CSCI 1103: Arrays

Chris Kauffman

Last Updated:
Fri Oct 13 09:18:58 CDT 2017
Logistics

Reading from Eck

- Ch 3.8 Intro to Arrays
- Ch 2.3.2-3 Classes, Objects, Strings

Goals

- Reference vs Primitive
- Arrays

Project 2

- Due Sunday
- Conditionals, loops arrays

Lab04: Loops
Will cover what we've been up to with while and for

Exam 1: Wed 10/11
Review Mon 10/9
Aggregate Types in Programming

- All programming languages provide some basic types like numbers and booleans
- Variable name refers to one of value of this kind, e.g.
  
  ```
  int i = 1;
  double x = 5.6;
  ```
  
- Most problems require more than this giving rise to aggregate types: a single name with multiple values
- Aggregate data can be
  - *Homogeneous*: groups of all the same
  - *Heterogeneous*: groups where some are different
Homogeneous Data

- All same data type
- Single name, multiple ints, multiple doubles, etc.
- Usually indexed by element number (4th elem, 9th elem)
- Example: arrays, collection of the same thing (*homogeneous*)
- Elements accessed via *array*[index]

Heterogeneous Data

- Data types different
- Single name, multiple values in an combination
- Example: need 1 int, 1 double, 2 booleans
- Usually indexed by *field name* as in
  - `myStudent.gpa = 3.91;`
  - `myStudent.name = "Sam";`
- Example: *classes/objects* in Java, grouped data

Now and Later

- Will discuss arrays and *Strings* now (homogeneous)
- Deal with classes/objects later (heterogeneous)
Two Kinds of types: Primitive and References

<table>
<thead>
<tr>
<th>Primitives</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Little types are primitives</td>
<td>- Big types including types you’ll create</td>
</tr>
<tr>
<td>- int, double, char, boolean, long, short, float...</td>
<td>- String, Scanner, File, Sauce, Exception, ... And all arrays</td>
</tr>
<tr>
<td>- Live directly inside a memory cell</td>
<td>- Contents of memory cell refer to another spot in memory where the thing actually resides</td>
</tr>
<tr>
<td>- Each primitive type has its own notion of a zero value: know what they are as all arrays are initialized to these values</td>
<td>- Usually refer to a heap location</td>
</tr>
<tr>
<td>- Only a small number of primitive types, can’t make new ones</td>
<td>- Identical to a pointer but operations are limited</td>
</tr>
<tr>
<td></td>
<td>- Have a single zero-value: null which points nowhere</td>
</tr>
</tbody>
</table>
Arrays: Lots of the Same Kind

- Declared with the square braces
  ```java
  int arr[];
  ```
- Initially null: zero value for reference types
  ```java
  if(arr1 == null) { ... }
  ```
- A fixed hunk of memory: must be explicitly allocated, state number of elements desired
  ```java
  arr = new int[5];
  ```
- Each element or slot holds one of the same type of data
- Each element referred to by index, 0-indexed (first element is at index 0)
- Elements can be assigned with square brace notation
  ```java
  arr[0] = 10;
  arr[1] = 15;
  ```
- Tracks length as a field
  ```java
  int size = arr.length;
  ```
- Last element is at arr.length-1
  ```java
  arr[ arr.length-1 ] = 35;
  ```
- Elements can be retrieved using square brace notation
  ```java
  int elem = arr[1];
  ```
Exercise: Array Pictures

```java
{ int arr1[];
 double arr2[];
 int idx;
 arr1 = new int[2];
 idx = 1;
 arr1[ idx ] = 15;
 arr2 = new double[3];
 ... }
```

Draw these changes

```java
{ ...
  arr1[0] = 25;
  arr2[2] = 1.234;
  arr1[1]++; 
  arr2 = new double[2];
  ...
 }
```
Answer: Array Pictures

```java
1 { int arr1[];
2      double arr2[];
3     int idx;
4 arr1 = new int[2];
5      idx = 1;
6 arr1[idx] = 15;
7 arr2 = new double[3];
8 ...

arr2 = new double[3];
10 ...
11 arr1[0] = 25;
12 arr2[2] = 1.234;
13 arr1[1]++;
14 arr2 = new double[2];
15 ...
16 }
```

Draw these changes
Memory Allocation and Garbage Collection

- Variables are either
  - Primitives: values in the box directly
  - References: value in box points to elsewhere

- Memory that is referenced from elsewhere must usually be allocated

- In Java, `new` keyword indicates an allocation
  ```java
  int a[] = new int[4]; // find me space for 5 ints
  ```

- Can stop referring to an area of memory
  ```java
  a = null; // stop pointing at that area
  ```

- Memory that was allocated but no longer referenced is subject to garbage collection

- Occasionally, program pauses and unloved memory is reclaimed, recycled for other uses
Array Initialization

- Initializing arrays can be tedious
- Java provides special syntax to ease this
- Will often use lines like
  ```java
  int arr[] = new int[]{5, 10, 15, 20};
  ```
  to set up problems
- Equivalent syntax
  ```java
  int arrA[];
  int [] arrB;
  ```
  Some prefer 2nd for readability:
  ```java
  integer array arrB
  ```

```java
1 // All these create roughly equivalent
2 // arrays with 3 elements: 15, 25, 35
3 public class ArrayInit{
4   public static
5     void main(String args[]){
6         int arrA[];
7         arrA = new int[3];
8         arrA[0] = 15;
11
12        int arrB[] = new int[3];
13        arrB[0] = 15;
16
17        int arrC[] = {15, 25, 35};
18
19        int arrD[];
20        arrD = new int[]{15, 25, 35};
21
22        int arrE[];
23        // DOESN’T WORK
24        // arrE = {15, 25, 35};
25     }
26 }
```
Exercise: Exceptional Behavior

Examine the two short programs below and determine their output.

```java
public class ArrayOOB{
    public static void main(String args[]){
        int arrA[] = new int[]{15, 25, 35};
    }
}
```

```java
public class ArrayNPE{
    public static void main(String args[]){
        int arrA[] = new int[]{15, 25, 35};
        arrA = null;
        System.out.printf("arrA[0] = %d\n",arrA[0]);
    }
}
```

Hint: Things may go sideways...
Answer: Exceptional Behavior

- Exceptions occur during runtime when problems occur
- Exceptions indicate line number but source may be elsewhere

Index out of Bounds

```java
1 // Throws an ArrayIndexOutOfBoundsException
2 public class ArrayOOB{
3     public static void main(String args[]) {
4         int arrA[] = new int[]{15, 25, 35};
6     }
7 }
```

> java ArrayOOB
Exception in thread "main"
    java.lang.ArrayIndexOutOfBoundsException: 3
at ArrayOOB.main(ArrayOOB.java:5)

- Attempt to access index beyond array size
- Usually a logic bug, check `arr.length` carefully

NullPointerException

```java
1 // Throws a NullPointerException
2 public class ArrayNPE{
3     public static void main(String args[]) {
4         int arrA[] = new int[]{15, 25, 35};
5         arrA = null;
6         System.out.printf("arrA[0] = %d\n",arrA[0]);
7     }
8 }
```

> java ArrayNPE
Exception in thread "main"
    java.lang.NullPointerException
at ArrayNPE.main(ArrayNPE.java:6)

- Attempt to dereference a pointer to nowhere
- All references, including arrays, subject to this one
Exercise: Arrays and Loops Go Hand-in-Hand

- Loops typically used to iterate over elements of arrays
- Loop bounds tied to `arr.length`

```java
1 // Typical loop to print all elements of an array
2 public class ArrayPrinting{
3    public static
4        void main(String args[]) {
5            int arr[] = {15, 25, 35, 45, 55, 65};
6
7            System.out.printf("Length of array is %d\n",arr.length);
8            for(int i=0; i<arr.length; i++){
9                System.out.printf("[%d] = %d\n",i,arr[i]);
10            }
11        }
12    }
```

Questions: `ArrayPrintingVariants.java`

- What is the output of this program?
- Can the array be changed without altering the loop?
- Change the loop to print out only odd indices 1,3,5 etc
- Change the loop to print out only elements larger than 30
- Change the loop to print even indices in reverse (!)
Answers: Arrays and Loops Go Hand-in-Hand

```java
1 // Typical loop to print all elements of an array
2 public class ArrayPrintingVariants{
3     public static
4         void main(String args[]){
5             int arr[] = {15, 25, 35, 45, 55, 65};
6 // int arr[] = {15, 25, 35, 22, 55, 65, 17}; // ALL INDICES
7 // [0] = 15
8         System.out.printf("Length of array is %d\n",arr.length);
9 // [1] = 25
10 System.out.printf("ALL INDICES\n"); // Print everything // [2] = 35
11     for(int i=0; i<arr.length; i++) { // [3] = 45
12         System.out.printf("[%d] = %d\n",i,arr[i]); // [4] = 55
13     } // [5] = 65
15
16 System.out.printf("ODD INDICES\n");
17     for(int i=1; i<arr.length; i+=2){ // Print only odd indices
18         System.out.printf("[%d] = %d\n",i,arr[i]);
19     }
20 // [7] = 45
21
22 System.out.printf("ELEMENTS > 30\n");
23     for(int i=0; i<arr.length; i++) { // Print elements > 30
24         if(arr[i] > 30){
25             System.out.printf("[%d] = %d\n",i,arr[i]);
26         }
27     }
28 // [8] = 45
29
30 System.out.printf("EVEN INDICES IN REVERSE\n");
31     int start = arr.length-1; // Find starting point
32     if(arr.length % 2 == 0){ // odd/even length differences
33         start--;
34     }
35 // [9] = 45
36     for(int i=start; i>=0; i-=2){ // Print even indices in reverse
37         System.out.printf("[%d] = %d\n",i,arr[i]);
38     }
39 }
40 }
```
Exercise: Sequence Reversal

A program to...

- Prompt for input size (positive integer)
- Allocate array of integers of given size
- In loop, read into array
- Print back in reverse order

Notes

- 4 to 5 different solution variants for this
- NOT possible to do this without an aggregate data type like arrays

> javac ReverseSequence.java

> java ReverseSequence
Enter sequence length:
8
Enter 8 integers: (ex: 13)
10 20 30 40 50 60 70 80
Sequence in reverse:
80 70 60 50 40 30 20 10

> java ReverseSequence
Enter sequence length:
5
Enter 5 integers: (ex: 13)
15 14 13 12 11
Sequence in reverse:
11 12 13 14 15

> java ReverseSequence
Enter sequence length:
3
Enter 3 integers: (ex: 13)
6 1 2
Sequence in reverse:
2 1 6
Answer: Sequence Reversal

1  public class ReverseSequence{
2     public static void main(String args[]) {
3         System.out.println("Enter sequence length:");
4         int seqLength = TextIO.getInt();  // get size from user
5         int sequence[] = new int[seqLength];  // allocate space for sequence
6         System.out.printf("Enter %d integers: (ex: 13)\n",
7                 seqLength);
8
9            for(int i=0; i<seqLength; i++) {  // input loop: read all
10                sequence[i] = TextIO.getInt();  // integers from user
11            }
12
13            System.out.println("Sequence in reverse:");
14            for(int i=seqLength-1; i>=0; i--){  // print out sequence in
15                System.out.printf("%d ",sequence[i]);  // reverse order
16            } System.out.println();
17         }
18     }
19 }

Common Solution Variants

- Read sequence into array from last to first, print in forward order
- Allocate second array, copy over in reverse order, print copy from front
- Reverse array in place, print from front
Exercise: Guessing Game with History PLAN

▶ Consider Code Demo to the right
▶ Guess up to 5 times
▶ Print high/low on incorrect guess
▶ Print history of guesses if correct

Answer the Following

▶ How many times to loop?
▶ What must be done every iteration unconditionally?
▶ How will history be tracked?
▶ Conditions inside loop?
▶ Conditions after loop?
▶ How to print history?

Form Your Plan (no code yet)

> javac GuessingHistory.java
> java GuessingHistory
Guess between 1 and 100: (Max 5 guesses)
50
Too big
30
Too small
40
Too small
48
Too big
44
Too big
Loser!
> java GuessingHistory
Guess between 1 and 100: (Max 5 guesses)
41
Too small
43
Too big
42
Correct! It took you 3 guesses which were: 41 43 42
> java GuessingHistory
Guess between 1 and 100: (Max 5 guesses)
98
Too big
17
Too small
31
Too small
42
Correct! It took you 4 guesses which were: 98 17 31 42
Exercise: Guessing Game with History CODE

- Consider Code Demo to the right
- Guess up to 5 times
- Print high/low on incorrect guess
- Print history of guesses if correct

Write Code for Game

- Will need an array, number of guesses
- Input loop with conditions in it
- Loop to print history for correct guess

```
> javac GuessingHistory.java
> java GuessingHistory
Guess between 1 and 100: (Max 5 guesses) 50
Too big
30
Too small
40
Too small
48
Too big
44
Too big

Loser!

> java GuessingHistory
Guess between 1 and 100: (Max 5 guesses) 41
Too small
43
Too big
42
Correct! It took you 3 guesses which were: 41 43 42

> java GuessingHistory
Guess between 1 and 100: (Max 5 guesses) 98
Too big
17
Too small
31
Too small
42
Correct! It took you 4 guesses which were: 98 17 31 42
```
public class GuessingHistory{
    public static void main(String args[]) {
        int secret = 42; // secret num for guessing
        int maxGuesses = 5; // limit guesses
        int history[] = new int[maxGuesses]; // array for history
        int nGuesses = 0; // current total guesses
        int guess = -1; // current guess

        System.out.printf("Guess between 1 and 100: (Max %d guesses)\n", maxGuesses);

        for(int i=0; i<maxGuesses; i++){
            guess = TextIO.getInt();
            history[nGuesses] = guess; // Update history
            nGuesses++;

            if(guess == secret){ // Check for correct guess
                break; // break from loop
            } else if(guess > secret){ // Hint if not correct
                System.out.println("Too big");
            } else if(guess < secret){
                System.out.println("Too small");
            }
        }

        if(guess == secret){ // Correct guess
            System.out.printf("Correct! It took you %d guesses which were:\n", nGuesses);
            for(int i=0; i<nGuesses; i++){
                System.out.printf("%d ",history[i]);
            }
            System.out.println();
        } else{ // Ran out of guesses
            System.out.println("Loser!");
        }
    }
}
Exercise: Arrays are a Reference Type

- Consider code to right
- Interesting assignment:
  ```java
  int arrB[] = arrA;
  ```
- Has a MAJOR effect on remaining program
- Predict output of this program

```java
public class ArrayAlias{
  public static void main(String args[]){
    int arrA[] = new int[]{15, 25, 35};
    int arrB[] = arrA; // !!!
    arrA[0] = 65;
    for(int i=0; i<arrA.length; i++){
      System.out.printf("%d ",arrA[i]);
    }
    System.out.println();
    for(int i=0; i<arrB.length; i++){
      System.out.printf("%d ",arrB[i]);
    }
    System.out.println();
    boolean arrsEqual = arrA == arrB;
    System.out.println(arrsEqual);
  }
}
```
Answer: Arrays are a Reference Type (Pictures)

- Assignment operation `x = y;` always copies a box value of `y` to box `x` in Java
- Effect for arrays is to create an alias: both variables refer to same area of memory

```java
javac ArrayAlias.java
java ArrayAlias
65 25 90
65 25 90
true
```
Distinct Arrays

- To get distinct arrays, must allocate memory twice
- The `new` keyword will appear twice for 2 arrays (roughly)
- Typical to use a loop copy from one array to the other

```java
public class ArraysDistinct{
    public static void main(String args[])
    {
        int arrA[] = new int[]{15, 25, 35};
        int arrB[] = new int[arrA.length];  // same size as arrA
        for(int i=0; i<arrA.length; i++)    // copy arrA elements
            arrB[i] = arrA[i];              // to arrB

        arrA[0] = 65;    // only arrA changed
        arrB[2] = 90;    // only arrB changed

        // arrA is {65, 25, 35}
        // arrB is {15, 25, 90}
        for(int i=0; i<arrA.length; i++){
            System.out.printf("%d ",arrA[i]);
        }
        System.out.println();
        for(int i=0; i<arrB.length; i++){
            System.out.printf("%d ",arrB[i]);
        }
        System.out.println();

        boolean arrsEqual = arrA == arrB;  // different locations
        System.out.println(arrsEqual);     // false
    }
}```
Meaning of Shallow Equality ==

- Operator == works for all kinds of things in Java: int, double, boolean, arrays...
- Compares contents of one box to another
- Only single boxes compared
- Common misconception

```java
int arrA[] = new int[]{5,7,9};
int arrB[] = new int[]{5,7,9};
if(arrA == arrB){
    System.out.println("Equal");
} else{
    System.out.println("Not Equal");
}
```

- For arrays, must use a loop to compare entire contents to one another
Exercise: Read two Arrays and Compare

Basic Behavior

```
> java CompareSequences
Enter sequence length: 3
Enter First 3 integers: 1 3 5
Enter Second 3 integers: 1 3 5
# seq1 seq2
0 1 1
1 3 3
2 5 5
Sequences equal: true
```

```
> java CompareSequences
Enter sequence length: 5
Enter First 5 integers: 10 20 30 40 50
Enter Second 5 integers: 10 22 30 44 50
# seq1 seq2
0  10  10
1  20  22
2  30  30
3  40  44
4  50  50
Sequences equal: false
```

```
> java CompareSequences
Enter sequence length: 4
Enter First 4 integers: 199 22 8 1011
Enter Second 4 integers: 199 22 8 1101
# seq1 seq2
0  199  199
1  22  22
2  8  8
3  1011  1101
Sequences equal: false
```

Implementation Notes

- Use a `printf()` to get nicely aligned columns
  - 1 char #/index, 4 chars seq1, 4 chars seq2
- Read both sequences first, then print both
- Use a loop to compare all elements
- Start with `areEqual = true;`
- If any differences found, flip to `false`
public class CompareSequences {
    public static void main(String args[]) {
        System.out.println("Enter sequence length:");
        int seqLength = TextIO.getInt(); // get size from user
        int seq1[] = new int[seqLength]; // allocate space for seq 1

        System.out.printf("Enter First %d integers:
", seqLength);
        for(int i=0; i<seqLength; i++) { // input loop: read seq 1
            seq1[i] = TextIO.getInt(); // integers from user
        }

        int seq2[] = new int[seqLength]; // allocate space for sequence 2
        System.out.printf("Enter Second %d integers:
", seqLength);
        for(int i=0; i<seqLength; i++) { // input loop: read all
            seq2[i] = TextIO.getInt(); // integers from user
        }

        System.out.printf("%2s %4s %4s
", // print table header
            ","#","seq1","seq2");

        for(int i=0; i<seqLength; i++) { // print out sequence table
            System.out.printf("%2d %4d %4d
", i,seq1[i],seq2[i]);
        }

        // Check for equality of all elements
        boolean areEqual = true; // assume equal
        for(int i=0; i<seqLength; i++) {
            if(seq1[i] != seq2[i]){ // detect differences
                areEqual = false; // change equal to not
            }
        }

        System.out.printf("Sequences equal: %b\n",areEqual);
    }
}