CSE 143
Lecture 4

Implementing `ArrayIntList`

slides created by Marty Stepp
http://www.cs.washington.edu/143/

Exercise

- Write a program that reads a file (of unknown size) full of integers and prints the integers in the reverse order to how they occurred in the file. Consider example file `data.txt`:

```
17
932085
-32053278
100
3
```

  - When run with this file, your program's output would be:

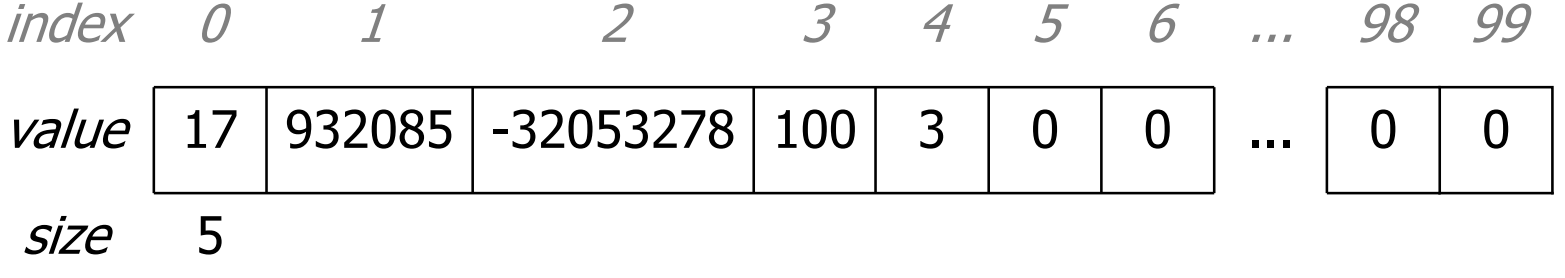
```
3
100
-32053278
932085
17
```

Solution using arrays

```
int[] nums = new int[100];   // make a really big array
int size = 0;

Scanner input = new Scanner(new File("data.txt"));
while (input.hasNextInt()) {
    nums[size] = input.nextInt();   // read each number
    size++;                         // into the array
}

for (int i = size - 1; i >= 0; i--) {
    System.out.println(nums[i]);    // print reversed
}
```

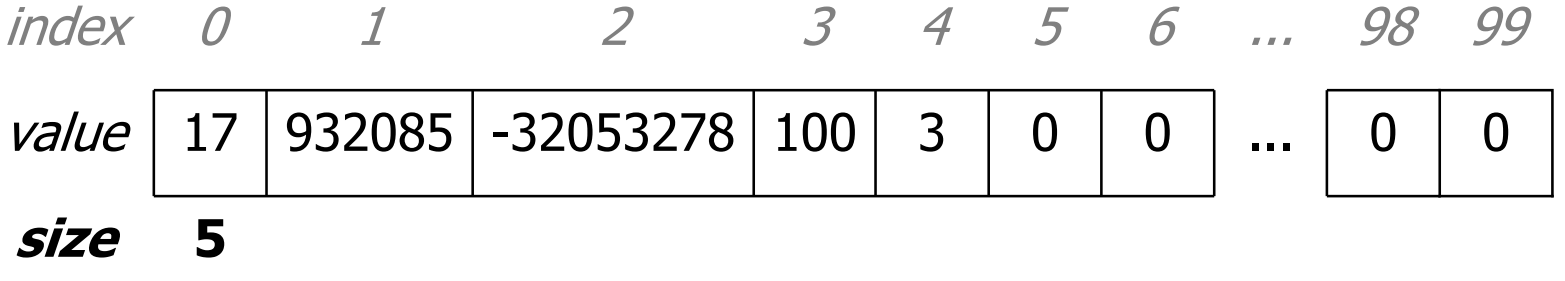


Unfilled arrays

```
int[] nums = new int[100];
int size = 0;
```

- We often need to store an unknown number of values.
  - Arrays can be used for this, but we must count the values.
  - Only the values at indexes [0, *size* - 1] are relevant.

- We are using an array to store a *list* of values.
  - What other operations might we want to run on lists of values?



Other possible operations

```
public static void add(int[] list, int size, int value, int index)
public static void remove(int[] list, int size, int index)
public static void find(int[] list, int size, int value)
public static void print(int[] list, int size)
...
```

- We could implement these operations as methods that accept a *list* array and its *size* along with other parameters.
  - But since the behavior and data are so closely related, it makes more sense to put them together into an object.
  - A list object can store an array of elements and a size, and can have methods for manipulating the list of elements.
    - Promotes **abstraction** (hides details of how the list works)

Exercise

- Let's write a class that implements a list using an `int[]`
  - We'll call it `ArrayIntList`
  - behavior:
    - `add(`**value**`),` `add(`**index**`,` **value**`)`
    - `get(`**index**`),` `set(`**index**`,` **value**`)`
    - `size()`
    - `remove(`**index**`)`
    - `indexOf(`**value**`)`
    - `toString()`

      ...
  - The list's *size* will be the number of elements added to it so far.
  - How will the list be used?...

Implementing `add`

- How do we add to the end of a list?

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
value	3	8	9	7	5	12	0	0	0	0
size	6									

- `list.add(42);`

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
value	3	8	9	7	5	12	**42**	0	0	0
size	**7**									

Implementing `add`, cont.

- To add to end of list, just store element and increase size:

```
public void add(int value) {
    list[size] = value;
    size++;
}
```

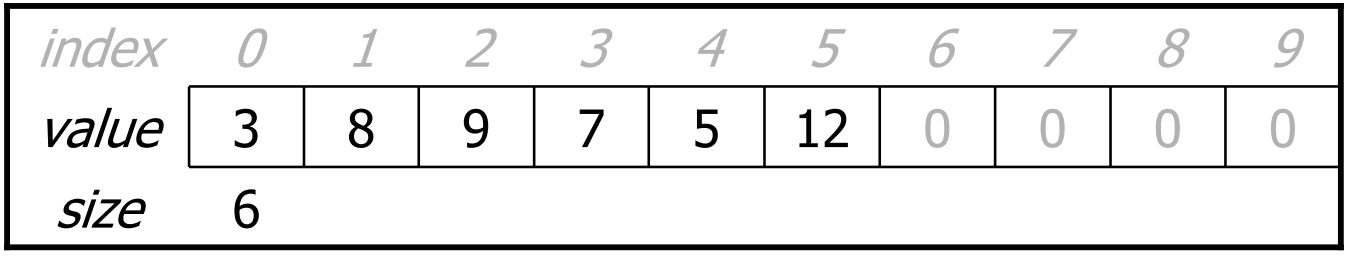
index	0	1	2	3	4	5	6	7	8	9
value	3	8	9	7	5	12	0	0	0	0
size	6									

- `list.add(42);`

index	0	1	2	3	4	5	6	7	8	9
value	3	8	9	7	5	12	**42**	0	0	0
size	**7**									

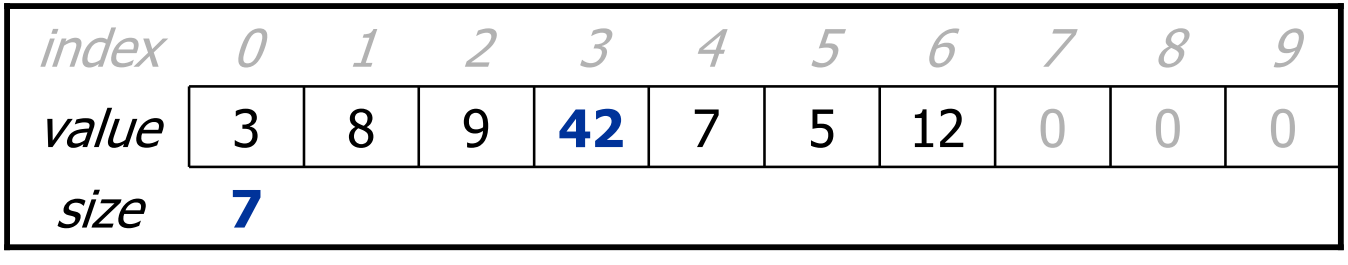
Implementing `add` (2)

- How do we add to the middle or end of the list?



index	0	1	2	3	4	5	6	7	8	9
value	3	8	9	7	5	12	0	0	0	0
size	6									

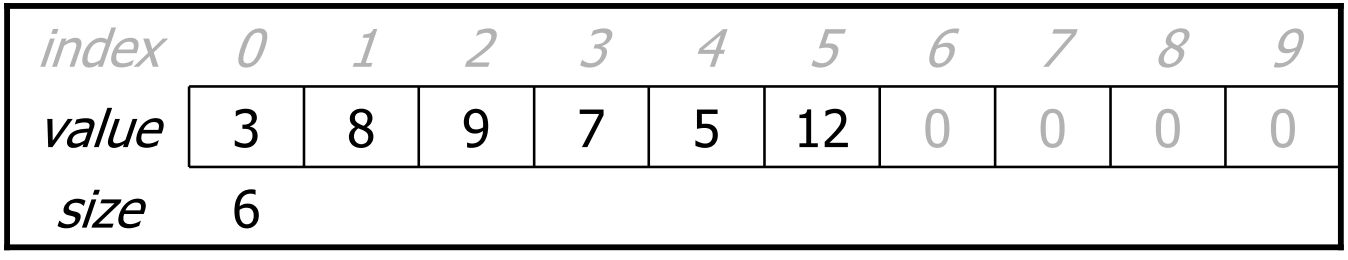
- `list.add(3, 42);` `// insert 42 at index 3`



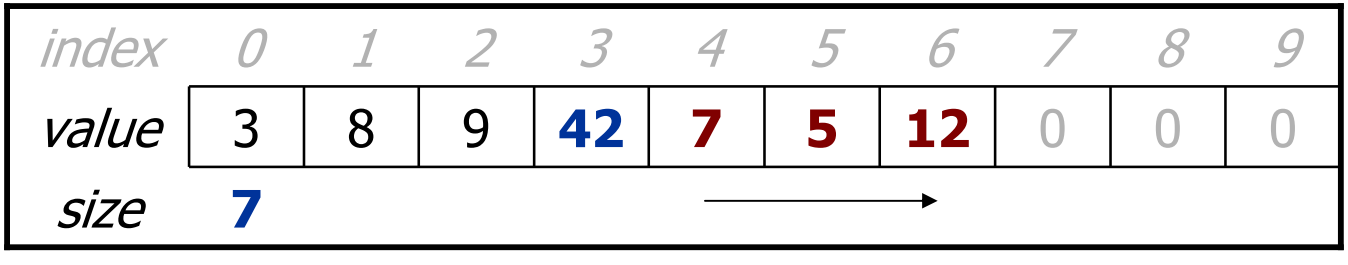
index	0	1	2	3	4	5	6	7	8	9
value	3	8	9	**42**	7	5	12	0	0	0
size	**7**									

Implementing `add` (2) cont.

- Adding to the middle or front is hard *(see book ch 7.3)*
  - must *shift* nearby elements to make room for the new value



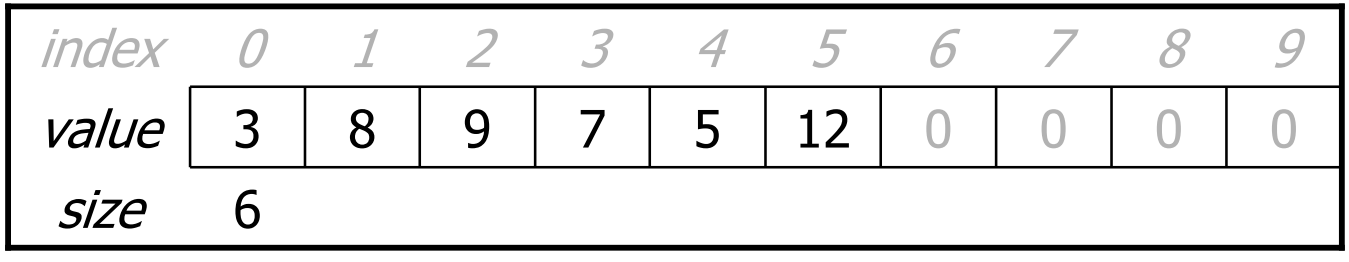
- `list.add(3, 42);  // insert 42 at index 3`



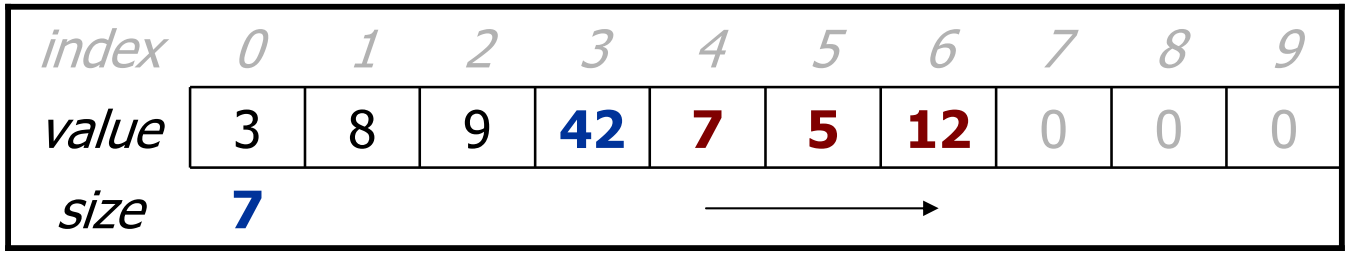
  - Note: The order in which you traverse the array matters!

Implementing `add` (2) code

```
public void add(int index, int value) {
    for (int i = size; i > index; i--) {
        list[i] = list[i - 1];
    }
    list[index] = value;
}
```



- `list.add(3, 42);`

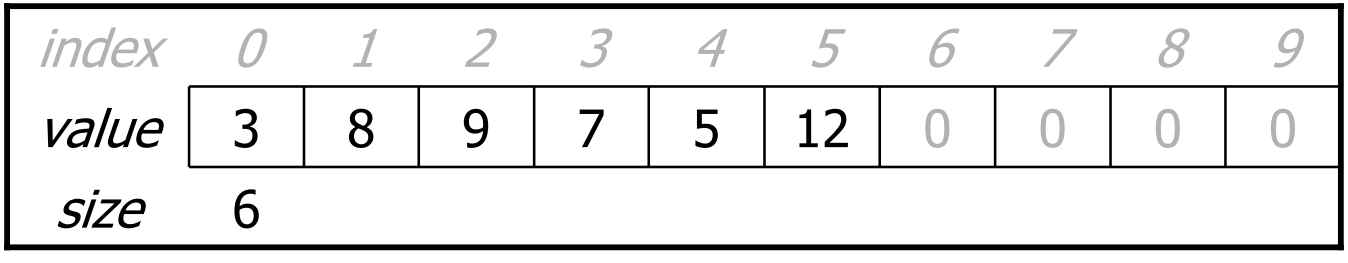


Other methods

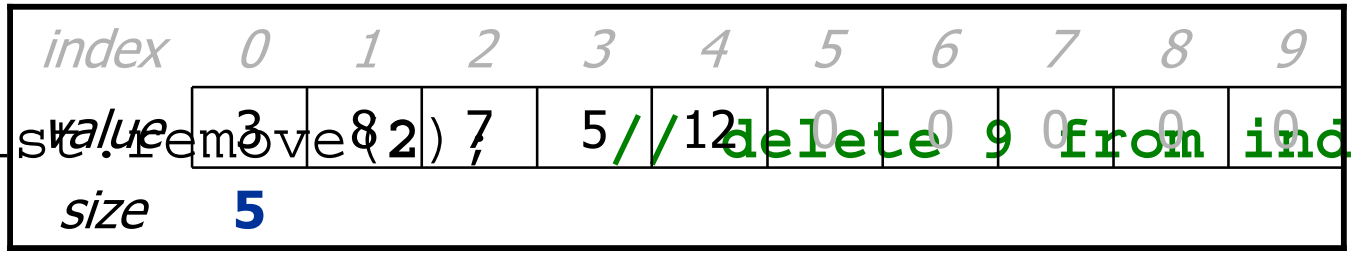
- Let's implement the following methods next:
  - size
  - get
  - set
  - toString

Implementing `remove`

- How can we remove an element from the list?

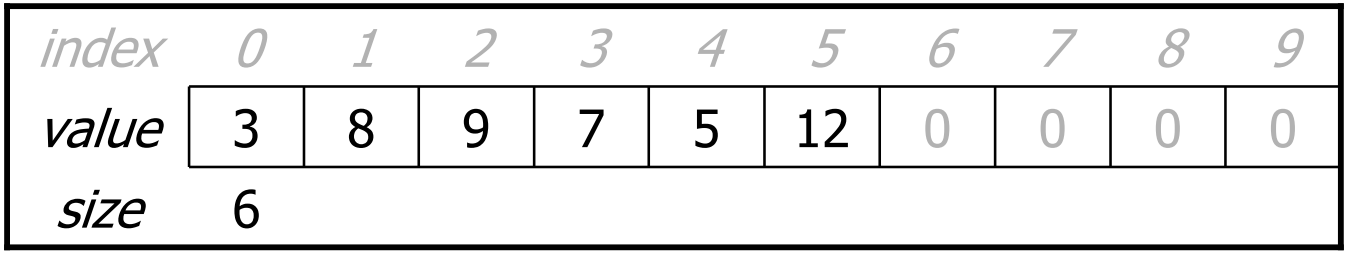


- `list.remove(2);` `// delete 9 from index 2`

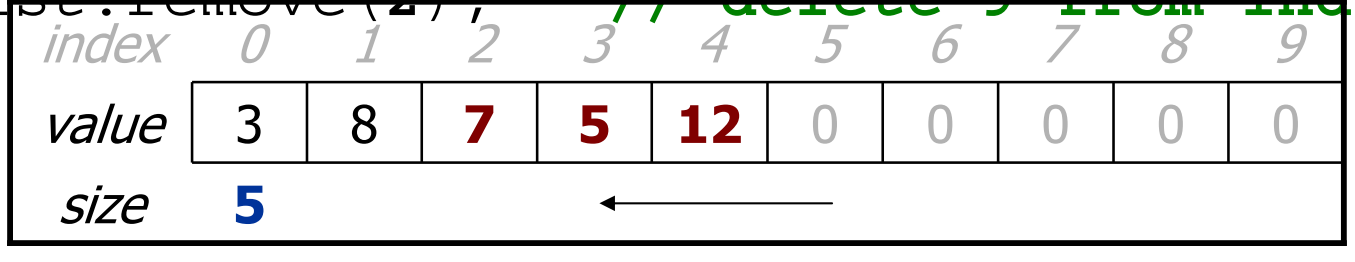


Implementing `remove`, cont.

- Again, we need to shift elements in the array
  - this time, it's a left-shift
  - in what order should we process the elements?
  - what indexes should we process?

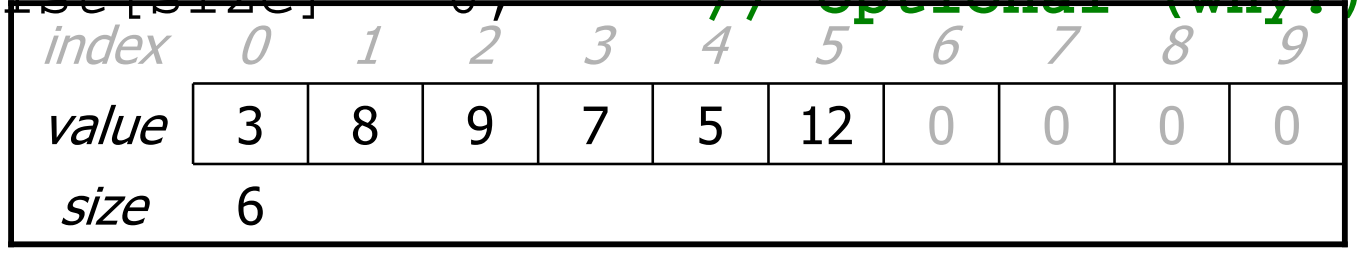


- `list.remove(2);  // delete 9 from index 2`

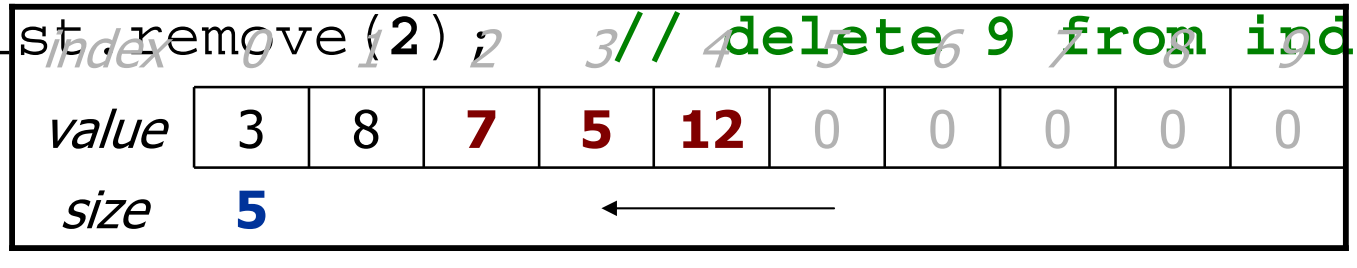


Implementing `remove` code

```
public void remove(int index) {
    for (int i = index; i < size; i++) {
        list[i] = list[i + 1];
    }
    size--;
    list[size] = 0;          // optional (why?)
}
```

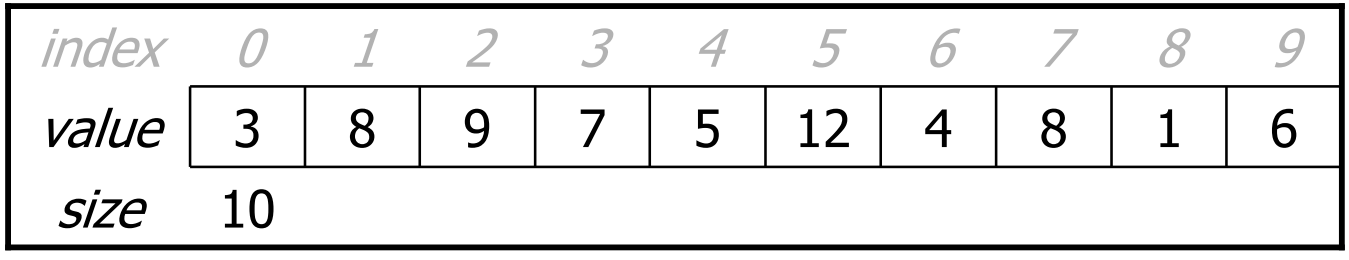


- `list.remove(2);  // delete 9 from index 2`



Running out of space

- What should we do if the client adds more than 10 elements?



- `list.add(15);` `// add an 11th element`