



# Building Java Programs

Chapter 10

`ArrayList`

**reading: 10.1**

# Welcome to CSE 143!



# Words exercise

- Write code to read a file and display its words in reverse order.
- A solution that uses an array:

```
String[] allWords = new String[1000];  
int wordCount = 0;
```

```
Scanner input = new Scanner(new File("words.txt"));  
while (input.hasNext()) {  
    String word = input.next();  
    allWords[wordCount] = word;  
    wordCount++;  
}
```

- What's wrong with this?

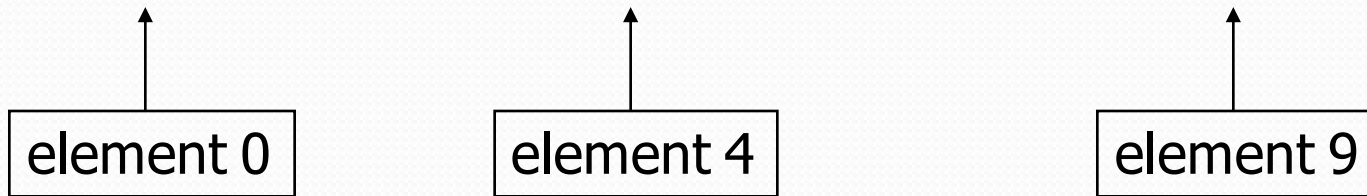
# Recall: Arrays (7.1)

- **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.

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

*value*    12   49   -2   26   5   17   -6   84   72   3

12	49	-2	26	5	17	-6	84	72	3
----	----	----	----	---	----	----	----	----	---



*length = 10*

# Array Limitations

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

# 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.

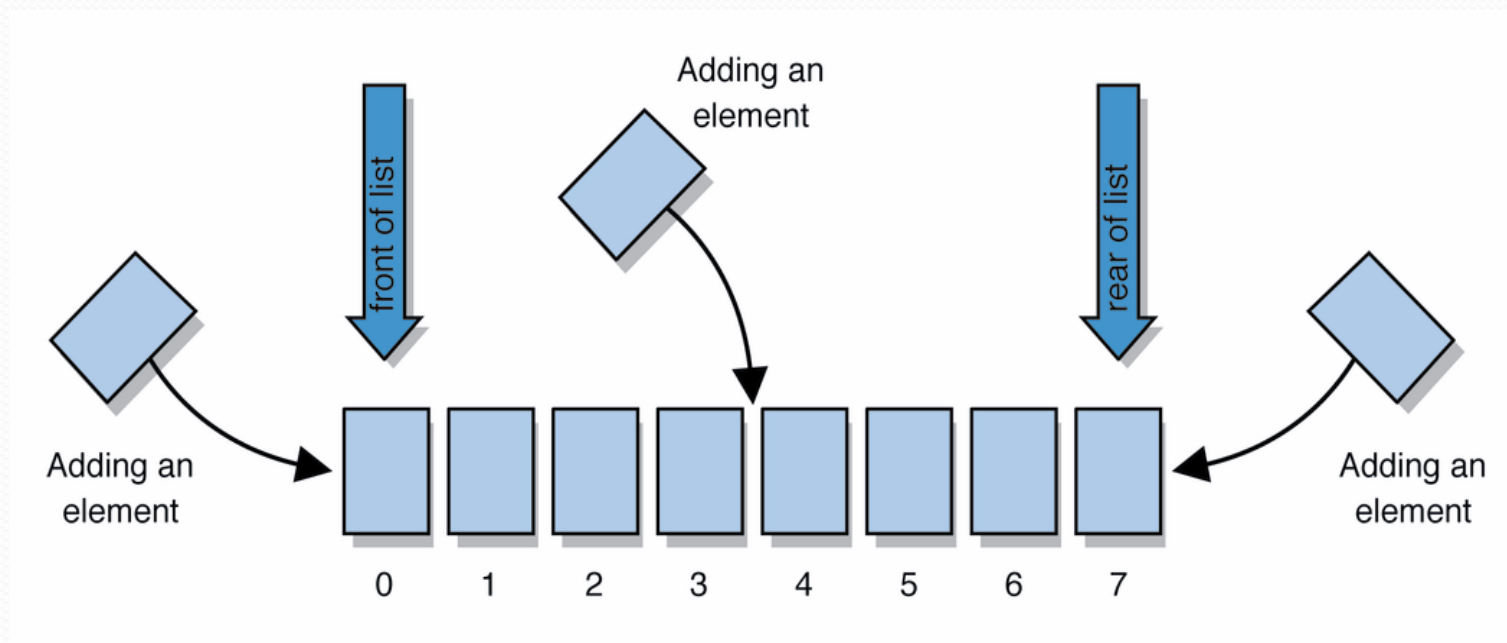
# Collections

- **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)
  - in Java, a list can be represented as an **ArrayList** object





# Type parameters (generics)

```
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 ( <b>value</b> )	appends value at end of list
add ( <b>index</b> , <b>value</b> )	inserts given value just before the given index, shifting subsequent values to the right
clear()	removes all elements of the list
indexOf ( <b>value</b> )	returns first index where given value is found in list (-1 if not found)
get ( <b>index</b> )	returns the value at given index
remove ( <b>index</b> )	removes/returns value at given index, shifting subsequent values to the left
set ( <b>index</b> , <b>value</b> )	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
```

# ArrayList as param/return

```
public static void name(ArrayList<Type> name) { // param
public static ArrayList<Type> name(params) //
return
```

- Example:

```
// Returns count of plural words in the given list.
```

```
public static int countPlural(ArrayList<String> list) {
    int count = 0;
    for (int i = 0; i < list.size(); i++) {
        String str = list.get(i);
        if (str.endsWith("s")) {
            count++;
        }
    }
    return count;
}
```

# Words exercise, revisited

- Write a program that reads a file and displays the words of that file as a list.
  - Then display the words in reverse order.
  - Then display them with all plurals (ending in "s") capitalized.
  - Then display them with all plural words removed.

# Exercise solution (partial)

```
ArrayList<String> allWords = new ArrayList<String>();
Scanner input = new Scanner(new File("words.txt"));
while (input.hasNext()) {
    String word = input.next();
    allWords.add(word);
}

// display in reverse order
for (int i = allWords.size() - 1; i >= 0; i--) {
    System.out.println(allWords.get(i));
}

// remove all plural words
for (int i = 0; i < allWords.size(); i++) {
    String word = allWords.get(i);
    if (word.endsWith("s")) {
        allWords.remove(i);
        i--;
    }
}
```

# ArrayList of primitives?

- The type you specify when creating an `ArrayList` must be an object type; it cannot be a primitive type.

```
// illegal -- int cannot be a type parameter  
ArrayList<int> list = new ArrayList<int>();
```

- But we can still use `ArrayList` with primitive types by using special classes called *wrapper* classes in their place.

```
// creates a list of ints  
ArrayList<Integer> list = new ArrayList<Integer>();
```



# Wrapper classes

Primitive Type	Wrapper Type
int	Integer
double	Double
char	Character
boolean	Boolean



- A wrapper is an object whose sole purpose is to hold a primitive value.
- Once you construct the list, use it with primitives as normal:

```
ArrayList<Double> grades = new ArrayList<Double>();  
grades.add(3.2);  
grades.add(2.7);  
...  
double myGrade = grades.get(0);
```