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

- Referece 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

Latin for: All the same VS Potentially 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

Primitives

- Little types are primitives
- `int`, `double`, `char`, `boolean`, `long`, `short`, `float...`
- Live directly inside a memory cell
- Each primitive type has its own notion of a zero value: know what they are as all arrays are initialized to these values
- Only a small number of primitive types, can't make new ones

References

- Big types including types you'll create
- `String`, `Scanner`, `File`, `Sauce`, `Exception`, ... And all arrays
- Contents of memory cell *refer* to another spot in memory where the thing actually resides
- Usually refer to a heap location
- Identical to a pointer but operations are limited
- Have a single zero-value: `null` which points nowhere

Arrays: Lots of the Same Kind

- Declared with the square braces

  ```
  int arr[];
  ```

- Initially `null`: zero value for reference types

  ```
  if(arr1 == null) { ... }
  ```

- A fixed hunk of memory: must be explicitly allocated, state number of elements desired

  ```
  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

  ```
  arr[0] = 10;
  arr[1] = 15;
  ```

- Tracks length as a field

  ```
  int size = arr.length;
  ```

- Last element is at `arr.length-1`

  ```
  arr[ arr.length-1 ] = 35;
  ```

- Elements can be retrieved using square brace notation

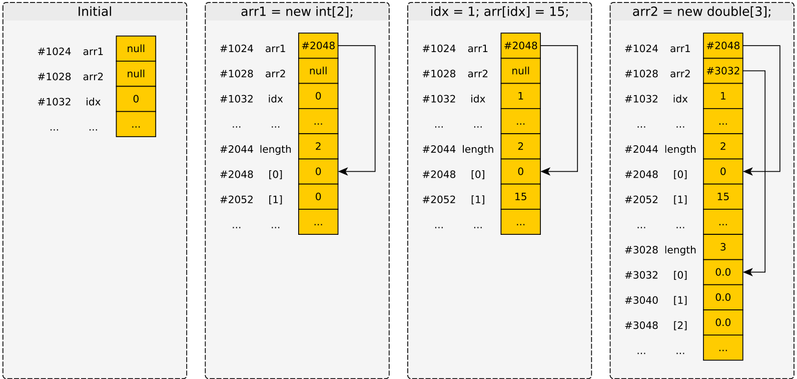
  ```
  int elem = arr[1];
  ```

Exercise: Array Pictures

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

Draw these changes

```
    ...
    arr1[0] = 25;
    arr2[2] = 1.234;
    arr1[1]++;
    arr2 = new double[2];
    ...
  }
```

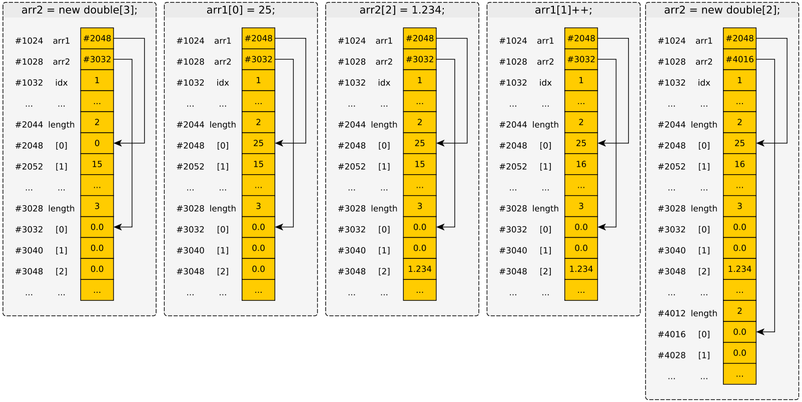


Answer: Array Pictures

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

Draw these changes

```
    ...
    arr1[0] = 25;
    arr2[2] = 1.234;
    arr1[1]++;
    arr2 = new double[2];
    ...
}
```



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

  ```
  int a[] = new int[4];  // find me space for 5 ints
  ```
- Can stop referring to an area of memory

  ```
  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

  ```
  int arr[] = new int[]{
    5, 10, 15, 20
  };
  ```

  to set up problems
- Equivalent syntax

  ```
  int arrA[];
  int [] arrB;
  ```

  Some prefer 2nd for readablness:
  integer array arrB

```
// All these create roughly equivalent
// arrays with 3 elements: 15, 25, 35
public class ArrayInit{
  public static
  void main(String args[]) {
    int arrA[];
    arrA = new int[3];
    arrA[0] = 15;
    arrA[1] = 25;
    arrA[2] = 35;

    int arrB[] = new int[3];
    arrB[0] = 15;
    arrB[1] = 25;
    arrB[2] = 35;

    int arrC[] = {15, 25, 35};

    int arrD[];
    arrD = new int[]{15, 25, 35};

    int arrE[];
    // DOESN'T WORK
    // arrE = {15, 25, 35};
  }
}
```

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};
    System.out.printf("arrA[3] = %d\n",arrA[3]);
  }
}
```

```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

```
1  // Throws an ArrayIndexOutOfBoundsException
2  public class ArrayOOB{
3    public static void main(String args[]) {
4      int arrA[] = new int[]{15, 25, 35};
5      System.out.printf("arrA[3] = %d\n",arrA[3]);
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`

```
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`

```
// Typical loop to print all elements of an array
public class ArrayPrinting{
  public static
  void main(String args[]) {
    int arr[] = {15, 25, 35, 45, 55, 65};

    System.out.printf("Length of array is %d\n",arr.length);
    for(int i=0; i<arr.length; i++){
      System.out.printf("[%d] = %d\n",i,arr[i]);
    }
  }
}
```

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

```
// Typical loop to print all elements of an array
public class ArrayPrintingVariants{
  public static
  void main(String args[]) {
    int arr[] = {15, 25, 35, 45, 55, 65};
    // int arr[] = {15, 25, 35, 22, 55, 65, 17};                     // ALL INDICES
                                                                     // [0] = 15
    System.out.printf("Length of array is %d\n",arr.length);         // [1] = 25
                                                                     // [2] = 35
    System.out.printf("ALL INDICES\n");           // Print everything  // [3] = 45
    for(int i=0; i<arr.length; i++){                                 // [4] = 55
      System.out.printf("[%d] = %d\n",i,arr[i]);                     // [5] = 65
    }                                                                // [6] = 13

    System.out.printf("ODD INDICES\n");
    for(int i=1; i<arr.length; i+=2){             // Print only odd indices
      System.out.printf("[%d] = %d\n",i,arr[i]);
    }

    System.out.printf("ELEMENTS > 30\n");
    for(int i=0; i<arr.length; i++){              // Print elements > 30
      if(arr[i] > 30){
        System.out.printf("[%d] = %d\n",i,arr[i]);
      }
    }

    System.out.printf("EVEN INDICES IN REVERSE\n");
    int start = arr.length-1;                     // Find starting point
    if(arr.length % 2 == 0){                      // odd/even length differences
      start--;
    }
    for(int i=start; i>=0; i-=2){                 // Print even indices in reverse
      System.out.printf("[%d] = %d\n",i,arr[i]);
    }
  }
}
```

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

```java
public class ReverseSequence{
  public static void main(String args[]) {
    System.out.println("Enter sequence length:");
    int seqLength = TextIO.getInt();          // get size from user
    int sequence[] = new int[seqLength];      // allocate space for sequence
    System.out.printf("Enter %d integers: (ex: 13)\n",
                      seqLength);

    for(int i=0; i<seqLength; i++){           // input loop: read all
      sequence[i] = TextIO.getInt();          // integers from user
    }

    System.out.println("Sequence in reverse:");
    for(int i=seqLength-1; i>=0; i--){        // print out sequence in
      System.out.printf("%d ",sequence[i]);   // reverse order
    }
    System.out.println();
  }
}
```

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
```

Answer: Guessing Game with History CODE

```java
// Guessing game with history staored in an array
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);

    // Get guesses from user, store in array, break out on correct guess
    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");
      }
    }

    // Could end loop with either a correct guess or running out of
    // guesses, need to figure out which it is
    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++){            // Print history
        System.out.printf("%d ",history[i]);
      }
      System.out.println();
    }
    else{                                       // Ran out of guesses
      System.out.println("Loser!");
    }
  }
}
```

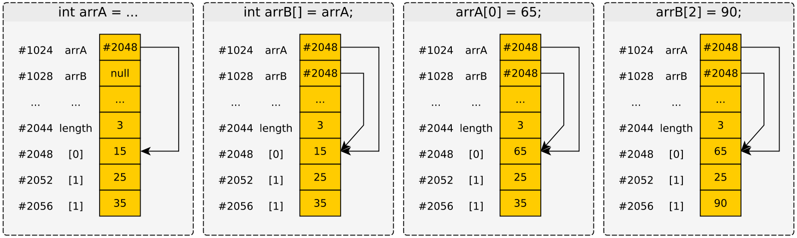
Exercise: Arrays are a Reference Type

- Consider code to right
- Interesting assignment:

  `int arrB[] = arrA;`
- Has a MAJOR effect on remaining program
- Predict output of this program

```
public class ArrayAlias{
  public static
  void main(String args[]) {
    int arrA[] = new int[]{15, 25, 35};
    int arrB[] = arrA;          // !!!

    arrA[0] = 65;
    arrB[2] = 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;
    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

```
> 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

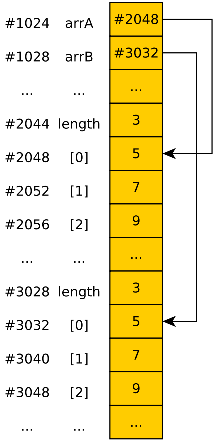
```java
public class ArraysDistinct{
  public static void main(String args[]) {
    int arrA[] = new int[]{15, 25, 35};
    int arrB[] = new int[arrA.length];   // same size as arrA
    for(int i=0; i<arrA.length; i++){    // copy arrA elements
      arrB[i] = arrA[i];                 // to arrB
   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

    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

```
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`

Answer: Read two Arrays and Compare

```java
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:\n", 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:\n",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\n",        // print table header
                      "#","seq1","seq2");

    for(int i=0; i<seqLength; i++){           // print out sequence table
      System.out.printf("%2d %4d %4d\n",
                        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);
  }
}
```