Questions during Class? Raise hand or send here

sli.do  #cse123

BEFORE WE START

Talk to your neighbors:

What’s your favorite form of potato?

Music: 123 24su Lecture Tunes ☀️

Instructor: Joe Spaniac

TAs: Andras, Eric, Sahej, Trien, Zach
Lecture Outline

• Announcements/Reminders

• LinkedList
  - ListNode cont.

• Why curr?

• Modifying LinkedLists
  - Special cases (MFEF)
Announcements

• R1 and P1 feedback releasing tonight sometime after lecture

• Creative Project 2 due tonight (7/9) at 11:59pm
  - Submit *something* so we can provide some feedback!

• Check-in 2 in section on Thursday (7/10)
  - Very, very similar problem to what you might see on a quiz
  - Guaranteed to get feedback before the quiz on Tuesday if you attend

• Programming Project 2 releases tomorrow (7/10)
  - One of the trickier assignments in the course
  - 2 weeks to complete this one! Feel free to take a breather if necessary but get started sooner than later

• Quiz 2 this upcoming Tuesday (7/15)
  - Topics: Abstract Classes, ArrayIntList, LinkedList
Lecture Outline

• Announcements/Reminders

• LinkedIntList
  - ListNodes cont.

• Why curr?

• Modifying LinkedLists
  - Special cases (MFEE)
Reminder: Implementing Data Structures

• No different from designing any other class!
  - Specified behavior (List interface):

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(E value)</td>
<td>Adds the given value to the end of the list</td>
</tr>
<tr>
<td>add(int index, E value)</td>
<td>Adds the given value at the given index</td>
</tr>
<tr>
<td>remove(E value)</td>
<td>Removes the given value if it exists</td>
</tr>
<tr>
<td>remove(int index)</td>
<td>Removes the value at the given index</td>
</tr>
<tr>
<td>get(int index)</td>
<td>Returns the value at the given index</td>
</tr>
<tr>
<td>set(int index, int value)</td>
<td>Updates the value at the given index to the one given</td>
</tr>
<tr>
<td>size()</td>
<td>Returns the number of elements in the list</td>
</tr>
</tbody>
</table>

• Choose appropriate fields based on behavior

• Just requires some thinking outside the box
LinkedIntList

• Goal: leverage non-contiguous memory usage
  - How? LinkedNodes!

• What field(s) do we need to keep track of?
  - ListNode front; // First node in the chain
  - int size; // Strictly necessary?

![Diagram of LinkedIntList]

- Front node
- Data
- Next
- Front node
- Data
- Next
- Front node
- Data
- Next
- Front node
- Data
- Next
- Front node
- Data
- Next
ListNodes cont.

• Now that we have LinkedIntList, will a client ever need to use a ListNode?
  - No! Not something they should have to worry about

• How can we abstract ListNodes away from them?
  - Leaving them in a public file is pretty obvious...

• What if we made ListNode a private class inside LinkedIntList?
  - We can still access it (just like private fields)
  - Clients won’t even know the class exists!

• Do fields need to be private if the entire class is private?
Lecture Outline

• Announcements/Reminders

• LinkedList
  - ListNode cont.

• Why curr?

• Modifying LinkedLists
  - Special cases (MFEE)
Reminder: Iterating over ListNode

- General pattern iteration code will follow:

```java
ListNode curr = front;
while (curr != null) {
    // Do something
    curr = curr.next;
}
```

Why do we need a ListNode curr?
public static void main(String[] args) {
    ListNode front = new ListNode(1, new ListNode(2, new ListNode(3)));
}

public static void printList(ListNode front) {
    while (front != null) {
        System.out.print(front.data + " ");
        front = front.next;
    }
    System.out.println();
}
Why curr?

```java
public static void main(String[] args) {
    ListNode front = new ListNode(1, new ListNode(2, new ListNode(3)));
}

public static void printList(ListNode front) {
    while (front != null) {
        System.out.print(front.data + " ");
        front = front.next;
    }
    System.out.println();
}
```

Diagram:
- **main()**
  - `front` with values 1, 2, 3
- **printList()**
  - `front` iterates through the list 1 -> 2 -> 3

The diagram shows the list nodes with values 1, 2, and 3, and the `printList()` method printing the list in reverse order: 3, 2, 1.
Why curr?

```java
public static void main(String[] args) {
    ListNode front = new ListNode(1, new ListNode(2, new ListNode(3)));
}

public static void printList(ListNode front) {
    while (front != null) {
        System.out.print(front.data + " ");
        front = front.next;
    }
    System.out.println();
}
```

public static void main(String[] args) {
    ListNode front = new ListNode(1, new ListNode(2, new ListNode(3)));
}

public static void printList(ListNode front) {
    while (front != null) {
        System.out.print(front.data + " ");
        front = front.next;
    }
    System.out.println();
}
Why curr?

public class LinkedIntList {
    private ListNode front;

    public void printList() {
        while (front != null) {
            System.out.print(front.data + " ");
            front = front.next;
        }
        System.out.println();
    }
}
Why curr?

public class LinkedIntList {
    private ListNode front;

    public void printList() {
        while (front != null) {
            System.out.print(front.data + " ");
            front = front.next;
        }
        System.out.println();
    }
}

LinkedIntList

front

1 — 2 — 3
Why curr?

public class LinkedIntList {
    private ListNode front;

    public void printList() {
        while (front != null) {
            System.out.print(front.data + " ");
            front = front.next;
        }
        System.out.println();
    }
}

![Diagram of LinkedIntList with nodes 2 and 3 connected]
Why curr?

```java
public class LinkedIntList {
    private ListNode front;

    public void printList() {
        while (front != null) {
            System.out.print(front.data + " ");
            front = front.next;
        }
        System.out.println();
    }
}
```

Modifying `front` now modifies the list!
public class LinkedIntList {
    private ListNode front;

    public void printList() {
        ListNode curr = front;
        while (curr != null) {
            System.out.print(curr.data + " ");
            curr = curr.next;
        }
        System.out.println();
    }
}
Why curr?

```java
public class LinkedIntList {
    private ListNode front;

    public void printList() {
        ListNode curr = front;
        while (curr != null) {
            System.out.print(curr.data + " ");
            curr = curr.next;
        }
        System.out.println();
    }
}
```

Diagram:
- `front` of `LinkedIntList`
- `curr`
- `printList()`

Diagram shows a list with nodes 1, 2, and 3 connected in a line, with 3 having a self-loop.
Why curr?

public class LinkedIntList {
    private ListNode front;

    public void printList() {
        ListNode curr = front;
        while (curr != null) {
            System.out.print(curr.data + " ");
            curr = curr.next;
        }
        System.out.println();
    }
}
Why curr?

public class LinkedIntList {
    private ListNode front;

    public void printList() {
        ListNode curr = front;
        while (curr != null) {
            System.out.print(curr.data + " ");
            curr = curr.next;
        }
        System.out.println();
    }
}

LinkedIntList

front

printList() curr

1 -> 2 -> 3
Lecture Outline

• Announcements/Reminders

• LinkedList
  - ListNodes cont.

• Why curr?

• Modifying LinkedLists
  - Special cases (MFEE)
Modifying LinkedLists

• Remember: using a curr variable to iterate over nodes
• We want to insert a 3 node between 2 and 4

• Does changing curr actually update our chain?
  - What will?
Modifying LinkedLists

• Remember: using a curr variable to iterate over nodes
• We want to insert a 3 node between 2 and 4

• Does changing curr actually update our chain?
  - 1. Make a new node storing 3 pointing to 4
Modifying LinkedLists

• Remember: using a curr variable to iterate over nodes
• We want to insert a 3 node between 2 and 4

• Does changing curr actually update our chain?
  - 1. Make a new node storing 3 pointing to 4
  - 2. Make 2 point to 3
Modifying LinkedLists

• Remember: using a `curr` variable to iterate over nodes
• We want to insert a 3 node between 2 and 4

• Does changing `curr` actually update our chain?
  - `curr.next = new ListNode(3, curr.next);`
Modifying LinkedLists

• Remember: using a `curr` variable to iterate over nodes
• We want to insert a 3 node between 2 and 4

• Does changing `curr` actually update our chain?
  - What will? Changing `curr.next`
Modifying LinkedLists

• Remember: using a curr variable to iterate over nodes
• We want to insert a 0 node before 1

• Is there anyway for us to do this with curr?
  - No!
Modifying LinkedLists

• Remember: using a curr variable to iterate over nodes
• We want to insert a 0 node before 1

• Is there anyway for us to do this with curr?
  - 1. Make a new node storing 0 pointing to 1
Modifying LinkedLists

• Remember: using a curr variable to iterate over nodes
• We want to insert a 0 node before 1

• Is there anyway for us to do this with curr?
  - 1. Make a new node storing 0 pointing to 1
  - 2. Make front point to 0
Modifying LinkedLists

• Remember: using a curr variable to iterate over nodes
• We want to insert a 0 node before 1

• Is there anyway for us to do this with curr?
  - this.front = new ListNode(0, this.front);
Modifying LinkedLists

• Remember: using a curr variable to iterate over nodes
• We want to insert a 0 node before 1

• So, what will actually change our list?
  - Changing curr.next, changing front
  - Need to stop one early to make changes
Modifying LinkedLists

• Remember: using a curr variable to iterate over nodes
• We want to insert a 3 node between 2 and 4

• Does changing curr actually update our chain?
  - What will? Changing curr.next, changing front
  - Need to stop one early to make changes

• Often a number of cases to watch out for:
  - M(iddle) – Modifying node in the middle of the list (general)
  - Front – Modifying the first node
  - E(mpty) – What if the list is empty?
  - E(nd) – Rare, do we need to do something with the end of the list?