Data Structures

List

Ja-Hee Kim@seoultech

Contents

01

ADT: List

02

Array List

03

Linked List

04

Variations

Data Structures . List

ADT: List

List

- a set of names, numbers etc, usually written one below the other, for example so that you can remember or check them

ADT List

List

Responsibilities

add: append element x

add: insert element x into i^{th} position

remove: delete element x

remove: delete i^{th} element

get: let the caller know what element is the i^{th} element

indexOf: let the caller know what is the i^{th} element

clear: remove all entries form the list

size: gets the number of entries currently in the list.

ADT List

List is the ordered collection of data, and it allows duplicate elements

- add(newEntry: T): Boolean

  Apend newEntry to the list

  - Input
    - newEntry is the object to be added.
  - Postcondition: the list contains newEntry
  - Return:
    - true if adding is success
    - false if adding is fail
- add(givenPosition: integer , newEntry: T): Boolean

  Adds newEntry into the givenPoistion of the list

  - Input
    - givenPosition: the index where the newEntry will be located
    - newEntry is the object to be added.
  - Postcondition: the list contains newEntry in givenPosition
  - Return:
    - true if adding is success
    - false if adding is fail

ADT List

- remove(anEntry: T): Boolean
  - Removes the first or only occurrence of anEntry from the list.
  - Input: anEntry is the object to be removed.
  - Postcondition: the first anEntry does not exist in the list
  - Return: Returns true if anEntry was located and removed, or false if not. In the latter case, the list remains unchanged.

- remove(givenPosition: integer): T
  - Removes and returns the entry at position givenPosition.
  - Input: givenPosition is an integer.
  - Postcondition: the element in the givenPosition does not exist in the list
  - Return: the object at the index givenPosition.

ADT List

- get(givenPosition: integer): T
    - Gets the element in the givenPosition
    - Input: givenPosition is an index.
    - Precondition: givenPosition has an element
    - Return: the object at the index givenPosition

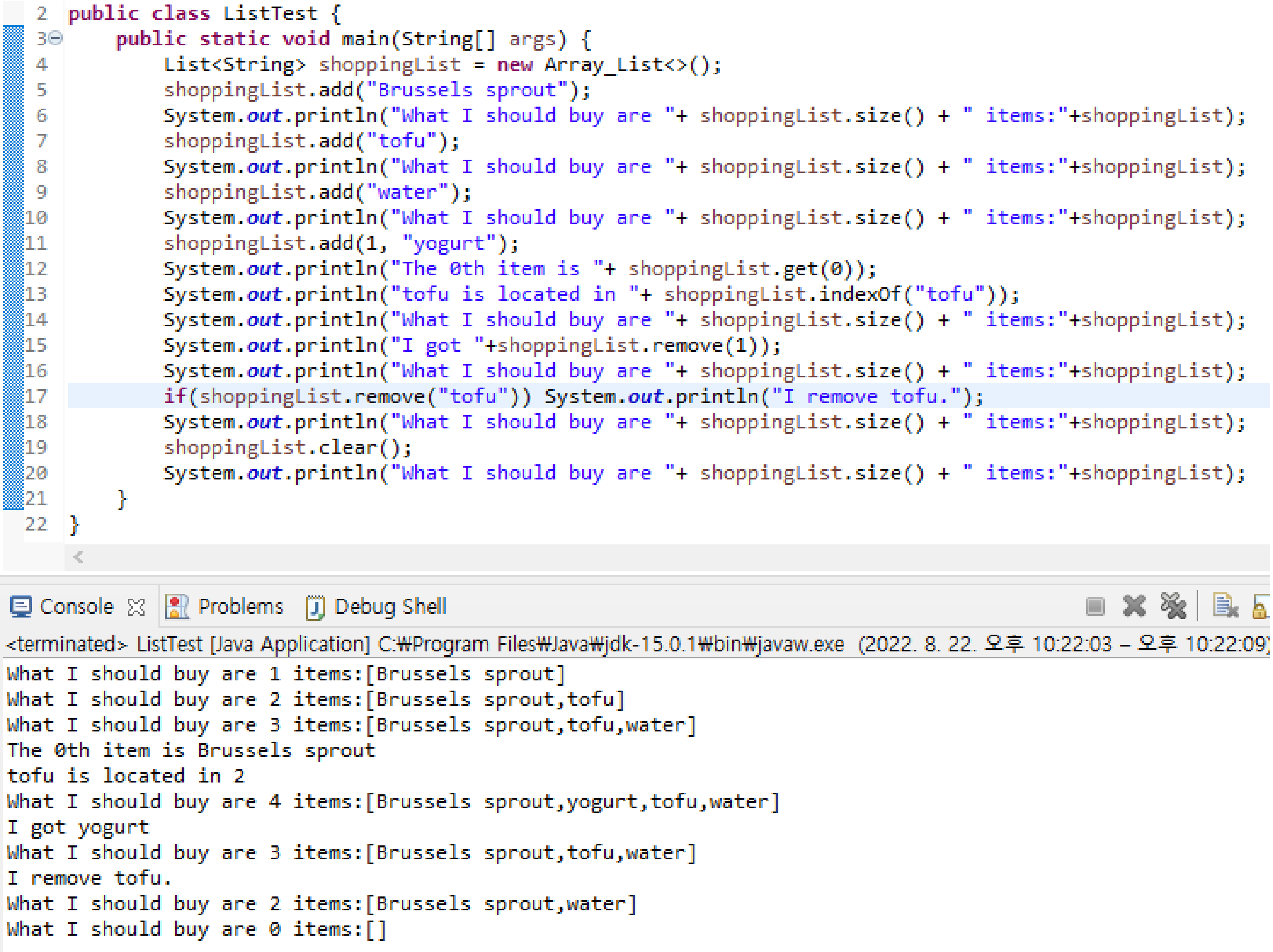
- indexOf(anEntry: T): integer
    - Gets the position of the first or only occurrence of anEntry.
    - Input: anEntry is the object to be found.
    - Return: the position of anEntry if it occurs in the list. Otherwise, returns the position where anEntry would occur in the list, but as a negative integer

ADT List

- clear(): void
  - Removes all entries from the list.
  - Postcondition: the list does not have any element.

- size(): integer
  - Gets the number of entries currently in the list.
  - Return: the number of entries currently in the list.

Example



```java
public class ListTest {
    public static void main(String[] args) {
        List<String> shoppingList = new Array_List<>();
        shoppingList.add("Brussels sprout");
        System.out.println("What I should buy are "+ shoppingList.size() + " items:"+shoppingList);
        shoppingList.add("tofu");
        System.out.println("What I should buy are "+ shoppingList.size() + " items:"+shoppingList);
        shoppingList.add("water");
        System.out.println("What I should buy are "+ shoppingList.size() + " items:"+shoppingList);
        shoppingList.add(1, "yogurt");
        System.out.println("The 0th item is "+ shoppingList.get(0));
        System.out.println("tofu is located in "+ shoppingList.indexOf("tofu"));
        System.out.println("What I should buy are "+ shoppingList.size() + " items:"+shoppingList);
        System.out.println("I got "+shoppingList.remove(1));
        System.out.println("What I should buy are "+ shoppingList.size() + " items:"+shoppingList);
        if(shoppingList.remove("tofu")) System.out.println("I remove tofu.");
        System.out.println("What I should buy are "+ shoppingList.size() + " items:"+shoppingList);
        shoppingList.clear();
        System.out.println("What I should buy are "+ shoppingList.size() + " items:"+shoppingList);
    }
}
```

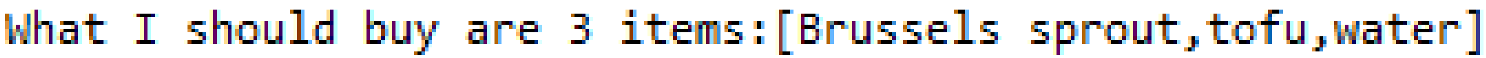
```
<terminated> ListTest [Java Application] C:₩Program Files₩Java₩jdk-15.0.1₩bin₩javaw.exe  (2022. 8. 22. 오후 10:22:03 – 오후 10:22:09)
What I should buy are 1 items:[Brussels sprout]
What I should buy are 2 items:[Brussels sprout,tofu]
What I should buy are 3 items:[Brussels sprout,tofu,water]
The 0th item is Brussels sprout
tofu is located in 2
What I should buy are 4 items:[Brussels sprout,yogurt,tofu,water]
I got yogurt
What I should buy are 3 items:[Brussels sprout,tofu,water]
I remove tofu.
What I should buy are 2 items:[Brussels sprout,water]
What I should buy are 0 items:[]
```

Data Structures . List

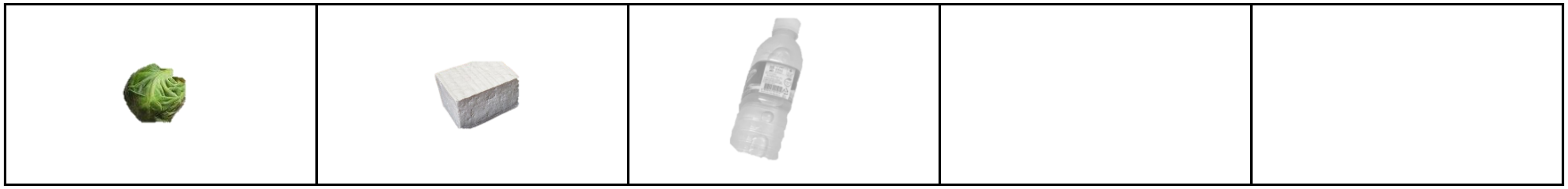
Array List

Concept

```
public class ListTest {
        public static void main(String[] args) {
                List<String> shoppingList = new Array_List<>(5);
                shoppingList.add("Brussels sprout");
                shoppingList.add("tofu");
                shoppingList.add("water");
                System.out.println("What I should buy are "+ shoppingList.size() + "
                items:"+shoppingList);
                shoppingList.add(1, "yogurt");
                System.out.println("The 0th item is "+ shoppingList.get(0));
                System.out.println("tofu is located in "+ shoppingList.indexOf("tofu"));
                System.out.println("What I should buy are "+ shoppingList.size() + " items:"+shoppingList);
                System.out.println("I got "+shoppingList.remove(1));
                if(shoppingList.remove("tofu")) System.out.println("I remove tofu.");
                shoppingList.clear();
                System.out.println("What I should buy are "+ shoppingList.size() + " items:"+shoppingList);
        }
}
```



```
What I should buy are 3 items:[Brussels sprout,tofu,water]
```

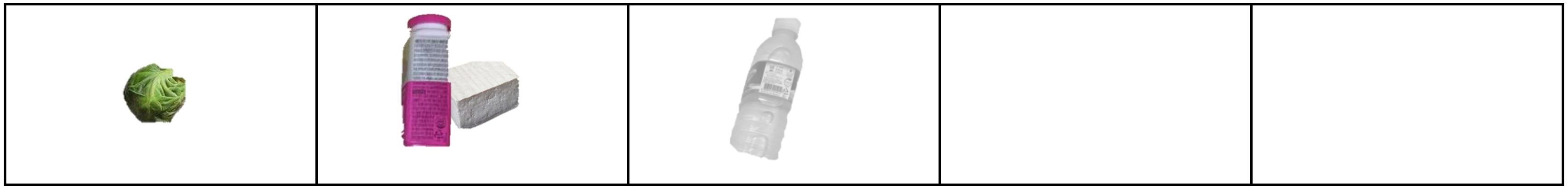


Concept

```
public class ListTest {
        public static void main(String[] args) {
                List<String> shoppingList = new Array_List<>(5);
                shoppingList.add(0, "Brussels sprout");
                shoppingList.add(1, "tofu");
                shoppingList.add(2, "water");
                System.out.println("What I should buy are "+ shoppingList.size() + " items:"+shoppingList);
                shoppingList.add(1, "yogurt");
                System.out.println("The 0th item is "+ shoppingList.get(0));
                System.out.println("tofu is located in "+ shoppingList.indexOf("tofu"));
                System.out.println("What I should buy are "+ shoppingList.size() + " items:"+shoppingList);
                System.out.println("I got "+shoppingList.remove(1));
                if(shoppingList.remove("tofu")) System.out.println("I remove tofu.");
                shoppingList.clear();
                System.out.println("What I should buy are "+ shoppingList.size() + " items:"+shoppingList);
        }
}
```

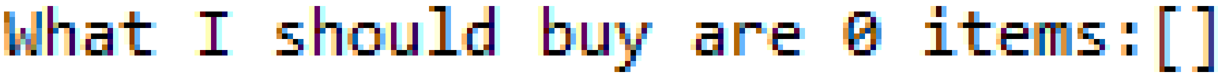
```
The 0th item is Brussels sprout
tofu is located in 2
What I should buy are 4 items:[Brussels sprout,yogurt,tofu,water]
```

```
I got yogurt
I remove tofu.
```

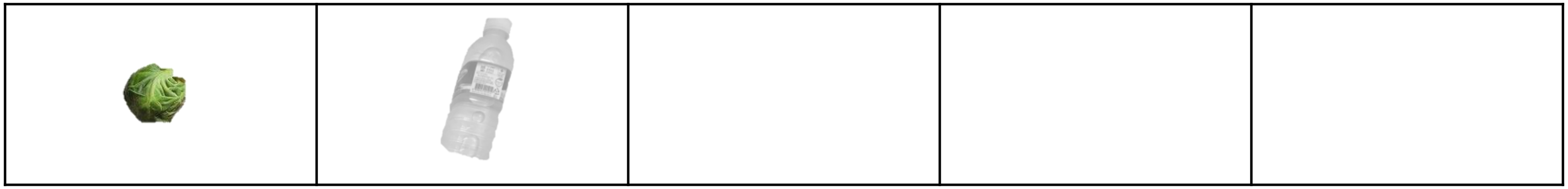


Concept

```
public class ListTest {
        public static void main(String[] args) {
                List<String> shoppingList = new Array_List<>(5);
                shoppingList.add(0, "Brussels sprout");
                shoppingList.add(1, "tofu");
                shoppingList.add(2, "water");
                System.out.println("What I should buy are "+ shoppingList.size() + " items:"+shoppingList);
                shoppingList.add(1, "yogurt");
                System.out.println("The 0th item is "+ shoppingList.get(0));
                System.out.println("tofu is located in "+ shoppingList.indexOf("tofu"));
                System.out.println("What I should buy are "+ shoppingList.size() + " items:"+shoppingList);
                System.out.println("I got "+shoppingList.remove(1));
                if(shoppingList.remove("tofu")) System.out.println("I remove tofu.");
                shoppingList.clear();
                System.out.println("What I should buy are "+ shoppingList.size() + " items:"+shoppingList);
        }
}
```



```
What I should buy are 0 items:[]
```



Design Issue 1

- How to declare data field
- Array_List will hold a collection of objects

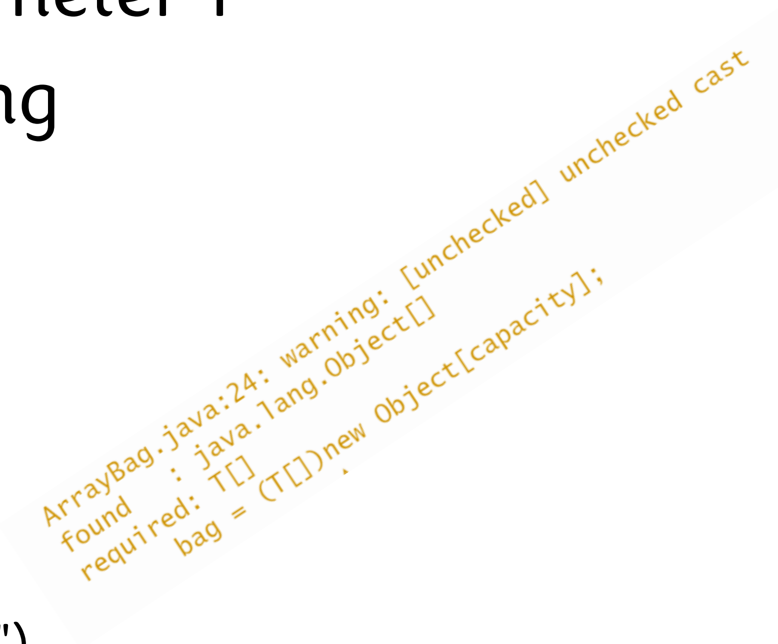
  one field can be an array of these object
- Capacity:
  - the length of array vs. the number of element
  - user defined vs. default number

```
private final T list[];
private int numberOfEntries;
private static final int DEFAULT_CAPACITY=25;
```

Design Issue 2

- initializing the array of parameter T
- What is the current initializing
  - Syntax error
    - list = new T[capacity]
    - list = new Object[capacity]
  - Missing type checking
    - list = (T[])new Object[capacity];
  - Suggestion

    @SuppressWarnings("unchecked")
    T[] tempList = (T[])new Object[desiredCapacity];
    list = tempList;



```
ArrayBag.java:24: warning: [unchecked] unchecked cast
found   : java.lang.Object[]
required: T[]
      bag = (T[])new Object[capacity];
                 ^
```

Suppressing compiler warnings

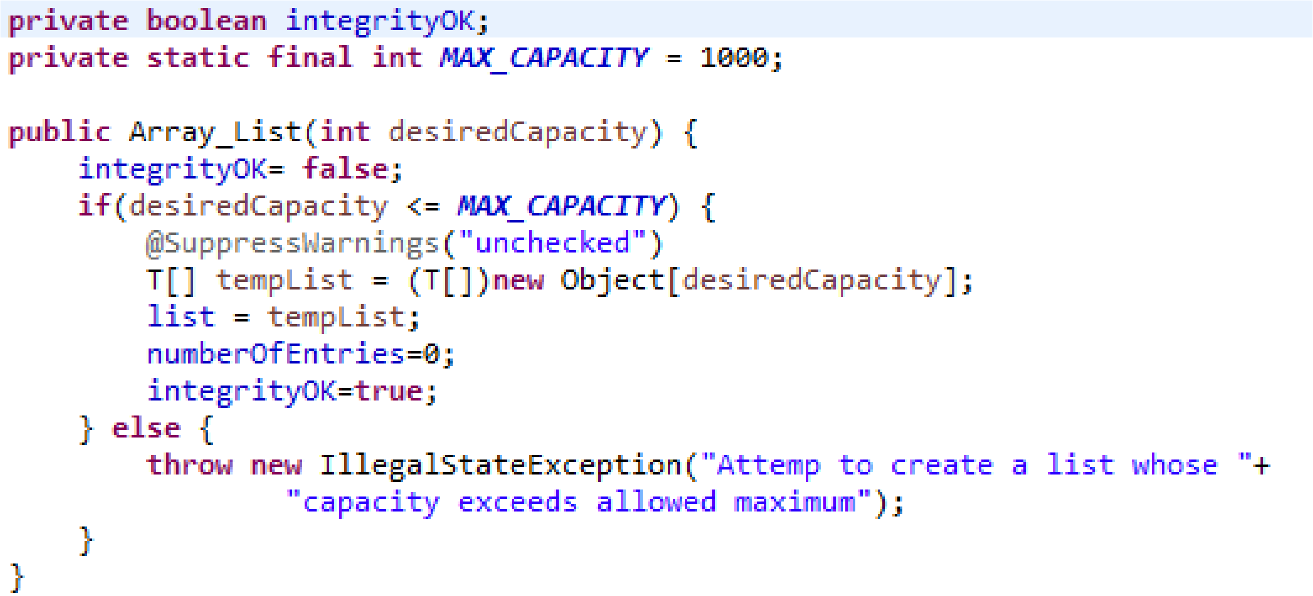
To suppress an unchecked-cast warning from the compiler, you precede the flagged statements with the instruction

    @SuppressWarnings("unchecked")

Note that this instruction can precede only a method definition or a variable declaration.

Design Issue 3

- Fail-safe programming:
  - What happens if a client tries to create a List whose capacity exceeds a given limit?
  - What happens if a constructor does not execute completely?



```
private boolean integrityOK;
private static final int MAX_CAPACITY = 1000;

public Array_List(int desiredCapacity) {
    integrityOK= false;
    if(desiredCapacity <= MAX_CAPACITY) {
        @SuppressWarnings("unchecked")
        T[] tempList = (T[])new Object[desiredCapacity];
        list = tempList;
        numberOfEntries=0;
        integrityOK=true;
    } else {
        throw new IllegalStateException("Attemp to create a list whose "+
                "capacity exceeds allowed maximum");
    }
}
```

Design Issue 4

```
public T[] toArray(){
   return list;
}
```

```
public T[] toArray() {
   checkIntegrity();
   @SuppressWarnings("unchecked")
   T[] result = (T[])new Object[numberOfEntries];
   for(int i = 0 ; i < numberOfEntries ; i++)
       result[i] = list[i];
    return result;
}
```

A private data field should be changed only by the method.
However, if a client know the reference of a private data field, it can change freely without method of the ADT.

Ex:

```
list=shoppingList.toArray();
list[0] = null;
```

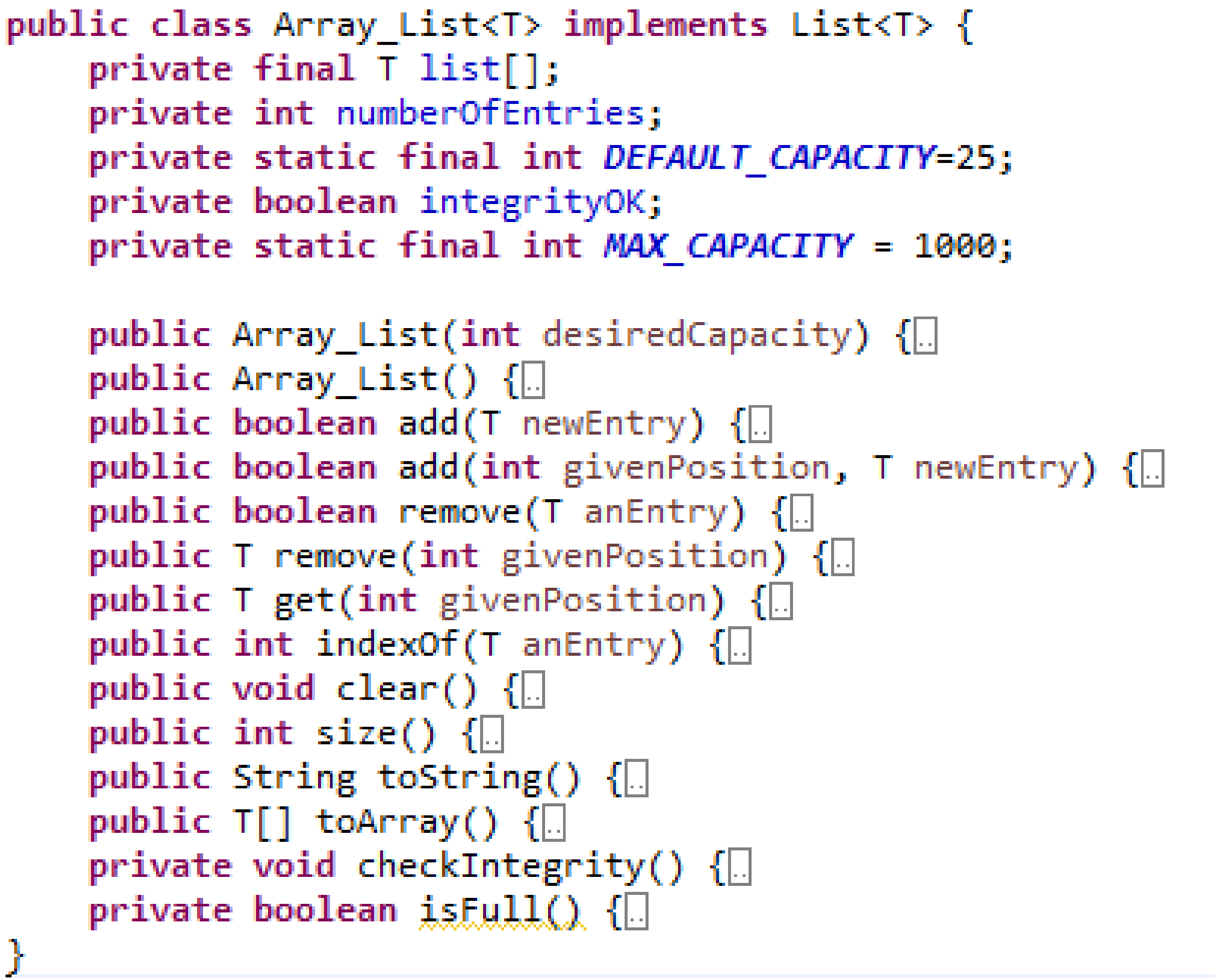
Design Issue 5

Some code may be duplicated.

1. Integrity checking
2. Finding a certain entry:
   indexOf (T anEntry)
   remove(T anEntry)
3. Remove an item in a certain location
   remove(int givenPosition): boolean
   remove(T anEntry): anEntry
   clear()

- checkIntegrity() : void
- indexOf (T anEntry)
- removeEntry(int index): T

Array_List



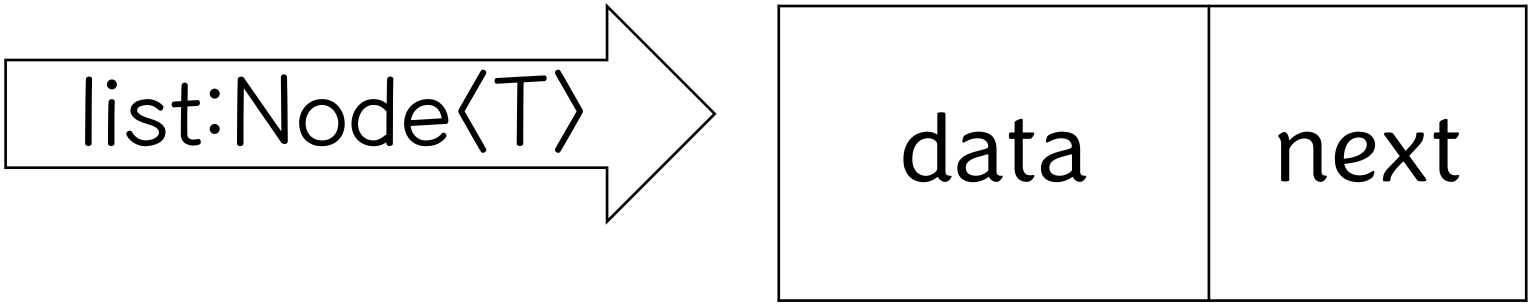
```java
public class Array_List<T> implements List<T> {
    private final T list[];
    private int numberOfEntries;
    private static final int DEFAULT_CAPACITY=25;
    private boolean integrityOK;
    private static final int MAX_CAPACITY = 1000;

    public Array_List(int desiredCapacity) {[..]
    public Array_List() {[..]
    public boolean add(T newEntry) {[..]
    public boolean add(int givenPosition, T newEntry) {[..]
    public boolean remove(T anEntry) {[..]
    public T remove(int givenPosition) {[..]
    public T get(int givenPosition) {[..]
    public int indexOf(T anEntry) {[..]
    public void clear() {[..]
    public int size() {[..]
    public String toString() {[..]
    public T[] toArray() {[..]
    private void checkIntegrity() {[..]
    private boolean isFull() {[..]
}
```

Data Structures . List

Linked List

Analogy-Train



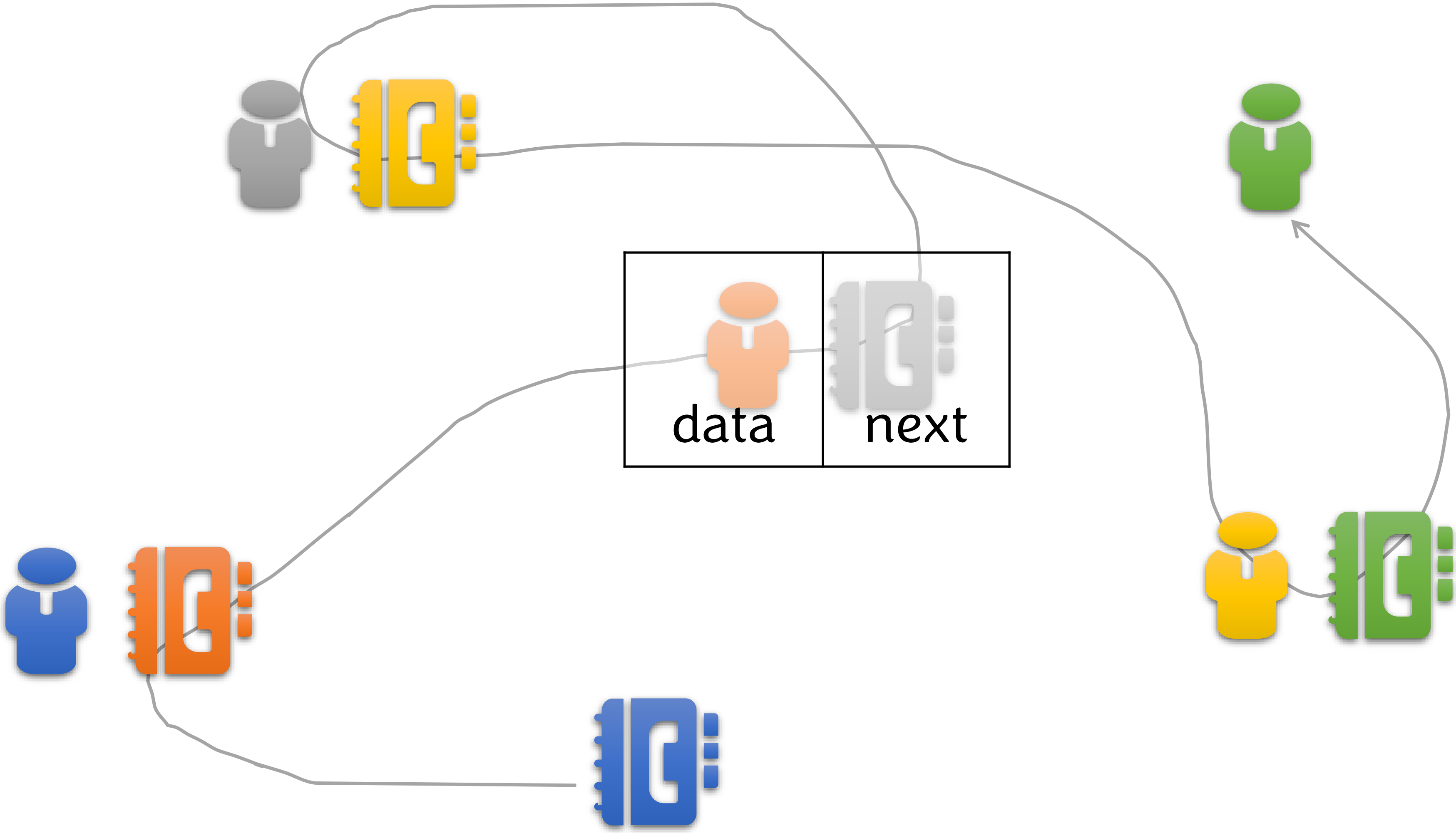
list:Node<T>

data

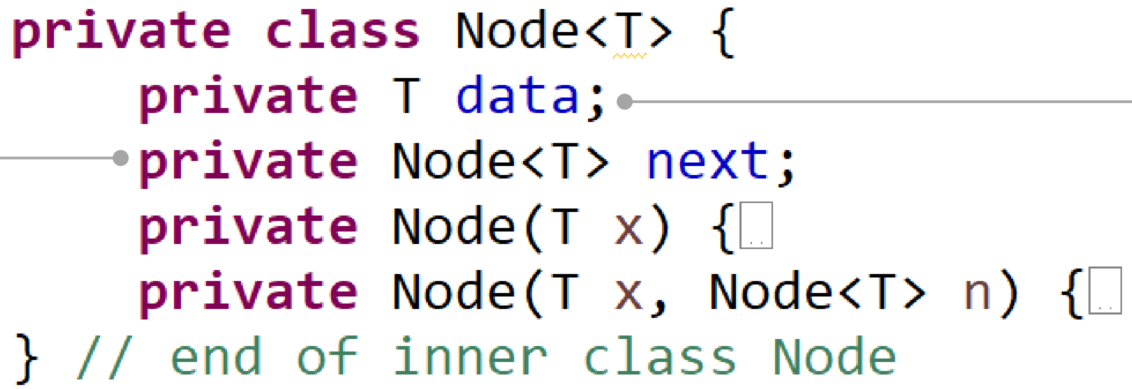
next

Analogy

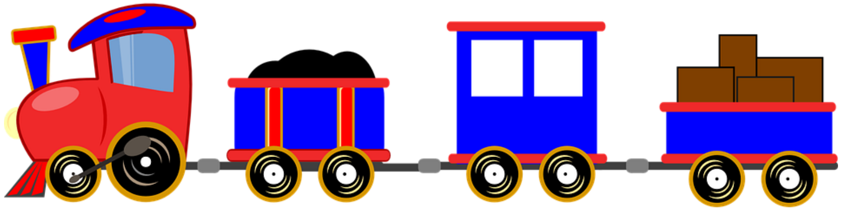
- Network of Emergency Contacts



Node

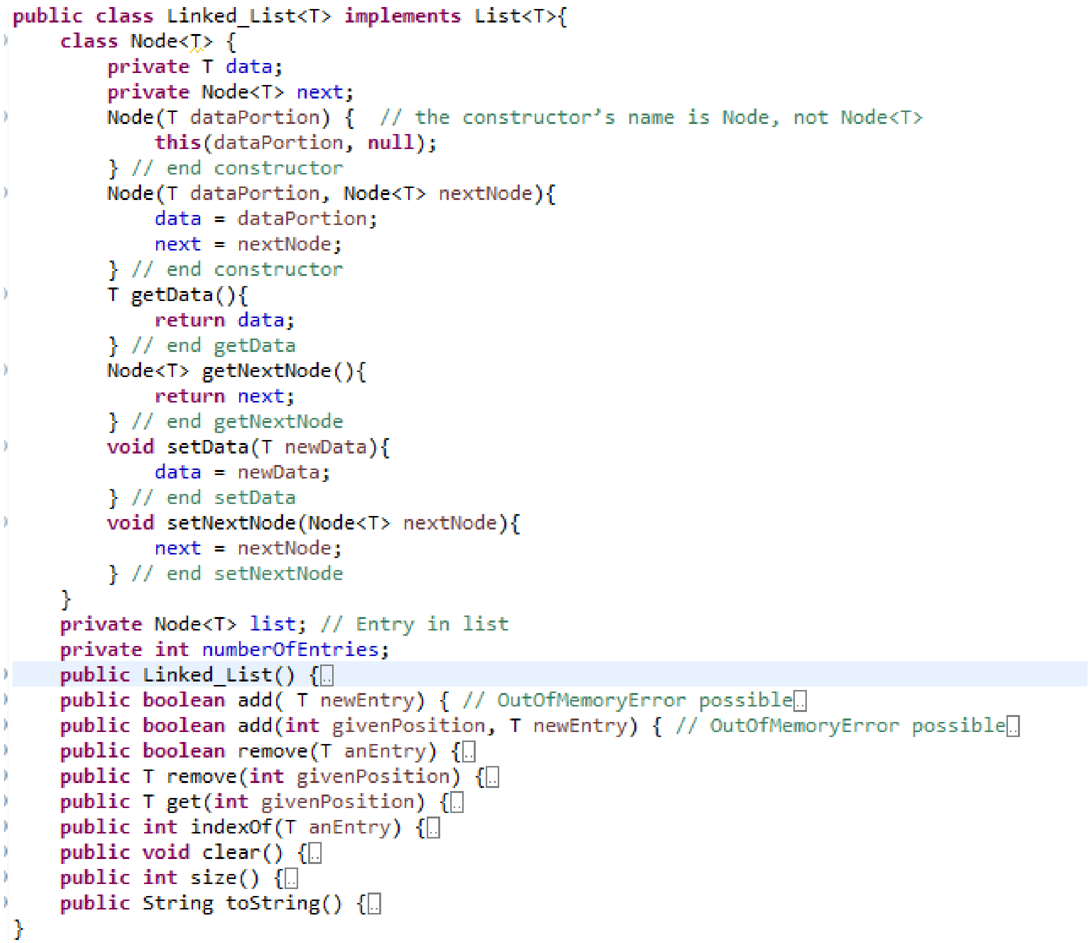


```
private class Node<T> {
      private T data;
      private Node<T> next;
      private Node(T x) {□
      private Node(T x, Node<T> n) {□
} // end of inner class Node
```



Node

- Nested class:
  - defined in another class definition
- Inner class:
  - a nested class that is not static
- Outer class = enclosing class:
  - embeds a nested class
- Top-level class = outermost class:
  - outer class that is not a nest class



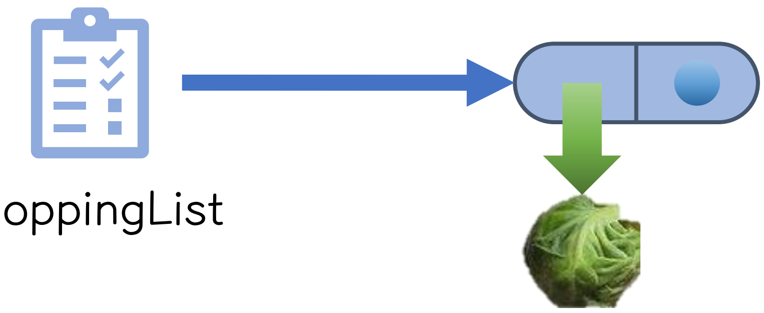
```
public class Linked_List<T> implements List<T>{
    class Node<T> {
        private T data;
        private Node<T> next;
        Node(T dataPortion) {  // the constructor’s name is Node, not Node<T>
            this(dataPortion, null);
        } // end constructor
        Node(T dataPortion, Node<T> nextNode){
            data = dataPortion;
            next = nextNode;
        } // end constructor
        T getData(){
            return data;
        } // end getData
        Node<T> getNextNode(){
            return next;
        } // end getNextNode
        void setData(T newData){
            data = newData;
        } // end setData
        void setNextNode(Node<T> nextNode){
            next = nextNode;
        } // end setNextNode
    }
    private Node<T> list; // Entry in list
    private int numberOfEntries;
    public Linked_List() {□
    public boolean add( T newEntry) { // OutOfMemoryError possible□
    public boolean add(int givenPosition, T newEntry) { // OutOfMemoryError possible□
    public boolean remove(T anEntry) {□
    public T remove(int givenPosition) {□
    public T get(int givenPosition) {□
    public int indexOf(T anEntry) {□
    public void clear() {□
    public int size() {□
    public String toString() {□
}
```

Concept: add

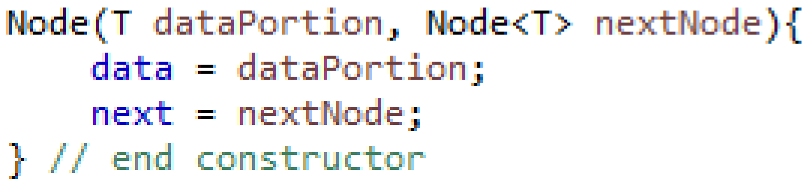
- Add

```
List<String> shoppingList = new Linked_List<>();
shoppingList.add("Brussels sprout");
```

Increase number of entries
if the list is empty
    shoppingList points the new node



numberOfEntries=0



```
Node(T dataPortion, Node<T> nextNode){
    data = dataPortion;
    next = nextNode;
} // end constructor
```

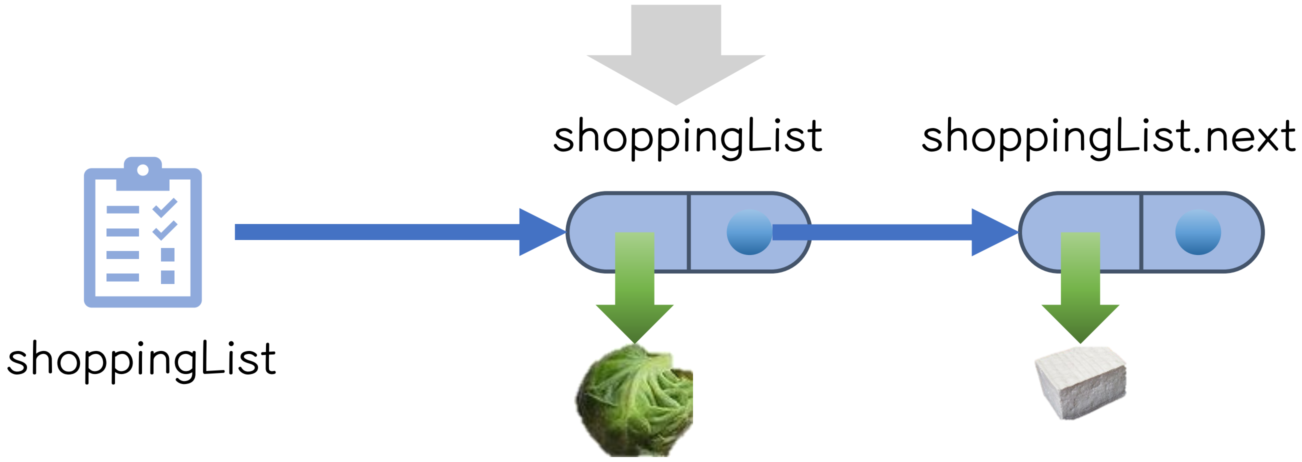
Concept: add

- Add

```
shoppingList.add("tofu");
```

```
if the list is not empty
    find the last node
    the last node points the new node
```

How can we know the last node?



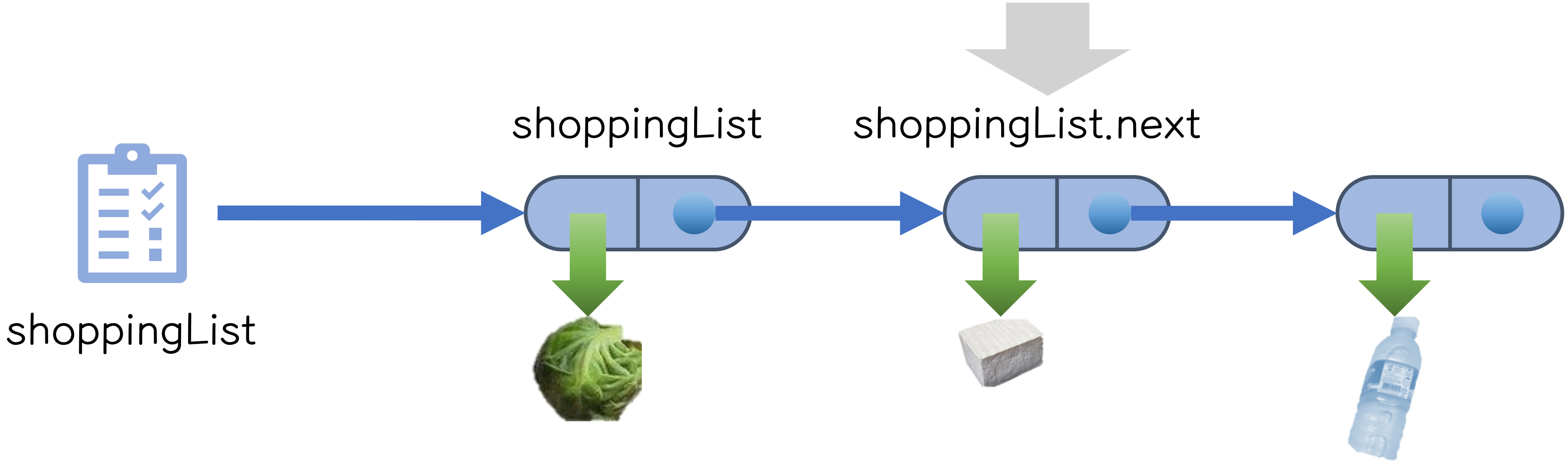
numberOfEntries=1

Concept: add

- Add

```
shoppingList.add(“water");
```

```
if the list is not empty
    find the last node
    the last node points the new node
```



shoppingList

shoppingList.next

shoppingList

numberOfEntries

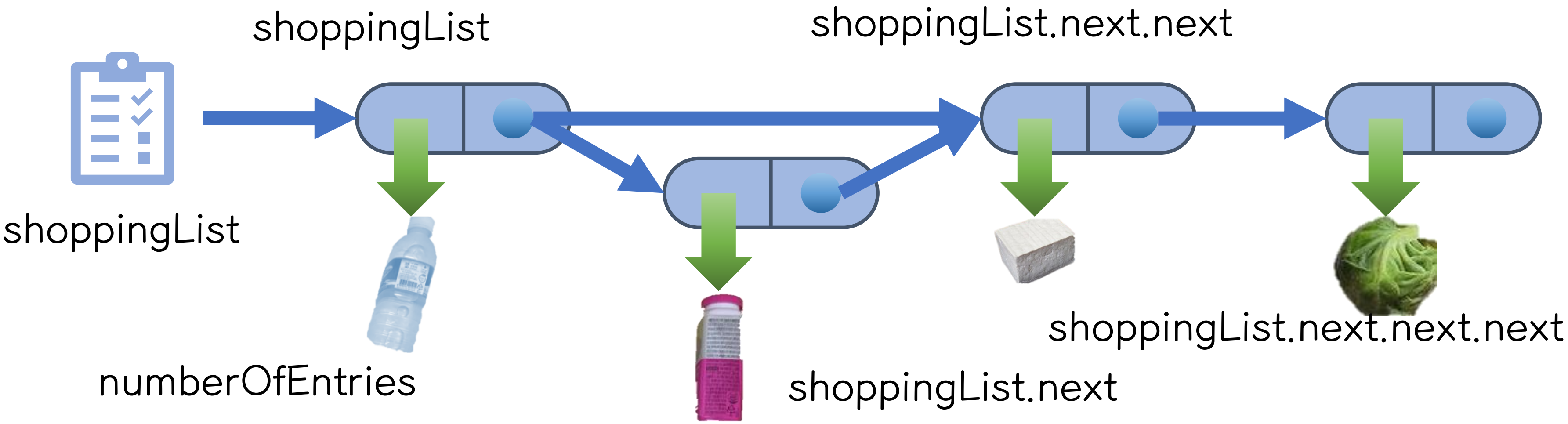
Concept: add

- Add

```
shoppingList.add(1, “yogurt");
```

Q: How can we find the location with the given index?

A: temporary point

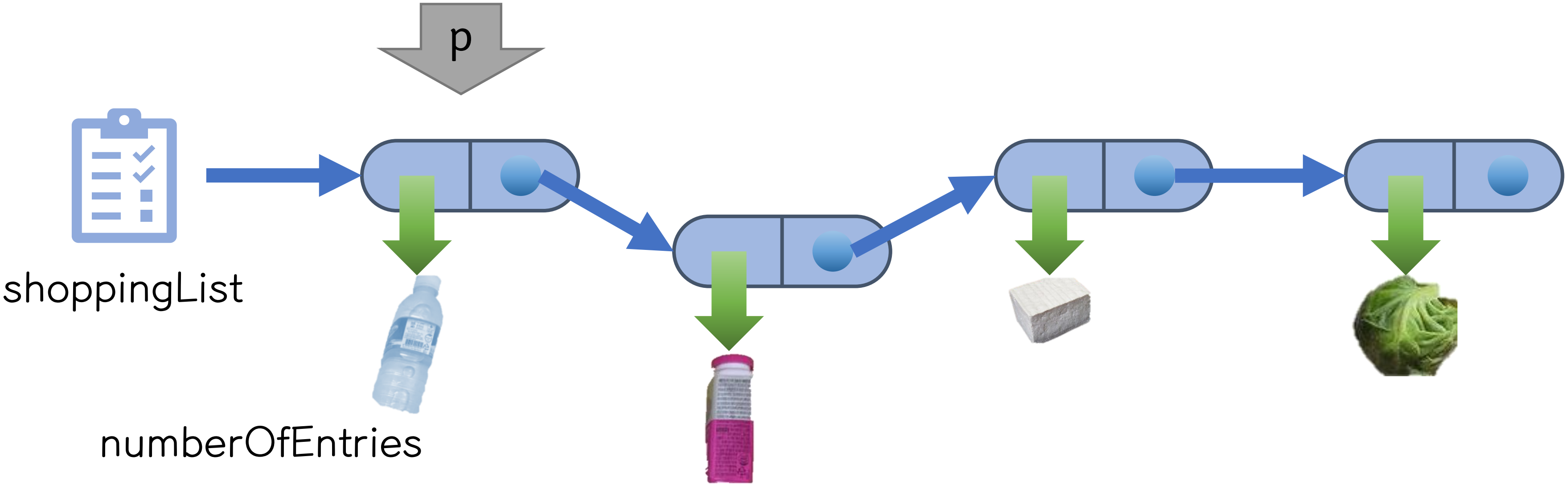


Current point p

- Counting with integer variable: i
- Pointing the current node: p

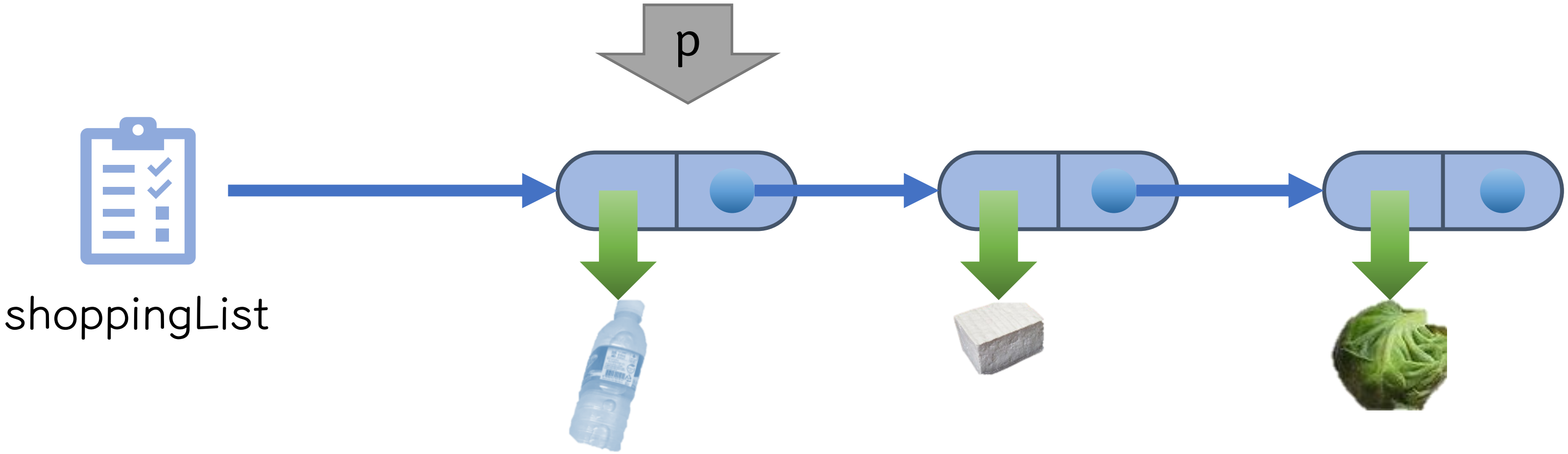
i = 0

1



size, toString

```
System.out.println(
        "What I should buy are “
        + shoppingList.size() + " items:"+shoppingList);
```



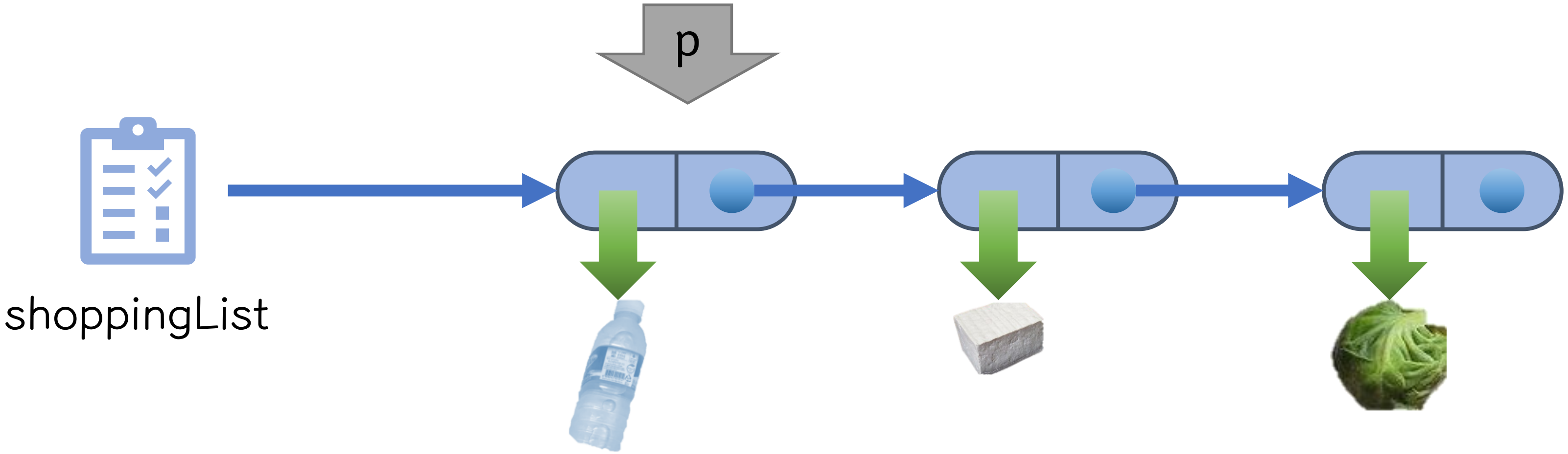
p

shoppingList

numberOfEntries=3

indexOf

```
System.out.println(
        "tofu is located in "+ shoppingList.indexOf("tofu"));
Index= 1
```



p

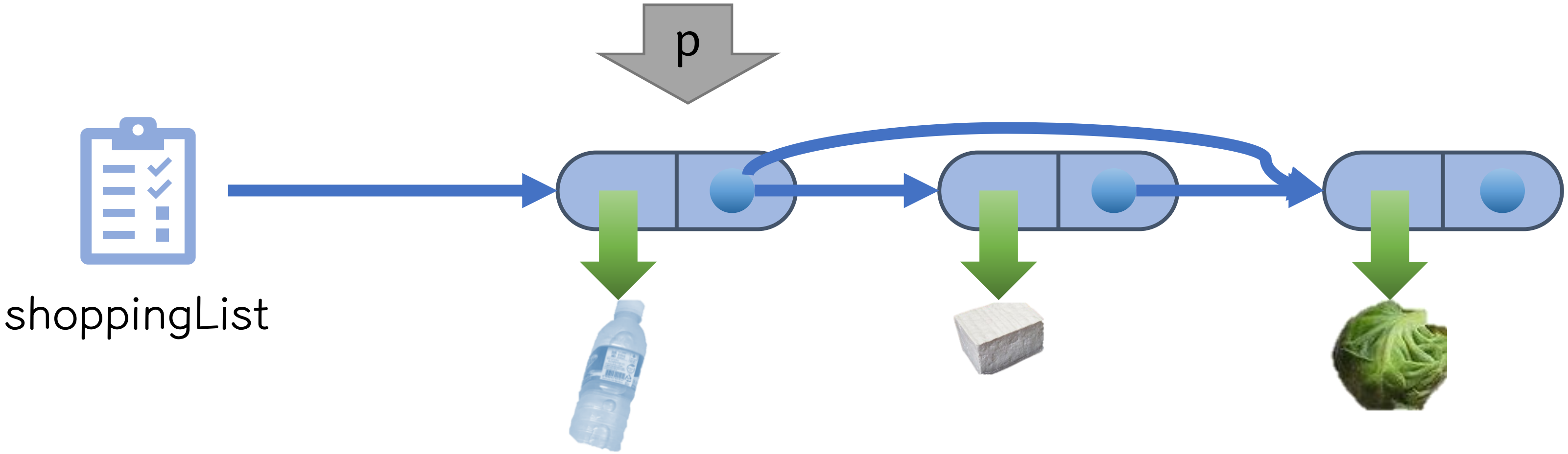
shoppingList

numberOfEntries=3

Remove with index

```
System.out.println("I got "+shoppingList.remove(1));
```

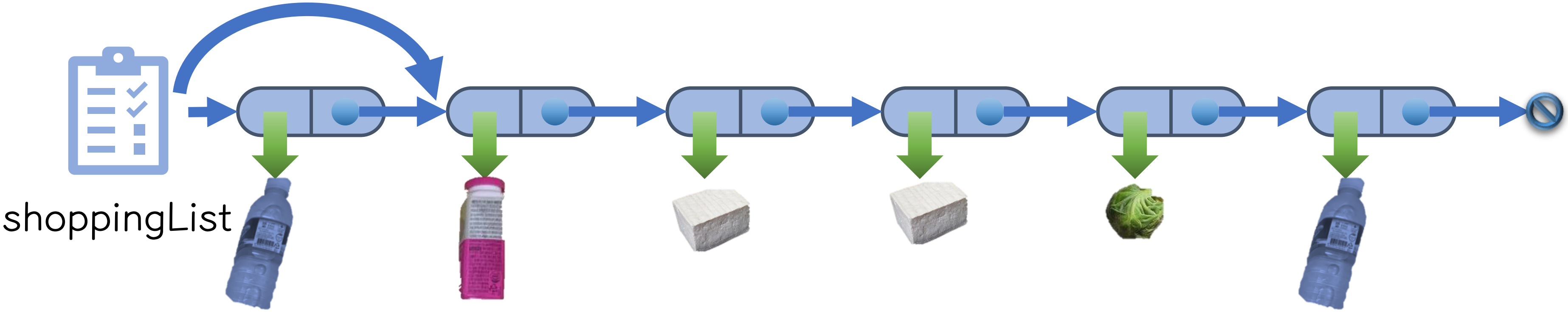
How to know who's next is me?



numberOfEntries= 2

Removing case

- Removing a givenPosition
  - Case 1: the list is empty-return false
  - Case 2: index 0

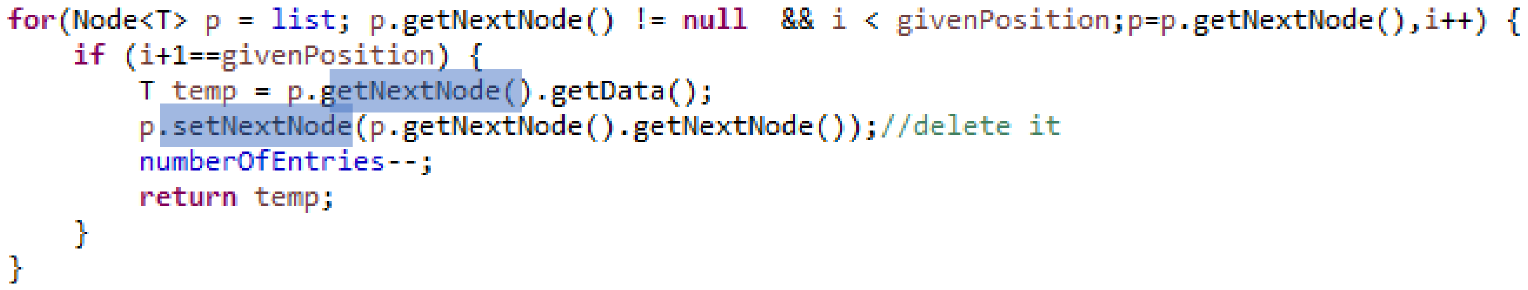


  - Case 3: index is bigger than the size
  - Case 4: other case

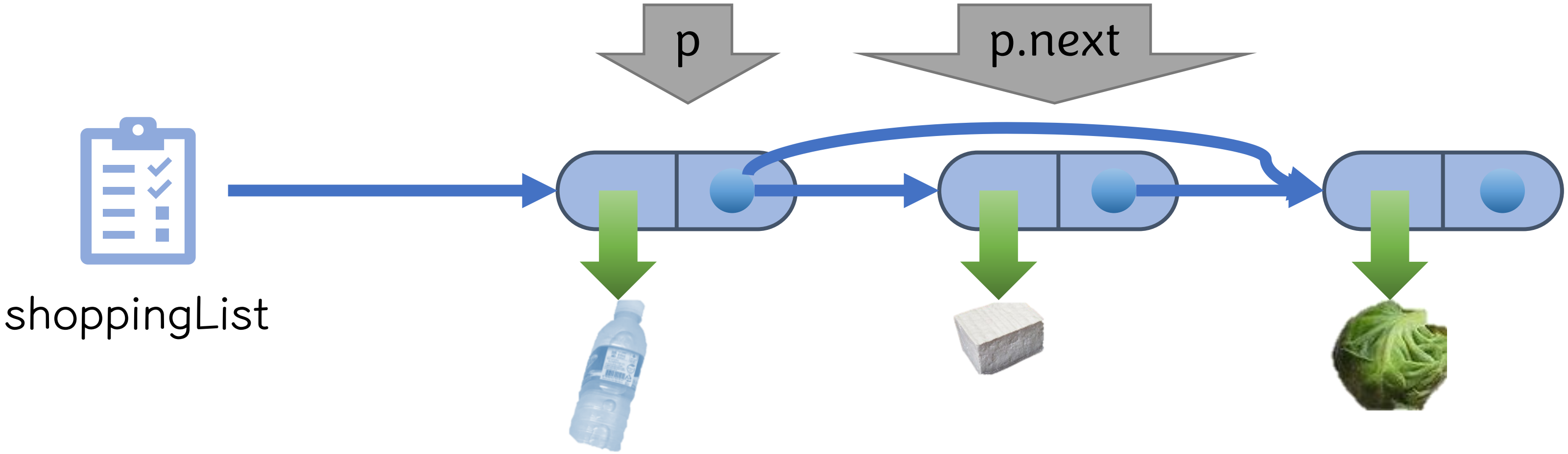
Removing

- Case 4

check the data of p.next not p



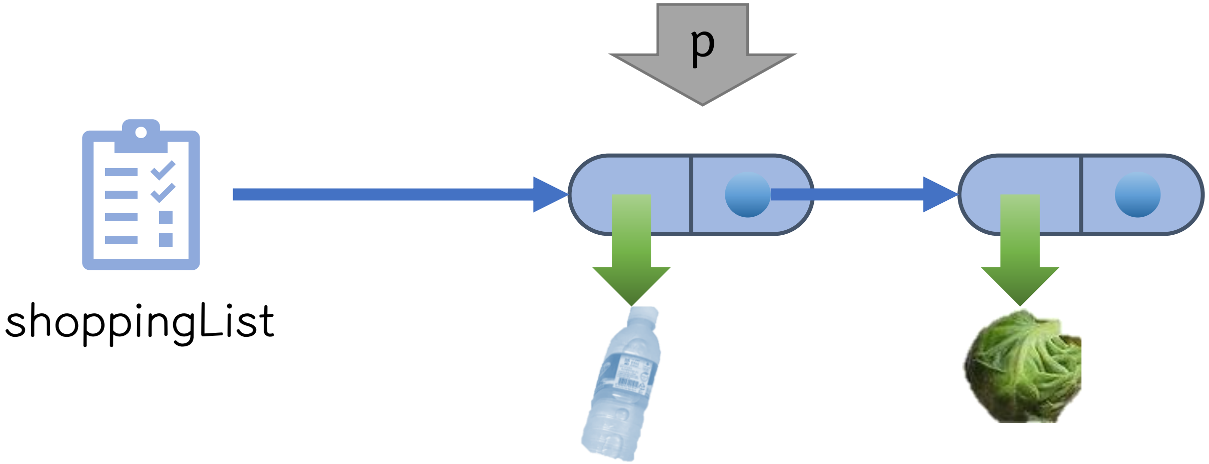
```
for(Node<T> p = list; p.getNextNode() != null  && i < givenPosition;p=p.getNextNode(),i++) {
    if (i+1==givenPosition) {
        T temp = p.getNextNode().getData();
        p.setNextNode(p.getNextNode().getNextNode());//delete it
        numberOfEntries--;
        return temp;
    }
}
```



numberOfEntries= 2

Remove with data

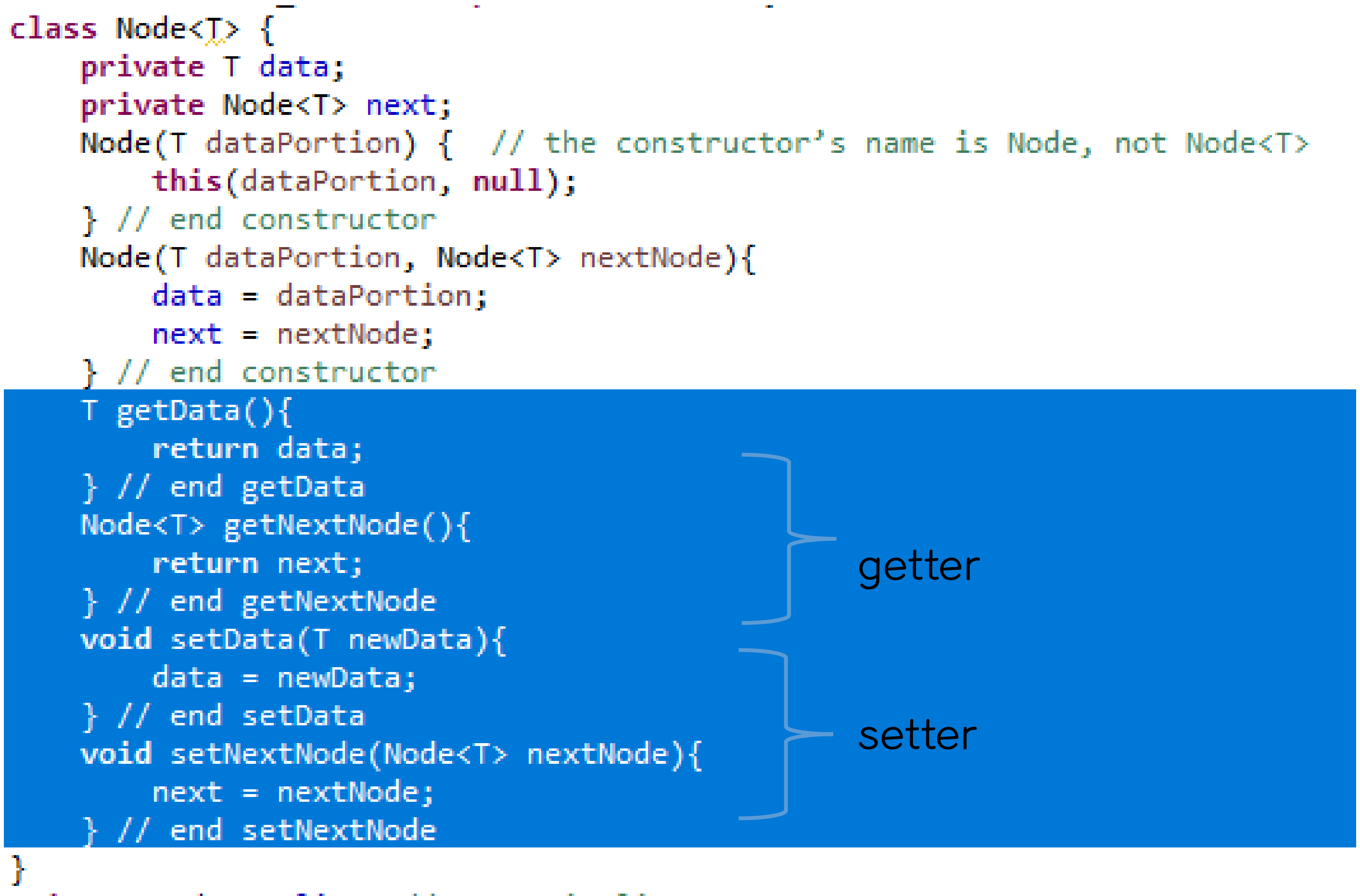
```
if(shoppingList.remove("tofu"))
        System.out.println("I remove tofu.");
```



numberOfEntries=2

Design issue

- Inner class vs package
  - We need getter and setter methods if you define Node outside of Linked_List.

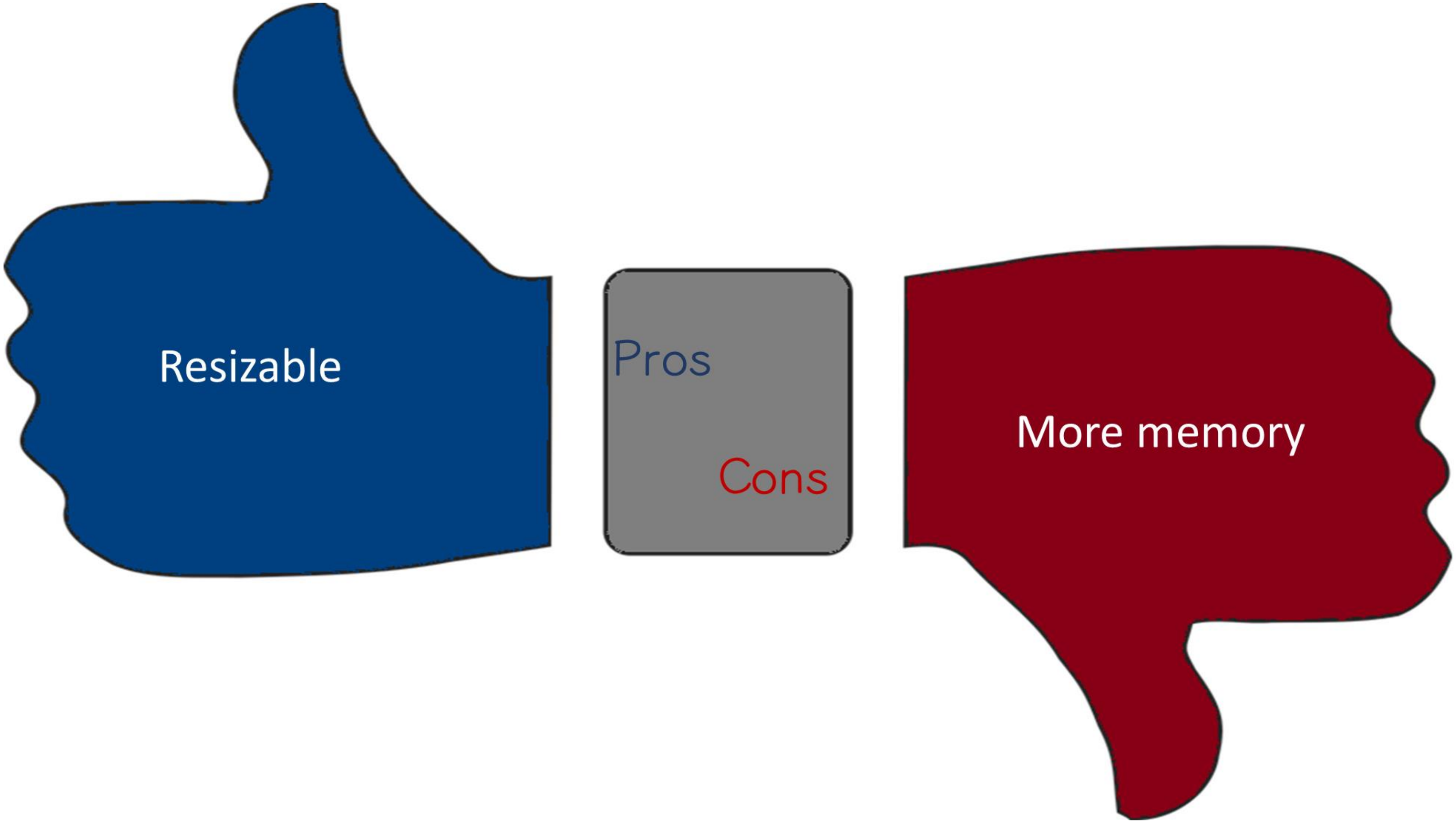


```
class Node<T> {
    private T data;
    private Node<T> next;
    Node(T dataPortion) {  // the constructor's name is Node, not Node<T>
        this(dataPortion, null);
    } // end constructor
    Node(T dataPortion, Node<T> nextNode){
        data = dataPortion;
        next = nextNode;
    } // end constructor
    T getData(){
        return data;
    } // end getData
    Node<T> getNextNode(){
        return next;
    } // end getNextNode
    void setData(T newData){
        data = newData;
    } // end setData
    void setNextNode(Node<T> nextNode){
        next = nextNode;
    } // end setNextNode
}
```

getter

setter

Comparing to Array_List



Resizable

Pros

Cons

More memory

Time complexity

ADT operations	Array	Linked
add	O(n)	O(n)
remove	O(n)	O(n)
indexOf	O(n)	O(n)
clear	O(1)	O(1)
size, toArray	O(n)	O(n)

Data Structures . List

Variations

Agenda

- Sorted List
- Singly linked list
- Doubly linked list
- Circular linked list
  - The last node points to the first node instead of null.

ADT List

Sorted List

Responsibilities

add: Adds newEntry to the sorted list so that the list remains sorted

remove: delete element x

remove: delete i^{th} element

get: let the caller know what element is the i^{th} element

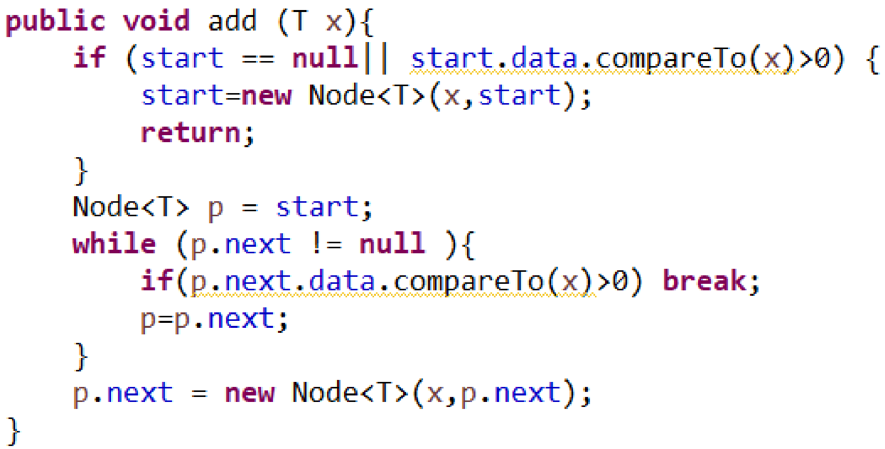
indexOf: let the caller know what is the i^{th} element

clear: remove all entries form the list

size: gets the number of entries currently in the list.

add

- precondition: the list is in ascending order;
- postcondition: the list is in ascending order, and it contains x
- Cases
  - Case 1: empty sorted list
  - Case 2: insertion at the front of the list
  - Case 3: attach after the last node
  - Case 4: between two nodes

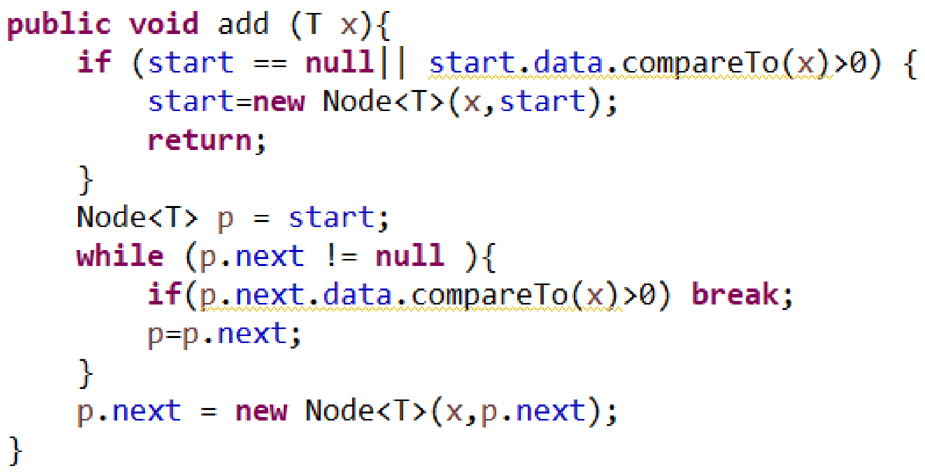


```
public void add (T x){
    if (start == null|| start.data.compareTo(x)>0) {
        start=new Node<T>(x,start);
        return;
    }
    Node<T> p = start;
    while (p.next != null ){
        if(p.next.data.compareTo(x)>0) break;
        p=p.next;
    }
    p.next = new Node<T>(x,p.next);
}
```

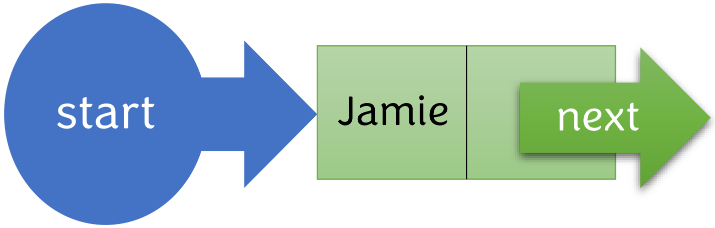
Case 1: empty sorted list

- Client program:

    list.add("Jamie");



```
public void add (T x){
    if (start == null|| start.data.compareTo(x)>0) {
        start=new Node<T>(x,start);
        return;
    }
    Node<T> p = start;
    while (p.next != null ){
        if(p.next.data.compareTo(x)>0) break;
        p=p.next;
    }
    p.next = new Node<T>(x,p.next);
}
```

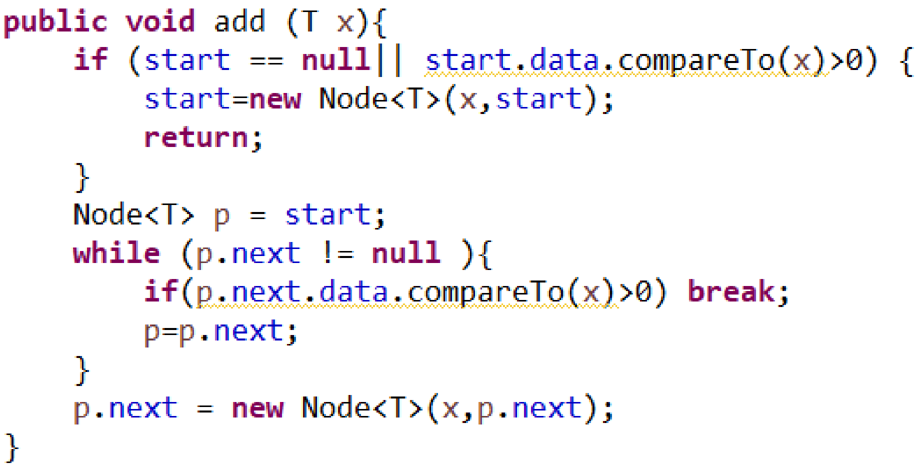


Case 2: insertion at the front of the list

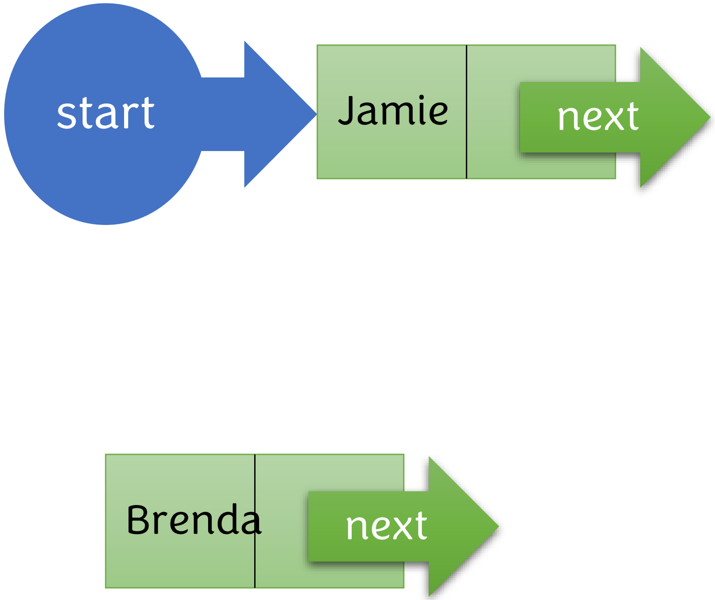
- Client program:

  list.add(“Jamie”);

  list.add(“Brenda”);

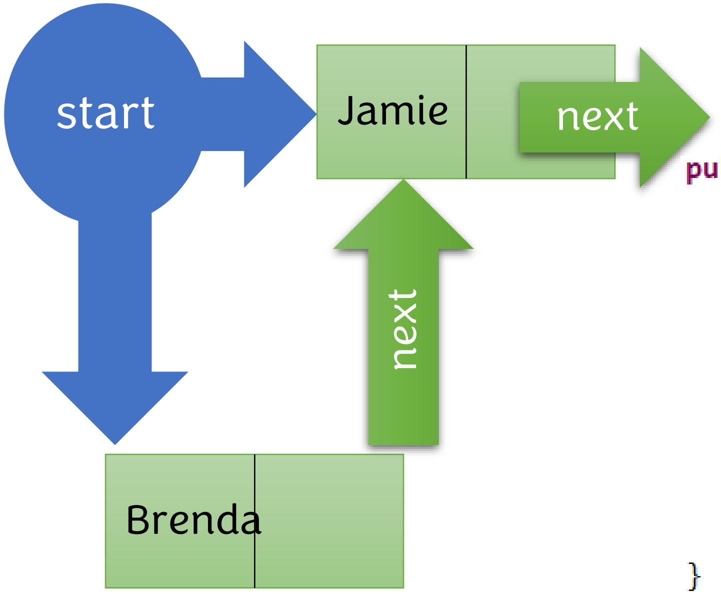


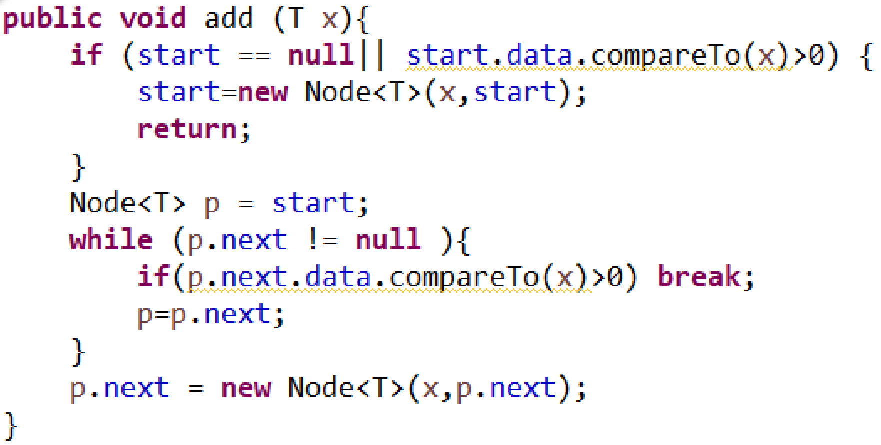
```
public void add (T x){
    if (start == null|| start.data.compareTo(x)>0) {
        start=new Node<T>(x,start);
        return;
    }
    Node<T> p = start;
    while (p.next != null ){
        if(p.next.data.compareTo(x)>0) break;
        p=p.next;
    }
    p.next = new Node<T>(x,p.next);
}
```



Case 2: insertion at the front of the list

- Case 2: insertion at the front of the list
  - the next of the new node to start.next
  - start.next points the new node





```
public void add (T x){
    if (start == null|| start.data.compareTo(x)>0) {
        start=new Node<T>(x,start);
        return;
    }
    Node<T> p = start;
    while (p.next != null ){
        if(p.next.data.compareTo(x)>0) break;
        p=p.next;
    }
    p.next = new Node<T>(x,p.next);
}
```

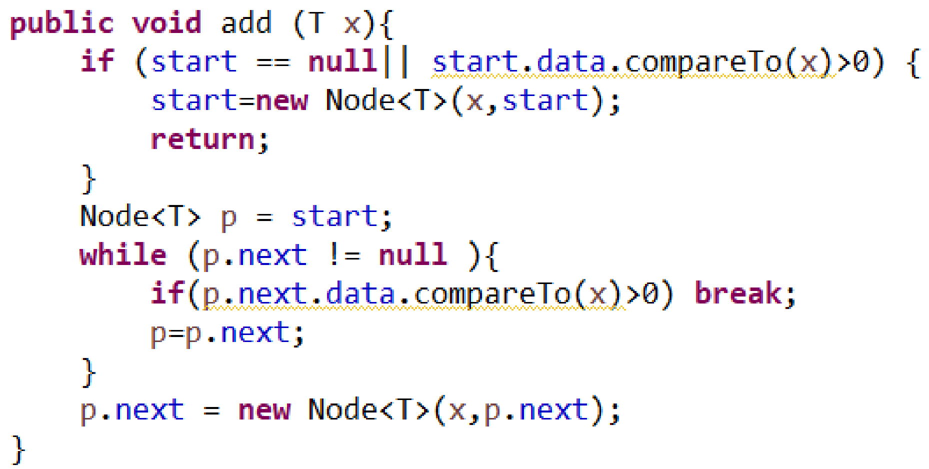
Case 3: attach after the last node

- Client program:

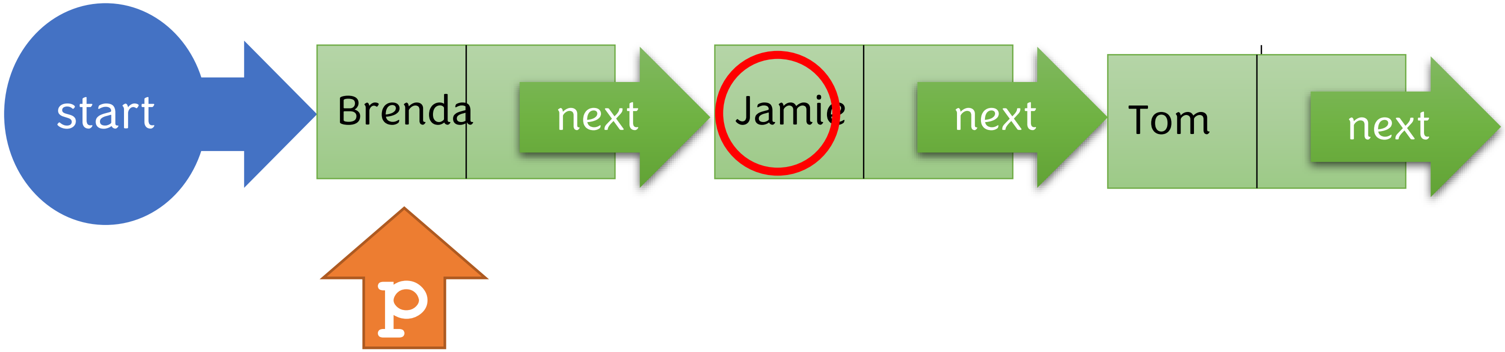
  list.add(“Jamie”);

  list.add(“Brenda”);

  list.add("Tom");



```
public void add (T x){
    if (start == null|| start.data.compareTo(x)>0) {
        start=new Node<T>(x,start);
        return;
    }
    Node<T> p = start;
    while (p.next != null ){
        if(p.next.data.compareTo(x)>0) break;
        p=p.next;
    }
    p.next = new Node<T>(x,p.next);
}
```



Case 4: between two nodes

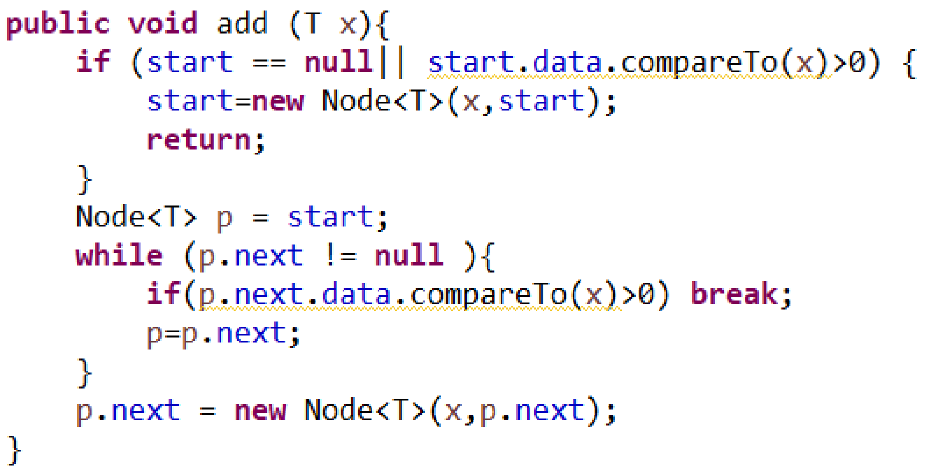
- Client program:

  list.add(“Jamie”);

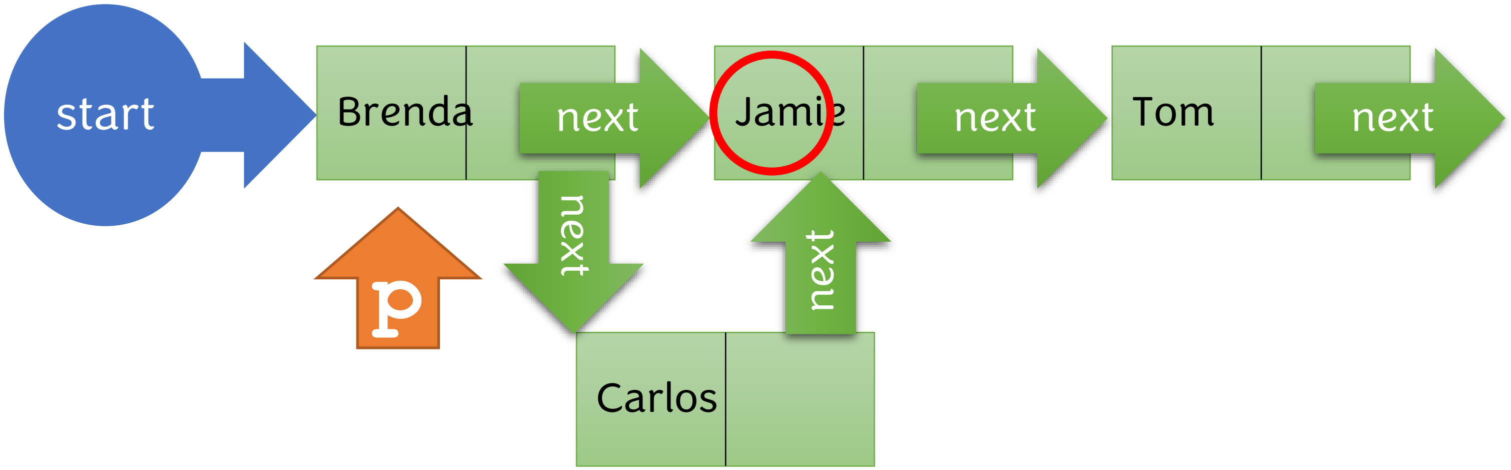
  list.add(“Brenda”)

  list.add("Tom");

  list.add("Carlos");

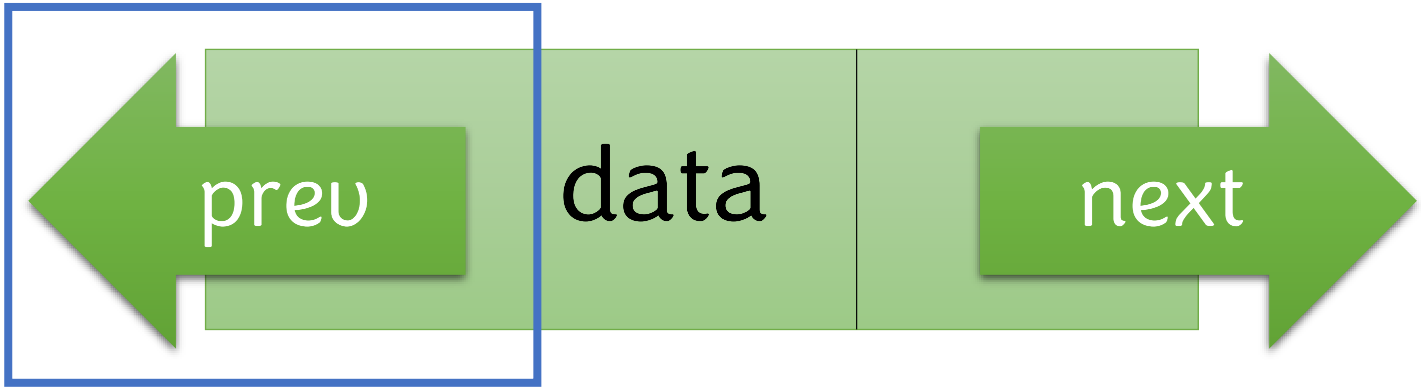


```
public void add (T x){
    if (start == null|| start.data.compareTo(x)>0) {
        start=new Node<T>(x,start);
        return;
    }
    Node<T> p = start;
    while (p.next != null ){
        if(p.next.data.compareTo(x)>0) break;
        p=p.next;
    }
    p.next = new Node<T>(x,p.next);
}
```



doubly linked list

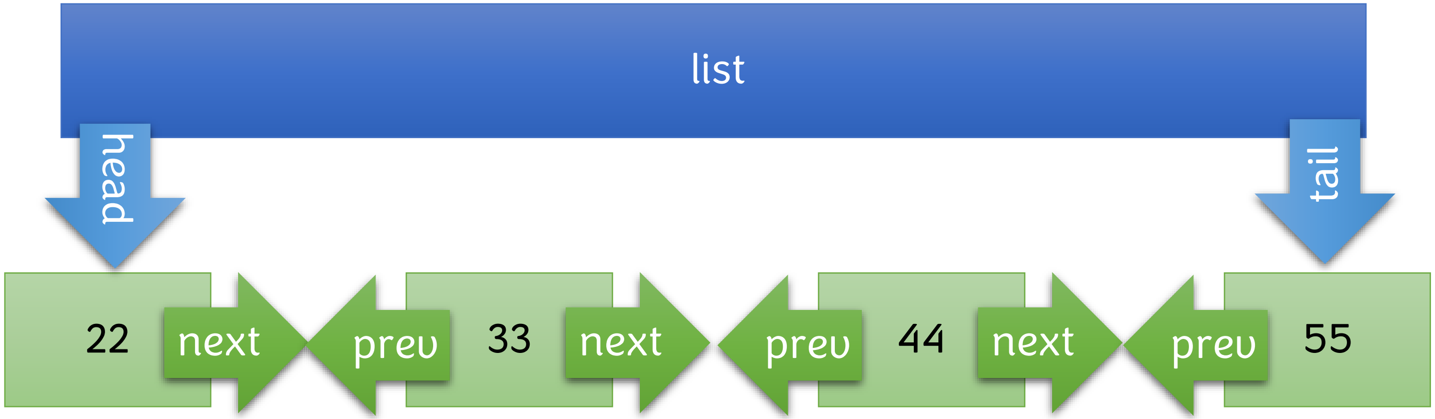
- A node of a doubly linked list
- a node = data + previous point + next point



```
Class Node<T> {
    T data; // for data field
    Node prev; // for the previous node
    Node next; // for the next node
}
```

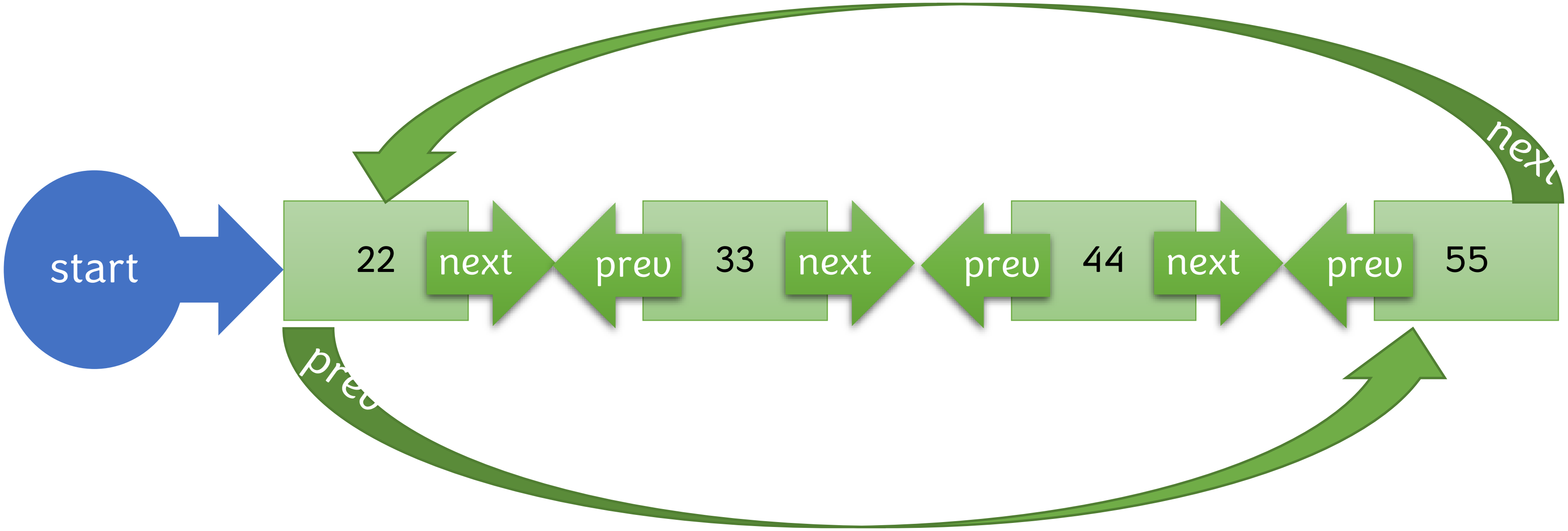
doubly linked list

- Structure of a doubly linked list
- The first and last nodes of a doubly linked list are immediately accessible
- Advantage
  - Allows traversal of nodes in both direction
  - Deque can be implemented using a doubly linked list easily



Circular Linked List

- The last node points to the first node instead of null.
- It can be a singly linked list or a doubly linked list



Thank you!