Linked Structures

The *Java Collections Framework* provides many useful interfaces and implementations (classes). They all serve the same purpose: to store a collection of objects. Each type of collection has its trade-offs, and there is no one "best solution" for every storage problem.

Manager: Recorder:

Presenter: Reflector:

Content Learning Objectives

After completing this activity, students should be able to:

- Explain how items are inserted into an ArrayList vs a LinkedList.
- Summarize performance trade-offs for ArrayList and LinkedList.
- Decide whether to use an ArrayList or a LinkedList in a program.

Process Skill Goals

During the activity, students should make progress toward:

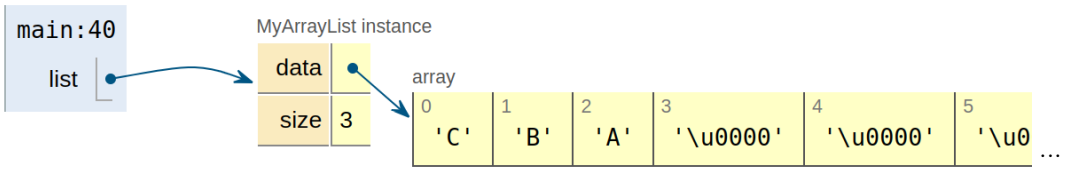
- Making connections between list diagrams and source code. (Information Processing)

Model 1 Array Lists

An "ArrayList" is a list that uses an array to store elements. The following code is a *simplified* version of `java.util.ArrayList`.

```
public class MyArrayList {

    private char[] data = new char[10];
    private int size;

    public void add(char item) {
        if (size == data.length) {
            grow();
        } else {
            shift();
        }
        data[0] = item;
        size++;
    }

    private void grow() {
        // make the new data array 50% larger
        char[] old = data;
        int newLen = (int) (old.length * 1.5);
        data = new char[newLen];
        // copy and shift from old to new array
        for (int i = size; i > 0; i--) {
            data[i] = old[i - 1];
        }
    }

    private void shift() {
        for (int i = size; i > 0; i--) {
            data[i] = data[i - 1];
        }
    }

    public static void main(String[] args) {
        MyArrayList list = new MyArrayList();
        list.add('A');
        list.add('B');
        list.add('C');
    }

}
```

The diagram shows the state of memory at the end of `main()`:



Questions (10 min) Start time:

1. Based on the code and the diagram, what is the value of . . . ?

a) `list.size`

b) `list.data.length`

2. Does the `add()` method insert at the beginning or append the end of the list? Justify your answer using the diagram.

3. How many array updates (that is, `array[index] = value` statements) were run for each of the following lines of `main()`?

a) `list.add('A');`

b) `list.add('B');`

c) `list.add('C');`

4. In general, if an ArrayList has *n* items, how many array updates are needed to insert the next item? Explain why.

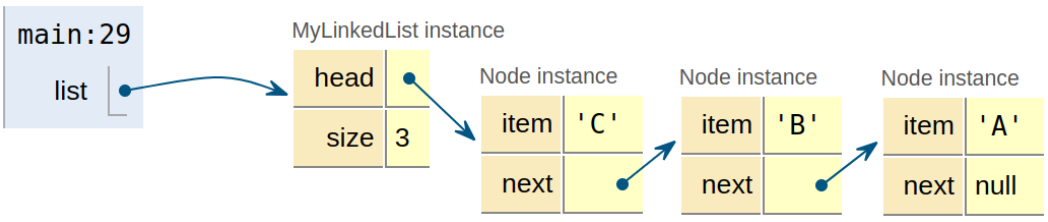
5. Imagine the `data` array is full and `size` is 10. If one more item is added to the list, how large will the new `data` array be? (*Hint:* Look at the `grow()` method.)

Model 2 Linked Lists

A "LinkedList" is a list that uses references to link elements. The following code is a *simplified* version of `java.util.LinkedList`.

```
1  public class MyLinkedList {
2
3      private Node head;  // see inner class below
4      private int size;
5
6      public void add(char item) {
7          Node oldHead = head;
8          head = new Node(item, oldHead);
9          size++;
10     }
11
12     private static class Node {
13         public char item;
14         public Node next;
15
16         public Node(char item, Node next) {
17             this.item = item;
18             this.next = next;
19         }
20     }
21
22     public static void main(String[] args) {
23         MyLinkedList list = new MyLinkedList();
24         list.add('A');
25         list.add('B');
26         list.add('C');
27     }
28
29 }
```

The diagram shows the state of memory at the end of `main()`:



Questions (20 min) Start time:

6. How many classes are defined in the code?

7. Describe the two attributes of a `Node` object.

8. Based on the diagram, what is the difference between `list` and `head`?

9. Based on the code, describe the steps for adding a new item at the head of the list.

10. Imagine a list with one million elements. Describe the advantage **LinkedList** has over ArrayList when **adding** a new item **at the head** of the list.

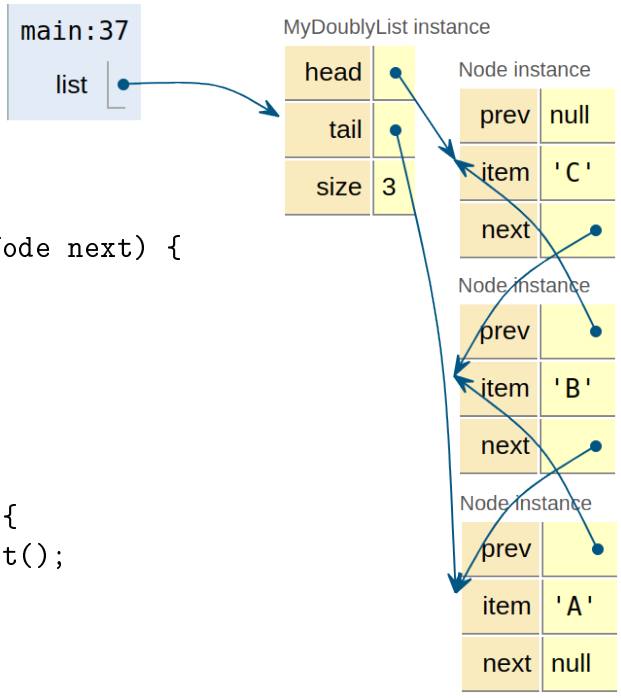
11. Imagine a list with one million elements. Describe the advantage **ArrayList** has over LinkedList when **getting** an item **in the middle** of the list.

12. (Optional) How much memory is needed to store a `MyLinkedList` of one million elements? How does that amount compare to using a `MyArrayList`?

Model 3 Doubly-Linked

Java's implementation of `LinkedList` stores two references in each node: one for the *previous*, and one for the *next*. In addition, both the *head* and the *tail* of the list are stored.

```
1  public class MyDoublyList {
2
3      private Node head;  // the first node
4      private Node tail;  // the last node
5      private int size;
6
7      public void add(char item) {
8          Node oldHead = head;
9          head = new Node(null, item, oldHead);
10         if (size == 0) {
11             tail = head;
12         } else {
13             oldHead.prev = head;
14         }
15         size++;
16     }
17
18     private static class Node {
19         public Node prev;
20         public char item;
21         public Node next;
22
23         public Node(Node prev, char item, Node next) {
24             this.prev = prev;
25             this.item = item;
26             this.next = next;
27         }
28     }
29
30     public static void main(String[] args) {
31         MyDoublyList list = new MyDoublyList();
32         list.add('A');
33         list.add('B');
34         list.add('C');
35     }
36
37 }
```



Questions (15 min) Start time:

13. At the end of `main()`, what is the value of . . . ?

a) `list.head.item`

b) `list.tail.item`

c) `list.head.prev`

d) `list.tail.next`

e) `list.head.next.item`

f) `list.tail.prev.prev.item`

14. Based on the code, describe the steps for adding a new item **at the head** of the list.

15. How different would the steps be for adding a new item **at the tail** of the list?

16. Imagine a list with one thousand elements. How would you insert a value at index 990?

17. What problems of singly-linked lists do doubly-linked lists solve? (In other words, what do the `prev` and `tail` references make possible?)

18. If your program requires a `List` collection, how would you decide which implementation to use? (`ArrayList` or `LinkedList`)

19. Explain why `java.util.ArrayList` is a poor choice of `List` in the program below:

```
1     public static void main(String[] args) {
2         List<String> list = new ArrayList<>();
3         System.out.println("Start");
4         addAndRemove(list);
5         System.out.println("Done!");
6     }
7
8     public static void addAndRemove(List<String> list) {
9         System.out.println("Adding...");
10        for (int i = 0; i < 1000000; i++) {
11            list.add(0, "A");  // insert at index 0
12        }
13        System.out.println("Removing...");
14        for (int i = 0; i < 1000000; i++) {
15            list.remove(0);  // remove at index 0
16        }
17    }
```

20. Explain why `java.util.LinkedList` is a poor choice of `List` in the program below.

```
1     public static void main(String[] args) {
2         List<String> list = new LinkedList<>();
3         System.out.println("Start");
4         addAndGet(list);
5         System.out.println("Done!");
6     }
7
8     public static void addAndGet(List<String> list) {
9         System.out.println("Adding...");
10        for (int i = 0; i < 1000000; i++) {
11            list.add("A");  // append at the end
12        }
13        System.out.println("Getting...");
14        for (int i = 0; i < 1000000; i++) {
15            list.get(list.size() / 2);  // get the middle
16        }
17    }
```