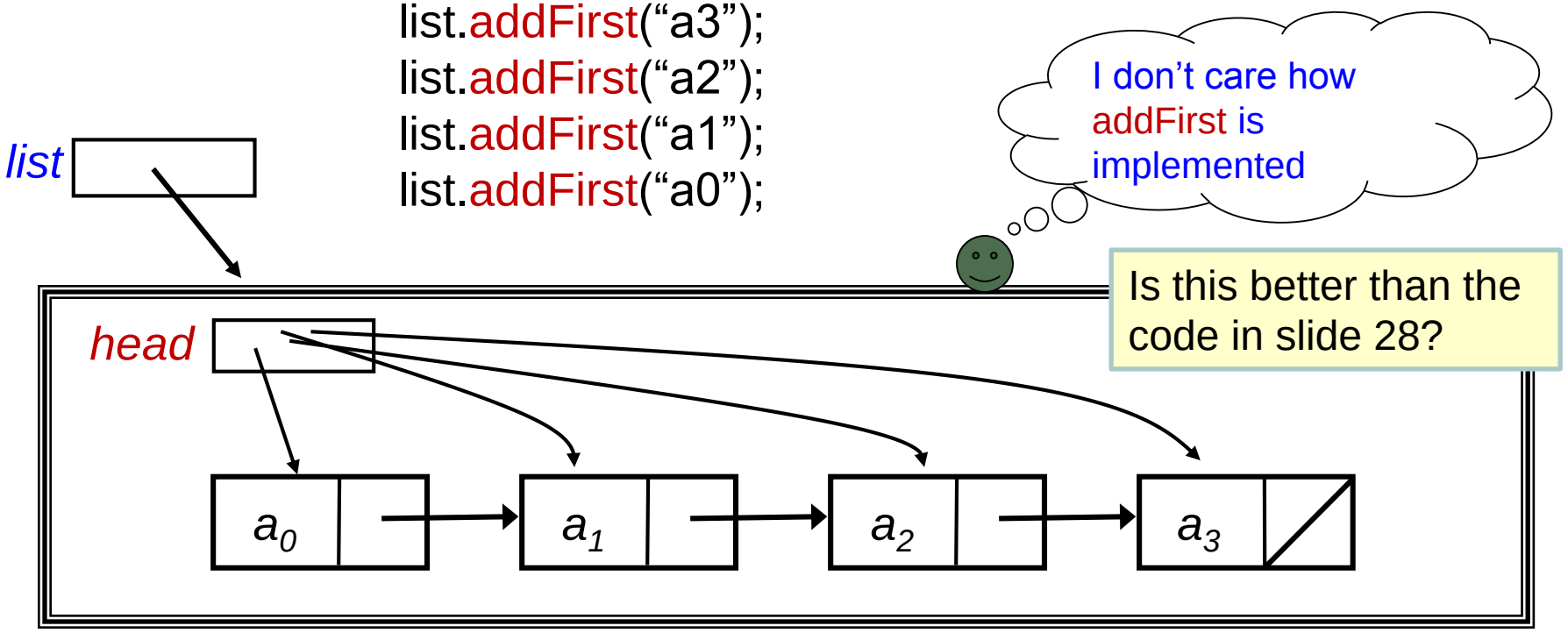
- We explore different implementations of Linked List ADT
- For example, if one such implementation is called LinkedList:



4. ADT of a Linked List (2/3)

- For a sequence of 4 items < a_0, a_1, a_2, a_3 >, we can build as follows:

```
LinkedList <String> list = new LinkedList <String> ();
list.addFirst("a3");
list.addFirst("a2");
list.addFirst("a1");
list.addFirst("a0");
```



4. ADT of a Linked List (3/3)

LinkedListInterface.java

```
import java.util.*;

public interface LinkedListInterface <E> {

  public boolean  isEmpty();
  public int      size();
  public E        getFirst() throws NoSuchElementException;
  public boolean  contains(E item);
  public void     addFirst(E item);
  public E        removeFirst() throws NoSuchElementException;

  public void     print();

  // ....etc....
}
```

- The // …etc…. part should be added (which is omitted here) to make it different from ListInterface
- Note that in general an interface can extend another interface (which is not done in this example)

4.1 Basic Linked List (1/5)

- Using ListNode to define BasicLinkedList

BasicLinkedList.java

```
import java.util.*;
class BasicLinkedList <E> implements LinkedListInterface <E> {
  protected ListNode <E> head = null;
  protected int num_nodes = 0;

  public boolean isEmpty() { return (num_nodes == 0); }

  public int size() { return num_nodes; }

  public E getFirst() throws NoSuchElementException {
    if (head == null)
      throw new NoSuchElementException("can't get from an empty list");
    else return head.element;
  }

  public boolean contains(E item) {
    for (ListNode <E> n = head; n!= null; n=n.next)
      if (n.getElement().equals(item)) return true;
    return false;
  }
```

4.1 Basic Linked List (2/5)

- The adding and removal of items

BasicLinkedList.java

```
public void addFirst(E item) {
   head = new ListNode <E> (item, head);
   num_nodes++;
}

public E removeFirst() throws NoSuchElementException {
   ListNode <E> ln;
   if (head == null)
     throw new NoSuchElementException("can't remove from empty list");
   else {
     ln = head;
     head = head.next;
     num_nodes--;
     return ln.element;
   }
}
```

4.1 Basic Linked List (3/5)

- Printing of the linked list

BasicLinkedList.java

```
public void print2() throws NoSuchElementException {
   if (head == null)
      throw new NoSuchElementException("Nothing to print...");

   ListNode <E> ln = head;
   System.out.print("List is: " + ln.element);
   for (int i=1; i < num_nodes; i++) {
      ln = ln.next;
      System.out.print(", " + ln.element);
   }
   System.out.println(".");
}
```

4.1 Basic Linked List (4/5)

- Example use #1

TestBasic1.java

```
import java.util.*;

class TestBasic1 {
  public static void main(String [] args)
                          throws NoSuchElementException {
    BasicLinkedList <String> bl = new BasicLinkedList <String> ();
    bl.addFirst("aaa");
    bl.addFirst("bbb");
    bl.addFirst("ccc");
    bl.print();

    System.out.println("testing removal");
    bl.removeFirst();
    bl.print();

    System.out.println("testing removal");
    bl.removeFirst();
    if (bl.contains("aaa")) bl.addFirst ("xxxx");
    bl.print();
  }
}
```

Answer?

4.1 Basic Linked List (5/5)

- Example use #2

TestBasic2.java

```
import java.util.*;

class TestBasic2 {
  public static void main(String [] args)
                          throws NoSuchElementException {
    BasicLinkedList <Integer> bl = new BasicLinkedList <Integer> ();

    bl.addFirst(34);
    bl.addFirst(12);
    bl.addFirst(9);
    bl.print();

    System.out.println("testing removal");
    bl.removeFirst();
    bl.print();
  }
}
```

Answer?