In some languages (C++), -> is used for dereferencing
Implementing `add(2)`

// Inserts the given value at the given index.
public void add(int index, int value) {
    ...
}

- Exercise: Implement the two-parameter `add` method.
Write a method `addSorted` that accepts an `int` as a parameter and adds it to a sorted list in sorted order.

- **Before `addSorted(17)`:**

  Front = [ ]  
  
<table>
<thead>
<tr>
<th>Element 0</th>
<th>Element 1</th>
<th>Element 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>-4</td>
<td>8</td>
<td>22</td>
</tr>
</tbody>
</table>

- **After `addSorted(17)`:**

  Front = [ ]  
  
<table>
<thead>
<tr>
<th>Element 0</th>
<th>Element 1</th>
<th>Element 2</th>
<th>Element 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>-4</td>
<td>8</td>
<td>17</td>
<td>22</td>
</tr>
</tbody>
</table>
The common case

- Adding to the middle of a list:
  \texttt{addSorted(17)}

Which references must be changed?

What sort of loop do we need?

When should the loop stop?
First attempt

- An incorrect loop:

```java
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.
Another case to handle

- Adding to the end of a list:
  \[\text{addSorted}(42)\]

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

Why does our code crash?

What can we change to fix this case?
Third case to handle

- Adding to the front of a list:
  \[ \text{addSorted}(-10) \]

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

- Adding to (the front of) an empty list:
  \texttt{addSorted(42)}

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

- **middle**: "typical" case in the middle of an existing list
- **back**: special case at the back of an existing list
- **front**: special case at the front of an existing list
- **empty**: special case of an empty list
Other list features

- Add the following methods to the `LinkedIntList`:
  - `size`
  - `isEmpty`
  - `clear`
  - `toString`
  - `indexOf`
  - `contains`
  - `remove`

- Add preconditions and exception tests to appropriate methods.
• **interface**: A list of methods that a class can promise to implement.

  • Inheritance gives you an is-a relationship *and* code sharing.
    • A **Lawyer** can be treated as an **Employee** and inherits its code.

  • Interfaces give you an is-a relationship *without* code sharing.
    • A **Rectangle** object can be treated as a **Shape** but inherits no code.

• Always declare variables using the **interface** type.

  ```java
  List<String> list = new ArrayList<String>();
  ```