CSE 143
Lecture 1

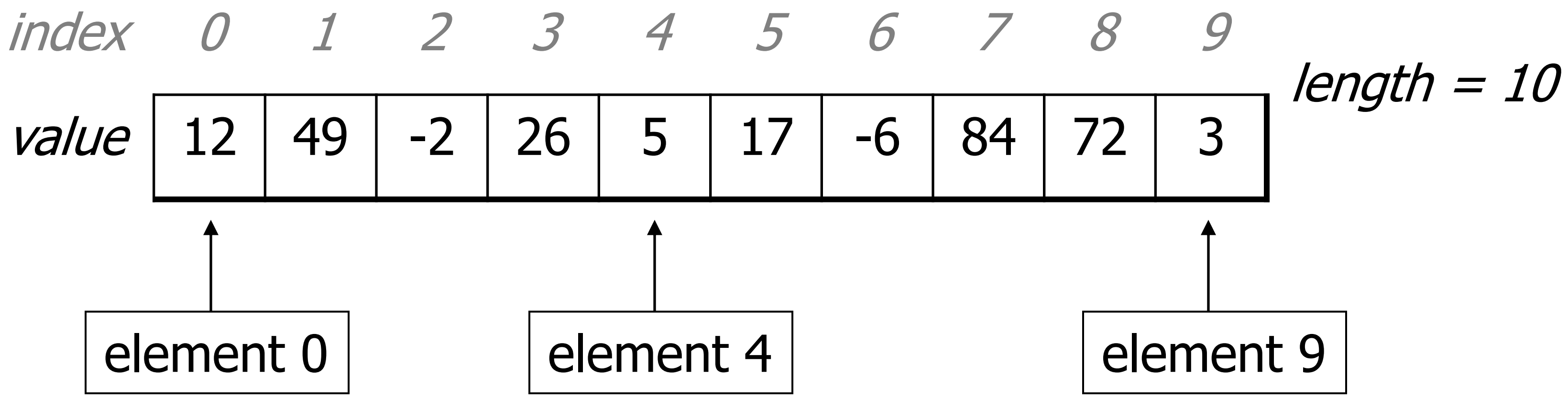
Arrays (review); `ArrayList`

reading: 10.1

slides adapted from Marty Stepp and Hélène Martin
http://www.cs.washington.edu/143/

Arrays (7.1)

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



Array declaration

type`[]` **name** `= new` **type**`[`**length**`];`

– Length explicitly provided. All elements' values initially 0.

```
int[] numbers = new int[5];
```

index	*0*	*1*	*2*	*3*	*4*
value	0	0	0	0	0

type`[]` **name** `= {`**value**`,` **value**`,` ... **value**`};`

– Infers length from number of values provided. Example:

```
int[] numbers = {12, 49, -2, 26, 5, 17, -6};
```

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*
value	12	49	-2	26	5	17	-6

Accessing elements; length

```
name[index]                    // access
name[index] = value;           // modify
name.length
```

- Legal indexes: between **0** and the **array's length - 1**.

```
numbers[3] = 88;
for (int i = 0; i < numbers.length; i++) {
    System.out.print(numbers[i] + " ");
}
System.out.println(numbers[-1]);  // exception
System.out.println(numbers[7]);   // exception
```

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*
value	12	49	-2	**88**	5	17	-6

Array as param/return

```
public static void name(type[] name) {     // param
public static type[] name(params)          // return
```

- Example:

```
public static int[] stutter(int[] a) {
    int[] result = new int[a.length * 2];
    for (int i = 0; i < result.length; i++) {
        result[i] = a[i / 2];
    }
    return result;
}
```

- Call:

```
int[] nums = {2, -4, 7};
int[] result = stutter(nums);
               // {2, 2, -4, -4, 7, 7}
```

The `Arrays` class

- Class `Arrays` in package `java.util` has useful static methods for manipulating arrays:

Method name	**Description**
`binarySearch(`**array**`, `**value**`)`	returns the index of the given value in a <u>sorted</u> array (< 0 if not found)
`copyOf(`**array**`, `**length**`)`	returns a new array with same elements
`equals(`**array1**`, `**array2**`)`	returns `true` if the two arrays contain the same elements in the same order
`fill(`**array**`, `**value**`)`	sets every element in the array to have the given value
`sort(`**array**`)`	arranges the elements in the array into ascending order
`toString(`**array**`)`	returns a string representing the array, such as `"[10, 30, 17]"`

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

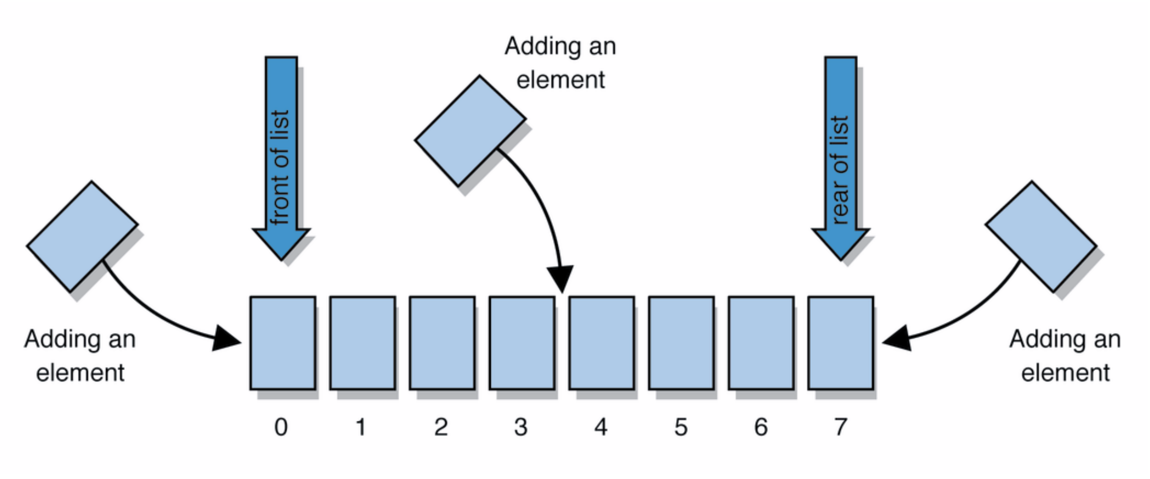
  - Is this good code? Why or why not?

Collections and lists

- **collection**: an object that stores data ("**elements**")

```
import java.util.*;    // to use Java's collections
```

- **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



Idea of a list

- An `ArrayList` is like an array that resizes to fit its contents.

- When a list is created, it is initially empty.

```
[]
```

- You can add items to the list. (By default, adds at end of 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.

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.

```
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]"`

* *(a partial list; see 10.1 for other methods)*

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
```

Out-of-bounds

- Legal indexes are between **0** and the **list's size() - 1**.
  - Reading or writing any index outside this range will cause an `IndexOutOfBoundsException`.

```
ArrayList<String> names = new ArrayList<String>();
names.add("Marty");    names.add("Kevin");
names.add("Vicki");    names.add("Larry");
System.out.println(names.get(0));          // okay
System.out.println(names.get(3));          // okay
System.out.println(names.get(-1));         // exception
names.add(9, "Aimee");                     // exception
```

index	*0*	*1*	*2*	*3*
value	Marty	Kevin	Vicki	Larry

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;
}
```

ArrayList of primitives?

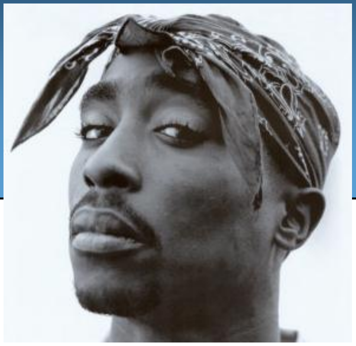
- The type you specify when creating an `ArrayList` must be an object/class type; it cannot be a primitive type.

```
// 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.

```
// legal; 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);
```

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 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--;
    }
}
```