Let’s write a collection class named `LinkedIntList`.

- Has the same methods as `ArrayIntList`:
  - `add`, `add`, `get`, `indexOf`, `remove`, `size`, `toString`

- The list is internally implemented as a chain of linked nodes
  - The `LinkedIntList` keeps a reference to its `front` as a field
  - `null` is the end of the list; a `null` `front` signifies an empty list

---

**Implementing `add`**

// Adds the given value to the end of the list.
public void add(int value) {
  ...
}

- How do we add a new node to the end of a list?
- Does it matter what the list’s contents are before the add?

---

**Adding to an empty list**

- Before adding 20:

```
front = null
```

- After:

```
front = data
```

- We must create a new node and attach it to the list.
The \textit{add} method, 1st try

\begin{verbatim}
// Adds the given value to the end of the list.
public void add(int value) {
    if (front == null) {
        // adding to an empty list
        front = new ListNode(value);
    } else {
        // adding to the end of an existing list
        ...
    }
}
\end{verbatim}

Adding to non-empty list

- Before adding value 20 to end of list:

\begin{center}
\begin{tikzpicture}
    \node (front) [node_style] {front} ;
    \node [node_style, right of=front] (42) {42} ;
    \node [node_style, right of=42] (next) {\textbf{data}\ next\ -3\ \ \ } ;
    \node [node_style, right of=next] (end) {\textbf{data}\ next\ } ;

    \draw [->] (front) -- (42) ;
    \draw [->] (42) -- (next) ;
    \draw [->] (next) -- (end) ;

    \node at (front) [below] {element 0} ;
    \node at (42) [below] {element 1} ;
\end{tikzpicture}
\end{center}

- After:

\begin{center}
\begin{tikzpicture}
    \node (front) [node_style] {front} ;
    \node [node_style, right of=front] (42) {42} ;
    \node [node_style, right of=42] (next) {-3\ \ } ;
    \node [node_style, right of=next] (20) {\textbf{data}\ next\ 20\ } ;

    \draw [->] (front) -- (42) ;
    \draw [->] (42) -- (next) ;
    \draw [->] (next) -- (20) ;

    \node at (front) [below] {element 0} ;
    \node at (next) [below] {element 1} ;
    \node at (20) [below] {element 2} ;
\end{tikzpicture}
\end{center}

Don't fall off the edge!

- To add/remove from a list, you must modify the next reference of the node \textit{before} the place you want to change.

\begin{center}
\begin{tikzpicture}
    \node (front) [node_style] {front} ;
    \node [node_style, right of=front] (42) {42} ;
    \node [node_style, right of=42] (next) {\textbf{data}\ next\ -3\ \ } ;

    \draw [->] (front) -- (42) ;
    \draw [->] (42) -- (next) ;

    \node at (front) [below] {element 0} ;
    \node at (42) [below] {element 1} ;
\end{tikzpicture}
\end{center}

- Where should \textit{current} be pointing, to add 20 at the end?
- What loop test will stop us at this place in the list?

The \textit{add} method

\begin{verbatim}
// Adds the given value to the end of the list.
public void add(int value) {
    if (front == null) {
        // adding to an empty list
        front = new ListNode(value);
    } else {
        // adding to the end of an existing list
        ListNode current = front;
        while (current.next != null) {
            current = current.next;
        }
        current.next = new ListNode(value);
    }
}
\end{verbatim}
Implementing `get`

// Returns value in list at given index.
public int get(int index) {
    ...
}

- Exercise: Implement the `get` method.

```
front = data: 42  next: element 0
         data: -3  next: element 1
         data: 17  next: element 2
```

The `get` method

// Returns value in list at given index.
// Precondition: 0 <= index < size()
public int get(int index) {
    ListNode current = front;
    for (int i = 0; i < index; i++) {
        current = current.next;
    }
    return current.data;
}

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.

```
front = data: 42  next: element 0
         data: -3  next: element 1
         data: 17  next: element 2
```

The `add` method (2)

// Inserts the given value at the given index.
// Precondition: 0 <= index <= size()
public void add(int index, int value) {
    if (index == 0) {
        // adding to an empty list
        front = new ListNode(value, front);
    } else {
        // inserting into an existing list
        ListNode current = front;
        for (int i = 0; i < index - 1; i++) {
            current = current.next;
        }
        current.next = new ListNode(value, current.next);
    }
}
Implementing `get`

```java
// Returns value in list at given index.
public int get(int index) {
    ...
}
```

- Exercise: Implement the `get` method.

Warm-up 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?
- What effect does this code have on a `LinkedIntList`?
  ```java
  ListNode current = front;
  current = null;
  ```
- Implement `removeFromFront()`, which removes the item at the head of a linked list. (Recall: member variable `front` references the start of the list.)

Warm-up 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();`
- 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.
- `void remove() { front = front.next; }`
Implementing `remove`

```java
// Removes and returns the list's first value.
public int remove() {
    ...
}
```

- How do we remove the front node from a list?
- Does it matter what the list's contents are before the remove?

Removing from a list

- Before removing element at index 1:
  - **Before:**
    - `front` = 42
    - `front` = -3
    - `front` = 20
  - **After:**
    - `front` = 42
    - `front` = 20
    - `front` = -3
    - `element 0`
    - `element 1`
    - `element 2`

Removing from the front

- Before removing element at index 0:
  - **Before:**
    - `front` = 42
    - `front` = -3
    - `front` = 20
  - **After:**
    - `front` = 42
    - `front` = -3
    - `front` = 20
    - `element 0`
    - `element 1`
    - `element 2`

Removing the only element

- Before:
  - **Before:**
    - `front` = 20
    - `front` = -3
    - `front` = 20
  - **After:**
    - `front` = null
    - `front` = -3
    - `front` = 20

- We must change the front field to store `null` instead of a node.
- Do we need a special case to handle this?
remove solution

// Removes value at given index from list.
// Precondition: 0 <= index < size()
public void remove(int index) {
    if (index == 0) {
        // special case: removing first element
        front = front.next;
    } else {
        // removing from elsewhere in the list
        ListNode current = front;
        for (int i = 0; i < index - 1; i++) {
            current = current.next;
        }
        current.next = current.next.next;
    }
}

The common case

• Adding to the middle of a list:
  addSorted(17)

  front = data next data next data next
          -4           8           22
  element 0          element 1          element 2

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

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):

    front = data next data next data next
            -4           8           22
    element 0          element 1          element 2

  – After addSorted(17):

    front = data next data next data next
            -4           17           22
    element 0          element 2          element 3

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

- Corrected version of the loop:

```java
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:

```java
addSorted(42)
```

```
front = data next data next data next
-4 8 22
```

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:

```java
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:

```java
addSorted(-10)
```

```
front = data next data next data next
-4 8 22
```

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

- Another correction to our code:

```java
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

```java
// Adds given value to list in sorted order.
// Precondition: Existing elements are 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;
        }
    }
}
```

Redundant client code

```java
public class ListClient {
    public static void main(String[] args) {
        ArrayIntList list1 = new ArrayIntList();
        list1.add(18);
        list1.add(27);
        list1.add(93);
        System.out.println(list1);
        list1.remove(1);
        System.out.println(list1);
        LinkedList list2 = new LinkedList();
        list2.add(18);
        list2.add(27);
        list2.add(93);
        System.out.println(list2);
        list2.remove(1);
        System.out.println(list2);
    }
}
```
public class ListClient {
    public static void main(String[] args) {
        IntList list1 = new ArrayIntList();
        process(list1);
        IntList list2 = new LinkedIntList();
        process(list2);
    }
    public static void process(IntList list) {
        list.add(18);
        list.add(27);
        list.add(93);
        System.out.println(list);
        list.remove(1);
        System.out.println(list);
        System.out.println(list);
    }
}