Building Java Programs

Chapter 15

`ArrayIntList`

reading: 15.1

Welcome to CSE 143!

Go to pollev.com/cse143

Context for CSE 143

CSE 142

- Control: loops, if/else, methods, parameters, returns
- I/O: Scanners, user input, files
- Data: primitive types (int, double, etc.), *arrays*, *classes*

CSE 143

- Control: recursion
- Data
  - Java collections
  - Classes + Object Oriented Programming
- Best of CS

Road Map

CS Concepts

- Client/Implementer
- Efficiency
- Recursion
- Regular Expressions
- Grammars
- Sorting
- Backtracking
- Hashing
- Huffman Compression

Java Language

- Exceptions
- Interfaces
- References
- Generics
- Comparable
- Inheritance/Polymorphism
- Abstract Classes

Data Structures

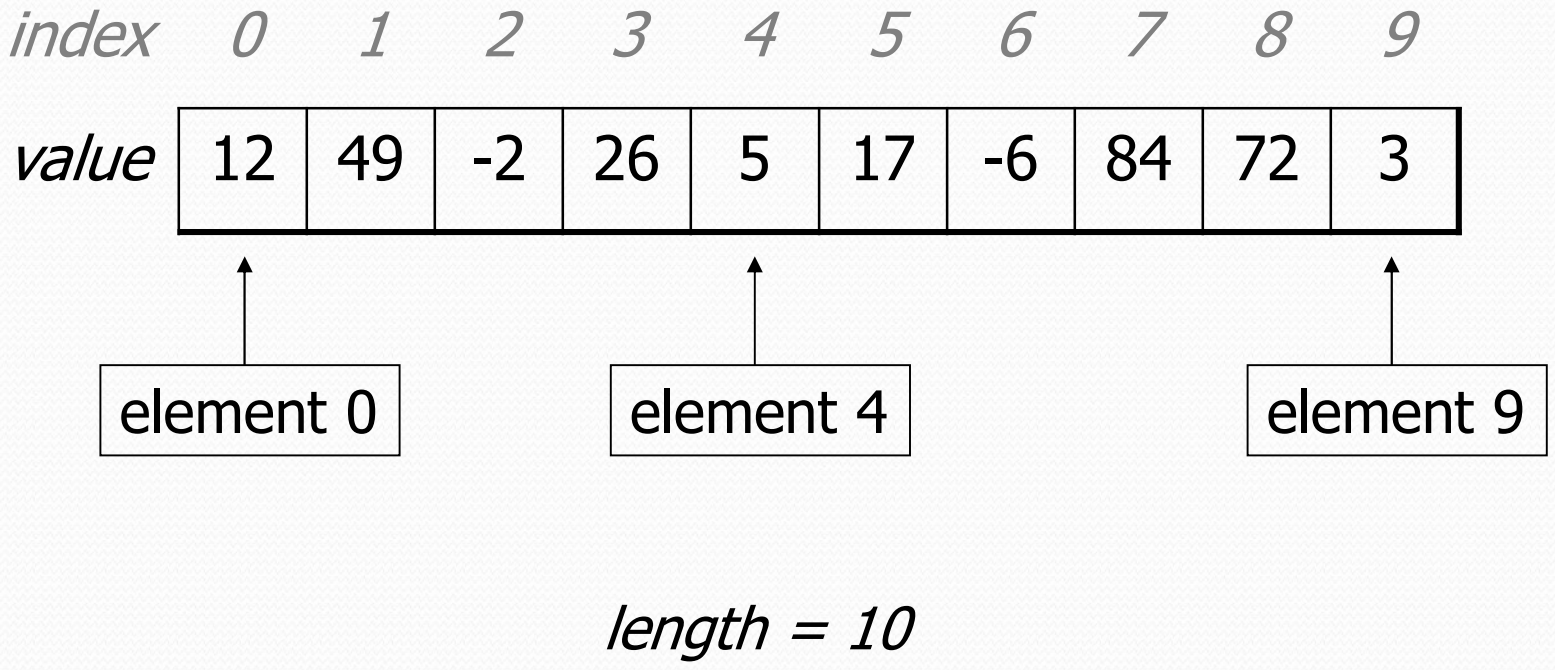
- Lists
- Stacks
- Queues
- Sets
- Maps
- Priority Queues

Java Collections

- Arrays
- ArrayList
- LinkedList
- Stack
- TreeSet / TreeMap
- HashSet / HashMap
- PriorityQueue

Recall: Arrays (7.1)

- **array**: object that stores many values of the same type.
  - **element**: One value in an array.
  - **index**: 0-based integer to access an element from an array.
  - **length**: Number of elements in the array.



Array Limitations

- Fixed-size
- Adding or removing from middle is hard
- Not much built-in functionality (need Arrays class)

Collections

- **collection**: an object that stores data; a.k.a. "data structure"
  - the objects stored are called **elements**
  - some collections maintain an ordering; some allow duplicates
  - typical operations: *add*, *remove*, *clear*, *contains* (search), *size*

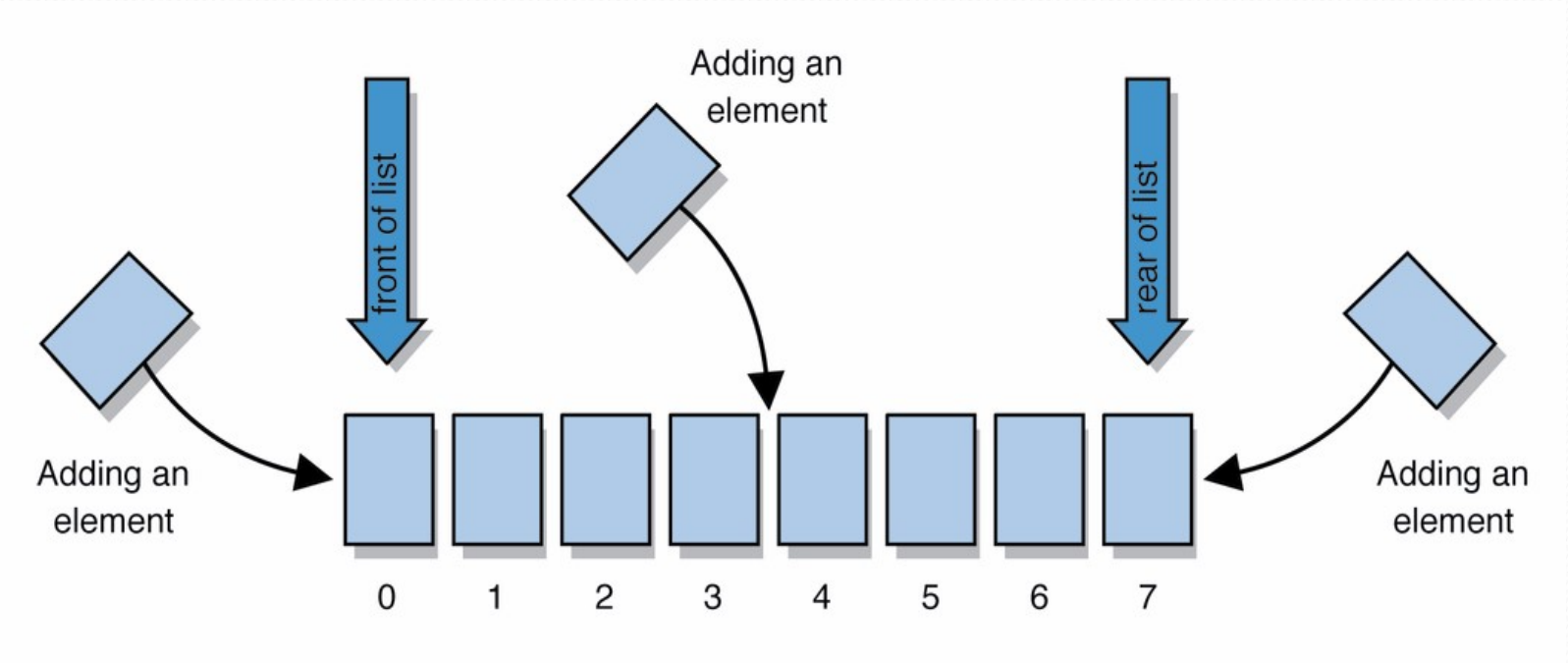
  - examples found in the Java class libraries:
    (covered in this course!)
    - `ArrayList`, `LinkedList`, `HashMap`, `TreeSet`, `PriorityQueue`

  - all collections are in the `java.util` package

```
import java.util.*;
```

Lists

- **list**: a collection of elements with 0-based **indexes**
  - elements can be added to the front, back, or elsewhere
  - a list has a **size** (number of elements that have been added)
  - This is just a high level idea, haven't said how to do it in Java



List Abstraction

- Like an array that resizes to fit its contents.
- When a list is created, it is initially empty.

```
[]
```

- Use `add` methods to add to different locations in list

```
[hello, ABC, goodbye, okay]
```

  - The list object keeps track of the element values that have been added to it, their order, indexes, and its total size.
  - You can add, remove, get, set, ... any index at any time.

ArrayList

```
ArrayList<Type> name = new ArrayList<Type>();
```

- When constructing an `ArrayList`, you must specify the type of its elements in `< >`
  - This is called a *type parameter* ; `ArrayList` is a *generic* class.
  - Allows the `ArrayList` class to store lists of different types.
  - Arrays use a similar idea with **Type**`[]`

```
ArrayList<String> names = new ArrayList<String>();
names.add("Marty Stepp");
names.add("Stuart Reges");
```

`ArrayList` methods (10.1)*

`add(`**value**`)`	appends value at end of list
`add(`**index**`,` **value**`)`	inserts given value just before the given index, shifting subsequent values to the right
`clear()`	removes all elements of the list
`indexOf(`**value**`)`	returns first index where given value is found in list (-1 if not found)
`get(`**index**`)`	returns the value at given index
`remove(`**index**`)`	removes/returns value at given index, shifting subsequent values to the left
`set(`**index**`,` **value**`)`	replaces value at given index with given value
`size()`	returns the number of elements in list
`toString()`	returns a string representation of the list such as `"[3, 42, -7, 15]"`

ArrayList vs. array

- construction

```
String[] names = new String[5];
ArrayList<String> list = new ArrayList<String>();
```

- storing a value

```
names[0] = "Jessica";
list.add("Jessica");
```

- retrieving a value

```
String s = names[0];
String s = list.get(0);
```

ArrayList vs. array

```
String[] names = new String[5];                    // construct
names[0] = "Jessica";                              // store
String s = names[0];                               // retrieve
for (int i = 0; i < names.length; i++) {
    if (names[i].startsWith("B")) { ... }
}                                                  // iterate
```

```
ArrayList<String> list = new ArrayList<String>();
list.add("Jessica");                               // store
String s = list.get(0);                            // retrieve
for (int i = 0; i < list.size(); i++) {
    if (list.get(i).startsWith("B")) { ... }
}                                                  // iterate
```

pollev.com/cse143

- Suppose we had the following method:

```
// Returns count of plural words in the given list.
public static int removePlural(ArrayList<String> list) {
    for (int i = 0; i < list.size(); i++) {
        String str = list.get(i);
        if (str.endsWith("s")) {
            list.remove(i);
        }
    }
}
```

- What would the output be after the method call?

```
ArrayList<String> list = …; // [a, bs, c, ds, es, f]
removePlural(list);
System.out.println(list);
```