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:

- Show the contents of a List after adding and removing elements.
- Summarize performance trade-offs for ArrayList and LinkedList.

Process Skill Goals

During the activity, students should make progress toward:

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

Model 1 List Interface

In computer science, a ***list*** is a sequence of items with an ***index*** (or position) for each item.

index:	0	1	2	3	4	5	6	7	...
item:	`"Mer"`	`"Ven"`	`"Ear"`	`"Mar"`	`"Jup"`	`"Sat"`	`"Ura"`	`"Nep"`	...

Since we know which item is first, second, etc., we say that the list is ***ordered***. The first item is at the ***head*** of the list; the last item is at the ***tail*** of the list. Here is a subset of methods from `interface List<E>` in the Java API.

Modifier and Type	Method and Description
`boolean`	**`add`**`(E e)` Appends the specified element to the end of this list (optional operation).
`void`	**`add`**`(int index, E element)` Inserts the specified element at the specified position in this list (optional operation).
`E`	**`get`**`(int index)` Returns the element at the specified position in this list.
`boolean`	**`isEmpty`**`()` Returns `true` if this list contains no elements.
`E`	**`remove`**`(int index)` Removes the element at the specified position in this list (optional operation).
`E`	**`set`**`(int index, E element)` Replaces the element at the specified position in this list with the specified element (optional operation).
`int`	**`size`**`()` Returns the number of elements in this list.

Questions (15 min) Start time:

1. Give several examples of lists from everyday life.

2. What is the index of the head of a list? In general, what is the index of the tail of a list?

3. In Java lists, what does the `<E>` represent?

4. Of the seven methods shown from the Java API, how many change the contents of the list?

5. Fill in the table below to show the contents of the list after each method call. Note that the `add` method returns `true` if the list changes as a result, and the `set` method returns the element previously at the specified position.

Method Call		0	1	1	3	4		Return Value
`add(0, "A")`		A						true
`size()`		A						1
`add("N")`		A	N					true
`add(0, "R")`								
`add("G")`								
`size()`								
`remove(0)`								
`get(0)`								
`get(size()-1)`								
`set(1, "U")`								

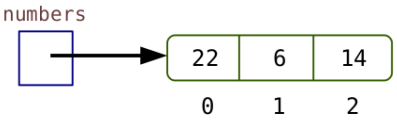
6. When adding or removing elements at the beginning, what did you have to do with the existing elements in the list?

7. In your own words, describe what a `List` collection is from a programmer's perspective.

Model 2 Array Lists

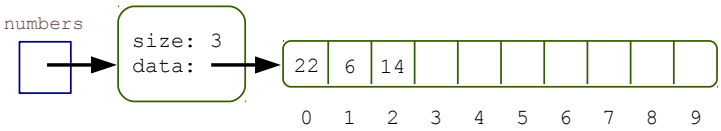
Arrays store elements in one ***contiguous*** block of memory. Since elements are stored together, you can immediately access any element by its index.

```
int[] numbers = {22, 6, 14};
System.out.println(numbers[1]);
```



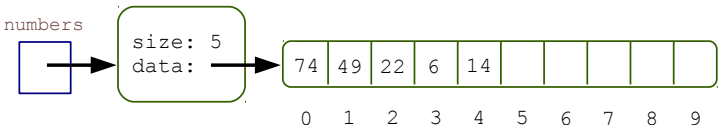
The `ArrayList` collection implements `List` and uses an array (internally) to store its elements.

```
ArrayList<Integer> numbers = new ArrayList<>();
numbers.add(22);
numbers.add(6);
numbers.add(14);
```



When new values are inserted, existing array elements are moved to the right.

```
numbers.add(0, 49);
numbers.add(0, 74);
```



If the array fills up, `ArrayList` automatically creates a new array about 50% larger. All current values must be copied into the new array, and the old array is then garbage collected.

Questions (15 min) Start time:

8. Why does Java use the name `ArrayList`? (What do the words `Array` and `List` indicate?)

9. How many array operations (i.e., integer assignments) were required to add 49 and 79 to the front of the second diagram in Model 2?

10. Imagine the internal array for `numbers` is full (i.e., with size=10 above). If you request one more element to be added (at the end), how big will the new array be?

11. Continuing the previous question, what operations are required to add one more element when the array is full? Briefly describe each operation, beginning with creating the new array.

12. Discuss why `ArrayList` is a poor choice of `List` in the program below:

```
1  import java.util.ArrayList;
2  import java.util.List;
3
4  public class ArraysAreBad {
5
6      public static void main(String[] args) {
7          List<String> list = new ArrayList<>();
8          System.out.println("Start");
9          addAndRemove(list);
10         System.out.println("Done!");
11     }
12
13     public static void addAndRemove(List<String> list) {
14         for (int i = 0; i < 1000000; i++) {
15             list.add(0, "A");  // add at index 0
16         }
17         for (int i = 0; i < 1000000; i++) {
18             list.remove(0);  // remove at index 0
19         }
20     }
21 }
```

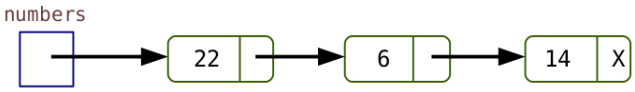
13. Arrays are simple and effective. Why would we want anything but ArrayList?

Model 3 Linked Lists

Linked structures "chain" elements using references. Each element of the list is called a ***node***.

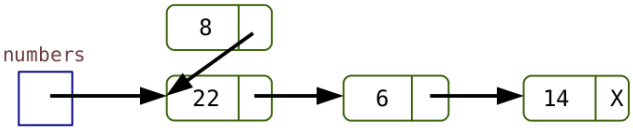
```
public class Node {

    private int value;
    private Node next;
    ...
}
```

```
Node node3 = new Node(14, null);
Node node2 = new Node(6, node3);
Node numbers = new Node(22, node2);
```

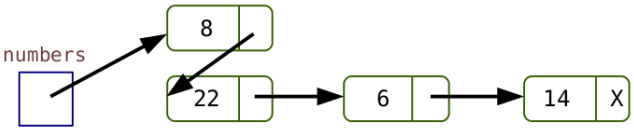


This organization allows fast insertions/deletions near the beginning. For example, to add 8:

```
Node temp = new Node(8, numbers);
```



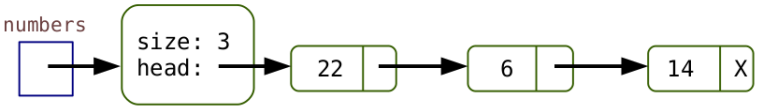

```
numbers = temp;
```



Instead of working with nodes directly, we can design a wrapper class to implement a list:

```
public class MyList {

    private int size;
    private Node head;
    ...
}
```

```
MyList numbers = new MyList();
numbers.addAtStart(14);
numbers.addAtStart(6);
numbers.addAtStart(22);
```



Questions (15 min) Start time:

14. In `MyList`, how many assignment operations are required to add 14 *at the front* of an empty list? Note that creating a `Node` takes two assignments (one for `value` and one for `next`).

15. In `MyList`, how many operations are required to add 22 at the front, after 14 and 6 have been added?

16. How many operations are required to add an element *at the end* of `MyList` with 3 elements?

17. How much memory is needed to store each element in the `LinkedList`? How does that amount compare with using an `ArrayList`?

18. Discuss why `LinkedList` is a poor choice of `List` in the program below.

```
1  import java.util.LinkedList;
2  import java.util.List;
3
4  public class LinksAreBad {
5
6      public static void main(String[] args) {
7          List<String> list = new LinkedList<>();
8          System.out.println("Start");
9          addAndGet(list);
10         System.out.println("Done!");
11     }
12
13     public static void addAndGet(List<String> list) {
14         for (int i = 0; i < 1000000; i++) {
15             list.add("A");  // add at the end
16         }
17         for (int i = 0; i < 1000000; i++) {
18             list.get(list.size() / 2);  // get the middle
19         }
20     }
21 }
```

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