"If somebody were to watch most of my life over the past few years, it would be me sitting in a quiet room by myself studying and laboring over mounds of information," he told NPR.

"I think that sometimes people just look at the end product of somebody’s hard work," he said, reflecting back on his own journey. "But you know, [by doing that] you kind of miss the part where people are doing all the work that it takes to become a success."

"[M]y philosophy has always been to be comfortable with being uncomfortable. And the more I can put myself into uncomfortable situations, the more I can grow."
In recent years, Allamby has been asked to speak publicly about his journey from fixing cars to saving lives. When he does, he avoids using language that makes him sound exceptional. In fact, he tries to do the opposite, stressing the methodical nature of his slow rise through the ranks of academia.

"There’s going to be times when you feel like giving up, but those are the times to really push forward and to rely on the people who surround you," Allamby said. "People who give you positive feedback in order to kind of fill your bucket back up so that you can keep going."

### Topics
- ADTs
- More linked lists

### Your to-dos
1. Read **before Fri**: Bailey, Ch 9.4–9.5.
2. Quiz 4, **due Saturday by noon**.
3. Lab 4, **due Tuesday 10/11 by 10pm**.

### Announcements
- Quizzes: earlier
- Colloquium: **What I Did Last Summer (Industry)**, 2:35pm in Wege Auditorium with **cookies**.
A **linked list** is a recursive data structure. A linked list is composed of simple pieces called **list nodes**. A list node contains **data** (of generic type `T`) and a **reference** (a “link”) to either another list node or null.

The empty list is defined as **null**.

Every other list has at least one list node.

There's only **one link** in each node, to the **rest** of the list.

How would I represent the above idea in **Java**?
The purpose of a class:
To “abstract away” implementation details.

Abstraction
Abstraction is the process of removing irrelevant information so that a program is easier to understand.

Do you see any similarities?
The two classes share the same interface.

Interface
An interface defines boundary between two systems across which they share information. An interface is a contract: calling a method defined in an interface returns the data as promised.

Because an interface contains no implementation, programmers who use them cannot rely on implementation details.

E.g., the List interface states that there must be an add method but does not say how it should be implemented.
List

A list is an ordered collection of items of an element of type E. It supports prepending an element to the front, appending (adding) and element to the end, finding an element, and element removal.

A Vector is a list.

A SinglyLinkedList is a list.

A DoublyLinkedList is a list.

Observe that this similarity is “deeper” than just what an interface provides....

Abstract Data Type

An abstract data type is a mathematical formulation of a data type. ADTs abstract away accidental properties of data structures (e.g., implementation details, programming language). Instead, ADTs contain only essential properties and are concisely defined by their logical behavior over a set of values and a set of operations.

In an ADT, precisely how data is represented on a computer does not matter.

By contrast: data structure

A data structure is the physical form of a data type, i.e., it is an implementation of an ADT. Generally, data structures are designed to efficiently support the logical operations described by the ADT.

For data structures, precisely how data is represented on a computer matters a lot. Simple data structures are often composed of simple representations, like primitives, while more complex data structures are composed of other data structures.

Vector, SinglyLinkedList, etc. are data structures.

A Vector is a List

```java
structure:
Class Vector<E>

extends AbstractList<E>
implements java.lang.Cloneable

All Implemented Interfaces:
java.lang.Cloneable, java.lang.Iterable<E>, List<E>, Structure<E>
```
A Linked List is a List

structure:
Class SinglyLinkedList<E>

java.lang.Object
|-- AbstractStructure<E>
|   |-- AbstractList<E>
|   |   `|-- SinglyLinkedList<E>

All Implemented Interfaces:
java.lang.Iterable<E>, List<E>, AbstractList<E>, AbstractCollection<E>, Cloneable

public class SinglyLinkedList<E>
extends AbstractList<E>

Vector Big-O

<table>
<thead>
<tr>
<th>operation</th>
<th>worst</th>
<th>best</th>
</tr>
</thead>
<tbody>
<tr>
<td>addFirst(E e)</td>
<td>O(n)</td>
<td>O(1)</td>
</tr>
<tr>
<td>get(int i)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>indexOf(E e)</td>
<td>O(n)</td>
<td>O(1)</td>
</tr>
<tr>
<td>remove(E e)</td>
<td>O(n)</td>
<td>O(1)</td>
</tr>
<tr>
<td>size()</td>
<td>O(n)</td>
<td>O(1)</td>
</tr>
</tbody>
</table>

Singly-Linked List Big-O

<table>
<thead>
<tr>
<th>operation</th>
<th>worst</th>
<th>best</th>
</tr>
</thead>
<tbody>
<tr>
<td>addFirst(E e)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>get(int i)</td>
<td>O(n)</td>
<td>O(1)</td>
</tr>
<tr>
<td>indexOf(E e)</td>
<td>O(n)</td>
<td>O(1)</td>
</tr>
<tr>
<td>remove(E e)</td>
<td>O(n)</td>
<td>O(1)</td>
</tr>
<tr>
<td>size()</td>
<td>O(n)</td>
<td>O(1)</td>
</tr>
</tbody>
</table>

Missing from Java: ADT behavior

Java provides no way of specifying behavior independently of implementation.

E.g., a List interface might require

    public void prepend(T elem)

But there’s no way to require that an implementation actually place the element at the beginning of the list.
The best we can do in Java: static types

Java uses types to stand in for ADTs.

However, Java provides some control over abstractness, and we can use this control to approximate what we want.

interface $\rightarrow$ fully abstract
abstract class $\rightarrow$ partially abstract
class $\rightarrow$ not abstract

Interface

An interface defines boundary between two systems across which they share information. An interface is a contract: calling a method defined in an interface returns the data as promised.

An interface contains no implementation!

You cannot specify behavior at all!

Recap & Next Class

Today:
- ADTs
- Lists

Next class:
- Sorting