CSE 143
Lecture 11

More Linked Lists

reading: 16.2 - 16.3

slides created by Marty Stepp
http://www.cs.washington.edu/143/

Conceptual questions

- What is the difference between a `LinkedIntList` and a `ListNode`?

- What is the difference between an empty list and a `null` list?
  - How do you create each one?

- Why are the fields of `ListNode` public? Is this bad style?

- What effect does this code have on a `LinkedIntList`?

```
ListNode current = front;
current = null;
```

Conceptual answers

- A list consists of 0 to many node objects.
  - Each node holds a single data element value.

- null list: `LinkedIntList list = null;`
  empty list: `LinkedIntList list = new LinkedIntList();`

- It's okay that the node fields are public, because client code never directly interacts with `ListNode` objects.

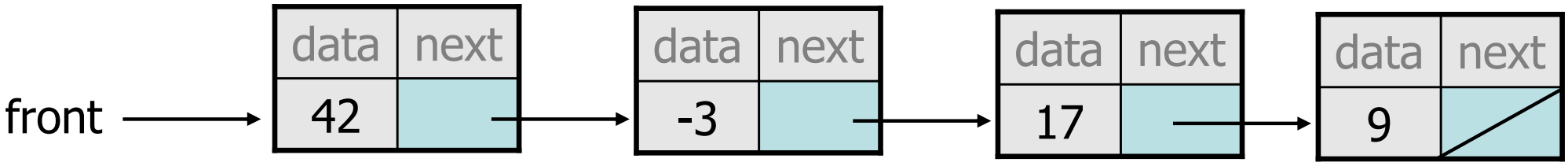
- The code doesn't change the list.
  You can change a list only in one of the following two ways:
  - Modify its `front` field value.
  - Modify the `next` reference of a node in the list.

Linked vs. array lists

- We have implemented the following two collection classes:
  - `ArrayIntList`

index	0	1	2	3
value	42	-3	17	9

  - `LinkedIntList`



  - They have similar behavior.
    We should be able to treat them the same way in client code.

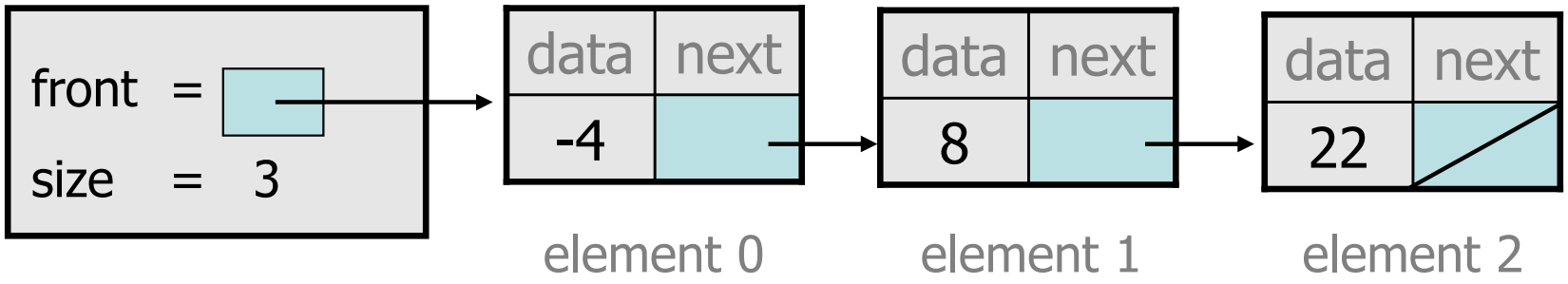
An IntList interface

```
// Represents a list of integers.
public interface IntList {
    public void add(int value);
    public void add(int index, int value);
    public int get(int index);
    public int indexOf(int value);
    public boolean isEmpty();
    public void remove(int index);
    public void set(int index, int value);
    public int size();
}
```

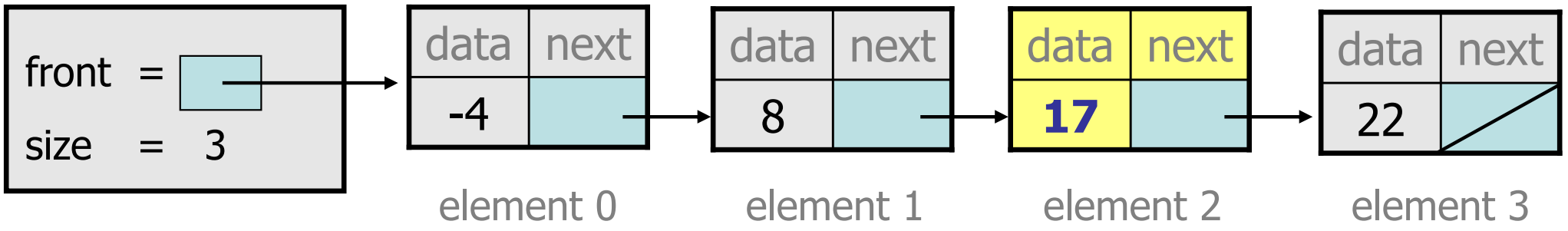
```
public class ArrayIntList implements IntList { …
public class LinkedIntList implements IntList { …
```

Exercise

- Write a method `addSorted` that accepts an integer value as a parameter and adds that value to a sorted list in sorted order.
  - Before `addSorted(17)`:



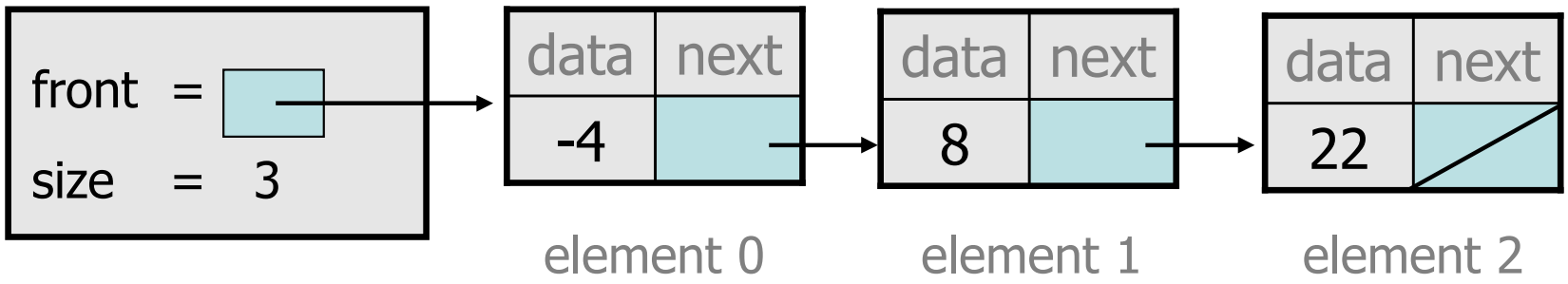
  - After `addSorted(17)`:



The common case

- Adding to the middle of a list:

```
addSorted(17)
```

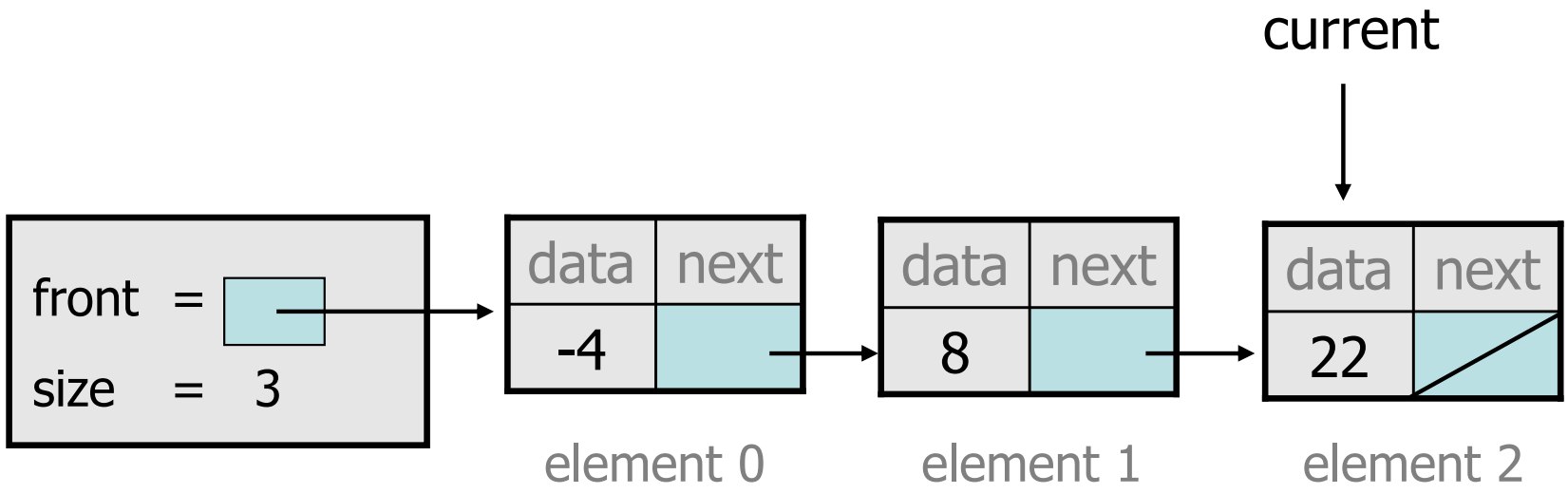


  - Which references must be changed?
  - What sort of loop do we need?
  - When should the loop stop?

First attempt

- An incorrect loop:

```
ListNode current = front;
while (current.data < value) {
    current = current.next;
}
```

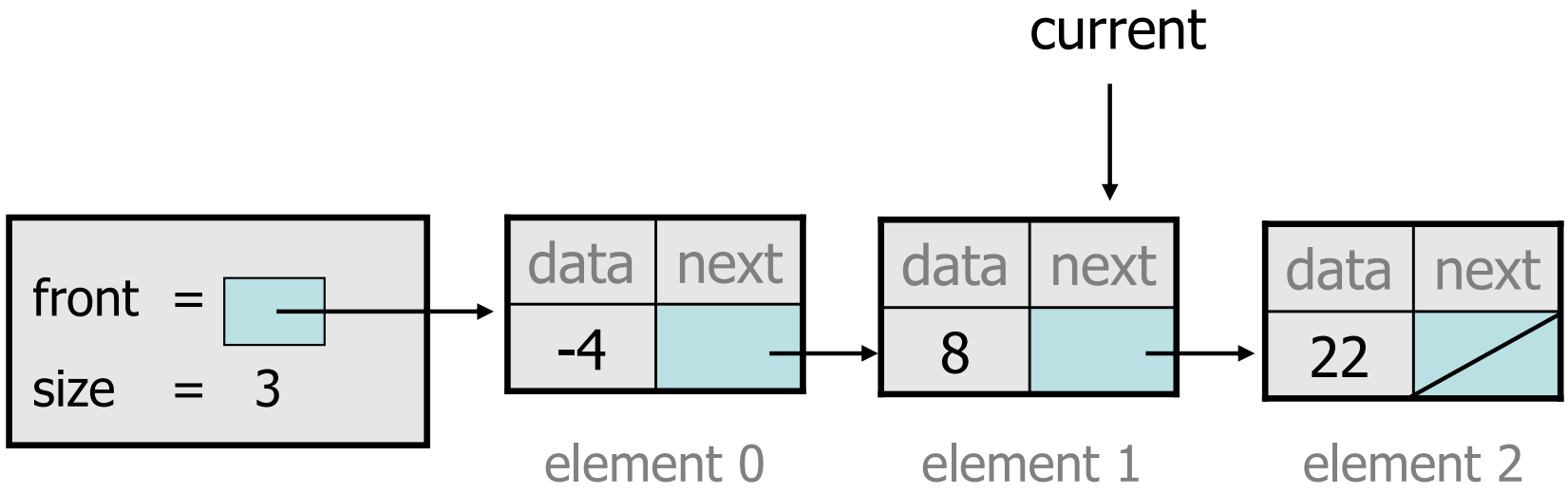


- What is wrong with this code?
  - The loop stops too late to affect the list in the right way.

Key idea: peeking ahead

- An incorrect loop:

```
ListNode current = front;
while (current.next.data < value) {
    current = current.next;
}
```

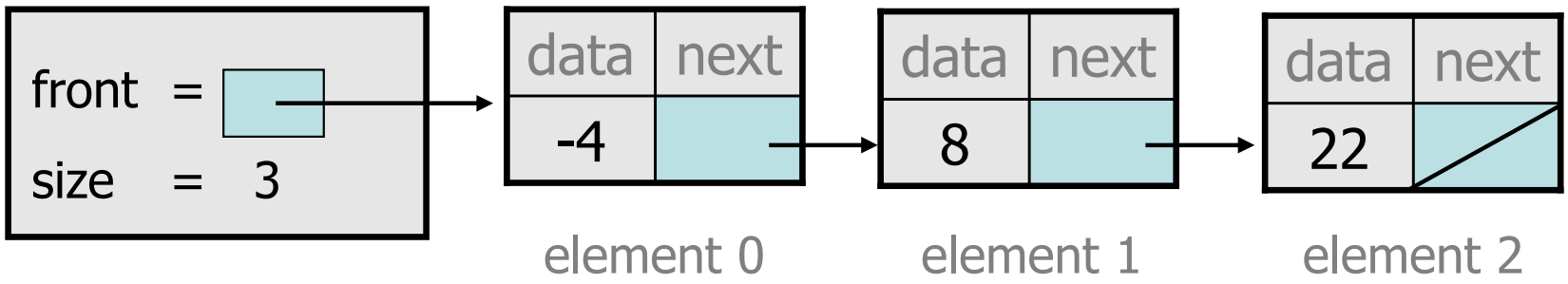


  - This time the loop stops in the right place.

Another case to handle

- Adding to the end of a list:

```
addSorted(42)
```



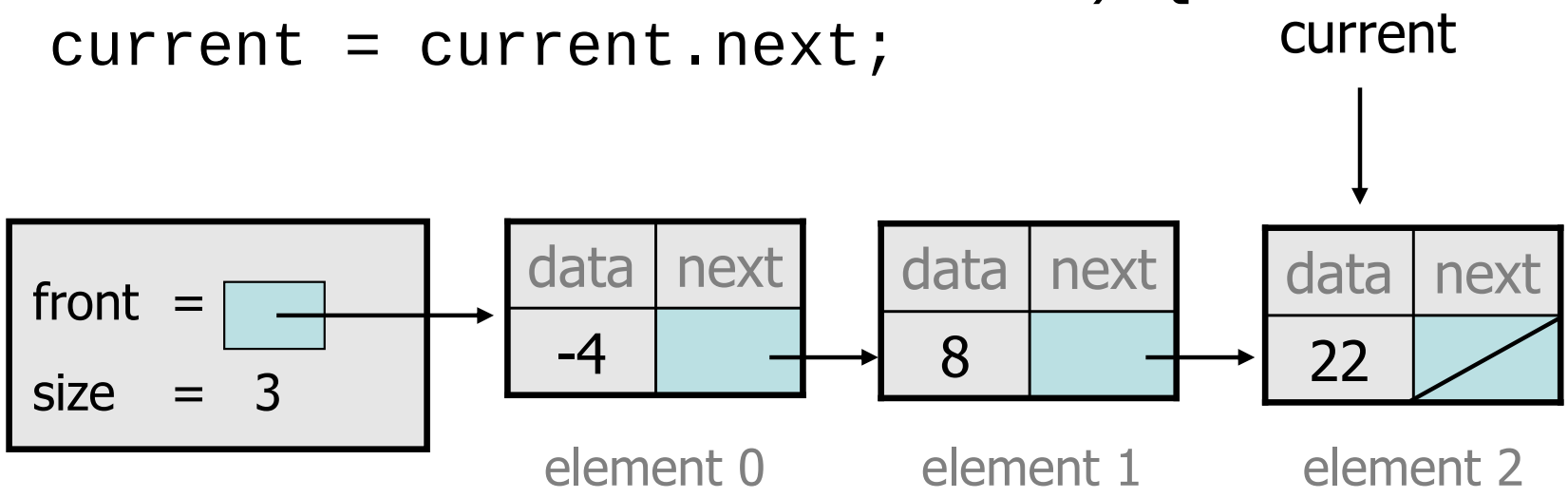

```
Exception in thread "main": java.lang.NullPointerException
```

- Why does our code crash?
- What can we change to fix this case?

Multiple loop tests

- A correction to our loop:

```
ListNode current = front;
while (current.next != null &&
       current.next.data < value) {
    current = current.next;
}
```

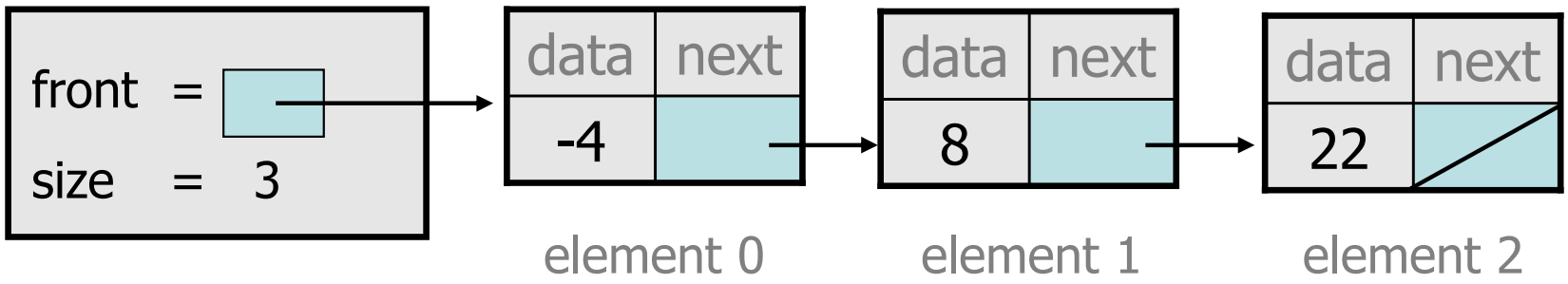


  - We must check for a `next` of `null` *before* we check its `.data`.

Third case to handle

- Adding to the front of a list:

```
addSorted(-10)
```



  - What will our code do in this case?
  - What can we change to fix it?

Handling the front

- Another correction to our code:

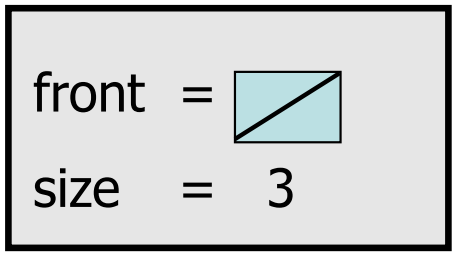
```
if (value <= front.data) {
    // insert at front of list
    front = new ListNode(value, front);
} else {
    // insert in middle of list
    ListNode current = front;
    while (current.next != null &&
           current.next.data < value) {
        current = current.next;
    }
}
```

  - Does our code now handle every possible case?

Fourth case to handle

- Adding to (the front of) an empty list:

```
addSorted(42)
```



  - What will our code do in this case?
  - What can we change to fix it?

Final version of code

```
// Adds given value to list in sorted order.
// Precondition: Existing list is sorted
public void addSorted(int value) {
    if (front == null || value <= front.data) {
        // insert at front of list
        front = new ListNode(value, front);
    } else {
        // insert in middle of list
        ListNode current = front;
        while (current.next != null &&
               current.next.data < value) {
            current = current.next;
        }
    }
}
```