

Building Java Programs

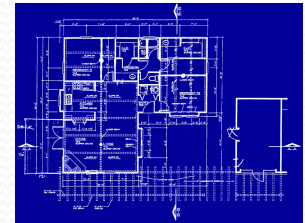
Chapter 15

Lecture 15-1: Implementing `ArrayIntList`

reading: 15.1 - 15.3

Recall: classes and objects

- **class:** A program entity that represents:
 - A complete program or module, or
 - A template for a type of objects.
 - (`ArrayList` is a class that defines a type.)



- **object:** An entity that combines **state** and **behavior**.
 - **object-oriented programming (OOP):** Programs that perform their behavior as interactions between objects.
 - **abstraction:** Separation between concepts and details. Objects provide abstraction in programming.

Elements of a class

```
public class BankAccount {
    private String name;           // fields:
    private int id;                // data encapsulated
    private double balance;       // inside each object

    public BankAccount(String name, int id) {
        this.name = name;         // constructor:
        this.id = id;             // initializes
        this.balance = 0.0;       // new objects
    }

    public void deposit(double amount) {
        this.balance += amount; // instance method:
    }                               // each object's
    ...                               // behavior
}
```

"implicit parameter": object on which a method was called

ArrayList implementation

- What is an ArrayList's behavior?
 - add, remove, indexOf, etc
- What is an ArrayList's state?
 - Many elements of the same type stored somehow
 - The number of elements stored in the list

<i>index</i>	<i>0</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>...</i>	<i>98</i>	<i>99</i>
<i>value</i>	17	932085	-32053278	100	3	0	0	...	0	0
<i>size</i>	5									

Implementing add

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

```
public void add(int value) { // put the element
    data[size] = value;      // in the last slot,
    size++;                 // and increase the size
}
```

<i>index</i>	0	1	2	3	4	5	6	7	8	9
<i>value</i>	3	8	9	7	5	12	0	0	0	0
<i>size</i>	6									

- `list.add(42);`

<i>index</i>	0	1	2	3	4	5	6	7	8	9
<i>value</i>	3	8	9	7	5	12	42	0	0	0
<i>size</i>	7									

Printing an `ArrayList`

- Let's add a method that allows clients to print a list's elements.

- You may be tempted to write a `print` method:

```
// client code
ArrayList list = new ArrayList();
...
list.print();
```

- Why is this a bad idea? What would be better?
 - It is more flexible to provide a method that returns a `String`, allowing a client to print, or use the `String` in some other way.

The toString method

- Tells Java how to convert an object into a String

```
ArrayList list = new ArrayList();  
System.out.println("list is " + list);  
// ("list is " + list.toString());
```

- Syntax:

```
public String toString() {  
    code that returns a suitable String;  
}
```

- Every class has a toString, even if it isn't in your code.
 - The default is the class's name and a hex (base-16) number:

```
ArrayList@9e8c34
```

toString solution

// Returns a String representation of the list.

```
public String toString() {
    if (size == 0) {
        return "[]";
    } else {
        String result = "[" + data[0];
        for (int i = 1; i < size; i++) {
            result += ", " + data[i];
        }
        result += "];";
        return result;
    }
}
```


Other methods

- Let's implement the following methods in our list:
 - `size()`
Returns the number of elements in the list.
 - `get(index)`
Returns the element value at a given index.
 - `set(index, value)`
Sets the list to store the given value at the given index.

Pre/Post Comments

- The get and set methods require that the index passed in is valid. The public comments on the method header should document that.
- Pre/Post commenting style
 - Pre: documents anything that you are before the method executes. Any
 - Post: documents any state modification happening in the method along with what the method returns. Should not say exactly how the method does the code, just what the behavior of the method is to the client's point of view.
 - We expect you to comment every method in every class you write, although you do not need to comment in this style.

Error Detection

- The get and set methods should not only inform the client what a valid index is, they should check to make sure the index actually is valid

- Throwing exceptions

```
private void checkIndex(int index, int upperBound) {  
    if (index < 0 || index >= upperBound) {  
        throw new IllegalArgumentException();  
    }  
}
```

- exception checking can usually be factored into a private helper method, because it is a redundant operation.
- `IndexOutOfBoundsException` would have been equally good to throw in this case.

Implementing add #2

- How do we add to the middle or end of the list?
 - must *shift* elements to make room for the value (see book 7.4)

<i>index</i>	0	1	2	3	4	5	6	7	8	9
<i>value</i>	3	8	9	7	5	12	0	0	0	0
<i>size</i>	6									

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

<i>index</i>	0	1	2	3	4	5	6	7	8	9
<i>value</i>	3	8	9	42	7	5	12	0	0	0
<i>size</i>	7									

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

add #2 code

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

<i>index</i>	0	1	2	3	4	5	6	7	8	9
<i>value</i>	3	8	9	7	5	12	0	0	0	0
<i>size</i>	6									

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

<i>index</i>	0	1	2	3	4	5	6	7	8	9
<i>value</i>	3	8	9	42	7	5	12	0	0	0
<i>size</i>	7	→								

Implementing `remove`

- 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?

<i>index</i>	0	1	2	3	4	5	6	7	8	9
<i>value</i>	3	8	9	7	5	12	0	0	0	0
<i>size</i>	6									

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

<i>index</i>	0	1	2	3	4	5	6	7	8	9
<i>value</i>	3	8	7	5	12	0	0	0	0	0
<i>size</i>	5									

←

Implementing `remove` code

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

<i>index</i>	0	1	2	3	4	5	6	7	8	9
<i>value</i>	3	8	9	7	5	12	0	0	0	0
<i>size</i>	6									

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

<i>index</i>	0	1	2	3	4	5	6	7	8	9
<i>value</i>	3	8	7	5	12	0	0	0	0	0
<i>size</i>	5 ←									

Other methods

- Additional methods included in the code that were not written as a class:
 - `indexOf(value)`
Returns the first occurrence of the given value in the list, or -1 if the value is not found
 - `contains(value)`
Returns true if the list contains the value at least once, false otherwise
 - `isEmpty()`
Returns `true` if the list contains no elements; else `false`.
(Why write this if we already have the `size` method?)