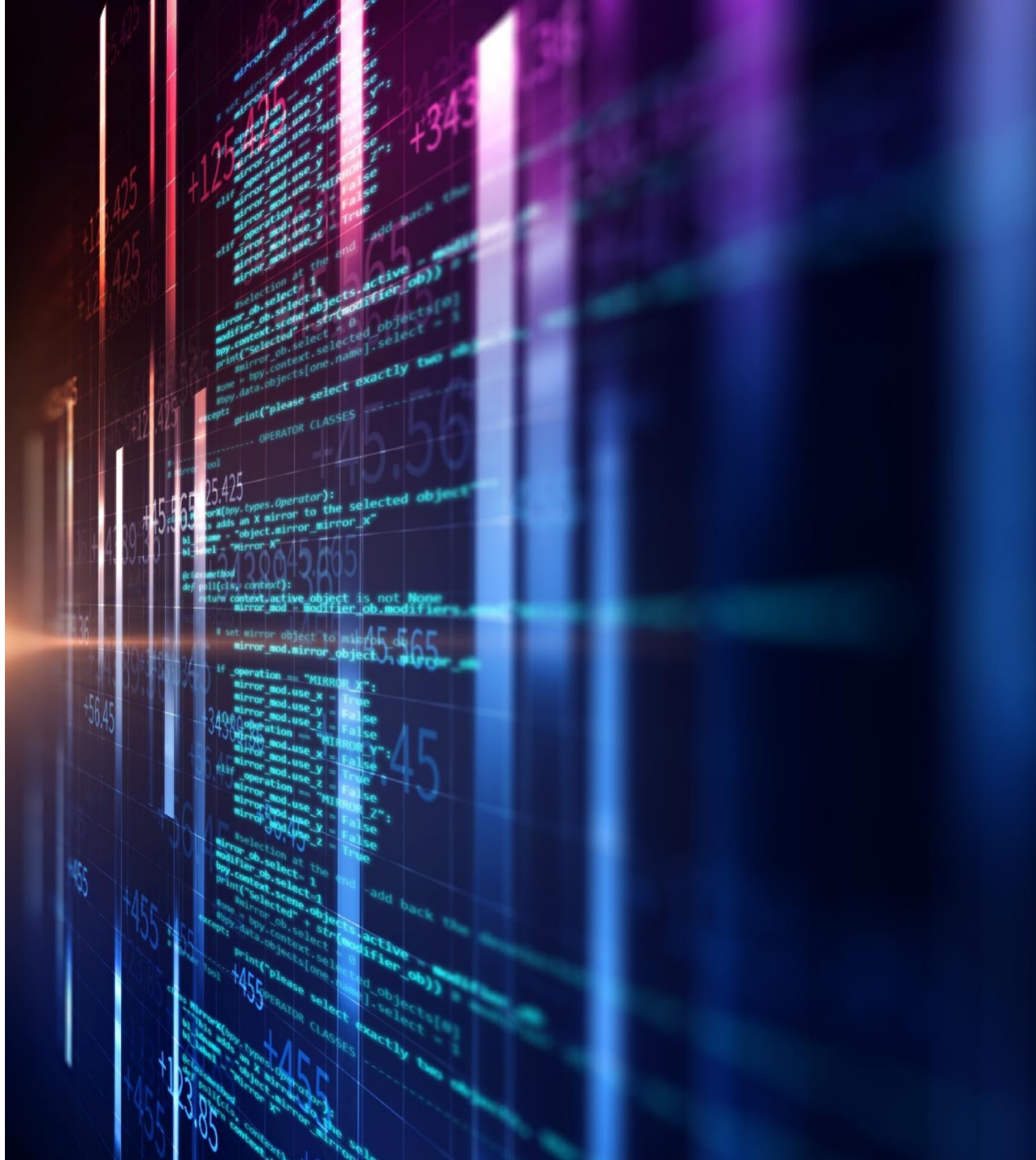
# More with Arrays

Brandon Krakowsky

Penn Engineering



# Variable Assignment

# Assignment by Value

- How are variables assigned in Java?
- For primitives (simple values) like ints, doubles, and booleans, variables are assigned *by value*

# Assignment by Value

- How are variables assigned in Java?
- For primitives (simple values) like ints, doubles, and booleans, variables are assigned *by value*

VariableAssignment.java

```java
import java.util.Arrays;

/**
 * Demonstrates assignment by value vs. assignment by reference.
 * @author lbrandon
 */
public class VariableAssignment {

    public static void main(String[] args) {

        /*
         * Assignment by value.
         */

        int a = 5;
        int b = a; //copies the value
        a = a + 1;
        System.out.println(a); //a stores the value 6
        System.out.println(b); //b stores the value 5

```

# Assignment by Reference

- For Objects (complex types) like arrays and Strings, variables are assigned *by reference*
  - This is equivalent to the concept of *pointers* in C

# Assignment by Reference

- For Objects (complex types) like arrays and Strings, variables are assigned *by reference*
  - This is equivalent to the concept of *pointers* in C

```
21
22          /*
23           * Assignment by reference.
24           */
25
26          int arr1[] = {1, 2, 3};
27          int arr2[] = arr1; //copies the reference
28          arr1[1] = 5;
29
30          //arr1 stores a reference to object [1, 5, 3]
31          System.out.println(Arrays.toString(arr1));
32          //arr2 stores a reference to the same object [1, 5, 3]
33          System.out.println(Arrays.toString(arr2));
34
```

# Copying Arrays

- How do you make a true copy of an array?

```
35
36          /*
37           * Copying arrays.
38           */
39
40          //How do you make a true copy of an array?
41          int[] myArr1 = {1, 2, -1};
42
43          //This DOES NOT COPY myArr1 to myArr2
44          int[] myArr2 = myArr1;
45          //this does not copy elements of myArr1 to myArr2
46          //myArr1 and myArr2 store references to the same array [1, 2, -1]
47
48          //How do we know?  Use == to compare object references
49          System.out.println(myArr1 == myArr2); //true
50
```

# Copying Arrays – Copy Elements

- You *can*, however, create a new array and copy the elements directly

```
51
52          //You can, however, create a new array and copy the elements directly
53          //Create an array myArr3 of same size as myArr1
54          int[] myArr3 = new int[myArr1.length];
55
56          //Copy elements of myArr1 to myArr3
57          for (int i = 0; i < myArr3.length; i++) {
58              myArr3[i] = myArr1[i];
59          }
60
61          //Use == to compare the objects
62          System.out.println(myArr1 == myArr3); //false
63
64          //And use the Arrays.equals method to compare the actual array contents (values)
65          System.out.println(Arrays.equals(myArr1, myArr3)); //true
66
```

# Copying Arrays - Cloning

- You can also clone (create an exact copy of) an array using the *clone* method
  - Many Java Objects support cloning

```
67
68          //Another way
69          //You can also clone (create an exact copy of) an array using the clone method
70          int[] anotherArr1 = {1, 8, 3};
71
72          //Copy elements of anotherArr1 to anotherArr2
73          int[] anotherArr2 = anotherArr1.clone();
74
75          //Use == to compare the objects
76          System.out.println(anotherArr1 == anotherArr2); //false
77
78          //Compare the actual array contents (values)
79          System.out.println(Arrays.equals(anotherArr1, anotherArr2)); //true
80
81      }
82  }
83
```

# Call by Value

- When we call a method, primitives (simple values) like ints, doubles, and booleans, are passed *by value*
- When you pass variables with primitives as arguments to a method, the values themselves are put into the method parameters
  - If the parameters are changed within the method, new local variables are created
  - The changes are not put back into the original arguments

# Call by Value

- When we call a method, primitives (simple values) like ints, doubles, and booleans, are passed *by value*
- When you pass variables with primitives as arguments to a method, the values themselves are put into the method parameters
  - If the parameters are changed within the method, new local variables are created
  - The changes are not put back into the original arguments

CallingMethods.java

```
1  import java.util.Arrays;
2
3  /**
4   * Demonstrates call by value vs. call by reference.
5   * @author lbrandon
6   */
7  public class CallingMethods {
```

# Call by Value

- When we call a method, primitives (simple values) like ints, doubles, and booleans, are passed *by value*

```
 8
 9    /**
10     * Updates the given primitive x.
11     * @param x to update
12     */
13    void alterPrimitive(int x) {
14
15        //increments x by 1
16        x = x + 1; //does not affect x in main method
17    }
18
```

# Call by Value

- When we call a method, primitives (simple values) like ints, doubles, and booleans, are passed *by value*

```
45    public static void main(String[] args) {
46
47        int x = 5;
48        System.out.println("Before altering x: " + x); //5
49
50        //call alterPrimitive
51        CallingMethods cm = new CallingMethods();
52        cm.alterPrimitive(x);
53
54        System.out.println("After altering x: " + x); //x still stores value 5
55        System.out.println();
```

# Call by Reference

- When we call a method, Objects (complex types) like arrays and Strings, are passed *by reference*
- When you pass variables with objects as arguments to a method, the arguments become references to the objects

# Call by Reference

- When we call a method, Objects (complex types) like arrays and Strings, are passed *by reference*
- When you pass variables with objects as arguments to a method, the arguments become references to the objects

```
19    /**
20     * Updates the given object a.
21     * @param a to update
22     */
23    void alterObject(int[] a) {
24
25        //changes contents of a
26        a[1] = 99; //does affect a in main method
```

# Call by Reference

- When we call a method, Objects (complex types) like arrays and Strings, are passed *by reference*

```
27
28          //create new variable b that points to a
29          int[] b = a;
30
31          //changes contents of b
32          b[2] = 12; //affects local a, AND a in main method
33      }
34
```

# Call by Reference

- When we call a method, Objects (complex types) like arrays and Strings, are passed *by reference*

```
57          int a[] = {1, 2, 3};
58
59          //[1, 2, 3]
60          System.out.println("Before altering a: " + Arrays.toString(a));
61
62          //call alterObject
63          cm.alterObject(a);
64
65          //a stores a reference to [1, 99, 12]
66          System.out.println("After altering a: " + Arrays.toString(a));
67      }
68  }
69
```

# Array Methods

- Arrays have very few attributes/methods
- *length* is useful, but there is no *add*, *remove*, *reverse*, etc.

**Ref:** https://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Array.html

# Example Programs

# Average Program

- Write a program that asks the user for 5 numbers (ints). It computes the average of the numbers. Allows the user to enter -1 to quit the program.

# Average Program

- Write a program that asks the user for 5 numbers (ints). It computes the average of the numbers. Allows the user to enter -1 to quit the program.

AverageProgram.java

```
1  import java.util.Scanner;
2
3  /**
4   * A program that asks the user for 5 numbers (ints).
5   * It computes the average of the numbers.
6   * Allows the user to enter -1 to quit the program.
7   *
8   * @author lbrandon
9   */
10 public class AverageProgram {
11
12     public static void main(String[] args) {
13
14         //declare and initialize int array with 5 slots
15         int[] numList = new int[5];
16
17         //declare and initialize count of numbers
18         int numCount = 0;
19
20         //create scanner
21         Scanner scan = new Scanner(System.in);
22
```

# Average Program

- Write a program that asks the user for 5 numbers (ints). It computes the average of the numbers. Allows the user to enter -1 to quit the program.

```
22
23          //set up loop to repeatedly get user input of an int
24          int num;
25          boolean playing = true;
26          while (playing == true) {
27
28              //get user input of an int
29              System.out.println("Enter num: ");
30              num = scan.nextInt();
31
32              //if the user inputs -1, exit the loop
33              if (num == -1) {
34                  playing = false; //will eventually exit loop
35              }
36
37              //put user input of int into array at index numCount
38              numList[numCount] = num;
39              numCount += 1; //increment numCount
40
41              //if the user has already provided 5 numbers, exit the loop
42              if (numCount > 4) {
43                  playing = false; //will eventually exit loop
44              }
45          }
46
```

# Average Program

- Write a program that asks the user for 5 numbers (ints). It computes the average of the numbers. Allows the user to enter -1 to quit the program.

```
46
47          //calculate sum of numbers
48          int numSum = 0;
49          for (int n : numList) {
50              numSum += n;
51          }
52
53          //calculate/print the average
54          double numAvg = numSum / numCount;
55          System.out.println("avg: " + numAvg);
56
57          //close scanner
58          scan.close();
59      }
60  }
61
```

# Average Program

- What's wrong with this program?

```
26          while (playing == true) {
27
28              //get user input of an int
29              System.out.println("Enter num: ");
30              num = scan.nextInt();
31
32              //if the user inputs -1, exit the loop
33              if (num == -1) {
34                  playing = false; //will eventually exit loop
35              }
36
37              //put user input of int into array at index numCount
38              numList[numCount] = num;
39              numCount += 1; //increment numCount
```

# Average Program

- What's wrong with this program?
- We're inserting -1 (to exit the program) into our array of numbers in our loop

```
26          while (playing == true) {
27
28              //get user input of an int
29              System.out.println("Enter num: ");
30              num = scan.nextInt();
31
32              //if the user inputs -1, exit the loop
33              if (num == -1) {
34                  playing = false; //will eventually exit loop
35              }
36
37              //put user input of int into array at index numCount
38              numList[numCount] = num;
39              numCount += 1; //increment numCount
```

# Average Program

- What's wrong with this program?
- We're inserting -1 (to exit the program) into our array of numbers in our loop
- Here's the fix!

```
26          while (playing == true) {
27
28              //get user input of an int
29              System.out.println("Enter num: ");
30              num = scan.nextInt();
31
32              //if the user inputs -1, exit the loop
33              if (num == -1) {
34                  playing = false; //will eventually exit loop
35              } else {
36                  //put user input of int into array at index numCount
37                  numList[numCount] = num;
38                  numCount += 1; //increment numCount
39              }
```