CS 10:
Problem solving via Object Oriented Programming

Lists Part 1

Agenda

1. Defining a List ADT
2. Generics
3. Singly linked list implementation
4. Exceptions
5. Visibility: public vs. private vs. protected vs. package

Abstract Data Types specify operations on a data set that defines overall behavior

Abstract Data Types (ADTs)

- ADTs specify a set of ***operations*** (e.g., *get, set, add*, …) that define how the ADT behaves on a collection of data elements

- At the ADT level we don't know (and don't really care) how data elements are stored (e.g., linked list or array or something else, it doesn't matter an abstract level)

- Also do not care about what kind of data the ADT holds (e.g., Strings, integers, Objects) – the ADT works the same regardless of what type of data it holds

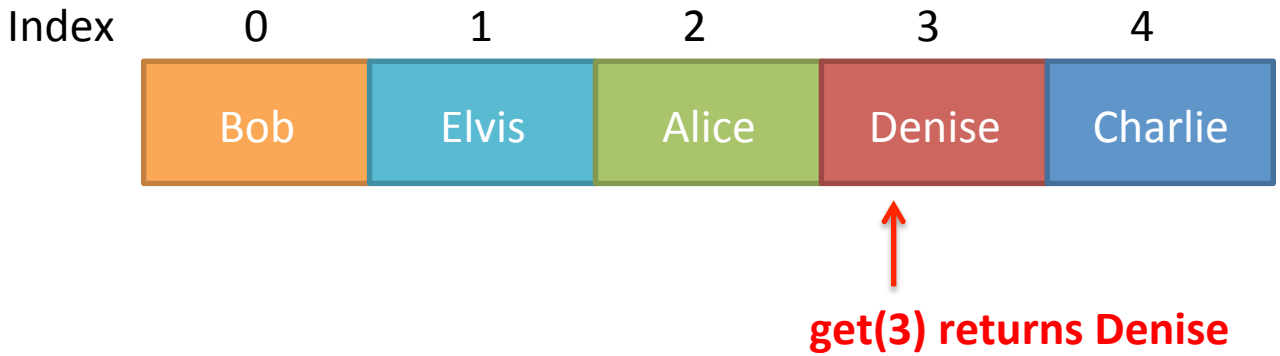
- Big idea: hide the way data is represented while allowing others to work with the data in a consistent manner

Example: List ADT defines a set of operations

List holds multiple elements (items) referenced by position in List

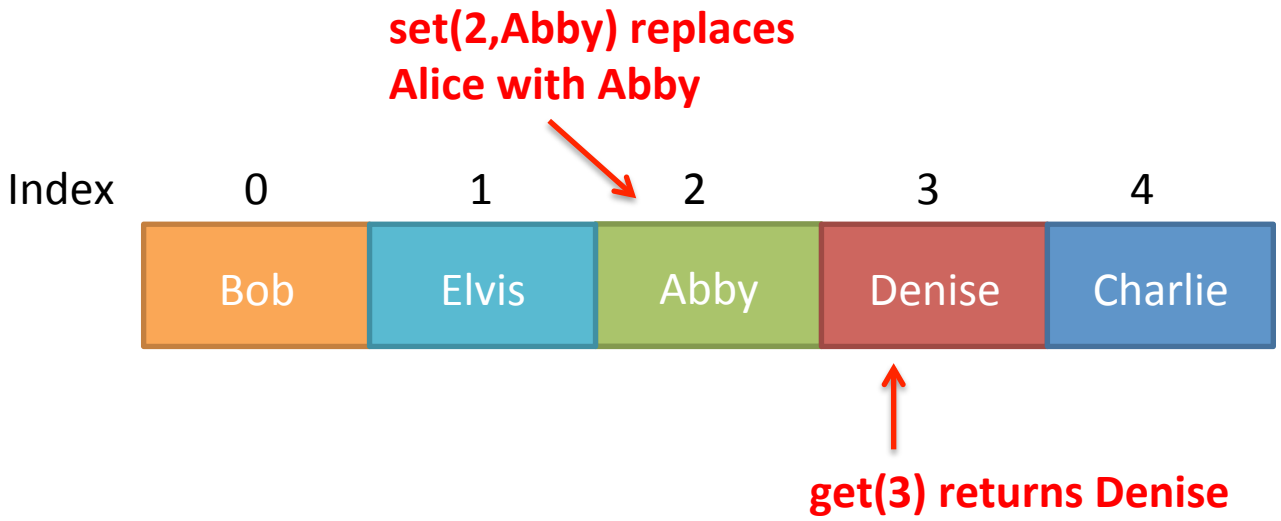
Example: List ADT defines a set of operations

List holds multiple elements (items) referenced by position in List



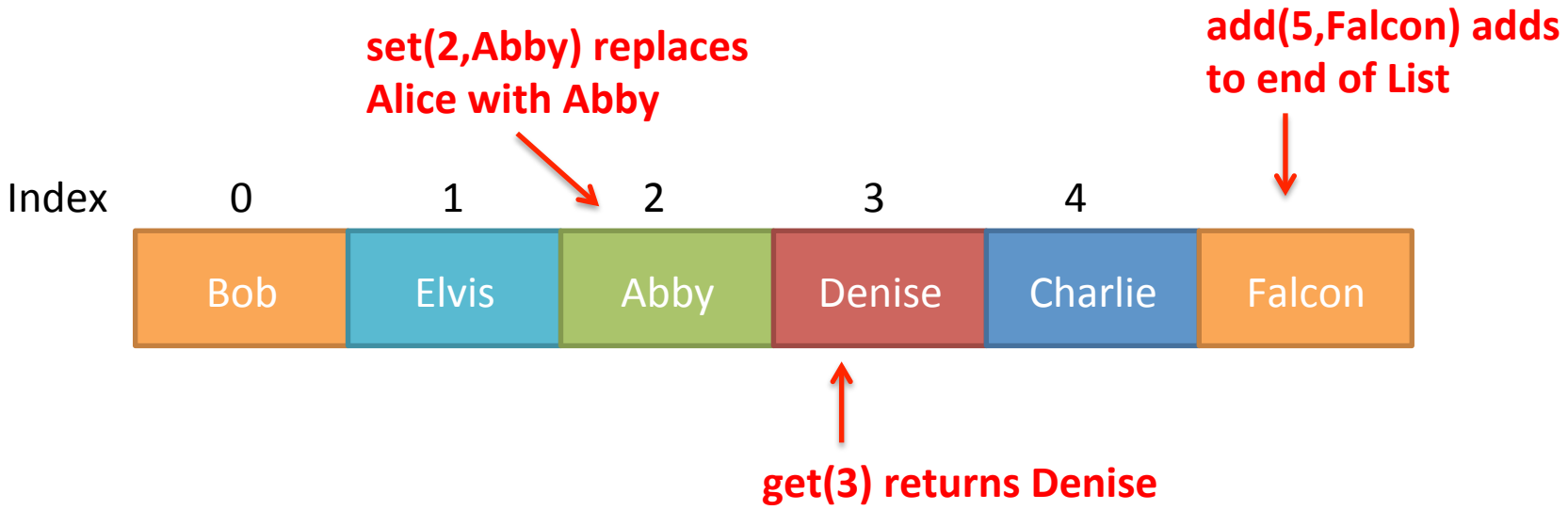
Example: List ADT defines a set of operations

List holds multiple elements (items) referenced by position in List



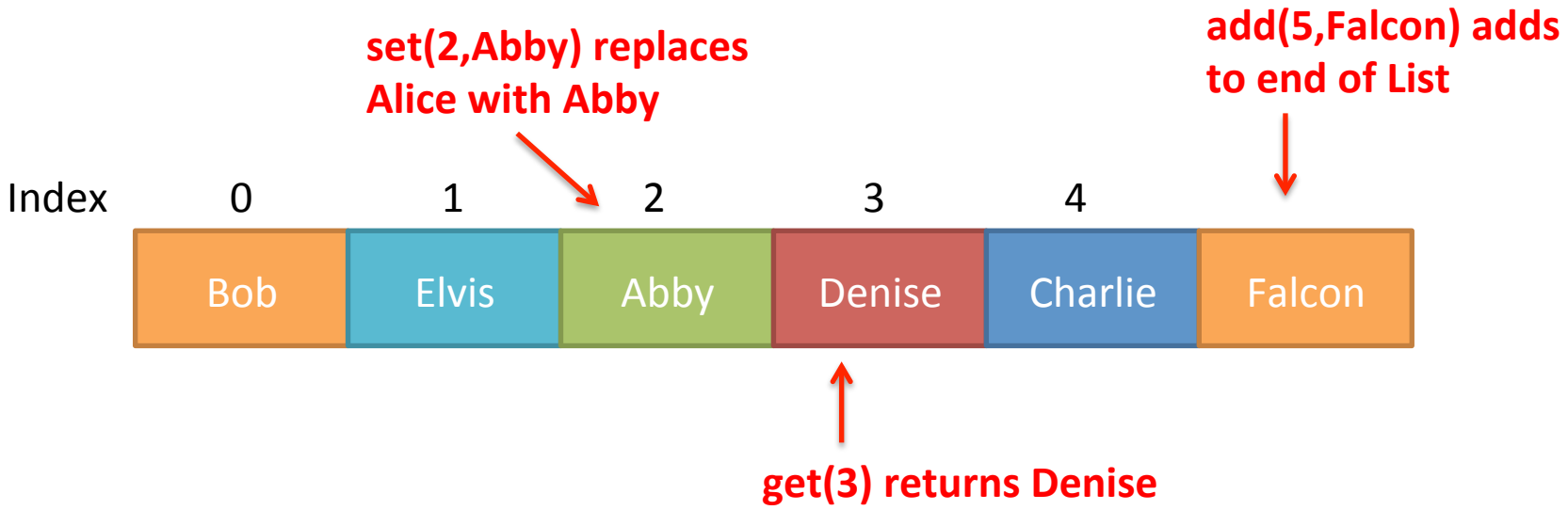
Example: List ADT defines a set of operations

List holds multiple elements (items) referenced by position in List



Example: List ADT defines a set of operations

List holds multiple elements (items) referenced by position in List

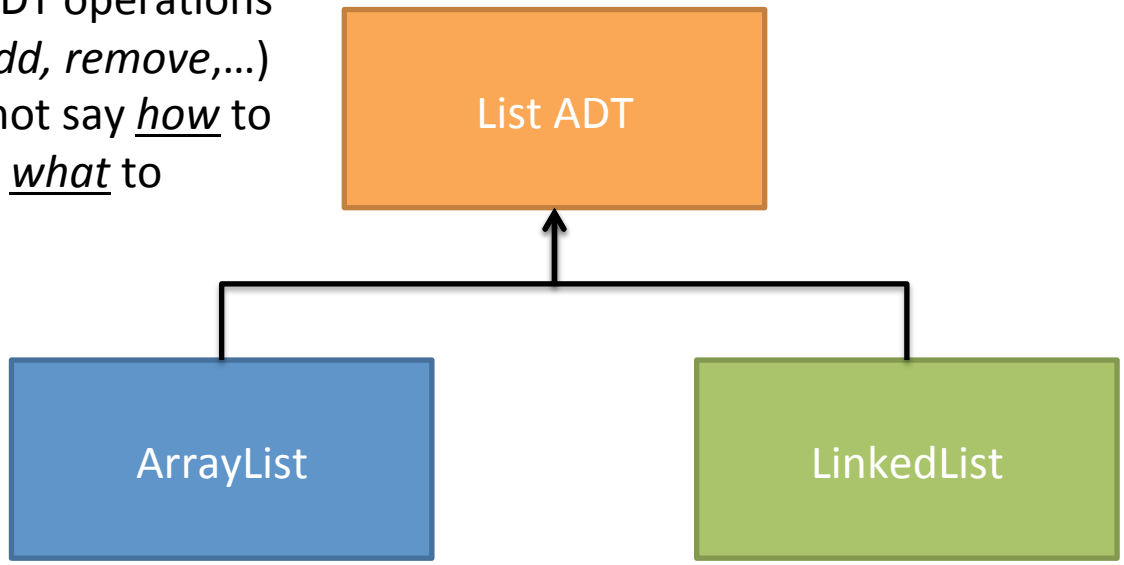


- **ADT defines these operations (and others)**
- **How were these items stored? Array? Linked List?**
  - We don't know and don't care at the ADT level, we just care that the operations (get, set, add, …) work as expected
- **What type of elements are these? Strings, Student Objects?**
  - See answer above – we don't care
  - The type of element does not affect how the ADT works!

ADTs can be implemented differently, but must provide common functionality

Java Interface:

- Defines set of ADT operations (e.g., *get, set, add, remove*,…)
- Interface does not say *how* to implement, just *what* to implement



Implementation:

- Code to implement operations that are defined by interface
- Can be written using different data structures
- MUST implement all functionality defined by interface
- Can include other functionality

Elements stored in an Array

Elements stored in an Linked List

Java has both ArrayList and LinkedList implementations of List
Both implementations provide the same functionality as required by interface, but store data differently

We will implement the List interface using both approaches

The List Interface describes several operations, but not implementations

List ADT

Operation	Description
`size()`	Return number of items in List
`isEmpty()`	True if no items in List, otherwise false
`get(i)`	Return the item at index *i*
`set(i,e)`	Replace the item at index *i* with item *e*
`add(i,e)`	Insert item *e* at index *i*, moving all subsequent items one index larger
`remove(i)`	Remove and return item at index *i*, move all subsequent items one index smaller

These operations MUST be implemented to complete the ADT
Free to implement other methods, but must have these
Notice the familiar look from Java's ArrayList

Interfaces go in one file, implementations go in another file(s)

Interface file
Specifies required operations
`SimpleList.java`

Uses keyword `interface`

Linked list implementation
`SinglyLinked.java`

OR

Array implementation

Implementation file
Actually implements required operations using a specific data structure

Same interface *could* be implemented in different ways (e.g., linked list *or* array)

Use keyword `implements` to implement an `interface`

SimpleList.java is an interface that specifies what operations MUST be implemented

```java
/**
 * A basic interface for a generic list ADT
 */
public interface SimpleList<T> {
    /**
     * Returns # elements in the list (they are indexed 0..size-1)
     */
    public int size();

    /**
     * Adds the item at the index, which must be between 0 and size
     * (since the current elements are 0..size-1, idx = size grows the list)
     */
    public void add(int idx, T item) throws Exception;

    /**
     * Removes the item at the index, which must be between 0 and size-1
     */
    public void remove(int idx) throws Exception;

    /**
     * Returns the item at the index, which must be between 0 and size-1
     */
    public T get(int idx) throws Exception;

    /**
     * Replaces the item at the index, which must be between 0 and size-1
     */
    public void set(int idx, T item) throws Exception;
}
```

Interface keyword tells Java this is an interface

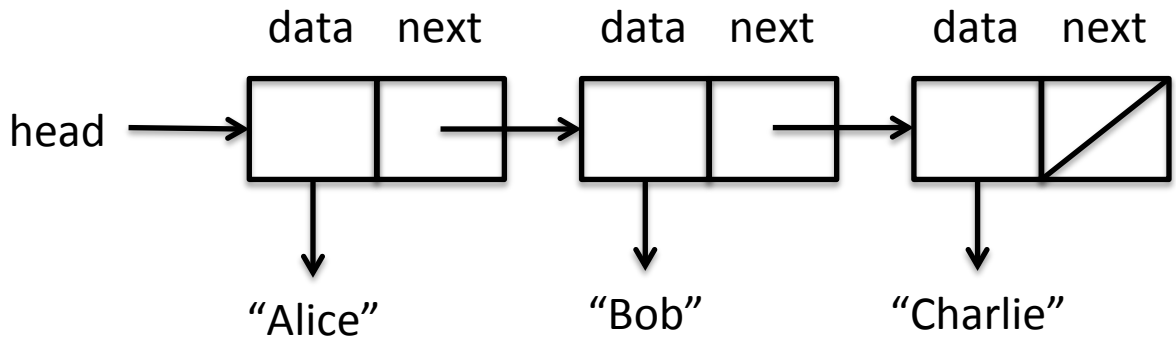
- **Methods defined to include parameters and return types (called a "signature")**
- **If you are going to create a List, then you MUST implement these methods**
- **How you implement is your business**

The List ADT could be *implemented* with a singly linked list *OR* an array; either works

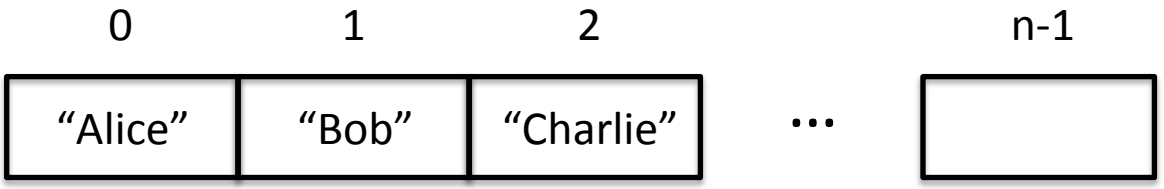
Examples of List implementation

- **We will implement a List both ways**
- **Each implementation has pros and cons**

Singly linked list



Array



Agenda

1. Defining a List ADT
2. Generics
3. Singly linked list implementation
4. Exceptions
5. Visibility: public vs. private vs. protected vs. package

Generics allow a variable to stand in for a Java type

Interface declaration

```
public interface SimpleList<T> {
  ...
  public T get(int index) throws Exception;
  public void add(int index, T item) throws Exception;
}
```

- T stands for whatever object *type* we instantiate
- SimpleList<Blob> then T always stands for Blob
- SimpleList<Point> then T always stands for Point
- Allows us to write *one* implementation that works regardless of what kind of Object we store in our data set
- Must use class version of primitives (Integer, Double, etc)
- By convention we name type of variables with a single uppercase letter, often T for “type”, later we’ll use K for key and V for value

Agenda

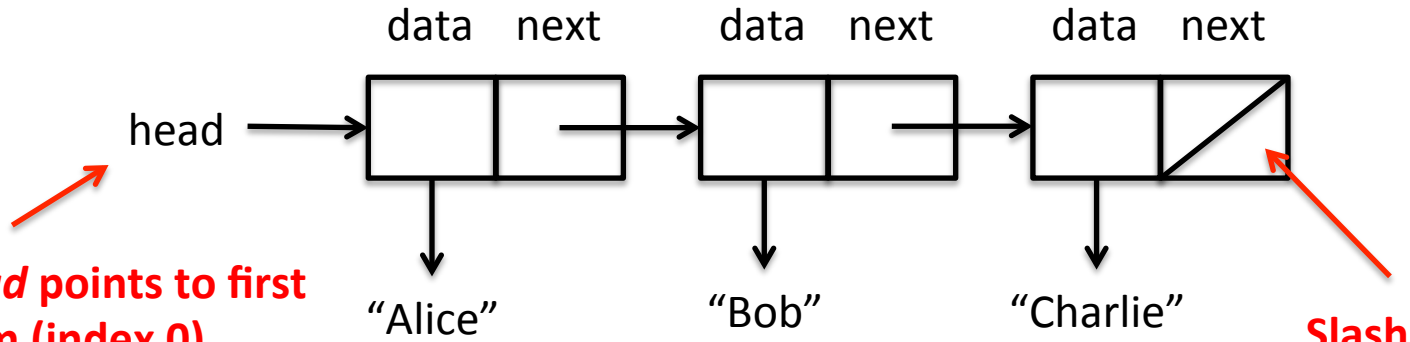
1. Defining a List ADT
2. Generics
3. Singly linked list implementation
4. Exceptions
5. Visibility: public vs. private vs. protected vs. package

Singly linked list review: elements have data and a next pointer

Singly linked list

"Box-and-pointer" diagram

- **Data in Box**
- **Pointer to next item in List**



***head* points to first item (index 0)**

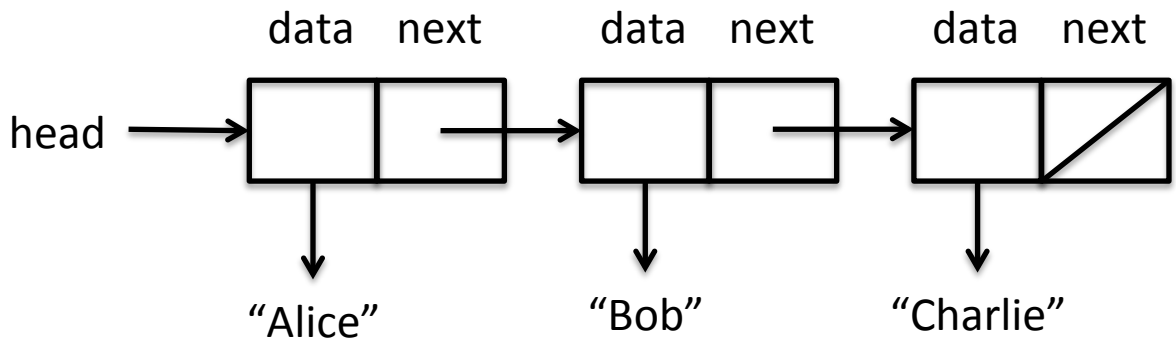
***null* if list is empty**

Slash indicates end of List

next* pointer is *null

To get an item at index i, start at head and march down

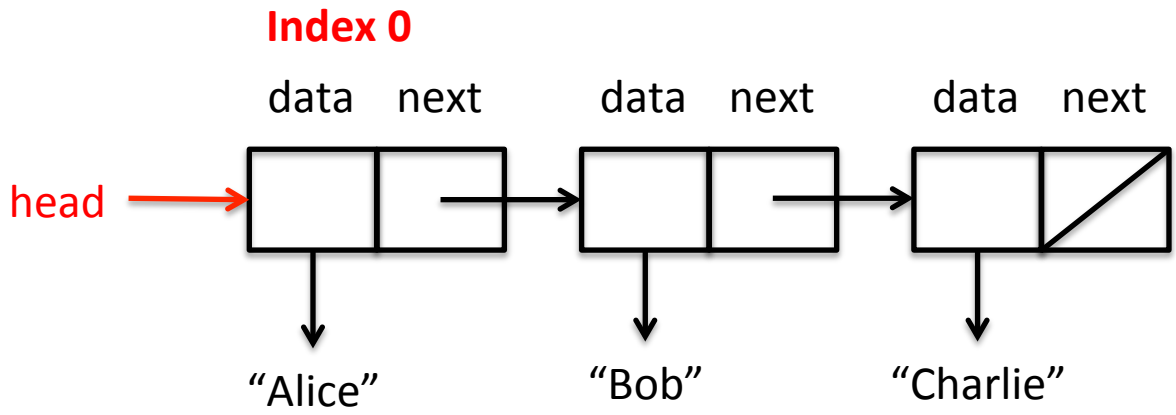
get(i) – return item at specified index



Get item at index 2

To get an item at index i, start at head and march down

get(i) – return item at specified index

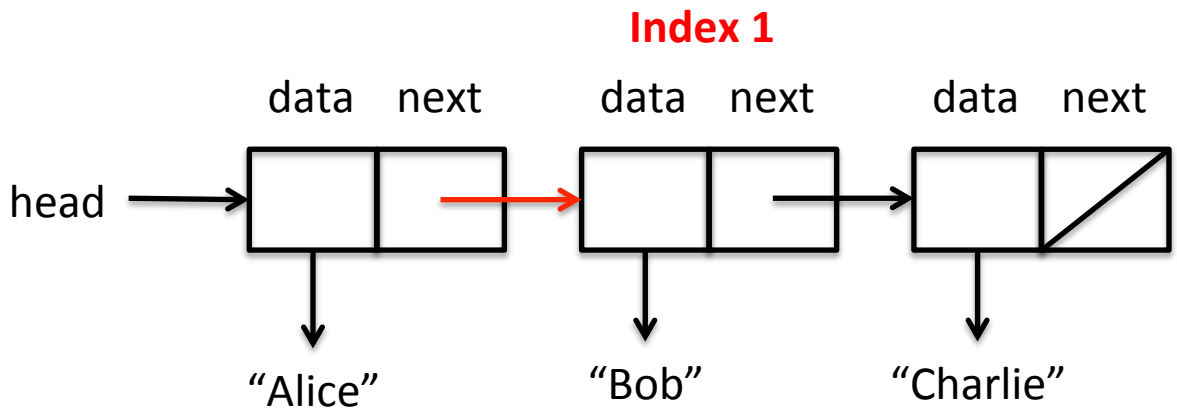


Get item at index 2

1. **Start at head (index 0)**

To get an item at index i, start at head and march down

get(i) – return item at specified index

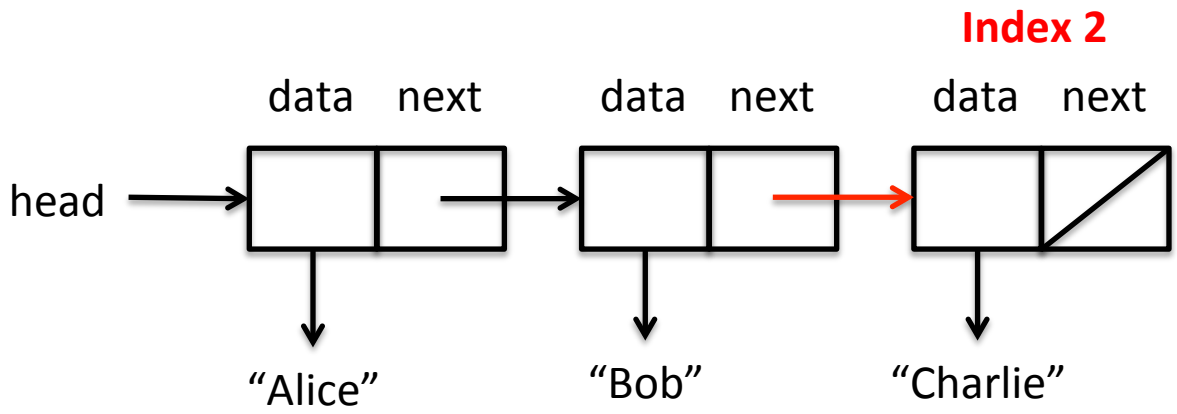


Get item at index 2

1. **Start at head (index 0)**
2. **Follow next pointer to index 1**

To get an item at index i, start at head and march down

get(i) – return item at specified index

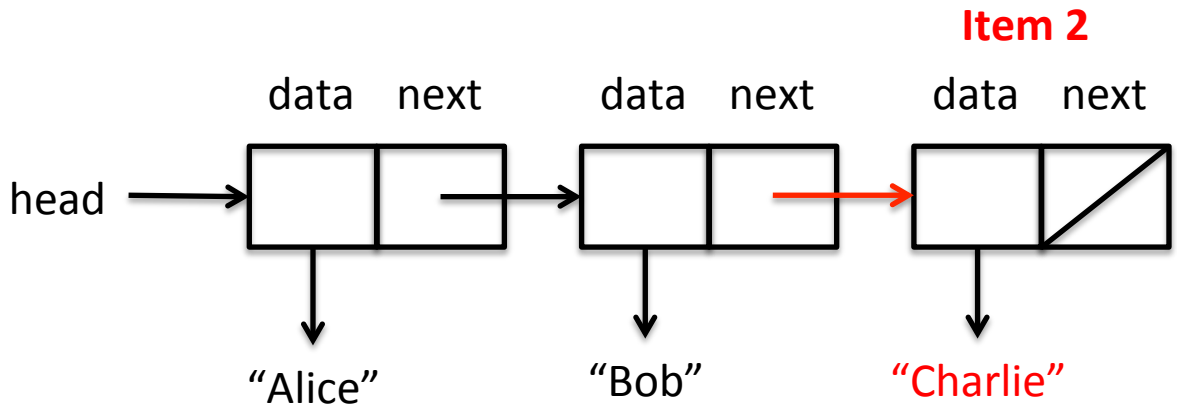


Get item at index 2

1. **Start at head (index 0)**
2. **Follow next pointer to index 1**
3. **Follow next pointer to index 2**

To get an item at index i, start at head and march down

get(i) – return item at specified index

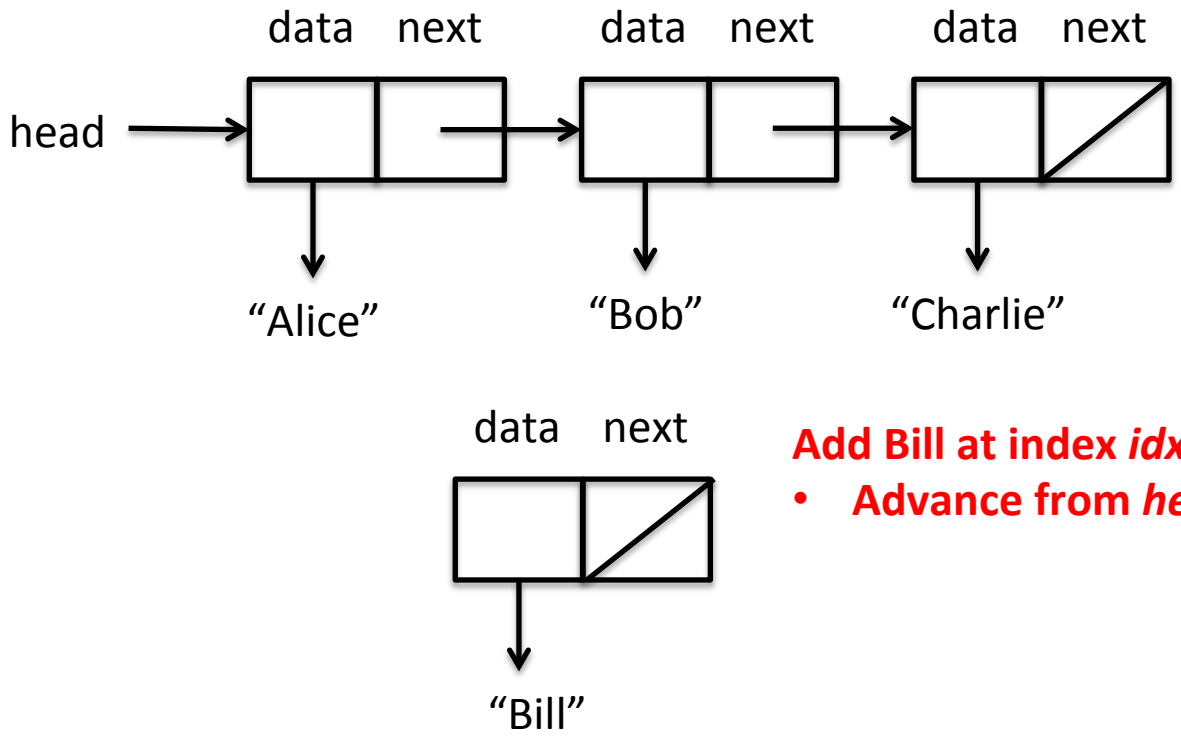


Get item at index 2

1. **Start at head (index 0)**
2. **Follow next pointer to index 1**
3. **Follow next pointer to index 2**
4. **Return "Charlie" (index 2)**

add() "splices in" a new object anywhere in the list by updating next pointers

add(1, "Bill")

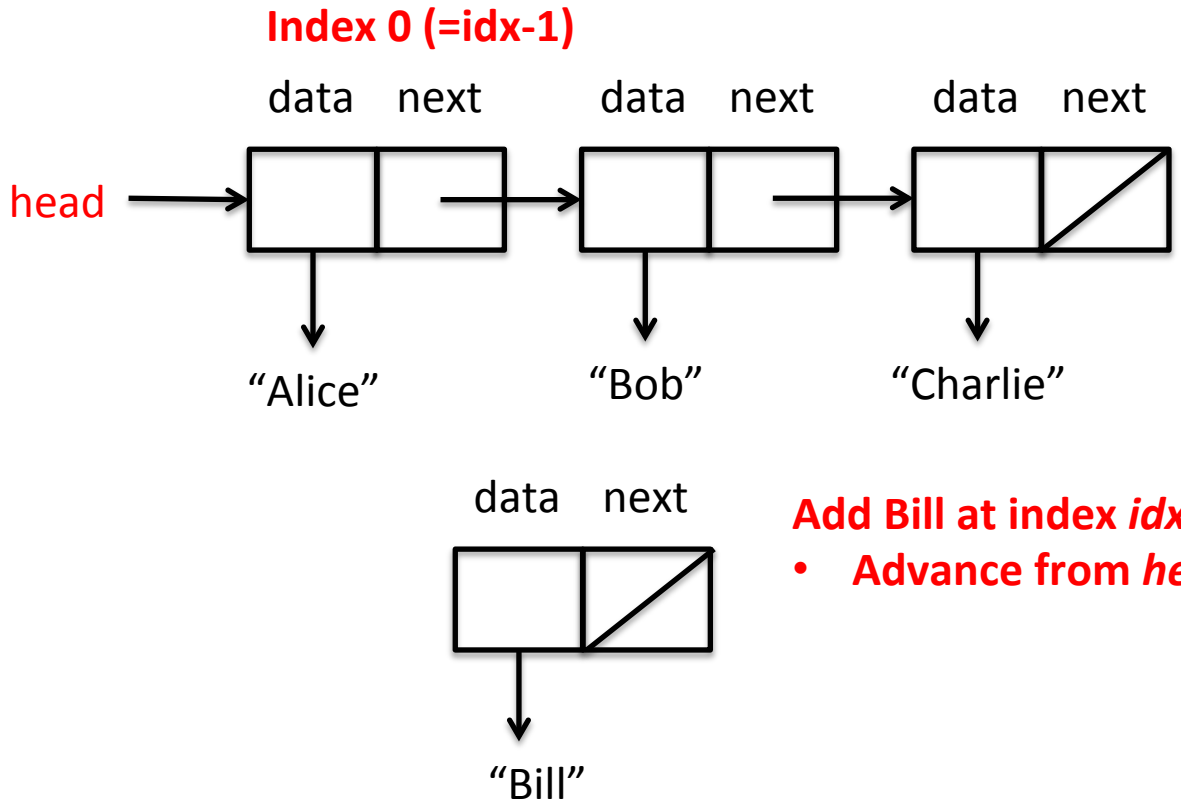


Add Bill at index *idx=1*

- **Advance from *head* to *idx-1* (Alice)**

add() "splices in" a new object anywhere in the list by updating next pointers

add(1, "Bill")

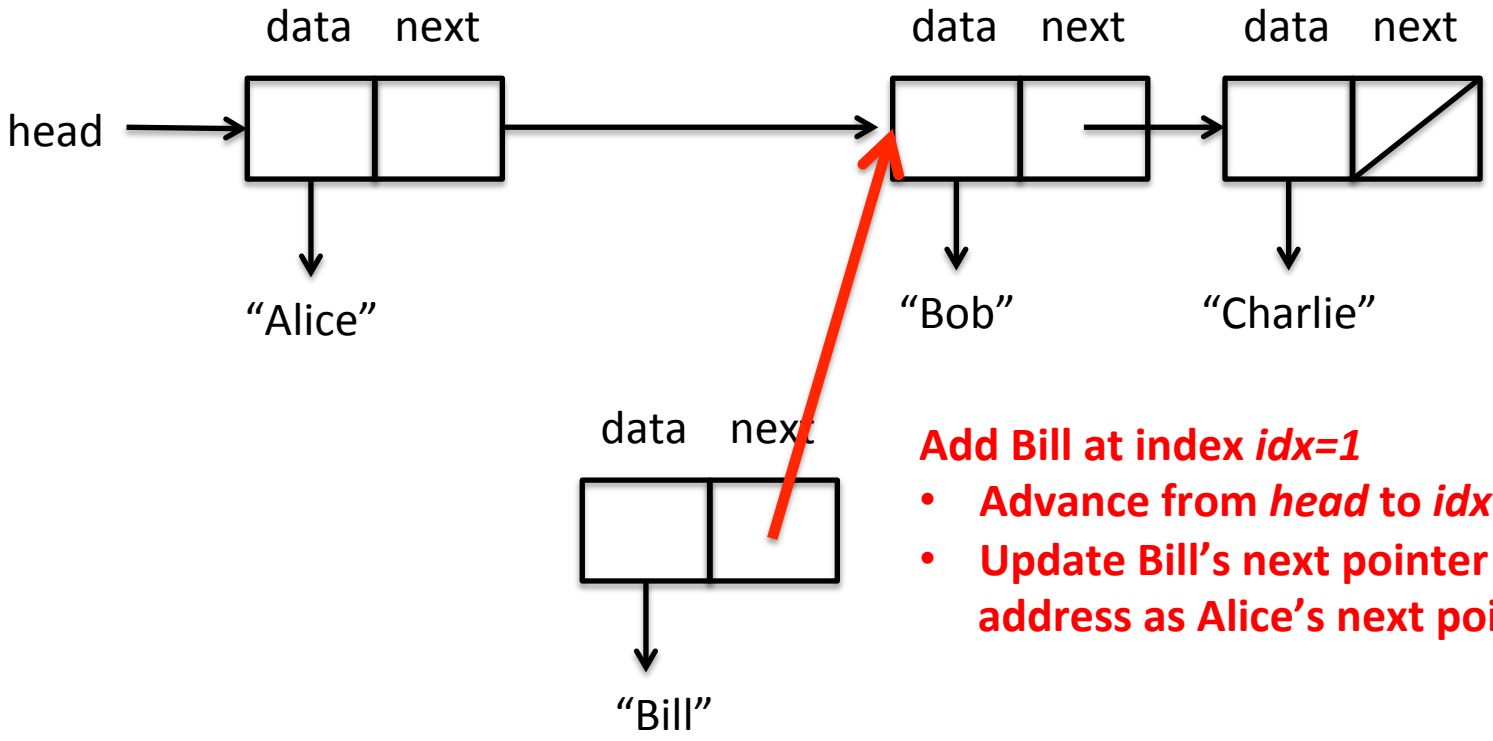


Add Bill at index *idx=1*

- **Advance from *head* to *idx-1* (Alice)**

add() "splices in" a new object anywhere in the list by updating next pointers

add(1, "Bill")

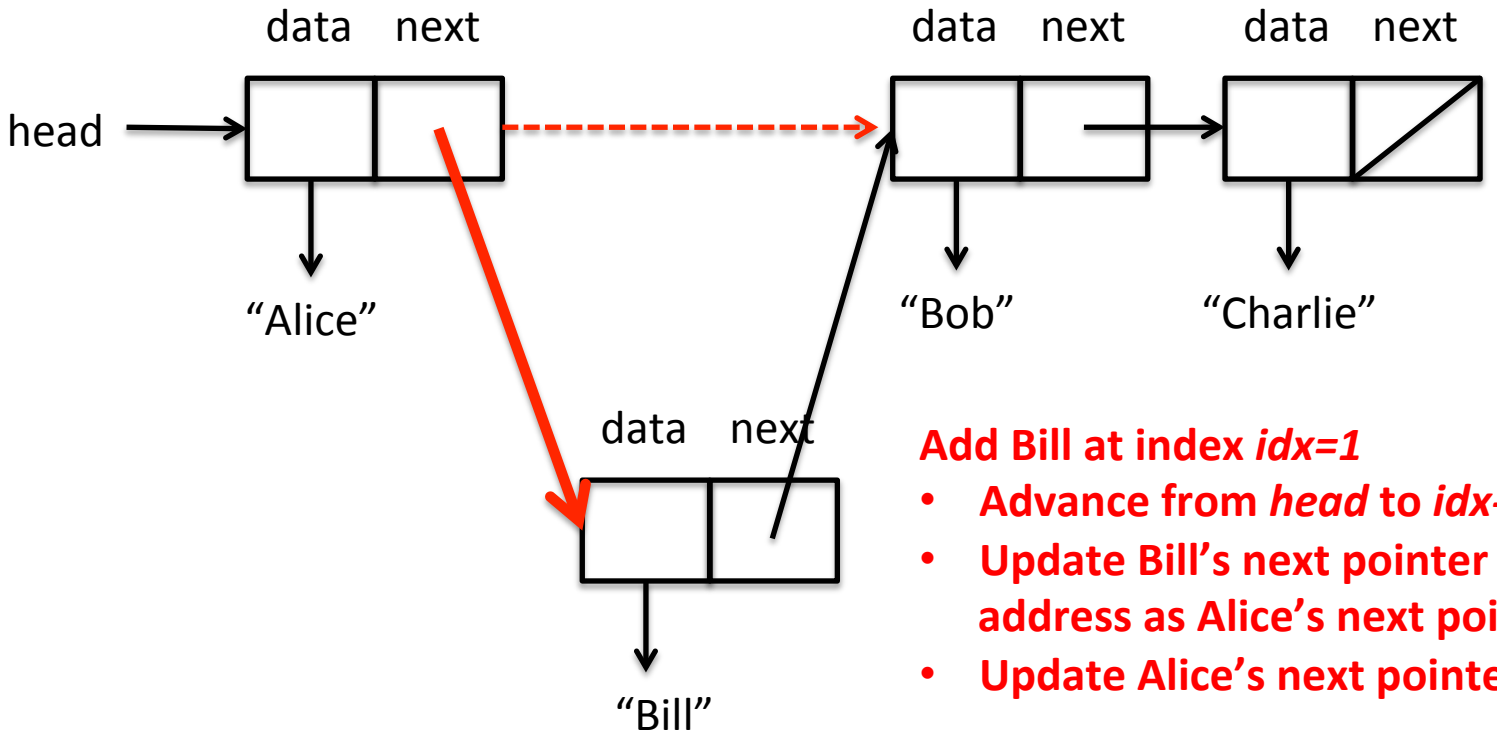


Add Bill at index *idx=1*

- **Advance from *head* to *idx-1* (Alice)**
- **Update Bill's next pointer to same address as Alice's next pointer**

add() "splices in" a new object anywhere in the list by updating next pointers

add(1, "Bill")

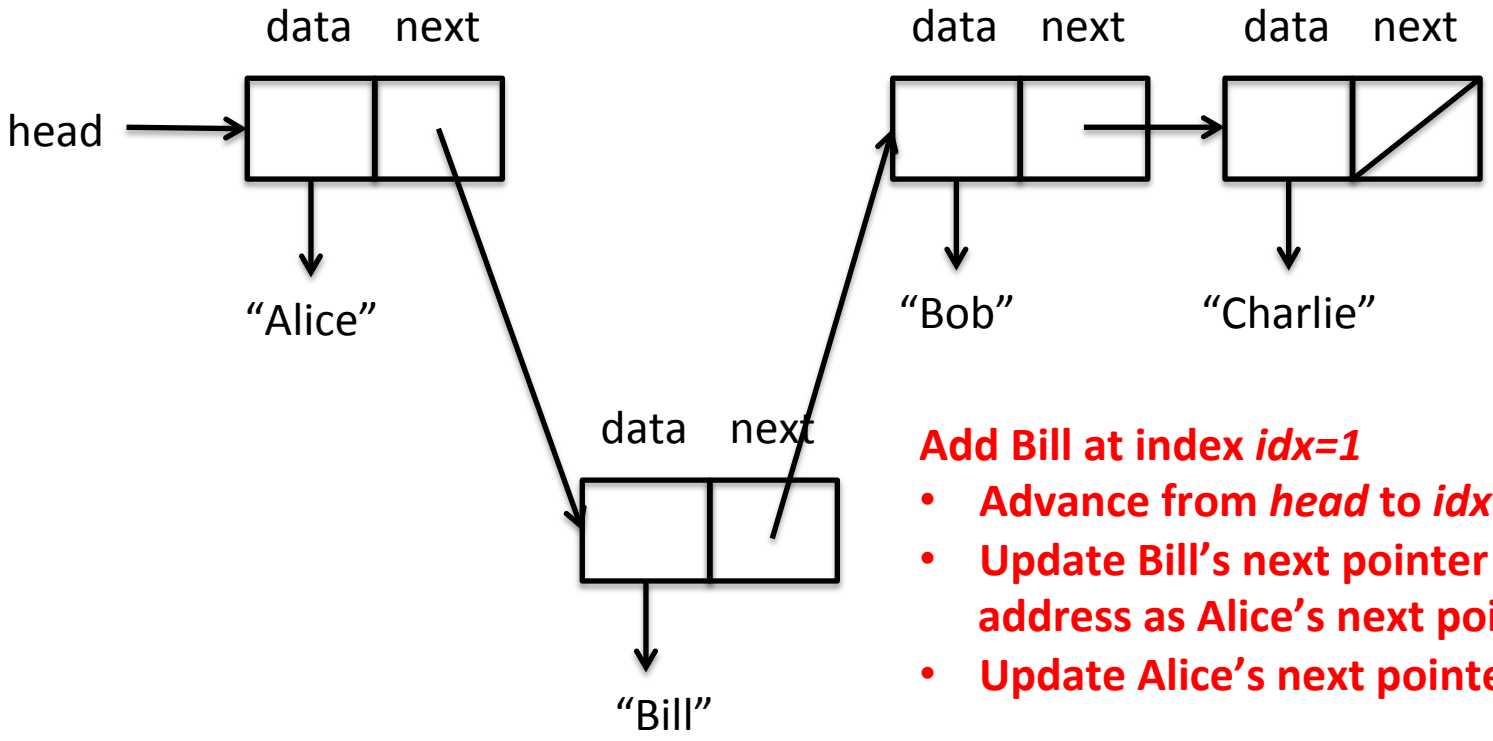


Add Bill at index *idx=1*

- **Advance from *head* to *idx-1* (Alice)**
- **Update Bill's next pointer to same address as Alice's next pointer**
- **Update Alice's next pointer to Bill**

add() "splices in" a new object anywhere in the list by updating next pointers

add(1, "Bill")

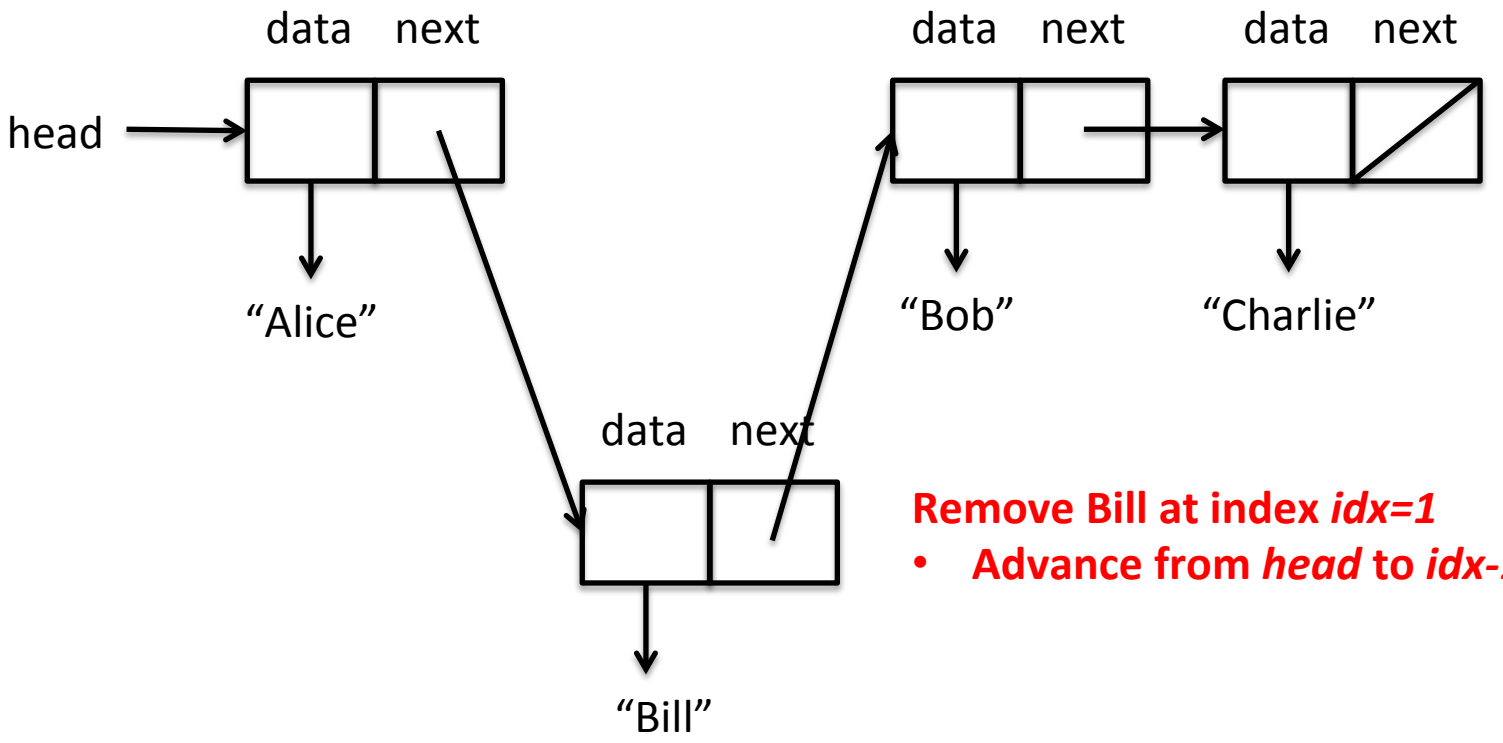


Add Bill at index *idx=1*

- **Advance from *head* to *idx-1* (Alice)**
- **Update Bill's next pointer to same address as Alice's next pointer**
- **Update Alice's next pointer to Bill**

remove() takes an item out of the list by updating next pointer

remove(1)

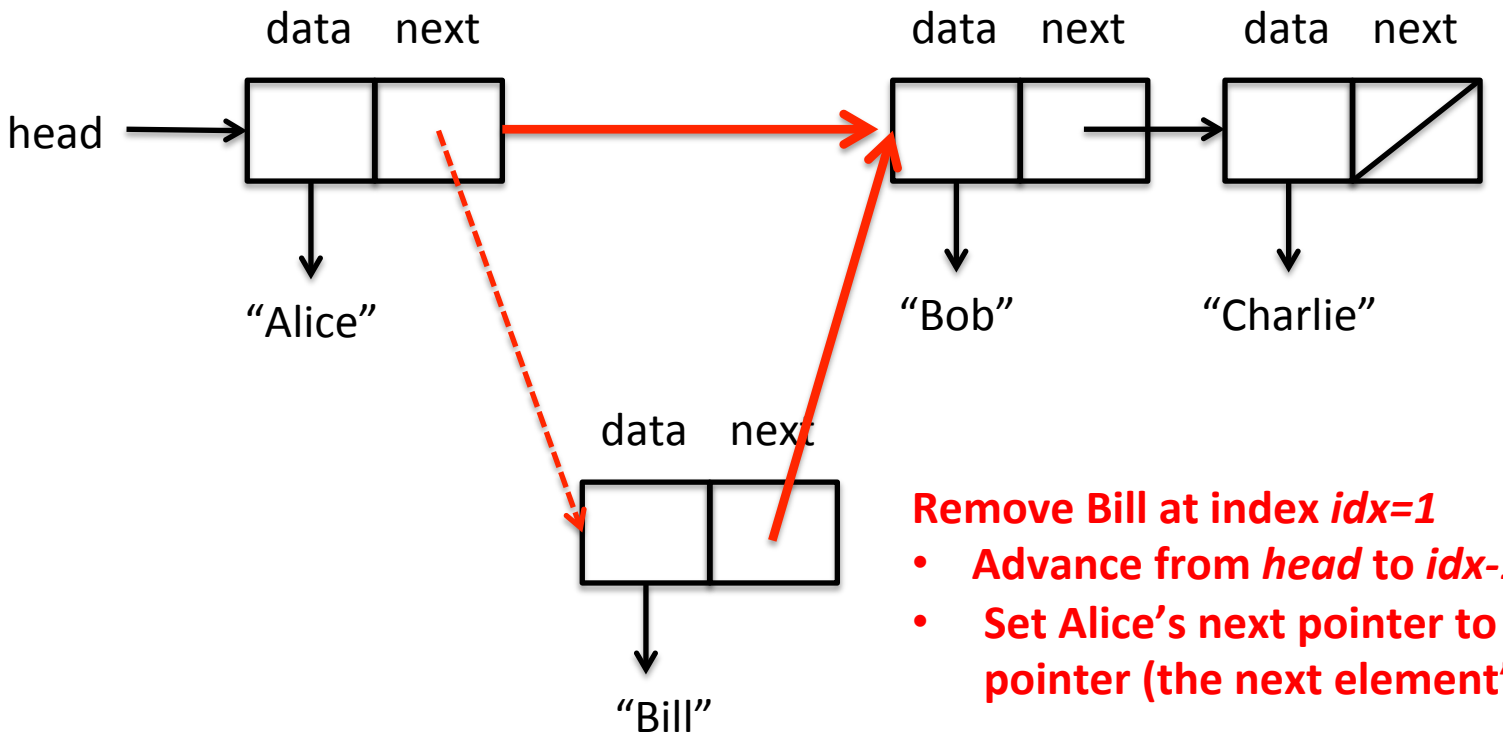


Remove Bill at index *idx=1*

- **Advance from *head* to *idx-1* (Alice)**

remove() takes an item out of the list by updating next pointer

remove(1)

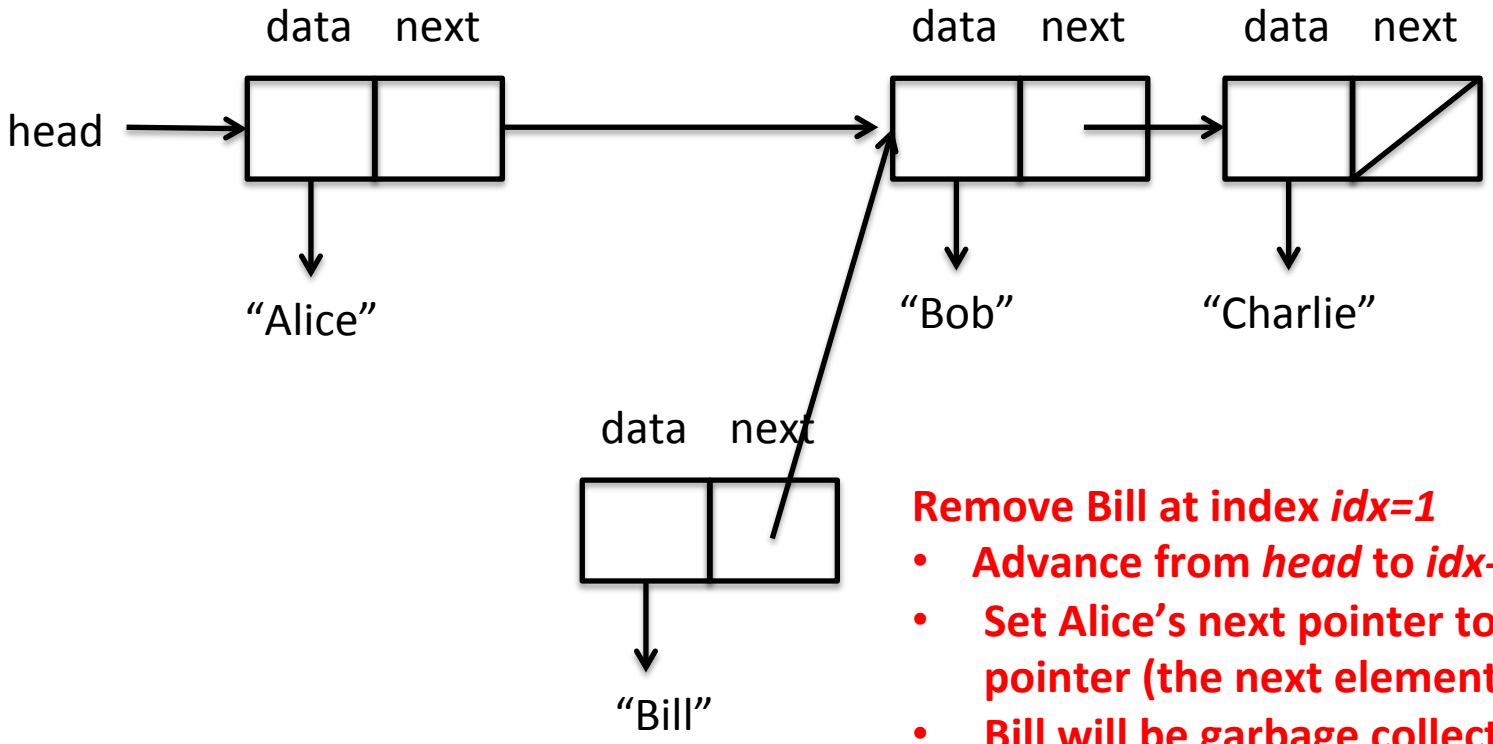


Remove Bill at index *idx=1*

- **Advance from *head* to *idx-1* (Alice)**
- **Set Alice's next pointer to Bill's next pointer (the next element's next)**

remove() takes an item out of the list by updating next pointer

remove(1)



Remove Bill at index *idx=1*

- **Advance from *head* to *idx-1* (Alice)**
- **Set Alice's next pointer to Bill's next pointer (the next element's next)**
- **Bill will be garbage collected (in C we have to call *free()*)**

SinglyLinked.java: Implementation of List interface

```
 8 public class SinglyLinked<T> implements SimpleList<T> {
 9     private Element head;    // front of the linked list
10     private int size;        // # elements in the list
11
12     /**
13      * The linked elements in the list: each has a piece of data and
14      */
15     private class Element {
16         private T data;
17         private Element next;
18
19         private Element(T data, Element next) {
20             this.data = data;
21             this.next = next;
22         }
23     }
24
25     public SinglyLinked() {
26         head = null;
27         size = 0;
28     }
29
30     public int size() {
31         return size;
32     }
33
34     /**
35      * Helper function, advancing to the nth Element in the list and
36      * (exception if not that many elements)
37      */
38     private Element advance(int n) throws Exception {
39         Element e = head;
40         while (n > 0) {
41             // Just follow the next pointers
42             e = e.next;
43             if (e == null) throw new Exception("invalid index");
44             n--;
45         }
46         return e;
47     }
48
```

"implements" is a promise to implement all required methods specified by Interface SimpleList

- ***isEmpty()***
- ***size()***
- ***add()***
- ***remove()***
- ***get()***
- ***set()***

SinglyLinked.java: Implementation of List interface

```
8  public class SinglyLinked<T> implements SimpleList<T> {
9      private Element head;    // front of the linked list
10     private int size;        // # elements in the list
11
12     /**
13      * The linked elements in the list: each has a piece of data and
14      */
15     private class Element {
16         private T data;
17         private Element next;
18
19         private Element(T data, Element next) {
20             this.data = data;
21             this.next = next;
22         }
23     }
24
25     public SinglyLinked() {
26         head = null;
27         size = 0;
28     }
29
30     public int size() {
31         return size;
32     }
33
34     /**
35      * Helper function, advancing to the nth Element in the list and
36      * (exception if not that many elements)
37      */
38     private Element advance(int n) throws Exception {
39         Element e = head;
40         while (n > 0) {
41             // Just follow the next pointers
42             e = e.next;
43             if (e == null) throw new Exception("invalid index");
44             n--;
45         }
46         return e;
47     }
```

- **Type of data is generic T**
- **Don't care what kind of data the List holds, could be Strings, Integers, Blob Objects,…**
- **This way we don't have to write a separate implementation if use Strings as elements, and other implementation if use Integers, and third implementation if use …**
- **Just implement the List once and hold whatever data type needed for the application**

SinglyLinked.java: Implementation of List interface

```
 8 public class SinglyLinked<T> implements SimpleList<T> {
 9     private Element head;    // front of the linked list
10     private int size;        // # elements in the list
11
12     /**
13      * The linked elements in the list: each has a piece of data and
14      */
15     private class Element {
16         private T data;
17         private Element next;
18
19         private Element(T data, Element next) {
20             this.data = data;
21             this.next = next;
22         }
23     }
24
25     public SinglyLinked() {
26         head = null;
27         size = 0;
28     }
29
30     public int size() {
31         return size;
32     }
33
34     /**
35      * Helper function, advancing to the nth Element in the list and
36      * (exception if not that many elements)
37      */
38     private Element advance(int n) throws Exception {
39         Element e = head;
40         while (n > 0) {
41             // Just follow the next pointers
42             e = e.next;
43             if (e == null) throw new Exception("invalid index");
44             n--;
45         }
46         return e;
47     }
```

- **Define a private class called *Element* to implement *data* and *next* pointers**
- ***Element* constructor takes *data* as type T and pointer to next *Element* (could be null)**
- ***Element* is private to SinglyLinked (internal to this file, no need for others to change it)**

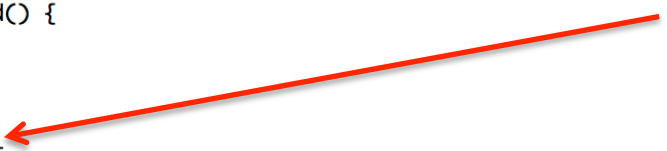
SinglyLinked.java: Implementation of List interface

```
 8 public class SinglyLinked<T> implements SimpleList<T> {
 9     private Element head;    // front of the linked list
10     private int size;        // # elements in the list
11
12     /**
13      * The linked elements in the list: each has a piece of data and
14      */
15     private class Element {
16         private T data;
17         private Element next;
18
19         private Element(T data, Element next) {
20             this.data = data;
21             this.next = next;
22         }
23     }
24
25     public SinglyLinked() {
26         head = null;
27         size = 0;
28     }
29
30     public int size() {
31         return size;
32     }
33
34     /**
35      * Helper function, advancing to the nth Element in the list and
36      * (exception if not that many elements)
37      */
38     private Element advance(int n) throws Exception {
39         Element e = head;
40         while (n > 0) {
41             // Just follow the next pointers
42             e = e.next;
43             if (e == null) throw new Exception("invalid index");
44             n--;
45         }
46         return e;
47     }
```

- **Creates *head* Element and *size* counter**
- **Constructor initializes *head* to null and *size* to 0**
- **Notice *head* is of type Element but is never "newed"**
- ***head* will be a pointer to first *Element* in the List**

SinglyLinked.java: Implementation of List interface

```
 8 public class SinglyLinked<T> implements SimpleList<T> {
 9     private Element head;    // front of the linked list
10     private int size;        // # elements in the list
11
12     /**
13      * The linked elements in the list: each has a piece of data and
14      */
15     private class Element {
16         private T data;
17         private Element next;
18
19         private Element(T data, Element next) {
20             this.data = data;
21             this.next = next;
22         }
23     }
24
25     public SinglyLinked() {
26         head = null;
27         size = 0;
28     }
29
30     public int size() {
31         return size;
32     }
33
34     /**
35      * Helper function, advancing to the nth Element in the list and
36      * (exception if not that many elements)
37      */
38     private Element advance(int n) throws Exception {
39         Element e = head;
40         while (n > 0) {
41             // Just follow the next pointers
42             e = e.next;
43             if (e == null) throw new Exception("invalid index");
44             n--;
45         }
46         return e;
47     }
```



- ***size()* method just returns instance variable *size***
- ***size* will be incremented on *add()*, decremented on *remove()***

SinglyLinked.java: Implementation of List interface

```
 8 public class SinglyLinked<T> implements SimpleList<T> {
 9     private Element head;    // front of the linked list
10     private int size;        // # elements in the list
11
12     /**
13      * The linked elements in the list: each has a piece of data and
14      */
15     private class Element {
16         private T data;
17         private Element next;
18
19         private Element(T data, Element next) {
20             this.data = data;
21             this.next = next;
22         }
23     }
24
25     public SinglyLinked() {
26         head = null;
27         size = 0;
28     }
29
30     public int size() {
31         return size;
32     }
33
34     /**
35      * Helper function, advancing to the nth Element in the list and
36      * (exception if not that many elements)
37      */
38     private Element advance(int n) throws Exception {
39         Element e = head;
40         while (n > 0) {
41             // Just follow the next pointers
42             e = e.next;
43             if (e == null) throw new Exception("invalid index");
44             n--;
45         }
46         return e;
47     }
48
```

- advance() helper method
- Start at *head and* marches down *n* items (*e* not new'ed)
- Loop until hit n^{th} item or run out of items in List
- Return n^{th} item (or throw exception)
- Note: return type from *advance()* is *Element*
- *advance()* not specified by interface, but implementations can have more methods than required

SinglyLinked.java: Implementation of List interface

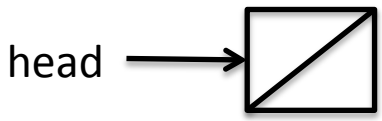
```
48
49    public void add(int idx, T item) throws Exception {
50        if (idx < 0) {
51            throw new Exception("invalid index");
52        }
53        else if (idx == 0) {
54            // Insert at head
55            head = new Element(item, head); //new item gets next pointer
56        }
57        else {
58            // It's the next thing after element # idx-1
59            Element e = advance(idx-1);
60            // Splice it in
61            e.next = new Element(item, e.next);  //create new element with
62                                                 //and prior element's nex
63        }
64        size++;
65    }
66
67    public void remove(int idx) throws Exception {
68        if (idx < 0) {
69            throw new Exception("invalid index");
70        }
71        else if (idx == 0) {
72            // Just pop off the head
73            if (head == null) throw new Exception("invalid index");
74            head = head.next;
75        }
76        else {
77            // It's the next thing after element # idx-1
78            Element e = advance(idx-1);
79            if (e.next == null) throw new Exception("invalid index");
80            // Splice it out
81            e.next = e.next.next;  //nice!
82        }
83        size--;
84    }
85
```

***add()/remove()* use *advance()* to march down list to item before index *idx*, then adjust pointers**

SinglyLinked.java: Implementation of List interface

```
48
49  public void add(int idx, T item) throws Exception {
50      if (idx < 0) {
51          throw new Exception("invalid index");
52      }
53      else if (idx == 0) {
54          // Insert at head
55          head = new Element(item, head); //new item gets next pointer
56      }
57      else {
58          // It's the next thing after element # idx-1
59          Element e = advance(idx-1);
60          // Splice it in
61          e.next = new Element(item, e.next);  //create new element with
62                                               //and prior element's nex
63      }
64      size++;
65  }
66
67  public void remove(int idx) throws Exception {
68      if (idx < 0) {
69          throw new Exception("invalid index");
70      }
71      else if (idx == 0) {
72          // Just pop off the head
73          if (head == null) throw new Exception("invalid index");
74          head = head.next;
75      }
76      else {
77          // It's the next thing after element # idx-1
78          Element e = advance(idx-1);
79          if (e.next == null) throw new Exception("invalid index");
80          // Splice it out
81          e.next = e.next.next;  //nice!
82      }
83      size--;
84  }
```

Safety check for negative index

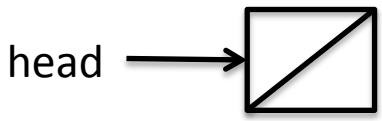


SinglyLinked.java: Implementation of List interface

```
48
49  public void add(int idx, T item) throws Exception {
50      if (idx < 0) {
51          throw new Exception("invalid index");
52      }
53      else if (idx == 0) {
54          // Insert at head
55          head = new Element(item, head); //new item gets next pointer
56      }
57      else {
58          // It's the next thing after element # idx-1
59          Element e = advance(idx-1);
60          // Splice it in
61          e.next = new Element(item, e.next);  //create new element with
62                                               //and prior element's nex
63      }
64      size++;
65  }
66
67  public void remove(int idx) throws Exception {
68      if (idx < 0) {
69          throw new Exception("invalid index");
70      }
71      else if (idx == 0) {
72          // Just pop off the head
73          if (head == null) throw new Exception("invalid index");
74          head = head.next;
75      }
76      else {
77          // It's the next thing after element # idx-1
78          Element e = advance(idx-1);
79          if (e.next == null) throw new Exception("invalid index");
80          // Splice it out
81          e.next = e.next.next;  //nice!
82      }
83      size--;
84  }
```

If adding at head (index 0)

add(0,15)



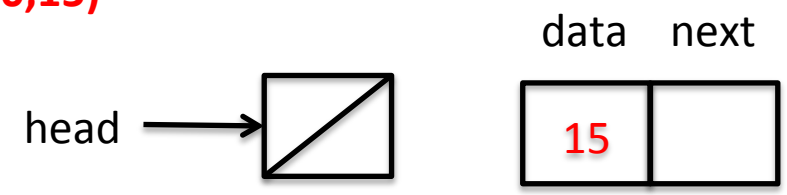
SinglyLinked.java: Implementation of List interface

```
48
49    public void add(int idx, T item) throws Exception {
50        if (idx < 0) {
51            throw new Exception("invalid index");
52        }
53        else if (idx == 0) {
54            // Insert at head
55            head = new Element(item, head); //new item gets next pointer
56        }
57        else {
58            // It's the next thing after element # idx-1
59            Element e = advance(idx-1);
60            // Splice it in
61            e.next = new Element(item, e.next);  //create new element with
62                                                 //and prior element's nex
63        }
64        size++;
65    }
66
67    public void remove(int idx) throws Exception {
68        if (idx < 0) {
69            throw new Exception("invalid index");
70        }
71        else if (idx == 0) {
72            // Just pop off the head
73            if (head == null) throw new Exception("invalid index");
74            head = head.next;
75        }
76        else {
77            // It's the next thing after element # idx-1
78            Element e = advance(idx-1);
79            if (e.next == null) throw new Exception("invalid index");
80            // Splice it out
81            e.next = e.next.next;  //nice!
82        }
83        size--;
84    }
```

If adding at head (index 0)

- **Create new element with data set to parameter *item***

add(0,15)



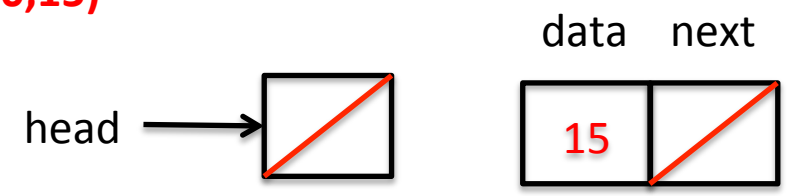
SinglyLinked.java: Implementation of List interface

```
48
49    public void add(int idx, T item) throws Exception {
50        if (idx < 0) {
51            throw new Exception("invalid index");
52        }
53        else if (idx == 0) {
54            // Insert at head
55            head = new Element(item, head); //new item gets next pointer
56        }
57        else {
58            // It's the next thing after element # idx-1
59            Element e = advance(idx-1);
60            // Splice it in
61            e.next = new Element(item, e.next);  //create new element with
62                                                 //and prior element's nex
63        }
64        size++;
65    }
66
67    public void remove(int idx) throws Exception {
68        if (idx < 0) {
69            throw new Exception("invalid index");
70        }
71        else if (idx == 0) {
72            // Just pop off the head
73            if (head == null) throw new Exception("invalid index");
74            head = head.next;
75        }
76        else {
77            // It's the next thing after element # idx-1
78            Element e = advance(idx-1);
79            if (e.next == null) throw new Exception("invalid index");
80            // Splice it out
81            e.next = e.next.next;  //nice!
82        }
83        size--;
84    }
```

If adding at head (index 0)

- **Create new element with data set to parameter *item***
- **Set new element next pointer to where ever *head* points**

add(0,15)



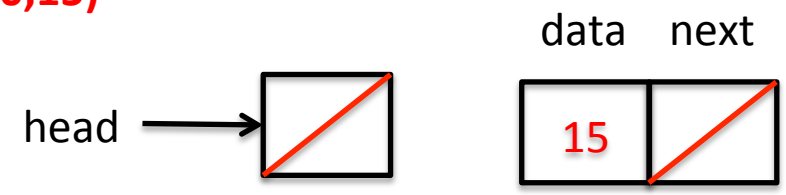
SinglyLinked.java: Implementation of List interface

```
48
49    public void add(int idx, T item) throws Exception {
50        if (idx < 0) {
51            throw new Exception("invalid index");
52        }
53        else if (idx == 0) {
54            // Insert at head
55            head = new Element(item, head); //new item gets next pointer
56        }
57        else {
58            // It's the next thing after element # idx-1
59            Element e = advance(idx-1);
60            // Splice it in
61            e.next = new Element(item, e.next);  //create new element with
62                                                 //and prior element's nex
63        }
64        size++;
65    }
66
67    public void remove(int idx) throws Exception {
68        if (idx < 0) {
69            throw new Exception("invalid index");
70        }
71        else if (idx == 0) {
72            // Just pop off the head
73            if (head == null) throw new Exception("invalid index");
74            head = head.next;
75        }
76        else {
77            // It's the next thing after element # idx-1
78            Element e = advance(idx-1);
79            if (e.next == null) throw new Exception("invalid index");
80            // Splice it out
81            e.next = e.next.next;  //nice!
82        }
83        size--;
84    }
```

If adding at head (index 0)

- **Create new element with data set to parameter *item***
- **Set new element next pointer to where ever *head* points**
- ***head* will initially point to null**

add(0,15)



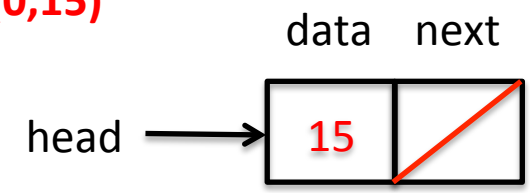
SinglyLinked.java: Implementation of List interface

```
48
49    public void add(int idx, T item) throws Exception {
50        if (idx < 0) {
51            throw new Exception("invalid index");
52        }
53        else if (idx == 0) {
54            // Insert at head
55            head = new Element(item, head); //new item gets next pointer
56        }
57        else {
58            // It's the next thing after element # idx-1
59            Element e = advance(idx-1);
60            // Splice it in
61            e.next = new Element(item, e.next);  //create new element with
62                                                 //and prior element's nex
63        }
64        size++;
65    }
66
67    public void remove(int idx) throws Exception {
68        if (idx < 0) {
69            throw new Exception("invalid index");
70        }
71        else if (idx == 0) {
72            // Just pop off the head
73            if (head == null) throw new Exception("invalid index");
74            head = head.next;
75        }
76        else {
77            // It's the next thing after element # idx-1
78            Element e = advance(idx-1);
79            if (e.next == null) throw new Exception("invalid index");
80            // Splice it out
81            e.next = e.next.next;  //nice!
82        }
83        size--;
84    }
```

If adding at head (index 0)

- **Create new element with data set to parameter *item***
- **Set new element next pointer to where ever *head* points**
- ***head* will initially point to null**
- **Set *head* to new element**

add(0,15)



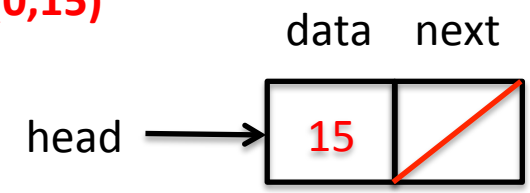
SinglyLinked.java: Implementation of List interface

```
48
49    public void add(int idx, T item) throws Exception {
50        if (idx < 0) {
51            throw new Exception("invalid index");
52        }
53        else if (idx == 0) {
54            // Insert at head
55            head = new Element(item, head); //new item gets next pointer
56        }
57        else {
58            // It's the next thing after element # idx-1
59            Element e = advance(idx-1);
60            // Splice it in
61            e.next = new Element(item, e.next);  //create new element with
62                                                 //and prior element's nex
63        }
64        size++;
65    }
66
67    public void remove(int idx) throws Exception {
68        if (idx < 0) {
69            throw new Exception("invalid index");
70        }
71        else if (idx == 0) {
72            // Just pop off the head
73            if (head == null) throw new Exception("invalid index");
74            head = head.next;
75        }
76        else {
77            // It's the next thing after element # idx-1
78            Element e = advance(idx-1);
79            if (e.next == null) throw new Exception("invalid index");
80            // Splice it out
81            e.next = e.next.next;  //nice!
82        }
83        size--;
84    }
```

If adding at head (index 0)

- **Create new element with data set to parameter *item***
- **Set new element next pointer to where ever *head* points**
- ***head* will initially point to null**
- **Set *head* to new element**
- **Finally increment *size***

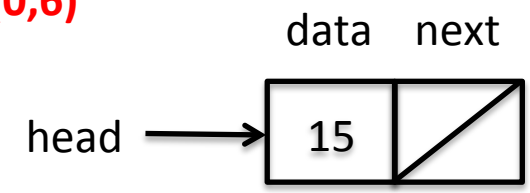
add(0,15)



SinglyLinked.java: Implementation of List interface

```
48
49    public void add(int idx, T item) throws Exception {
50        if (idx < 0) {
51            throw new Exception("invalid index");
52        }
53        else if (idx == 0) {
54            // Insert at head
55            head = new Element(item, head); //new item gets next pointer
56        }
57        else {
58            // It's the next thing after element # idx-1
59            Element e = advance(idx-1);
60            // Splice it in
61            e.next = new Element(item, e.next);  //create new element with
62                                                 //and prior element's nex
63        }
64        size++;
65    }
66
67    public void remove(int idx) throws Exception {
68        if (idx < 0) {
69            throw new Exception("invalid index");
70        }
71        else if (idx == 0) {
72            // Just pop off the head
73            if (head == null) throw new Exception("invalid index");
74            head = head.next;
75        }
76        else {
77            // It's the next thing after element # idx-1
78            Element e = advance(idx-1);
79            if (e.next == null) throw new Exception("invalid index");
80            // Splice it out
81            e.next = e.next.next;  //nice!
82        }
83        size--;
84    }
```

add(0,6)



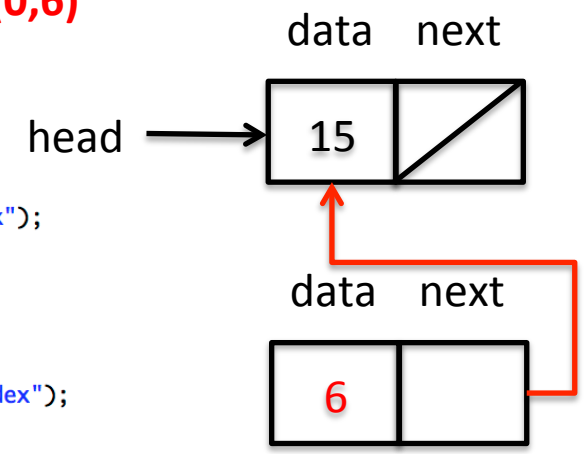
SinglyLinked.java: Implementation of List interface

```
48
49    public void add(int idx, T item) throws Exception {
50        if (idx < 0) {
51            throw new Exception("invalid index");
52        }
53        else if (idx == 0) {
54            // Insert at head
55            head = new Element(item, head); //new item gets next pointer
56        }
57        else {
58            // It's the next thing after element # idx-1
59            Element e = advance(idx-1);
60            // Splice it in
61            e.next = new Element(item, e.next);  //create new element with
62                                                 //and prior element's nex
63        }
64        size++;
65    }
66
67    public void remove(int idx) throws Exception {
68        if (idx < 0) {
69            throw new Exception("invalid index");
70        }
71        else if (idx == 0) {
72            // Just pop off the head
73            if (head == null) throw new Exception("invalid index");
74            head = head.next;
75        }
76        else {
77            // It's the next thing after element # idx-1
78            Element e = advance(idx-1);
79            if (e.next == null) throw new Exception("invalid index");
80            // Splice it out
81            e.next = e.next.next;  //nice!
82        }
83        size--;
84    }
85
```

If adding at head (index 0)

- **Create new element with data set to parameter *item***
- **Set new element next pointer to where ever *head* points**

add(0,6)



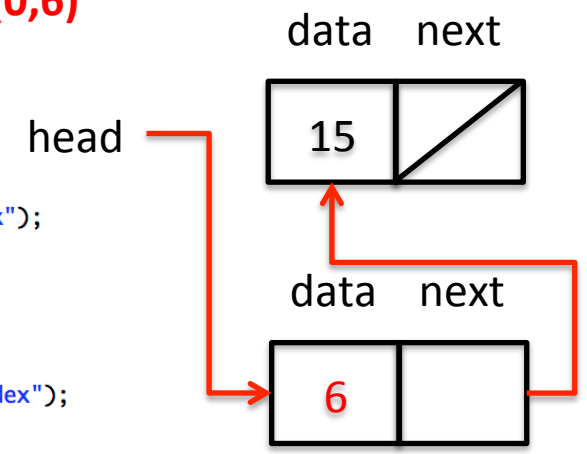
SinglyLinked.java: Implementation of List interface

```
48
49    public void add(int idx, T item) throws Exception {
50        if (idx < 0) {
51            throw new Exception("invalid index");
52        }
53        else if (idx == 0) {
54            // Insert at head
55            head = new Element(item, head); //new item gets next pointer
56        }
57        else {
58            // It's the next thing after element # idx-1
59            Element e = advance(idx-1);
60            // Splice it in
61            e.next = new Element(item, e.next);  //create new element with
62                                                 //and prior element's nex
63        }
64        size++;
65    }
66
67    public void remove(int idx) throws Exception {
68        if (idx < 0) {
69            throw new Exception("invalid index");
70        }
71        else if (idx == 0) {
72            // Just pop off the head
73            if (head == null) throw new Exception("invalid index");
74            head = head.next;
75        }
76        else {
77            // It's the next thing after element # idx-1
78            Element e = advance(idx-1);
79            if (e.next == null) throw new Exception("invalid index");
80            // Splice it out
81            e.next = e.next.next;  //nice!
82        }
83        size--;
84    }
```

If adding at head (index 0)

- **Create new element with data set to parameter *item***
- **Set new element next pointer to where ever *head* points**
- **Set *head* to new element**
- **Finally increment *size***

add(0,6)



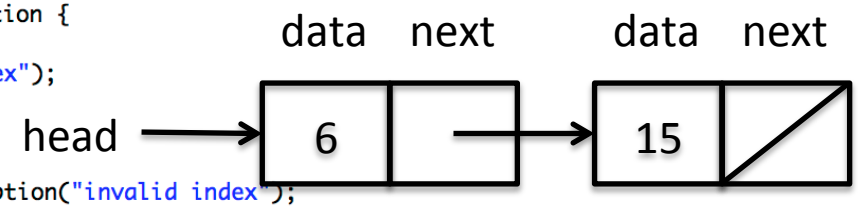
SinglyLinked.java: Implementation of List interface

```
48
49    public void add(int idx, T item) throws Exception {
50        if (idx < 0) {
51            throw new Exception("invalid index");
52        }
53        else if (idx == 0) {
54            // Insert at head
55            head = new Element(item, head); //new item gets next pointer
56        }
57        else {
58            // It's the next thing after element # idx-1
59            Element e = advance(idx-1);
60            // Splice it in
61            e.next = new Element(item, e.next);  //create new element with
62                                                 //and prior element's nex
63        }
64        size++;
65    }
66
67    public void remove(int idx) throws Exception {
68        if (idx < 0) {
69            throw new Exception("invalid index");
70        }
71        else if (idx == 0) {
72            // Just pop off the head
73            if (head == null) throw new Exception("invalid index");
74            head = head.next;
75        }
76        else {
77            // It's the next thing after element # idx-1
78            Element e = advance(idx-1);
79            if (e.next == null) throw new Exception("invalid index");
80            // Splice it out
81            e.next = e.next.next;  //nice!
82        }
83        size--;
84    }
```

If adding not at head

- ***advance*() to *e=(idx-1)th* item**

add(1,3)



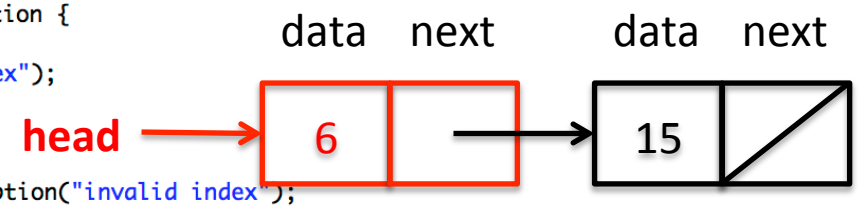
SinglyLinked.java: Implementation of List interface

```
48
49    public void add(int idx, T item) throws Exception {
50        if (idx < 0) {
51            throw new Exception("invalid index");
52        }
53        else if (idx == 0) {
54            // Insert at head
55            head = new Element(item, head); //new item gets next pointer
56        }
57        else {
58            // It's the next thing after element # idx-1
59            Element e = advance(idx-1);
60            // Splice it in
61            e.next = new Element(item, e.next);  //create new element with
62                                                 //and prior element's nex
63        }
64        size++;
65    }
66
67    public void remove(int idx) throws Exception {
68        if (idx < 0) {
69            throw new Exception("invalid index");
70        }
71        else if (idx == 0) {
72            // Just pop off the head
73            if (head == null) throw new Exception("invalid index");
74            head = head.next;
75        }
76        else {
77            // It's the next thing after element # idx-1
78            Element e = advance(idx-1);
79            if (e.next == null) throw new Exception("invalid index");
80            // Splice it out
81            e.next = e.next.next;  //nice!
82        }
83        size--;
84    }
85
```

If adding not at head

- ***advance*() to *e*=(*idx-1*)th item**

add(1,3)



Advance to item idx-1
Here (item 1)-1 = item 0
So *e* points to item 0 with data = 6

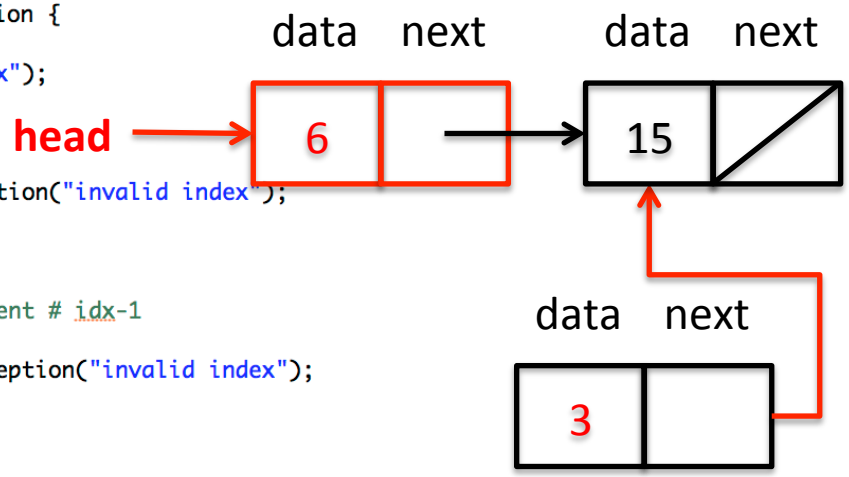
SinglyLinked.java: Implementation of List interface

```
48
49    public void add(int idx, T item) throws Exception {
50        if (idx < 0) {
51            throw new Exception("invalid index");
52        }
53        else if (idx == 0) {
54            // Insert at head
55            head = new Element(item, head); //new item gets next pointer
56        }
57        else {
58            // It's the next thing after element # idx-1
59            Element e = advance(idx-1);
60            // Splice it in
61            e.next = new Element(item, e.next);  //create new element with
62                                                 //and prior element's nex
63        }
64        size++;
65    }
66
67    public void remove(int idx) throws Exception {
68        if (idx < 0) {
69            throw new Exception("invalid index");
70        }
71        else if (idx == 0) {
72            // Just pop off the head
73            if (head == null) throw new Exception("invalid index");
74            head = head.next;
75        }
76        else {
77            // It's the next thing after element # idx-1
78            Element e = advance(idx-1);
79            if (e.next == null) throw new Exception("invalid index");
80            // Splice it out
81            e.next = e.next.next;  //nice!
82        }
83        size--;
84    }
```

If adding not at head

- ***advance*() to *e=(idx-1)th* item**
- **Create new *Element* with *data* set to *item* and *next* to *e.next***

add(1,3)



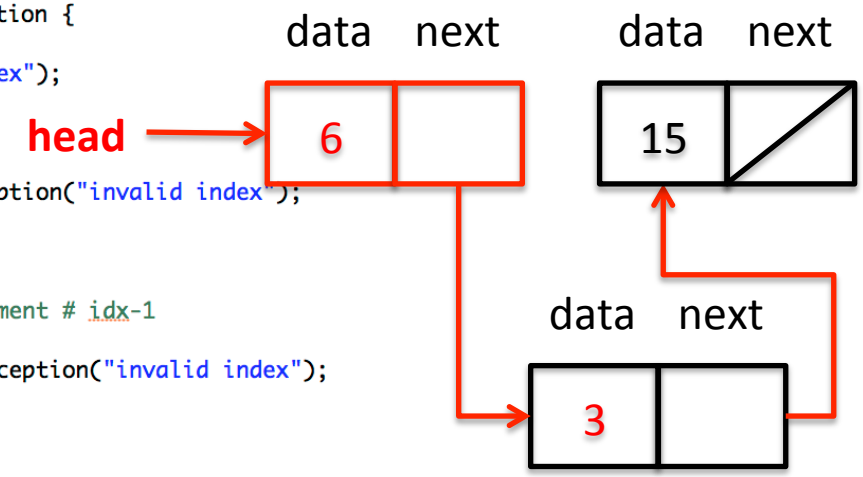
SinglyLinked.java: Implementation of List interface

```
48
49    public void add(int idx, T item) throws Exception {
50        if (idx < 0) {
51            throw new Exception("invalid index");
52        }
53        else if (idx == 0) {
54            // Insert at head
55            head = new Element(item, head); //new item gets next pointer
56        }
57        else {
58            // It's the next thing after element # idx-1
59            Element e = advance(idx-1);
60            // Splice it in
61            e.next = new Element(item, e.next);  //create new element with
62                                                 //and prior element's nex
63        }
64        size++;
65    }
66
67    public void remove(int idx) throws Exception {
68        if (idx < 0) {
69            throw new Exception("invalid index");
70        }
71        else if (idx == 0) {
72            // Just pop off the head
73            if (head == null) throw new Exception("invalid index");
74            head = head.next;
75        }
76        else {
77            // It's the next thing after element # idx-1
78            Element e = advance(idx-1);
79            if (e.next == null) throw new Exception("invalid index");
80            // Splice it out
81            e.next = e.next.next;  //nice!
82        }
83        size--;
84    }
```

If adding not at head

- ***advance*() to *e=(idx-1)th* item**
- **Create new *Element* with *data* set to *item* and *next* to *e.next***
- **Set *e.next* = new item**

add(1,3)



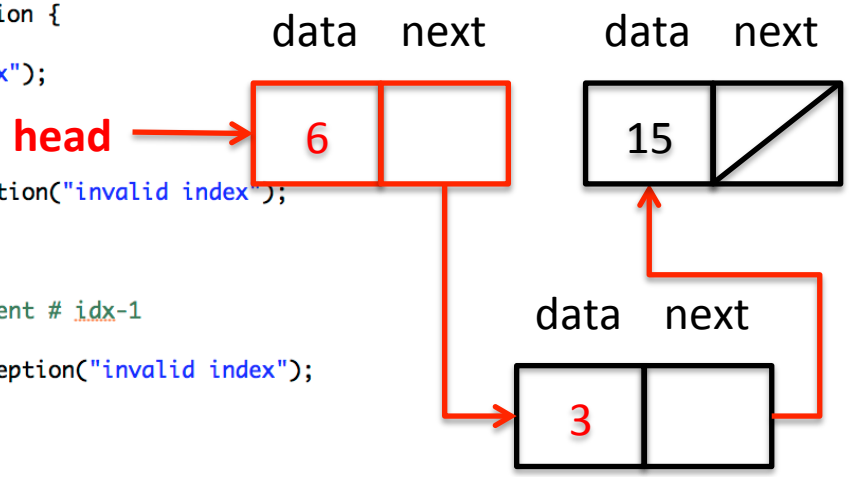
SinglyLinked.java: Implementation of List interface

```
48
49    public void add(int idx, T item) throws Exception {
50        if (idx < 0) {
51            throw new Exception("invalid index");
52        }
53        else if (idx == 0) {
54            // Insert at head
55            head = new Element(item, head); //new item gets next pointer
56        }
57        else {
58            // It's the next thing after element # idx-1
59            Element e = advance(idx-1);
60            // Splice it in
61            e.next = new Element(item, e.next);  //create new element with
62                                                 //and prior element's nex
63        }
64        size++;
65    }
66
67    public void remove(int idx) throws Exception {
68        if (idx < 0) {
69            throw new Exception("invalid index");
70        }
71        else if (idx == 0) {
72            // Just pop off the head
73            if (head == null) throw new Exception("invalid index");
74            head = head.next;
75        }
76        else {
77            // It's the next thing after element # idx-1
78            Element e = advance(idx-1);
79            if (e.next == null) throw new Exception("invalid index");
80            // Splice it out
81            e.next = e.next.next;  //nice!
82        }
83        size--;
84    }
```

If adding not at head

- ***advance*() to *e=(idx-1)th* item**
- **Create new *Element* with *data* set to *item* and *next* to *e.next***
- **Set *e.next* = new item**
- **Finally, increment *size***

add(1,3)



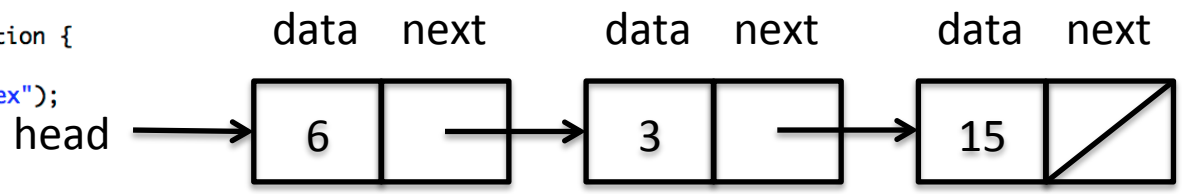
SinglyLinked.java: Implementation of List interface

```
48
49    public void add(int idx, T item) throws Exception {
50        if (idx < 0) {
51            throw new Exception("invalid index");
52        }
53        else if (idx == 0) {
54            // Insert at head
55            head = new Element(item, head); //new item gets next pointer
56        }
57        else {
58            // It's the next thing after element # idx-1
59            Element e = advance(idx-1);
60            // Splice it in
61            e.next = new Element(item, e.next);  //create new element with
62                                                 //and prior element's nex
63        }
64        size++;
65    }
66
67    public void remove(int idx) throws Exception {
68        if (idx < 0) {
69            throw new Exception("invalid index");
70        }
71        else if (idx == 0) {
72            // Just pop off the head
73            if (head == null) throw new Exception("invalid index");
74            head = head.next;
75        }
76        else {
77            // It's the next thing after element # idx-1
78            Element e = advance(idx-1);
79            if (e.next == null) throw new Exception("invalid index");
80            // Splice it out
81            e.next = e.next.next;  //nice!
82        }
83        size--;
84    }
```

If removing at *head*

- **Just set *head* to next**

remove(0)



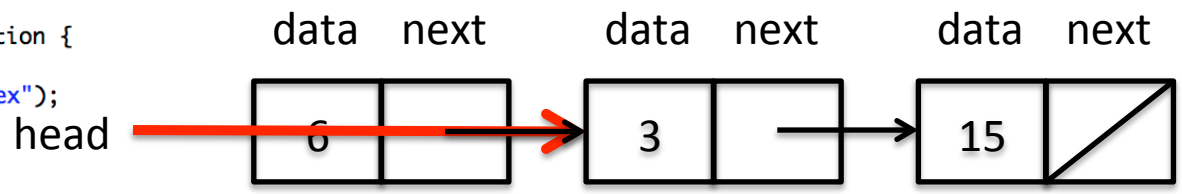
SinglyLinked.java: Implementation of List interface

```
48
49    public void add(int idx, T item) throws Exception {
50        if (idx < 0) {
51            throw new Exception("invalid index");
52        }
53        else if (idx == 0) {
54            // Insert at head
55            head = new Element(item, head); //new item gets next pointer
56        }
57        else {
58            // It's the next thing after element # idx-1
59            Element e = advance(idx-1);
60            // Splice it in
61            e.next = new Element(item, e.next);  //create new element with
62                                                 //and prior element's nex
63        }
64        size++;
65    }
66
67    public void remove(int idx) throws Exception {
68        if (idx < 0) {
69            throw new Exception("invalid index");
70        }
71        else if (idx == 0) {
72            // Just pop off the head
73            if (head == null) throw new Exception("invalid index");
74            head = head.next;
75        }
76        else {
77            // It's the next thing after element # idx-1
78            Element e = advance(idx-1);
79            if (e.next == null) throw new Exception("invalid index");
80            // Splice it out
81            e.next = e.next.next;  //nice!
82        }
83        size--;
84    }
```

If removing at *head*

- **Just set *head* to next**

remove(0)



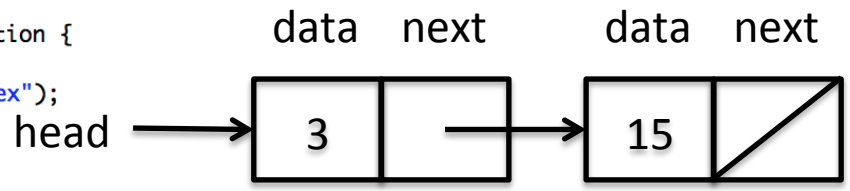
SinglyLinked.java: Implementation of List interface

```
48
49    public void add(int idx, T item) throws Exception {
50        if (idx < 0) {
51            throw new Exception("invalid index");
52        }
53        else if (idx == 0) {
54            // Insert at head
55            head = new Element(item, head); //new item gets next pointer
56        }
57        else {
58            // It's the next thing after element # idx-1
59            Element e = advance(idx-1);
60            // Splice it in
61            e.next = new Element(item, e.next);  //create new element with
62                                                 //and prior element's nex
63        }
64        size++;
65    }
66
67    public void remove(int idx) throws Exception {
68        if (idx < 0) {
69            throw new Exception("invalid index");
70        }
71        else if (idx == 0) {
72            // Just pop off the head
73            if (head == null) throw new Exception("invalid index");
74            head = head.next;
75        }
76        else {
77            // It's the next thing after element # idx-1
78            Element e = advance(idx-1);
79            if (e.next == null) throw new Exception("invalid index");
80            // Splice it out
81            e.next = e.next.next;  //nice!
82        }
83        size--;
84    }
```

If removing at *head*

- **Just set *head* to next**

remove(0)



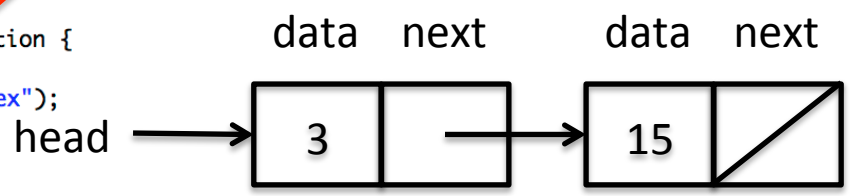
SinglyLinked.java: Implementation of List interface

```
48
49    public void add(int idx, T item) throws Exception {
50        if (idx < 0) {
51            throw new Exception("invalid index");
52        }
53        else if (idx == 0) {
54            // Insert at head
55            head = new Element(item, head); //new item gets next pointer
56        }
57        else {
58            // It's the next thing after element # idx-1
59            Element e = advance(idx-1);
60            // Splice it in
61            e.next = new Element(item, e.next);  //create new element with
62                                                 //and prior element's nex
63        }
64        size++;
65    }
66
67    public void remove(int idx) throws Exception {
68        if (idx < 0) {
69            throw new Exception("invalid index");
70        }
71        else if (idx == 0) {
72            // Just pop off the head
73            if (head == null) throw new Exception("invalid index");
74            head = head.next;
75        }
76        else {
77            // It's the next thing after element # idx-1
78            Element e = advance(idx-1);
79            if (e.next == null) throw new Exception("invalid index");
80            // Splice it out
81            e.next = e.next.next;  //nice!
82        }
83        size--;
84    }
```

If removing not at *head*

- ***advance()* to *idx-1* (data 3)**
- ***Set e.next to e.next.next***

remove(1)



SinglyLinked.java: Implementation of List interface

```
48
49    public void add(int idx, T item) throws Exception {
50        if (idx < 0) {
51            throw new Exception("invalid index");
52        }
53        else if (idx == 0) {
54            // Insert at head
55            head = new Element(item, head); //new item gets next pointer
56        }
57        else {
58            // It's the next thing after element # idx-1
59            Element e = advance(idx-1);
60            // Splice it in
61            e.next = new Element(item, e.next);  //create new element with
62                                                 //and prior element's nex
63        }
64        size++;
65    }
66
67    public void remove(int idx) throws Exception {
68        if (idx < 0) {
69            throw new Exception("invalid index");
70        }
71        else if (idx == 0) {
72            // Just pop off the head
73            if (head == null) throw new Exception("invalid index");
74            head = head.next;
75        }
76        else {
77            // It's the next thing after element # idx-1
78            Element e = advance(idx-1);
79            if (e.next == null) throw new Exception("invalid index");
80            // Splice it out
81            e.next = e.next.next;  //nice!
82        }
83        size--;
84    }
```

If removing not at *head*

- ***advance()* to *idx-1* (data 3)**
- ***Set e.next to e.next.next***

remove(1)

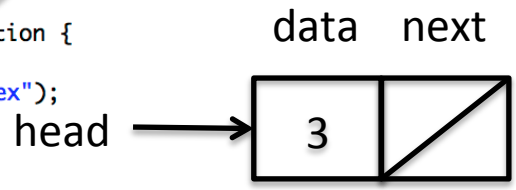
SinglyLinked.java: Implementation of List interface

```
48
49    public void add(int idx, T item) throws Exception {
50        if (idx < 0) {
51            throw new Exception("invalid index");
52        }
53        else if (idx == 0) {
54            // Insert at head
55            head = new Element(item, head); //new item gets next pointer
56        }
57        else {
58            // It's the next thing after element # idx-1
59            Element e = advance(idx-1);
60            // Splice it in
61            e.next = new Element(item, e.next);  //create new element with
62                                                 //and prior element's nex
63        }
64        size++;
65    }
66
67    public void remove(int idx) throws Exception {
68        if (idx < 0) {
69            throw new Exception("invalid index");
70        }
71        else if (idx == 0) {
72            // Just pop off the head
73            if (head == null) throw new Exception("invalid index");
74            head = head.next;
75        }
76        else {
77            // It's the next thing after element # idx-1
78            Element e = advance(idx-1);
79            if (e.next == null) throw new Exception("invalid index");
80            // Splice it out
81            e.next = e.next.next;  //nice!
82        }
83        size--;
84    }
```

If removing not at *head*

- ***advance()* to *idx-1 (data 3)***
- ***Set e.next to e.next.next***

remove(1)



SinglyLinked.java: Implementation of List interface

```
86      public T get(int idx) throws Exception {
87          if (idx < 0) {
88              throw new Exception("invalid index");
89          }
90          Element e = advance(idx);
91          return e.data;
92      }
93
94      public void set(int idx, T item) throws Exception {
95          if (idx < 0) {
96              throw new Exception("invalid index");
97          }
98          Element e = advance(idx);
99          e.data = item;
100     }
101
102     public String toString() {
103         String result = "";
104         for (Element x = head; x != null; x = x.next)
105             result += x.data + "->";
106         result += "[/]";
107
108         return result;
109     }
```

get()/set()* also use *advance()* to march down list, this time to index *idx

SinglyLinked.java: Implementation of List interface

```
86      public T get(int idx) throws Exception {
87          if (idx < 0) {
88              throw new Exception("invalid index");
89          }
90          Element e = advance(idx);
91          return e.data;
92      }
93
94      public void set(int idx, T item) throws Exception {
95          if (idx < 0) {
96              throw new Exception("invalid index");
97          }
98          Element e = advance(idx);
99          e.data = item;
100     }
101
102     public String toString() {
103         String result = "";
104         for (Element x = head; x != null; x = x.next)
105             result += x.data + "->";
106         result += "[/]";
107
108         return result;
109     }
```

get()/set()* also use *advance()* to march down list, this time to index *idx

Key point: all operations start at head and march down list

toString() overrides a Java Object method and allows us to print an object as desired

If toString() not overriden, defaults to printing the memory address of object

Return value type is String rather than actually printing

On an exam: make sure you return a String with toString(), don't print in toString()

ListTest.java: Test of List implementation

```
4 public class ListTest {
5     public static void main(String[] args) throws Exception {
6         SimpleList<String> list = new SinglyLinked<String>();
7         System.out.println(list);
8         list.add(0, "a");
9         System.out.println(list);
10        list.add(1, "c");
11        System.out.println(list);
12        list.add(1, "b");
13        System.out.println(list);
14        list.set(2, "e");
15        System.out.println(list.get(2));
16        list.add(0, "z");
17        System.out.println(list);
18        list.remove(2);
19        System.out.println(list);
20        list.remove(0);
21        System.out.println(list);
22        list.remove(1);
23        System.out.println(list);
24    }
25 }
```

Declare SinglyLinked List to hold Strings, so T = String in the implementation

Implementation is SinglyLinked which implemented SimpleList interface

Next class we'll look at an array implementation which will also be a SimpleList

Problems @ Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy

<terminated> ListTest [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Dec 30, 2017, 4:48:02 PM)

```
[/]
a->[/]
a->c->[/]
a->b->c->[/]
e
z->a->b->e->[/]
z->a->e->[/]
a->e->[/]
a->[/]
```

ListTest.java: Test of List implementation

```
4  public class ListTest {
5      public static void main(String[] args) throws Exception {
6          SimpleList<String> list = new SinglyLinked<String>();
7          System.out.println(list);
8          list.add(0, "a");
9          System.out.println(list);
10         list.add(1, "c");
11         System.out.println(list);
12         list.add(1, "b");
13         System.out.println(list);
14         list.set(2, "e");
15         System.out.println(list.get(2));
16         list.add(0, "z");
17         System.out.println(list);
18         list.remove(2);
19         System.out.println(list);
20         list.remove(0);
21         System.out.println(list);
22         list.remove(1);
23         System.out.println(list);
24     }
25 }
```

toString() method called in print statements

Problems @ Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy

<terminated> ListTest [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Dec 30, 2017, 4:48:02 PM)

```
[/]
a->[/]
a->c->[/]
a->b->c->[/]
e
z->a->b->e->[/]
z->a->e->[/]
a->e->[/]
a->[/]
```

ListTest.java: Test of List implementation

```
4 public class ListTest {
5     public static void main(String[] args) throws Exception {
6         SimpleList<String> list = new SinglyLinked<String>();
7         System.out.println(list);
8         list.add(0, "a");
9         System.out.println(list);
10        list.add(1, "c");
11        System.out.println(list);
12        list.add(1, "b");
13        System.out.println(list);
14        list.set(2, "e");
15        System.out.println(list.get(2));
16        list.add(0, "z");
17        System.out.println(list);
18        list.remove(2);
19        System.out.println(list);
20        list.remove(0);
21        System.out.println(list);
22        list.remove(1);
23        System.out.println(list);
24    }
25 }
```

toString() method called in print statements

Remember, toString() returns a String (doesn't do the printing)

Problems @ Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy

<terminated> ListTest [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Dec 30, 2017, 4:48:02 PM)

```
[/]
a->[/]
a->c->[/]
a->b->c->[/]
e
z->a->b->e->[/]
z->a->e->[/]
a->e->[/]
a->[/]
```

Agenda

1. Defining a List ADT
2. Generics
3. Singly linked list implementation
4. Exceptions
5. Visibility: public vs. private vs. protected vs. package

An exception indicates that something unexpected happened at run-time

- Cannot check for all errors at compile time
- What if we ask for element at an index of -1 in an array?
  - There is no clear, "always-do-this", answer
  - Maybe we should return null or maybe we should stop execution
- Exceptions provide a way to show something is amiss, and let calling functions deal with error (or not)
- Exceptions not handled by a method are passed to calling method. If exception not handled in *main()* or before, program stops
- "Throw" error with `throw new Exception("error description")`
- Java provides structured error-handling via *try/catch/finally* blocks
  - *catch* executes only if there is an exception in *try* body
  - *catch* block can specify the type of error it handles
  - Can have multiple *catch* blocks for each *try*
  - *Finally* block executes regardless whether *try* succeeds or fails

ListExceptions.java: Exceptions can be handled with try/catch/finally blocks

```
4  public class ListExceptions {
5      public static void main(String[] args) { // note: no "throws exception", as every method that could is
6
7          SimpleList<String> list = new SinglyLinked<String>();
8
9          try {
10             list.add(-1, "?");
11             System.out.println("I never run!");
12             System.out.println("Neither do I");
13         }
14         catch (Exception e) {
15             System.out.println("caught it!"); // will print -- we know this is bogus
16         }
17
18         try {
19             list.add(-1, "?");
20             System.out.println("Do I run?");
21             System.out.println("No I don't");
22         }
23         catch (Exception e) {
24             System.out.println("caught it again!"); // will print -- we know this is bogus
25             System.out.println(e);                  //will give us the error message
26         }
27         finally {
28             System.out.println("finally 1"); // executed whether or not caught an error
29         }
30
31         try {
32             list.add(0, "?");
33             System.out.println(list);
34         }
35         catch (Exception e) {
36             System.out.println("why did I catch it again!"); // won't print -- we know this code is fine
37         }
38         finally {
39             System.out.println("finally 2"); // executed whether or not caught an error
40         }
41 }
42 }
43
```

Create new SinglyLinked List

Problems Javadoc Declaration Console Debug Expressions Error Log Console Call Hierarchy

<terminated> ListExceptions [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Apr 6, 2018, 1:20:09 PM)

```
caught it!
caught it again!
java.lang.Exception: invalid index
finally 1
?->[/]
finally 2
```

ListExceptions.java: Exceptions can be handled with try/catch/finally blocks

```
4  public class ListExceptions {
5      public static void main(String[] args) { // note: no "throws exception", as every method that could is
6
7          SimpleList<String> list = new SinglyLinked<String>();
8
9          try {
10             list.add(-1, "?");
11             System.out.println("I never run!");
12             System.out.println("Neither do I");
13         }
14         catch (Exception e) {
15             System.out.println("caught it!"); // will print -- we know this is bogus
16         }
17
18         try {
19             list.add(-1, "?");
20             System.out.println("Do I run?");
21             System.out.println("No I don't");
22         }
23         catch (Exception e) {
24             System.out.println("caught it again!"); // will print -- we know this is bogus
25             System.out.println(e);                  //will give us the error message
26         }
27         finally {
28             System.out.println("finally 1"); // executed whether or not caught an error
29         }
30
31         try {
32             list.add(0, "?");
33             System.out.println(list);
34         }
35         catch (Exception e) {
36             System.out.println("why did I catch it again!"); // won't print -- we know this code is fine
37         }
38         finally {
39             System.out.println("finally 2"); // executed whether or not caught an error
40         }
41 }
42 }
43
```

Try block

Catch block
Only executes if exception in try block

Problems Javadoc Declaration Console Debug Expressions Error Log Console Call Hierarchy

<terminated> ListExceptions [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Apr 6, 2018, 1:20:09 PM)

```
caught it!
caught it again!
java.lang.Exception: invalid index
finally 1
?->[/]
finally 2
```

ListExceptions.java: Exceptions can be handled with try/catch/finally blocks

```
4  public class ListExceptions {
5      public static void main(String[] args) { // note: no "throws exception", as every method that could is
6
7          SimpleList<String> list = new SinglyLinked<String>();
8
9          try {
10             list.add(-1, "?");
11             System.out.println("I never run!");
12             System.out.println("Neither do I");
13         }
14         catch (Exception e) {
15             System.out.println("caught it!"); // will print -- we know this is bogus
16         }
17
18         try {
19             list.add(-1, "?");
20             System.out.println("Do I run?");
21             System.out.println("No I don't");
22         }
23         catch (Exception e) {
24             System.out.println("caught it again!"); // will print -- we know this is bogus
25             System.out.println(e);                  //will give us the error message
26         }
27         finally {
28             System.out.println("finally 1"); // executed whether or not caught an error
29         }
30
31         try {
32             list.add(0, "?");
33             System.out.println(list);
34         }
35         catch (Exception e) {
36             System.out.println("why did I catch it again!"); // won't print -- we know this code is fine
37         }
38         finally {
39             System.out.println("finally 2"); // executed whether or not caught an error
40         }
41 }
42 }
43
```

Trying to add at index -1 is an error, the catch block will execute because *add()* throws an exception for negative indices

Problems Javadoc Declaration Console Debug Expressions Error Log Console Call Hierarchy

<terminated> ListExceptions [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Apr 6, 2018, 1:20:09 PM)

```
caught it!
caught it again!
java.lang.Exception: invalid index
finally 1
?->[/]
finally 2
```

ListExceptions.java: Exceptions can be handled with try/catch/finally blocks

```
4  public class ListExceptions {
5      public static void main(String[] args) { // note: no "throws exception", as every method that could is
6
7          SimpleList<String> list = new SinglyLinked<String>();
8
9          try {
10             list.add(-1, "?");
11             System.out.println("I never run!");
12             System.out.println("Neither do I");
13         }
14         catch (Exception e) {
15             System.out.println("caught it!"); // will print -- we know this is bogus
16         }
17
18         try {
19             list.add(-1, "?");
20             System.out.println("Do I run?");
21             System.out.println("No I don't");
22         }
23         catch (Exception e) {
24             System.out.println("caught it again!"); // will print -- we know this
25             System.out.println(e);                  //will give us the error mess
26         }
27         finally {
28             System.out.println("finally 1"); // executed whether or not caught an
29         }
30
31         try {
32             list.add(0, "?");
33             System.out.println(list);
34         }
35         catch (Exception e) {
36             System.out.println("why did I catch it again!"); // won't print -- we
37         }
38         finally {
39             System.out.println("finally 2"); // executed whether or not caught an
40         }
41 }
42 }
43
```

Trying to add at index -1 is an error, the catch block will execute because *add()* throws an exception for negative indices

```
49     public void add(int idx, T item) throws Exception {
50         if (idx < 0) {
51             throw new Exception("invalid index");
52         }
53         else if (idx == 0) {
54             // Insert at head
55             head = new Element(item, head); //new item ge
56         }
57         else {
58             // It's the next thing after element # idx-1
59             Element e = advance(idx-1);
60             // Splice it in
61             e.next = new Element(item, e.next);  //create
62                                                  //and pri
63         }
64         size++;
65     }
66
```

SinglyLinked.java

Problems Javadoc Declaration Console Debug Expressions Error Log Console Call Hierarchy

<terminated> ListExceptions [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Apr 6, 2018, 1:20:09 PM)

```
caught it!
caught it again!
java.lang.Exception: invalid index
finally 1
?->[/]
finally 2
```

ListExceptions.java: Exceptions can be handled with try/catch/finally blocks

```
4  public class ListExceptions {
5      public static void main(String[] args) { // note: no "throws exception", as every method that could is
6
7          SimpleList<String> list = new SinglyLinked<String>();
8
9          try {
10             list.add(-1, "?");
11             System.out.println("I never run!");
12             System.out.println("Neither do I");
13         }
14         catch (Exception e) {
15             System.out.println("caught it!"); // will print -- we know this is bogus
16         }
17
18         try {
19             list.add(-1, "?");
20             System.out.println("Do I run?");
21             System.out.println("No I don't");
22         }
23         catch (Exception e) {
24             System.out.println("caught it again!"); // will print -- we know this is bogus
25             System.out.println(e);                  //will give us the error message
26         }
27         finally {
28             System.out.println("finally 1"); // executed whether or not caught an error
29         }
30
31         try {
32             list.add(0, "?");
33             System.out.println(list);
34         }
35         catch (Exception e) {
36             System.out.println("why did I catch it again!"); // won't print -- we know this code is fine
37         }
38         finally {
39             System.out.println("finally 2"); // executed whether or not caught an error
40         }
41 }
42 }
43
```

```
Problems  Javadoc  Declaration  Console  Debug  Expressions  Error Log  Console  Call Hierarchy
<terminated> ListExceptions [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Apr 6, 2018, 1:20:09 PM)
caught it!
caught it again!
java.lang.Exception: invalid index
finally 1
?->[/]
finally 2
```

Trying to add at index -1 is an error, the catch block will execute because *add()* throws an exception for negative indices

Catch block on line 15 executes because exception thrown on line 10

Lines 11 and 12 never execute because exception on line 10 stops execution in try block and starts running in catch block

"I never run" and "Neither do I" are not printed

If we didn't catch exception, the program would end because *main()* wouldn't have caught the exception (but we did catch it, so *main()* doesn't end execution)

ListExceptions.java: Exceptions can be handled with try/catch/finally blocks

```
4  public class ListExceptions {
5      public static void main(String[] args) { // note: no "throws exception", as every method that could is
6
7          SimpleList<String> list = new SinglyLinked<String>();
8
9          try {
10             list.add(-1, "?");
11             System.out.println("I never run!");
12             System.out.println("Neither do I");
13         }
14         catch (Exception e) {
15             System.out.println("caught it!"); // will print -- we know this is bogus
16         }
17
18         try {
19             list.add(-1, "?");
20             System.out.println("Do I run?");
21             System.out.println("No I don't");
22         }
23         catch (Exception e) {
24             System.out.println("caught it again!"); // will print -- we know this is bogus
25             System.out.println(e);                  //will give us the error message
26         }
27         finally {
28             System.out.println("finally 1"); // executed whether or not caught an error
29         }
30
31         try {
32             list.add(0, "?");
33             System.out.println(list);
34         }
35         catch (Exception e) {
36             System.out.println("why did I catch it again!"); // won't print -- we know this code is fine
37         }
38         finally {
39             System.out.println("finally 2"); // executed whether or not caught an error
40         }
41 }
42 }
43
```

Trying to add at index -1 is *still* an error, the catch block will execute

Problems Javadoc Declaration Console Debug Expressions Error Log Console Call Hierarchy

<terminated> ListExceptions [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Apr 6, 2018, 1:20:09 PM)

```
caught it!
caught it again!
java.lang.Exception: invalid index
finally 1
?->[/]
finally 2
```

ListExceptions.java: Exceptions can be handled with try/catch/finally blocks

```
4  public class ListExceptions {
5      public static void main(String[] args) { // note: no "throws exception", as every method that could is
6
7          SimpleList<String> list = new SinglyLinked<String>();
8
9          try {
10             list.add(-1, "?");
11             System.out.println("I never run!");
12             System.out.println("Neither do I");
13         }
14         catch (Exception e) {
15             System.out.println("caught it!"); // will print -- we know this is bogus
16         }
17
18         try {
19             list.add(-1, "?");
20             System.out.println("Do I run?");
21             System.out.println("No I don't");
22         }
23         catch (Exception e) {
24             System.out.println("caught it again!"); // will print -- we know this is bogus
25             System.out.println(e);                  //will give us the error message
26         }
27         finally {
28             System.out.println("finally 1"); // executed whether or not caught an error
29         }
30
31         try {
32             list.add(0, "?");
33             System.out.println(list);
34         }
35         catch (Exception e) {
36             System.out.println("why did I catch it again!"); // won't print -- we
37         }
38         finally {
39             System.out.println("finally 2"); // executed whether or not caught an
40         }
41 }
42 }
43
```

Problems Javadoc Declaration Console Debug Expressions Error Log Console Call Hierarchy

<terminated> ListExceptions [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Apr 6, 2018, 1:20:09 PM)

```
caught it!
caught it again!
java.lang.Exception: invalid index
finally 1
?->[/]
finally 2
```

Trying to add at index -1 is *still* an error, the catch block will execute

We can see what the exception was by printing e (it is just an object)

```
49     public void add(int idx, T item) throws Exception {
50         if (idx < 0) {
51             throw new Exception("invalid index");
52         }
53         else if (idx == 0) {
54             // Insert at head
55             head = new Element(item, head); //new item ge
56         }
57         else {
58             // It's the next thing after element # idx-1
59             Element e = advance(idx-1);
60             // Splice it in
```

"invalid index" was the message we included when we threw an error in the add method of SinglyLinked.java

ListExceptions.java: Exceptions can be handled with try/catch/finally blocks

```
4  public class ListExceptions {
5      public static void main(String[] args) { // note: no "throws exception", as every method that could is
6
7          SimpleList<String> list = new SinglyLinked<String>();
8
9          try {
10             list.add(-1, "?");
11             System.out.println("I never run!");
12             System.out.println("Neither do I");
13         }
14         catch (Exception e) {
15             System.out.println("caught it!"); // will print -- we know this is bogus
16         }
17
18         try {
19             list.add(-1, "?");
20             System.out.println("Do I run?");
21             System.out.println("No I don't");
22         }
23         catch (Exception e) {
24             System.out.println("caught it again!"); // will print -- we know this is bogus
25             System.out.println(e);                  //will give us the error message
26         }
27         finally {
28             System.out.println("finally 1"); // executed whether or not caught an error
29         }
30
31         try {
32             list.add(0, "?");
33             System.out.println(list);
34         }
35         catch (Exception e) {
36             System.out.println("why did I catch it again!"); // won't print -- we know this code is fine
37         }
38         finally {
39             System.out.println("finally 2"); // executed whether or not caught an error
40         }
41 }
42 }
43
```

- ***finally*** **always executes, regardless of whether exception in *try* block**
- ***catch*** **only executes if exception occurs in *try* block, otherwise *catch* code does not execute**
- **If exception in *try* block, execution in the *try* block stops at the point of the exception and picks up in first line of *catch* block**
- **Code in the *try* block after the line that caused the exception is not executed**

```
Problems  Javadoc  Declaration  Console  Debug  Expressions  Error Log  Console  Call Hierarchy
<terminated> ListExceptions [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Apr 6, 2018, 1:20:09 PM)
caught it!
caught it again!
java.lang.Exception: invalid index
finally 1
?->[/]
finally 2
```

ListExceptions.java: Exceptions can be handled with try/catch/finally blocks

```
4  public class ListExceptions {
5      public static void main(String[] args) { // note: no "throws exception", as every method that could is
6
7          SimpleList<String> list = new SinglyLinked<String>();
8
9          try {
10             list.add(-1, "?");
11             System.out.println("I never run!");
12             System.out.println("Neither do I");
13         }
14         catch (Exception e) {
15             System.out.println("caught it!"); // will print -- we know this is bogus
16         }
17
18         try {
19             list.add(-1, "?");
20             System.out.println("Do I run?");
21             System.out.println("No I don't");
22         }
23         catch (Exception e) {
24             System.out.println("caught it again!"); // will print -- we know this is bogus
25             System.out.println(e);                  //will give us the error message
26         }
27         finally {
28             System.out.println("finally 1"); // executed whether or not caught an error
29         }
30
31         try {
32             list.add(0, "?");
33             System.out.println(list);
34         }
35         catch (Exception e) {
36             System.out.println("why did I catch it again!"); // won't print -- we know this code is fine
37         }
38         finally {
39             System.out.println("finally 2"); // executed whether or not caught an error
40         }
41 }
42 }
43
```

This is valid, so *catch* block does not execute

Problems Javadoc Declaration Console Debug Expressions Error Log Console Call Hierarchy

```
<terminated> ListExceptions [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Apr 6, 2018, 1:20:09 PM)
caught it!
caught it again!
java.lang.Exception: invalid index
finally 1
?->[/]
finally 2
```

ListExceptions.java: Exceptions can be handled with try/catch/finally blocks

```
4  public class ListExceptions {
5      public static void main(String[] args) { // note: no "throws exception", as every method that could is
6
7          SimpleList<String> list = new SinglyLinked<String>();
8
9          try {
10             list.add(-1, "?");
11             System.out.println("I never run!");
12             System.out.println("Neither do I");
13         }
14         catch (Exception e) {
15             System.out.println("caught it!"); // will print -- we know this is bogus
16         }
17
18         try {
19             list.add(-1, "?");
20             System.out.println("Do I run?");
21             System.out.println("No I don't");
22         }
23         catch (Exception e) {
24             System.out.println("caught it again!"); // will print -- we know this is bogus
25             System.out.println(e);                  //will give us the error message
26         }
27         finally {
28             System.out.println("finally 1"); // executed whether or not caught an error
29         }
30
31         try {
32             list.add(0, "?");
33             System.out.println(list);
34         }
35         catch (Exception e) {
36             System.out.println("why did I catch it again!"); // won't print -- we know this code is fine
37         }
38         finally {
39             System.out.println("finally 2"); // executed whether or not caught an error
40         }
41 }
42 }
43
```

This is valid, so *catch* block does not execute

***finally* always executes, even if no exception in try block**

Problems Javadoc Declaration Console Debug Expressions Error Log Console Call Hierarchy

<terminated> ListExceptions [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Apr 6, 2018, 1:20:09 PM)

```
caught it!
caught it again!
java.lang.Exception: invalid index
finally 1
?->[/]
finally 2
```

Agenda

1. Defining a List ADT
2. Generics
3. Singly linked list implementation
4. Exceptions
5. Visibility: public vs. private vs. protected vs. package

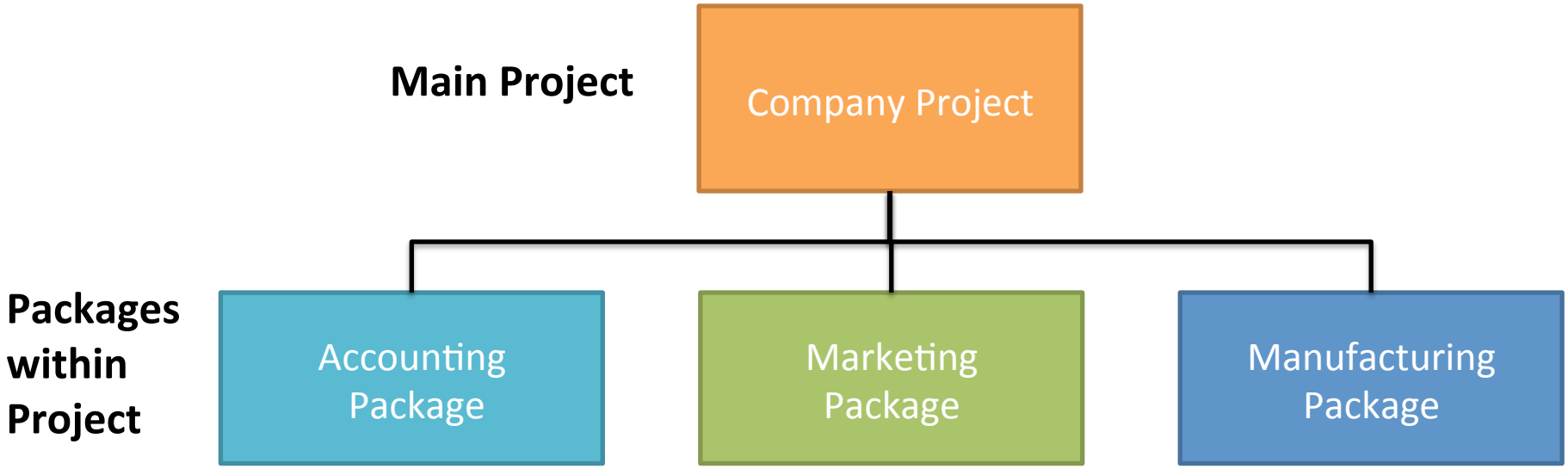
Java allows us to break up major portions of code into Projects, Packages and Classes

Example of master project for a company

Main Project

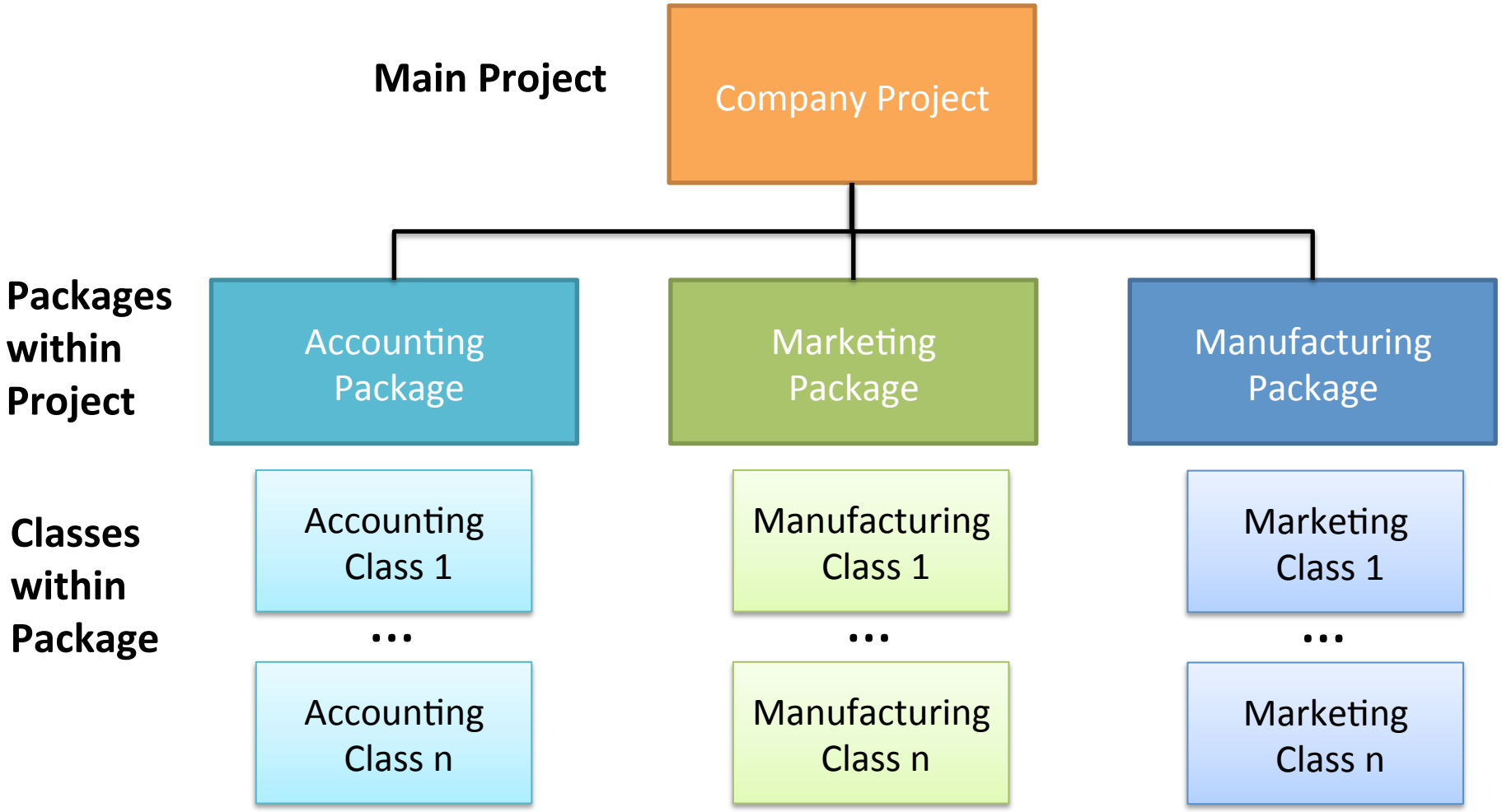
Java allows us to break up major portions of code into Projects, Packages and Classes

Example of master project for a company



Java allows us to break up major portions of code into Projects, Packages and Classes

Example of master project for a company

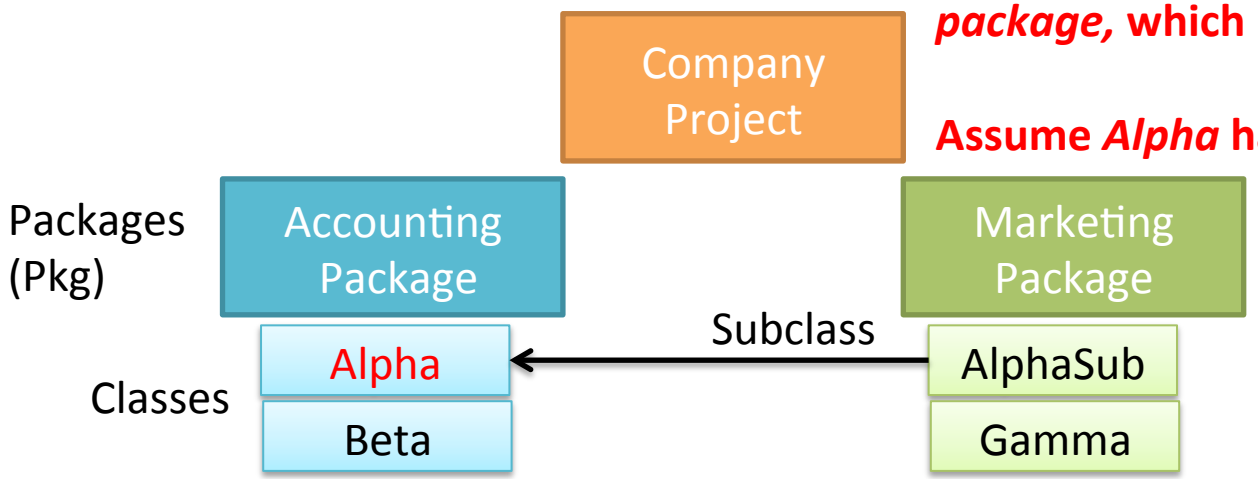


Visibility depends on modifier applied

Example: Visibility of Alpha class

Alpha* is a class in *Accounting package,* which is in *Company project

Assume *Alpha* has instance variable *x*



Y = can access
N = cannot access

	Alpha.x	Accounting Pkg		Marketing Pkg	
If Alpha.x is:	**accessed by:**	**Alpha**	**Beta**	**AlphaSub**	**Gamma**
public	Any class	Y	Y	Y	Y
protected	Pkg + Subclass	Y	Y	Y	N
No modifier	Pkg - Subclass	Y	Y	N	N
private	This class only	Y	N	N	N

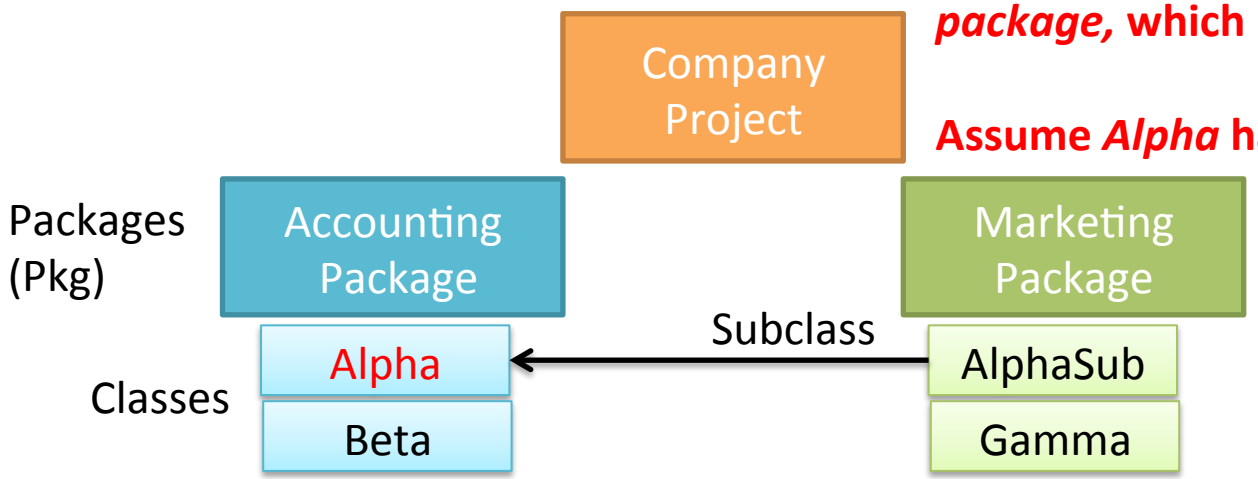
Adapted from Java documentation

Visibility depends on modifier applied

Example: Visibility of Alpha class

Alpha* is a class in *Accounting package,* which is in *Company project

Assume *Alpha* has instance variable *x*



Y = can access
N = cannot access

	Alpha.x	Accounting Pkg		Marketing Pkg	
If Alpha.x is:	**accessed by:**	**Alpha**	**Beta**	**AlphaSub**	**Gamma**
public	Any class	Y	Y	Y	Y
protected	Pkg + Subclass	Y	Y	Y	N
No modifier	Pkg - Subclass	Y	Y	N	N
private	This class only	Y	N	N	N

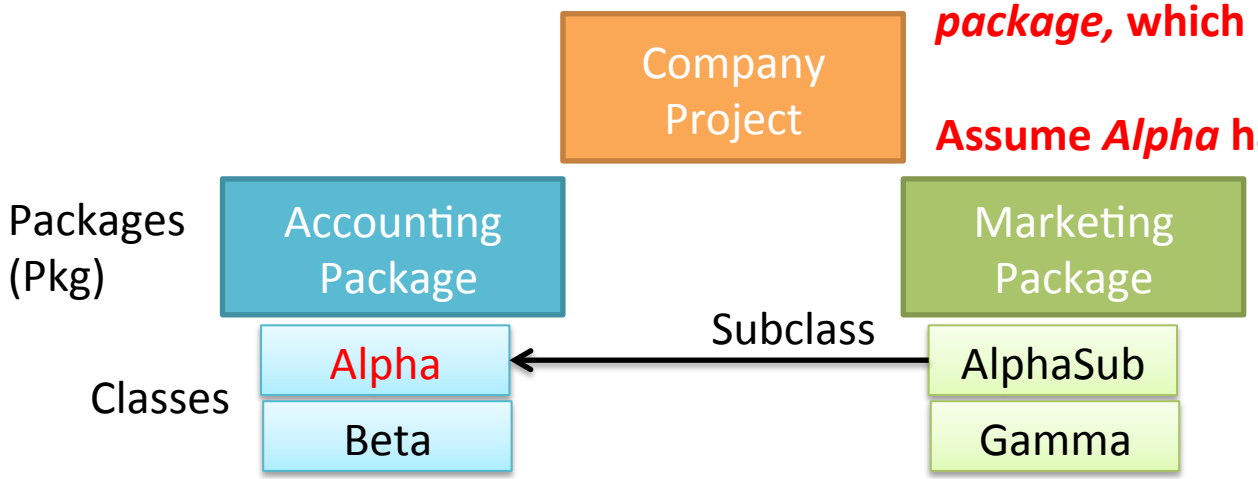
Adapted from Java documentation

Visibility depends on modifier applied

Example: Visibility of Alpha class

Alpha* is a class in *Accounting package,* which is in *Company project

Assume *Alpha* has instance variable *x*



Y = can access
N = cannot access

If Alpha.x is:	Alpha.x accessed by:	Accounting Pkg		Marketing Pkg	
		Alpha	Beta	AlphaSub	Gamma
public	Any class	Y	Y	Y	Y
protected	Pkg + Subclass	Y	Y	Y	N
No modifier	Pkg - Subclass	Y	Y	N	N
private	This class only	Y	N	N	N

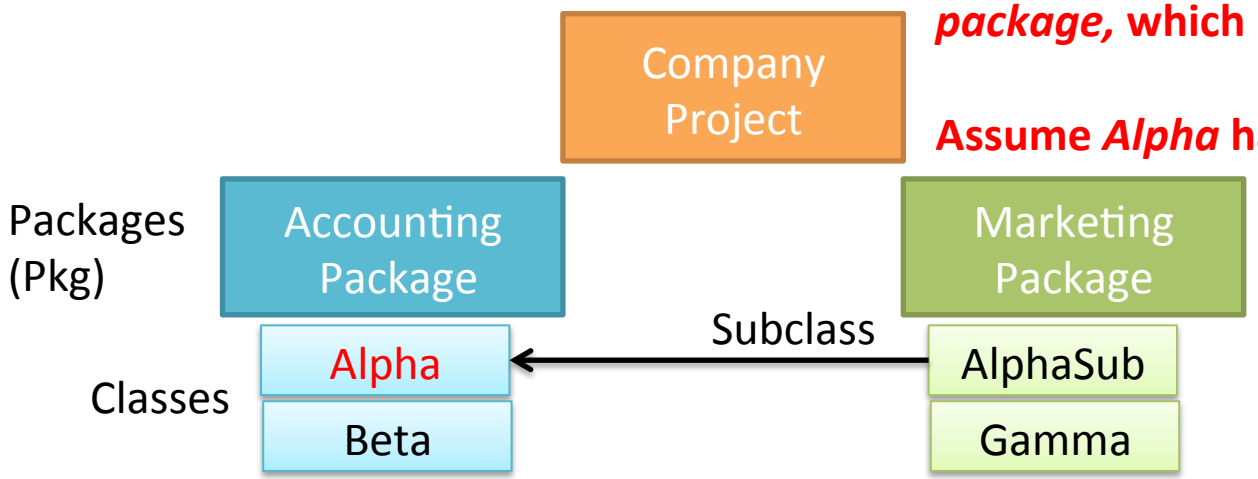
Adapted from Java documentation

Visibility depends on modifier applied

Example: Visibility of Alpha class

Alpha* is a class in *Accounting package,* which is in *Company project

Assume *Alpha* has instance variable *x*



Y = can access
N = cannot access

If Alpha.x is:	Alpha.x accessed by:	Accounting Pkg		Marketing Pkg	
		Alpha	Beta	AlphaSub	Gamma
public	Any class	Y	Y	Y	Y
protected	Pkg + Subclass	Y	Y	Y	N
No modifier	Pkg - Subclass	Y	Y	N	N
private	This class only	Y	N	N	N

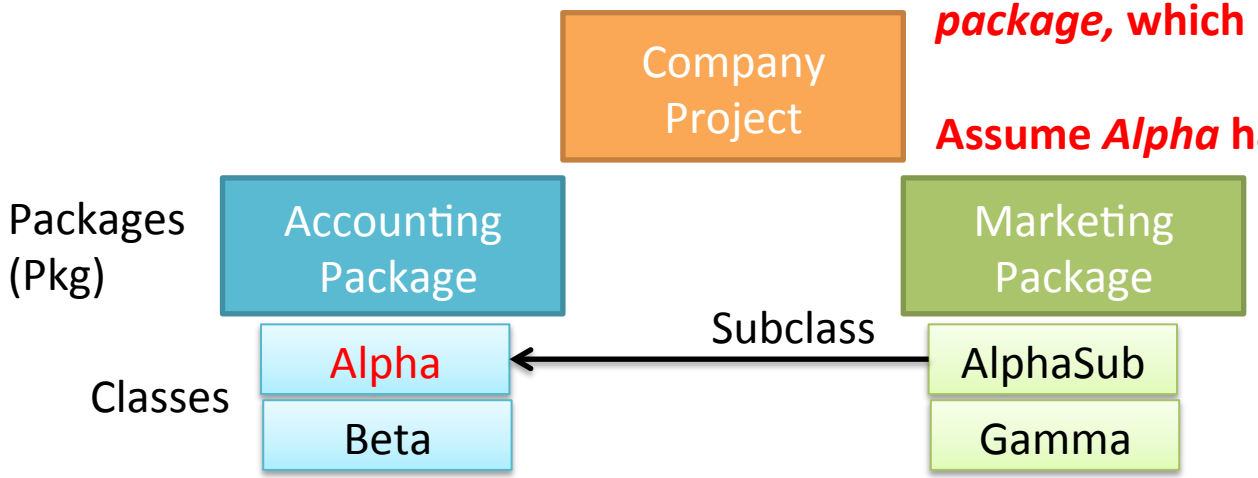
Adapted from Java documentation

Visibility depends on modifier applied

Example: Visibility of Alpha class

Alpha* is a class in *Accounting package,* which is in *Company project

Assume *Alpha* has instance variable *x*



Y = can access
N = cannot access

	Alpha.x	Accounting Pkg		Marketing Pkg	
If Alpha.x is:	**accessed by:**	**Alpha**	**Beta**	**AlphaSub**	**Gamma**
public	Any class	Y	Y	Y	Y
protected	Pkg + Subclass	Y	Y	Y	N
No modifier	Pkg - Subclass	Y	Y	N	N
private	This class only	Y	N	N	N

Adapted from Java documentation