CHAPTER 7 SINGLE-DIMENSIONAL ARRAYS CHAPTER 8 MULTIDIMENSIONAL ARRAYS

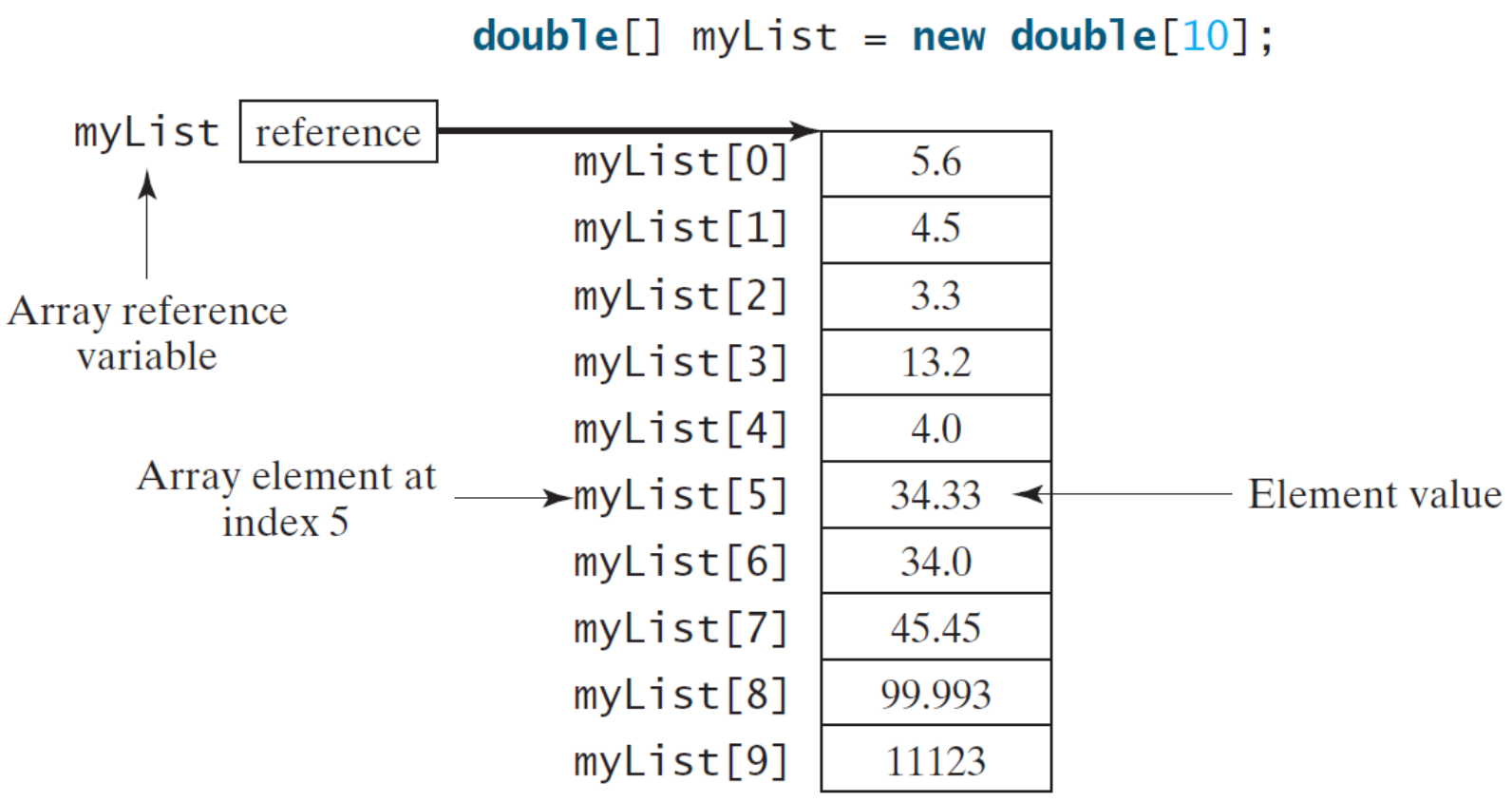
ACKNOWLEDGEMENT: THESE SLIDES ARE ADAPTED FROM SLIDES PROVIDED WITH INTRODUCTION TO JAVA PROGRAMMING, LIANG (PEARSON 2014)

MOTIVATION

- Read one hundred numbers, compute their average, and find out how many numbers are above the average.
- Store and manipulate large amounts of data
  - 52 playing cards in a deck
  - 3 thousand undergrads at UR
  - 140 characters per Tweet
  - 4 billion nucleotides in a DNA strand
  - 50 trillion cells in the human body
  - $6.022x10^{23}$ particles in a mole

INTRODUCING ARRAYS

- **Array** is a data structure that represents a collection of the same types of data.
- Array variables are **references**. More on this to come.



DECLARING ARRAY VARIABLES

- Array declaration

  ```
  datatype[] arrayRefVar;
  ```

  - Example:

    ```
    double[] myList;
    ```

- Alternate declaration form

  ```
  datatype arrayRefVar[]; // This style is allowed,
                          // but not preferred
  ```

  - Example:

    ```
    double myList[];
    ```

INITIALIZING ARRAYS

- Initializing array
  ```
  arrayRefVar = new datatype[arraySize];
  ```
  - Example:
    ```
    myList = new double[10];
    ```
- The values of the elements are default initialized
  - `false` for Boolean types
  - `0` or `0.0` for numeric types (integral or floating-point, respectively)
  - `'\u0000'` for Characters
  - `null` for Strings and object types

DECLARING AND INITIALIZING IN ONE STEP

- **datatype**[] arrayRefVar = **new** **datatype**[arraySize];
  - Example

    ```
    double[] myList = new double[10];
    ```

- **datatype** arrayRefVar[] = **new** **datatype**[arraySize];
  - Example

    ```
    double myList[] = new double[10];
    ```

THE LENGTH OF AN ARRAY

- Once an array is created, its size is fixed. It cannot be changed. You can find its size using

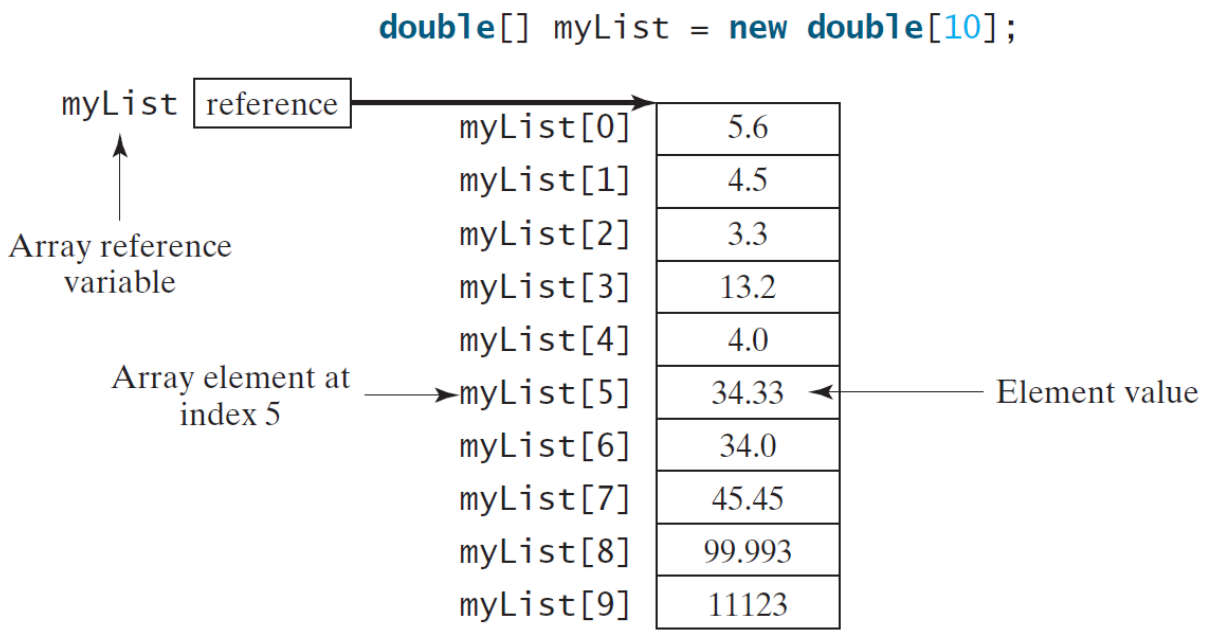
```
arrayRefVar.length
```

- For example,

```
int n = myList.length; //returns 10 based on
                       //prior construction
```

INDEXED VARIABLES

- The array elements are accessed through the index. The array indices are 0-based, i.e., it starts from `0` to `arrayRefVar.length-1`. In our example, `myList` holds ten double values and the indices are from 0 to 9.
- Each element in the array is represented using the following syntax, known as an indexed variable:
  `arrayRefVar[index];`



USING INDEXED VARIABLES

- After an array is created, an indexed variable can be used in the same way as a regular variable. For example, the following code adds the value in `myList[0]` and `myList[1]` to `myList[2]`.

```
myList[2] = myList[0] + myList[1];
```

ARRAY INITIALIZERS

- You may also use array initializers to give non-default values to arrays:

```
double[] myList =
    {1.9, 2.9, 3.4, 3.5};
```

- This shorthand syntax must be in one statement. Splitting it causes a syntax error.

- This shorthand notation is equivalent to the following statements:

```
1.double[] myList =
      new double[4];
2.myList[0] = 1.9;
3.myList[1] = 2.9;
4.myList[2] = 3.4;
5.myList[3] = 3.5;
```

TRACE PROBLEM WITH ARRAYS

```
1. int[] values = new int[5];
2.   for(int i = 1; i < 5; i++)
3.     values[i] = i + values[i-1];
4. values[0] = values[1] + values[4];
```

Declare, create, and initialize an array named values of size 5

Memory

values 0xA

0xA

Index	Value
0	0
1	0
2	0
3	0
4	0

TRACE PROBLEM WITH ARRAYS

```
1.int[] values = new int[5];
2.   for(int i = 1; i < 5; i++)
3.      values[i] = i + values[i-1];
4.values[0] = values[1] + values[4];
```

Memory

values 0xA

i 1

0xA

Index	Value
0	0
1	0
2	0
3	0
4	0

TRACE PROBLEM WITH ARRAYS

```
1.int[] values = new int[5];
2.   for(int i = 1; i < 5; i++)
3.      values[i] = i + values[i-1];
4.values[0] = values[1] + values[4];
```

i < 5 is true

Memory

values 0xA

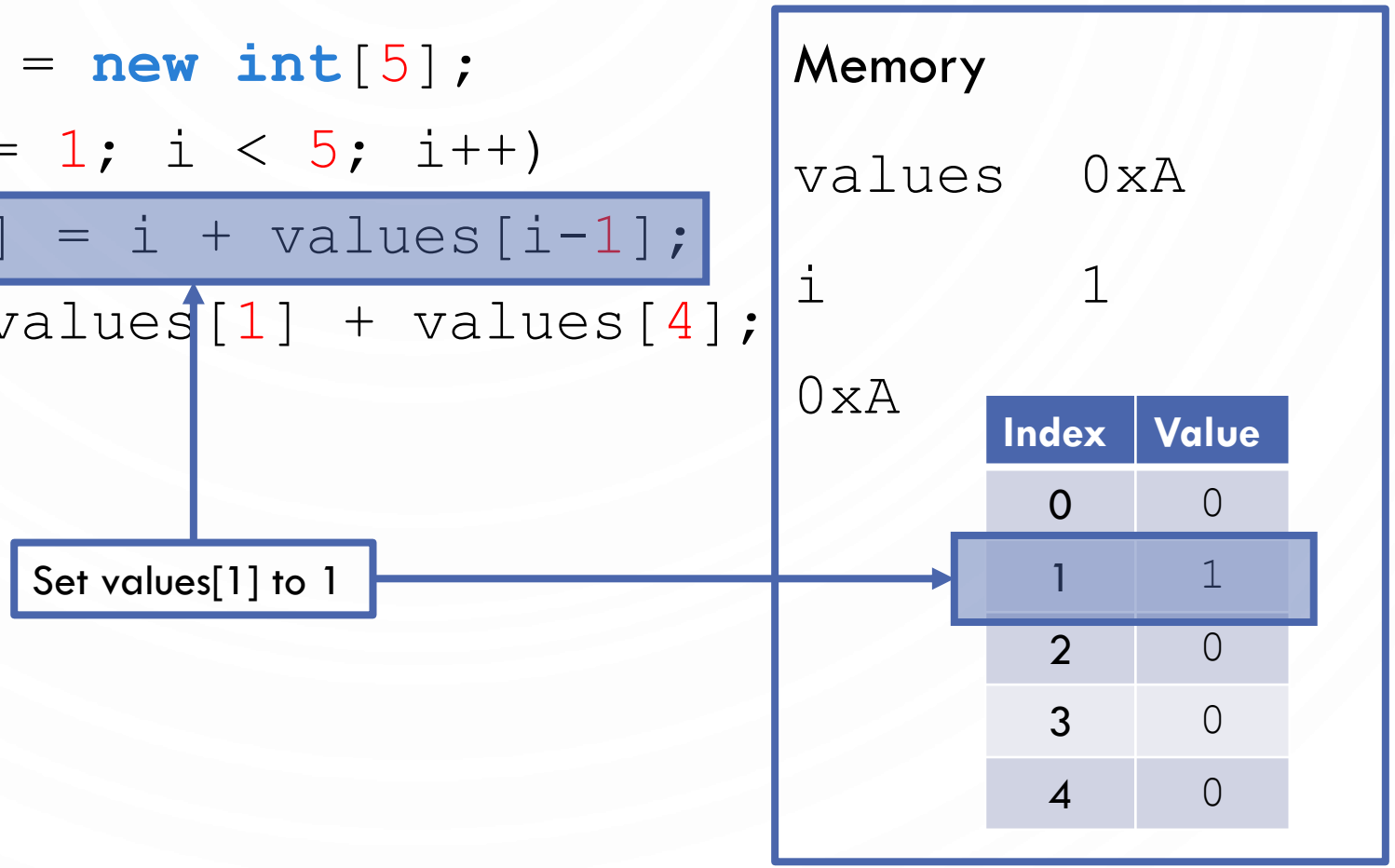
i 1

0xA

Index	Value
0	0
1	0
2	0
3	0
4	0

TRACE PROBLEM WITH ARRAYS

```
1.int[] values = new int[5];
2.   for(int i = 1; i < 5; i++)
3.      values[i] = i + values[i-1];
4.values[0] = values[1] + values[4];
```



TRACE PROBLEM WITH ARRAYS

```
1.int[] values = new int[5];
2.   for(int i = 1; i < 5; i++)
3.      values[i] = i + values[i-1];
4.values[0] = values[1] + values[4];
```

Memory

values 0xA

i 2

0xA

Index	Value
0	0
1	1
2	0
3	0
4	0

TRACE PROBLEM WITH ARRAYS

```
1.int[] values = new int[5];
2.   for(int i = 1; i < 5; i++)
3.      values[i] = i + values[i-1];
4.values[0] = values[1] + values[4];
```

i < 5 is true

Memory

values 0xA

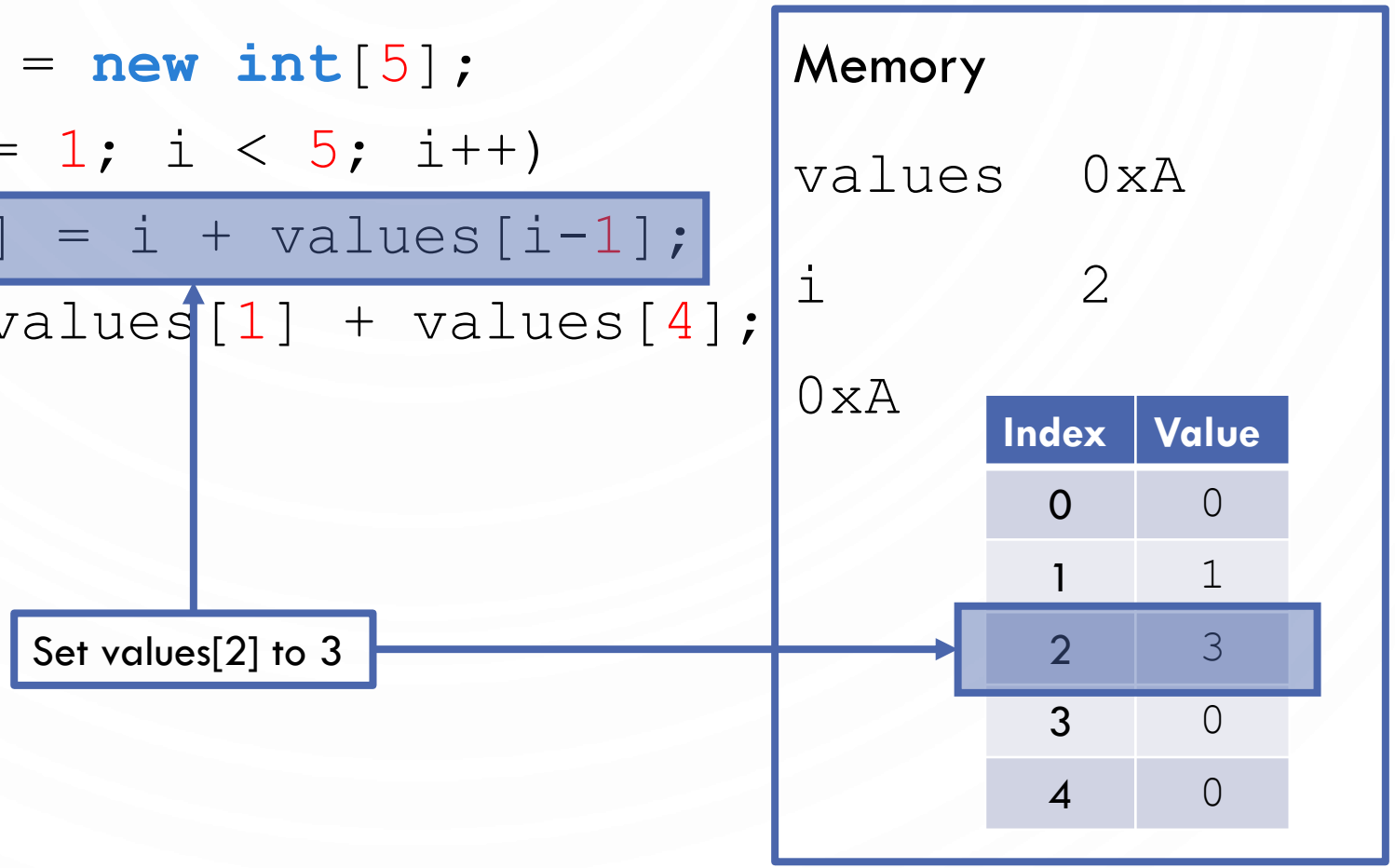
i 2

0xA

Index	Value
0	0
1	1
2	0
3	0
4	0

TRACE PROBLEM WITH ARRAYS

```
1.int[] values = new int[5];
2.   for(int i = 1; i < 5; i++)
3.      values[i] = i + values[i-1];
4.values[0] = values[1] + values[4];
```



TRACE PROBLEM WITH ARRAYS

```
1.int[] values = new int[5];
2.   for(int i = 1; i < 5; i++)
3.      values[i] = i + values[i-1];
4.values[0] = values[1] + values[4];
```

Memory

values 0xA

i 3

0xA

Index	Value
0	0
1	1
2	3
3	0
4	0

TRACE PROBLEM WITH ARRAYS

```
1.int[] values = new int[5];
2.   for(int i = 1; i < 5; i++)
3.      values[i] = i + values[i-1];
4.values[0] = values[1] + values[4];
```

i < 5 is true

Memory

values 0xA

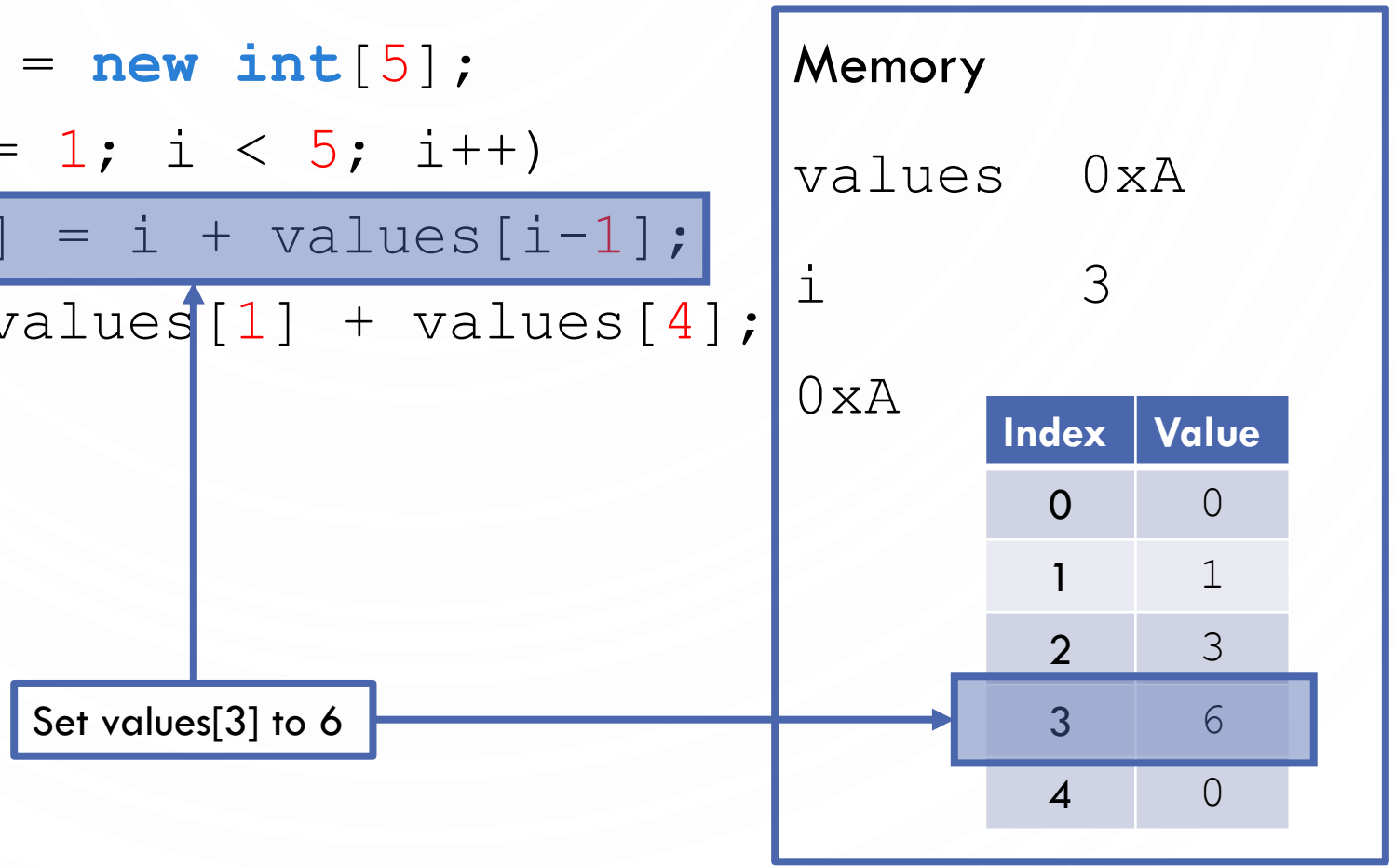
i 3

0xA

Index	Value
0	0
1	1
2	3
3	0
4	0

TRACE PROBLEM WITH ARRAYS

```
1.int[] values = new int[5];
2.   for(int i = 1; i < 5; i++)
3.      values[i] = i + values[i-1];
4.values[0] = values[1] + values[4];
```



TRACE PROBLEM WITH ARRAYS

```
1.int[] values = new int[5];
2.   for(int i = 1; i < 5; i++)
3.      values[i] = i + values[i-1];
4.values[0] = values[1] + values[4];
```

Memory

values 0xA

i 4

0xA

Index	Value
0	0
1	1
2	3
3	6
4	0

TRACE PROBLEM WITH ARRAYS

```
1.int[] values = new int[5];
2.   for(int i = 1; i < 5; i++)
3.      values[i] = i + values[i-1];
4.values[0] = values[1] + values[4];
```

i < 5 is true

Memory

values 0xA

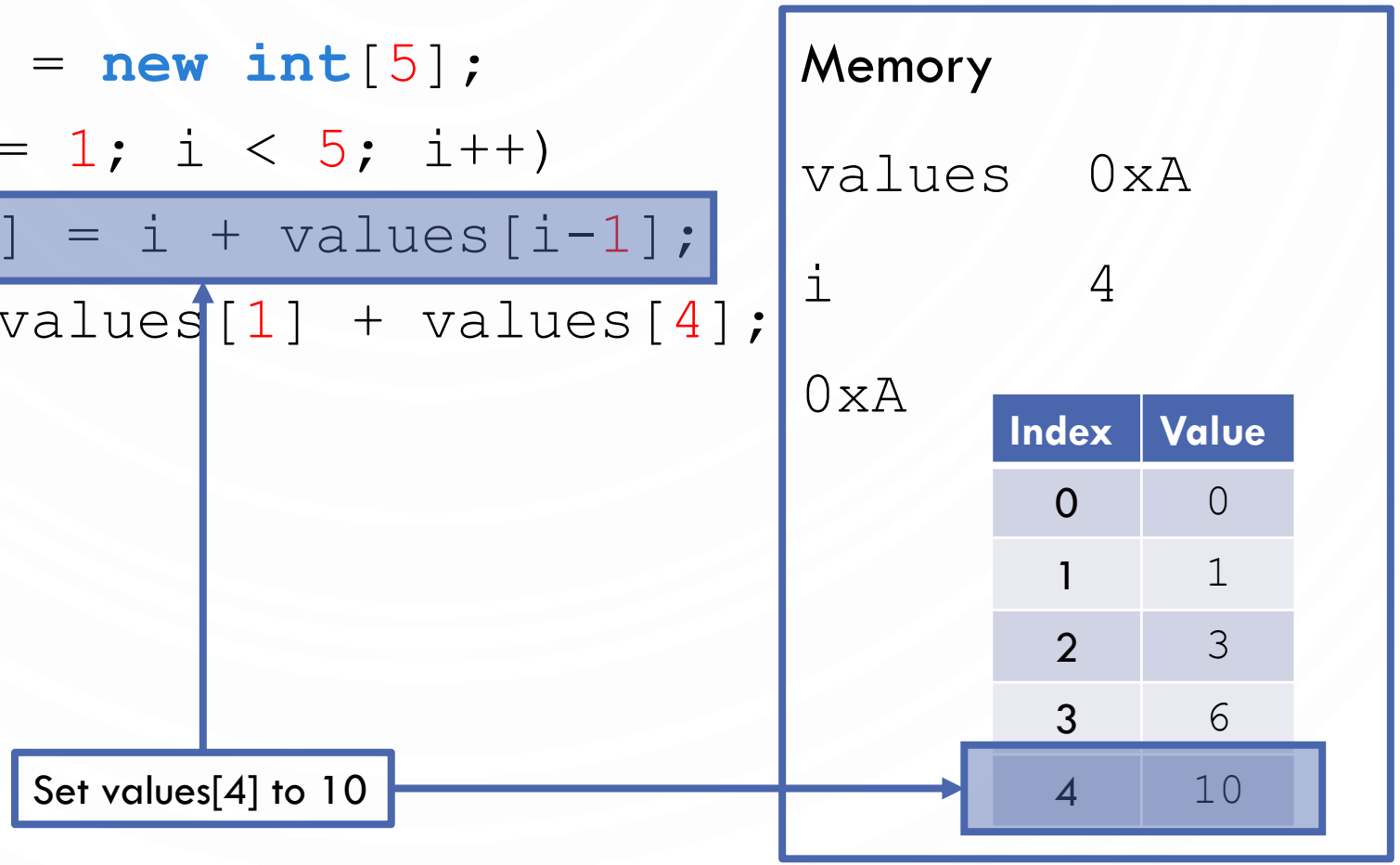
i 4

0xA

Index	Value
0	0
1	1
2	3
3	6
4	0

TRACE PROBLEM WITH ARRAYS

```
1.int[] values = new int[5];
2.   for(int i = 1; i < 5; i++)
3.      values[i] = i + values[i-1];
4.values[0] = values[1] + values[4];
```



TRACE PROBLEM WITH ARRAYS

```
1.int[] values = new int[5];
2.   for(int i = 1; i < 5; i++)
3.      values[i] = i + values[i-1];
4.values[0] = values[1] + values[4];
```

Memory

values 0xA

i 5

0xA

Index	Value
0	0
1	1
2	3
3	6
4	10

TRACE PROBLEM WITH ARRAYS

```
1.int[] values = new int[5];
2.   for(int i = 1; i < 5; i++)
3.      values[i] = i + values[i-1];
4.values[0] = values[1] + values[4];
```

i < 5 is false

Memory

values 0xA

i 5

0xA

Index	Value
0	0
1	1
2	3
3	6
4	10

TRACE PROBLEM WITH ARRAYS

```
1. int[] values = new int[5];
2.   for(int i = 1; i < 5; i++)
3.     values[i] = i + values[i-1];
4. values[0] = values[1] + values[4];
```

Set values[0] to 11

Memory

values 0xA

i 5

0xA

Index	Value
0	11
1	1
2	3
3	6
4	10

EXERCISES AS A TABLE

- 1 – Printing array of integers
- 2 – Summing an array of integers
- 3 – Finding the largest element
- 4 – Finding the largest index of the smallest element
- 1,3 – Random shuffling
- 2,4 – Rotate the elements by 1 index

ENHANCED FOR LOOP (FOR-EACH LOOP)

- JDK 1.5 introduced a new for loop that enables you to traverse the complete array sequentially without using an index variable. For example, the following code displays all elements in the array myList:

```
for(double value : myList)
  System.out.println(value);
```

- In general, the syntax is

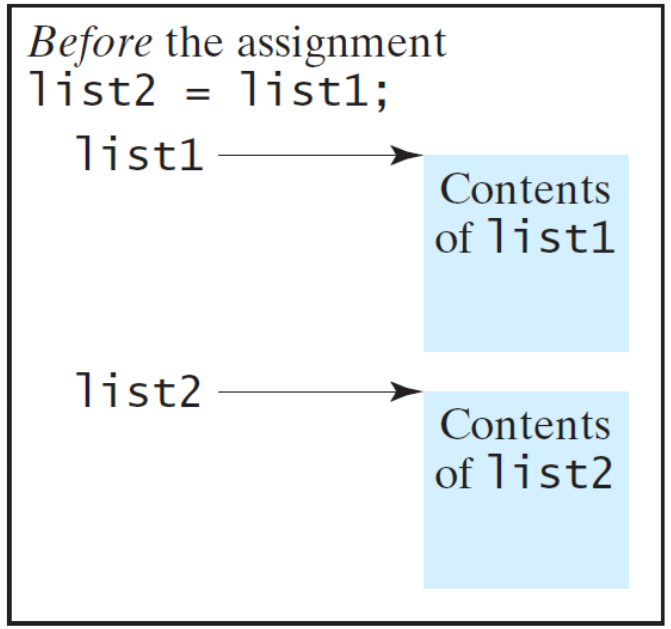
```
for(elementType value : arrayRefVar) {
  // Process the value
}
```

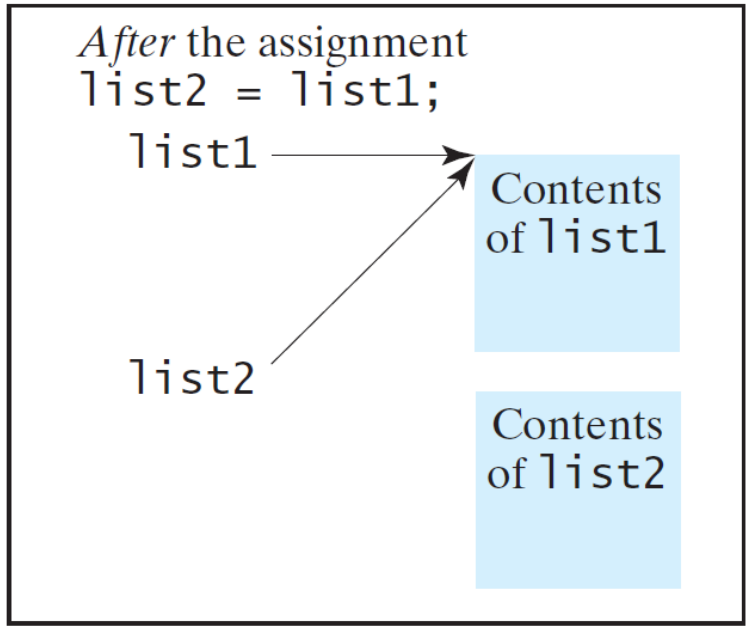
- You still have to use an index variable if you wish to traverse the array in a different order or change the elements in the array.

COPYING ARRAYS

- Often, in a program, you need to duplicate an array or a part of an array. In such cases you could attempt to use the assignment statement (=), as follows:

```
list2 = list1;
```





COPYING ARRAYS

- Using a loop:

```
1.int[] sourceArray = {2, 3, 1, 5, 10};
2.int[] targetArray = new int[sourceArray.length];
3.for(int i = 0; i < sourceArrays.length; i++)
4.    targetArray[i] = sourceArray[i];
```

PASSING ARRAYS TO METHODS

```
1.public static void printArray(int[] array) {
2.   for(int i = 0; i < array.length; i++)
3.      System.out.print(array[i] + " ");
4.}
```

Invoke the method

```
int[] list = {3, 1, 2, 6, 4, 2};
printArray(list);
```

Invoke the method with anonymous array

```
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

PASS BY VALUE

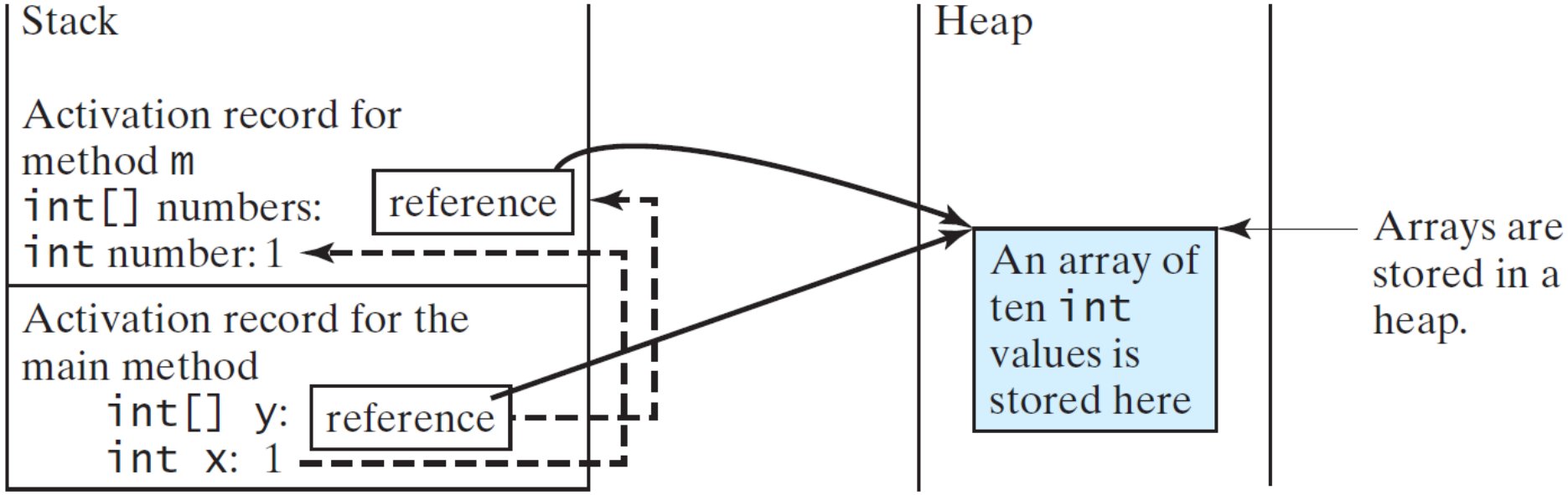
- Java uses **pass by value** to pass arguments to a method. There are important differences between passing a value of variables of primitive data types and passing arrays.
- For a parameter of a primitive type value, the actual value is passed. Changing the value of the local parameter inside the method does not affect the value of the variable outside the method.
- ***For a parameter of an array type, the value of the parameter contains a reference to an array; this reference is passed to the method. Any changes to the array that occur inside the method body will affect the original array that was passed as the argument.***

SIMPLE EXAMPLE

```
1. public class Test {
2.   public static void main(String[] args) {
3.     int x = 1;                  // x represents an int value
4.     int[] y = new int[10]; // y represents an array of int values
5.
6.     m(x, y);                    // Invoke m with arguments x and y
7.
8.     System.out.println("x is " + x);
9.     System.out.println("y[0] is " + y[0]);
10.  }
11.
12.  public static void m(int number, int[] numbers) {
13.    number = 1001;      // Assign a new value to local variable number
14.    numbers[0] = 5555; // Assign a new value to numbers[0]
15.  }
16.}
```

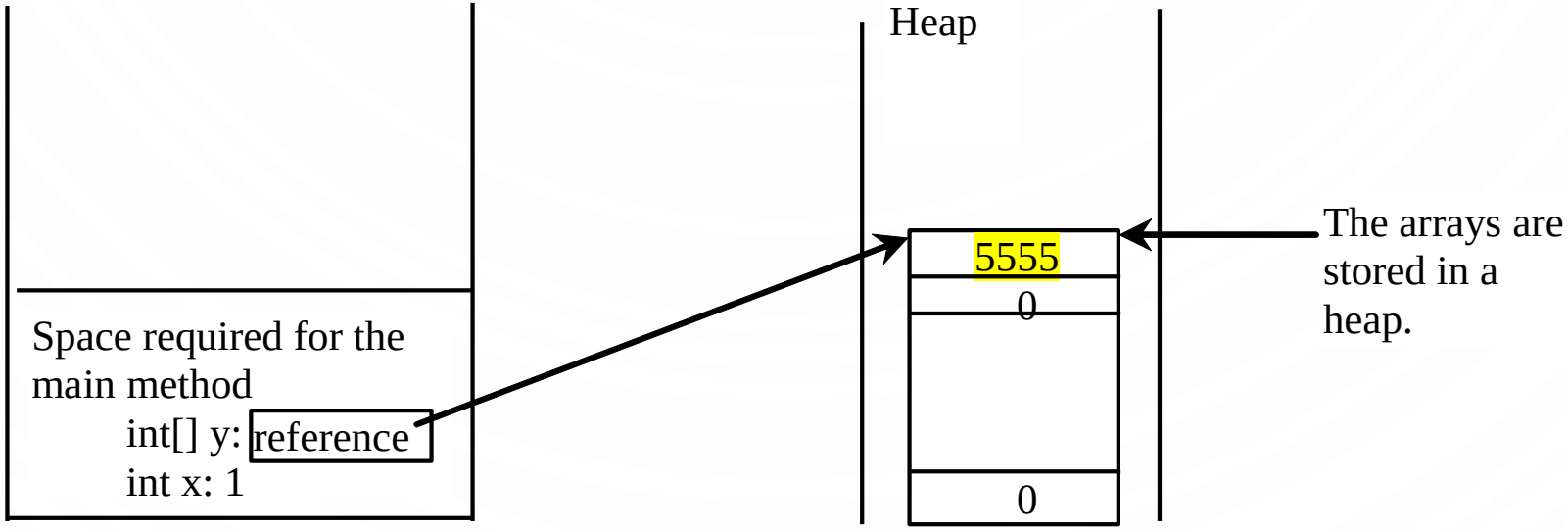
CALL STACK

- When invoking m(x, y), the values of x and y are passed to number and numbers. Since y contains the reference value to the array, numbers now contains the same reference value to the same array.



HEAP

- The JVM stores the array in an area of memory, called heap, which is used for dynamic memory allocation where blocks of memory are allocated and freed in an arbitrary order.



RETURNING AN ARRAY FROM A METHOD

```
1. int[] list1 =
2.     {1, 2, 3, 4, 5, 6};
3. int[] list2 = reverse(list1);
```

- Exercise: Trace the code. Show a detailed look at memory and how it changes.

```
1. public static int[] reverse(int[] list) {
2.   int[] result = new int[list.length];
3.
4.   for (int i = 0, j = result.length - 1;
5.        i < list.length; i++, j--) {
6.     result[j] = list[i];
7.   }
8.
9.   return result;
10.}
```

EXERCISE – EVERYONE CODE ALONG

EXERCISE

Banker of the Month!

- We work for JLDiablo's National Bank of Tristram.
- We manage savings, checking, and loan accounts for the citizens of Tristram, like good ole Wirt with his peg leg.
- We have all of the accounts stored in a file (accounts.txt). Each account has an id, a type, and an amount.
- We are gonna write a program to compute monthly changes based on the following:
  - Savings earn interest of 0.01%
  - Loans accrue interest of 0.4%
  - Checking accounts hava a monthly fee of 10 gold.
- After we are done, we are going to save to a new file

EXERCISE – START THE PROGRAM

```
1.    import java.util.Scanner;
2.    import java.io.File;
3.    import java.io.FileNotFoundException;
4.    import java.io.PrintWriter;
5.    public class Banking {
6.      public static void main(String[] args) throws FileNotFoundException {
7.        String inFileName = "accounts.txt";
8.        String outFileName = "accounts_new.txt";
9.        //Read each of the accounts into arrays
10.       //Process the account type
11.       //Output each of the accounts into the new file
12.     }
13.   }
```

EXERCISE – START FILLING OUT THE COMMENTS INTO CODE

```
1. //Read each account into an array
2. Scanner in = new Scanner(new File(inFileName));
3. int numAccounts = in.nextInt();
4. String[] accountTypes = new String[numAccounts];
5. double[] accountValues = new double[accountValues];
6. for(int i = 0; i < numAccounts; ++i) {
7.   accountTypes[i] = in.next();
8.   accountValues[i] = in.nextDouble();
9. }
10.in.close();
```

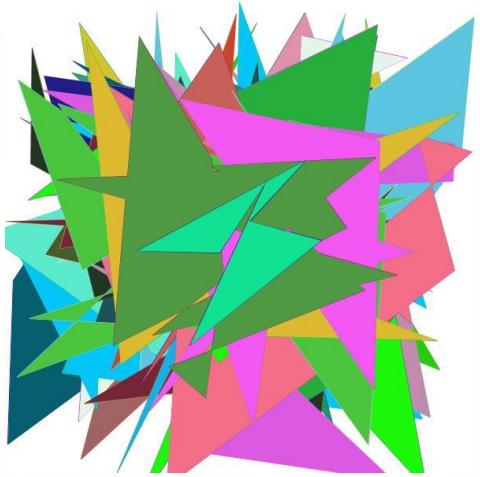
EXERCISE – COMPILE, TEST, AND CONTINUE

```
1. //Output each of the accounts into the new file
2. PrintWriter out = new PrintWriter(outFileName);
3. out.println(numAccounts);
4. for(int i = 0; i < numAccounts; ++i) {
5.   out.printf("%4d %8s %6.2f\n", i,
       accountTypes[i], accountValues[i]);
6. }
7. out.flush();
8. out.close();
```

EXERCISE – COMPILE, TEST, AND CONTINUE

```
1. //Process the account type
2. for(int i = 0; i < numAccounts; ++i) {
3.   if(accountTypes[i].equals("Savings"))
4.     accountValues[i] *= 1.0001; //0.01% interest
5.   else if(accountTypes[i].equals("Checking"))
6.     accountValues[i] -= 10;      //10 gold monthly fee
7.   else if(accountTypes[i].equals("Loan"))
8.     accountValues[i] *= 1.0004; //0.4% interest
9. }
```

EXERCISE IN PAIRS



1. Write a program that will generates a random polygon of $N = [3, 20]$ sides at random (x, y) points between $(-10, 10)$ – i.e., an array. Compute the center of mass $\left(\text{com}_\text{x} = \frac{\Sigma x_i}{N}, com_y = \frac{\Sigma y_i}{N}\right)$. Shift all points of the polygon by $(-com_x, -com_y)$. Write the polygon (array) to a file – first line is N, each line after is the points (output x y, not (x, y))
2. Write a program that reads your polygon file and shows it to the user using StdDraw. Use a random color to show the outline and a different random color to fill the polygon.
3. Make another program to allow a user to draw a polygon. Have them specify the number of points in the polygon. Let the user click in the window to specify the coordinates. Draw the polygon as they go. After they are done, write the polygon to a file and cleanly exit the program. Use #2 to view the polygon again.

MAIN METHOD IS JUST A REGULAR METHOD

- You can call a regular method by passing actual parameters. Can you pass arguments to main? Of course, yes. For example, the main method in class B is invoked by a method in A, as shown below:

```
public class A {
  public static void main(String[] args) {
    String[] strings = {"New York",
      "Boston", "Atlanta"};
    B.main(strings);
  }
}
```

```
class B {
  public static void main(String[] args) {
    for (int i = 0; i < args.length; i++)
      System.out.println(args[i]);
  }
}
```

COMMAND-LINE PARAMETERS

```
1. public class TestMain {
2.   public static void main(String[] args) {
3.     for(int i = 0; i < args.length; ++i)
4.       System.out.println(
5.         "Argument " + i + " is " + args[i]);
6.   }
7. }
```

- %java TestMain arg0 arg1 arg2 ... argn

EXERCISE WITH A PARTNER

- Write a program that will perform binary operations on integers. The program receives three parameters: an operator and two integers.
- Support erroneous input and the operators +, -, *, /, %

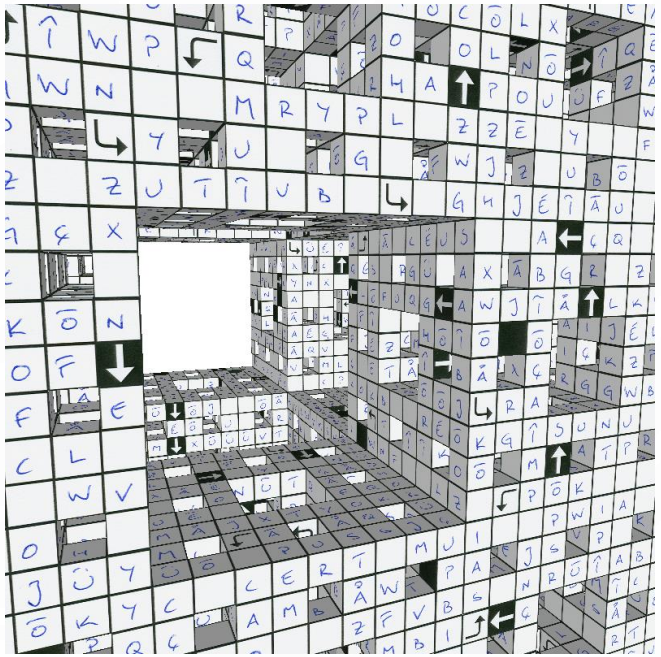
STRINGS REVISITED

- Strings are arrays of **char**! Well sort of…
- Strings 'underneath the hood' are arrays of **char**, but we use them differently
  - Java API for String
  - Use charAt(i) instead of [i]
  - Convert to **char**[] using toCharArray() and back to a String easily

```
String s = "Hello";
char[] c = s.toCharArray();
String s2 = new String(c);
```

  - Allows more, e.g., substring() which returns a portion of the String

MULTIDIMENSIONAL ARRAYS



MOTIVATIONS

- Thus far, you have used one-dimensional arrays to model linear collections of elements. You can use a two-dimensional array to represent a matrix or a table. For example, the following table that describes the distances between the cities can be represented using a two-dimensional array.

Distance Table (in miles)

	Chicago	Boston	New York	Atlanta	Miami	Dallas	Houston
Chicago	0	983	787	714	1375	967	1087
Boston	983	0	214	1102	1763	1723	1842
New York	787	214	0	888	1549	1548	1627
Atlanta	714	1102	888	0	661	781	810
Miami	1375	1763	1549	661	0	1426	1187
Dallas	967	1723	1548	781	1426	0	239
Houston	1087	1842	1627	810	1187	239	0

MOTIVATIONS

```
double[][] distances = {
    {0, 983, 787, 714, 1375, 967, 1087},
    {983, 0, 214, 1102, 1763, 1723, 1842},
    {787, 214, 0, 888, 1549, 1548, 1627},
    {714, 1102, 888, 0, 661, 781, 810},
    {1375, 1763, 1549, 661, 0, 1426, 1187},
    {967, 1723, 1548, 781, 1426, 0, 239},
    {1087, 1842, 1627, 810, 1187, 239, 0},
};
```

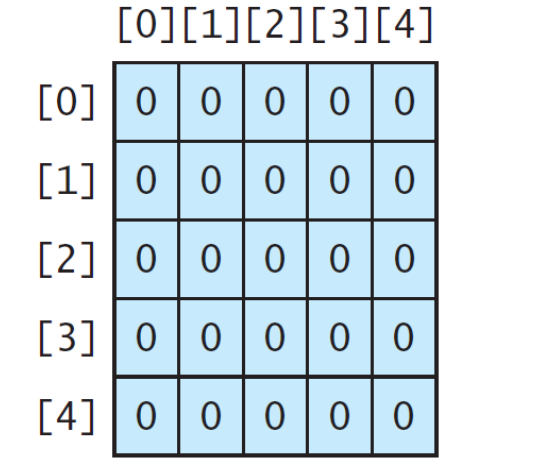
DECLARE/CREATE TWO-DIMENSIONAL ARRAYS

```
1.  // Declare array ref var
2.  dataType[][] refVar;
3.
4.  // Create array and assign its reference to variable
5.  refVar = new dataType[10][10];
6.
7.  // Combine declaration and creation in one statement
8.  dataType[][] refVar = new dataType[10][10];
9.
10. // Alternative syntax
11. dataType refVar[][] = new dataType[10][10];
```

DECLARE/CREATE TWO-DIMENSIONAL ARRAYS

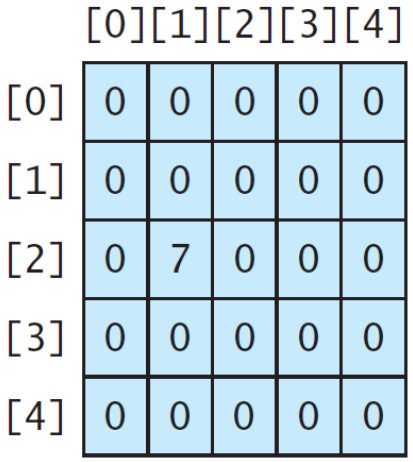
```
1.int[][] matrix = new int[10][10];
2.//or int matrix[][] = new int[10][10];
3.matrix[0][0] = 3;
4.
5.for(int i = 0; i < matrix.length; i++)
6.   for(int j = 0; j < matrix[i].length; j++)
7.     matrix[i][j] = (int)(Math.random() * 1000);
```

TWO-DIMENSIONAL ARRAY ILLUSTRATION



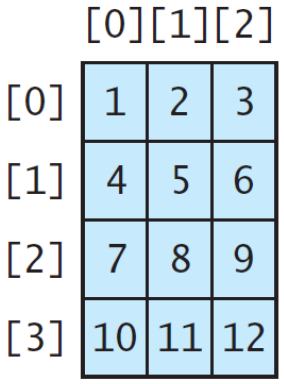

```
matrix = new int[5][5];
```

(a)




```
matrix[2][1] = 7;
```

(b)




```
int[][] array = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

(c)

matrix.length? 5

matrix[0].length? 5

array.length? 4

array[0].length? 3

DECLARING, CREATING, AND INITIALIZING USING SHORTHAND NOTATIONS

- You can also use an array initializer to declare, create and initialize a two-dimensional array. For example,

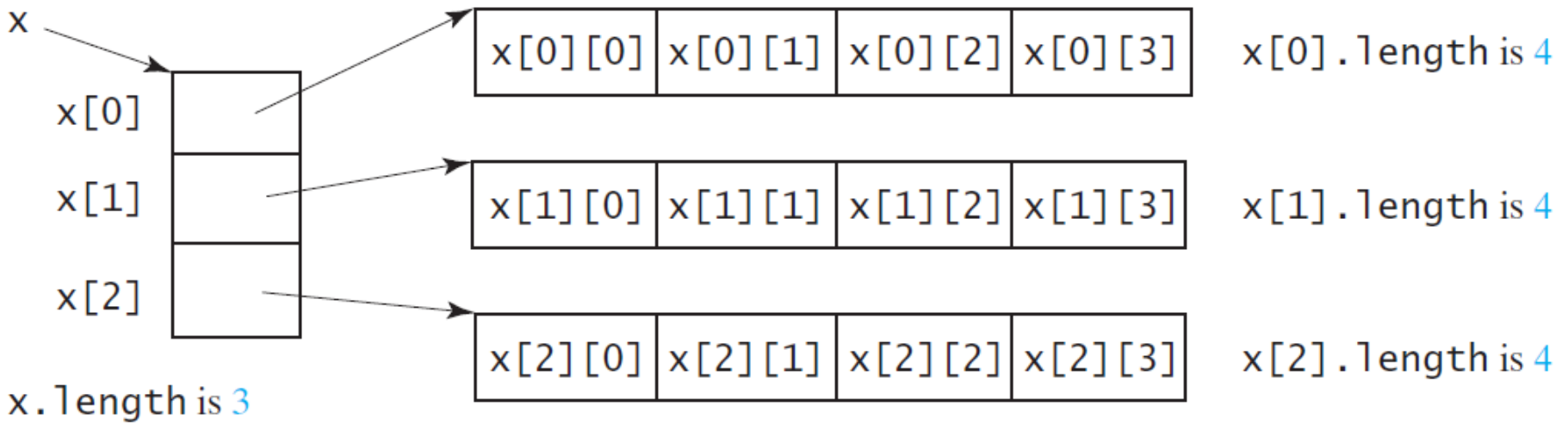
```
int[][] array = {
  { 1,  2,  3},
  { 4,  5,  6},
  { 7,  8,  9},
  {10, 11, 12}
};
```

Equivalent

```
int[][] array = new int[4][3];
array[0][0] =  1; array[0][1] =  2; array[0][2] =  3;
array[1][0] =  4; array[1][1] =  5; array[1][2] =  6;
array[2][0] =  7; array[2][1] =  8; array[2][2] =  9;
array[3][0] = 10; array[3][1] = 11; array[3][2] = 12;
```

LENGTHS OF TWO-DIMENSIONAL ARRAYS

- `int[][] x = new int[3][4];`



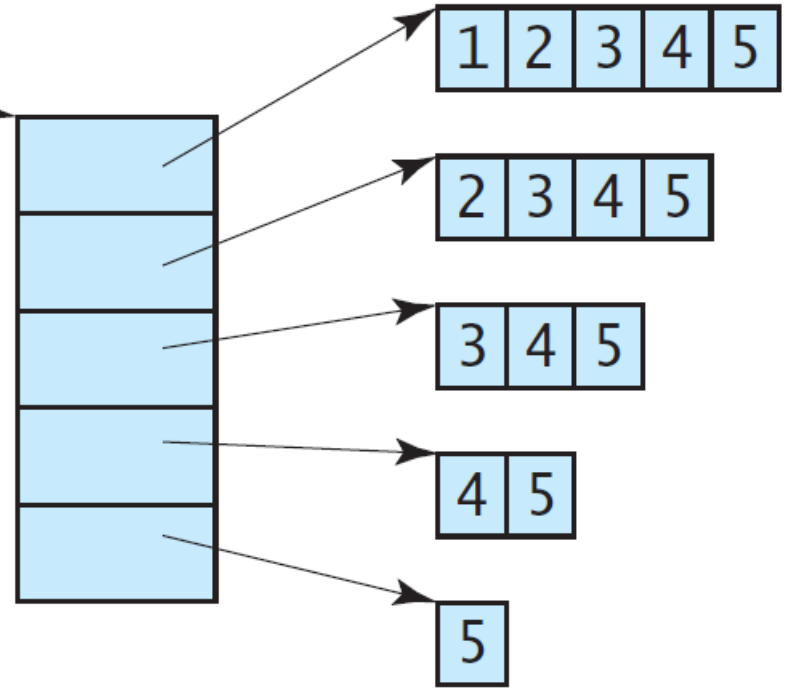
RAGGED ARRAYS

- Each row in a two-dimensional array is itself an array. So, the rows can have different lengths. Such an array is known as a **ragged array**. For example,

```
1. int[][] matrix = { //matrix.length is 5
2.    {1, 2, 3, 4, 5}, //matrix[0].length is 5
3.    {2, 3, 4, 5},    //matrix[1].length is 4
4.    {3, 4, 5},       //matrix[2].length is 3
5.    {4, 5},          //matrix[3].length is 2
6.    {5}              //matrix[4].length is 1
7. };
```

RAGGED ARRAYS

```
int[][] triangleArray = {
   {1, 2, 3, 4, 5},
   {2, 3, 4, 5},
   {3, 4, 5},
   {4, 5},
   {5}
};
```

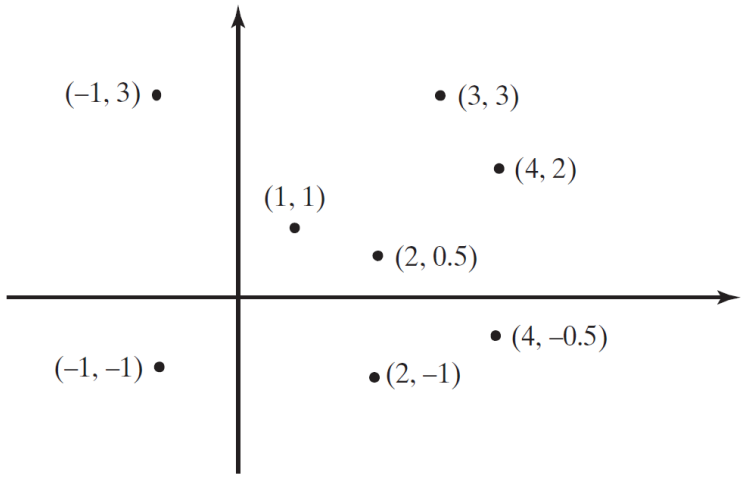


EXERCISE AS A TABLE

- 1 – Printing all elements into organized way
- 2 – Summing all elements
- 3 – Summing all elements by column into an array of sums
- 4 – Finding the maximum of each row into an array of maximums

EXERCISE WITH A PAIR

- Write an algorithm to find the two points nearest to each other
- Generalize to n-dimensional points



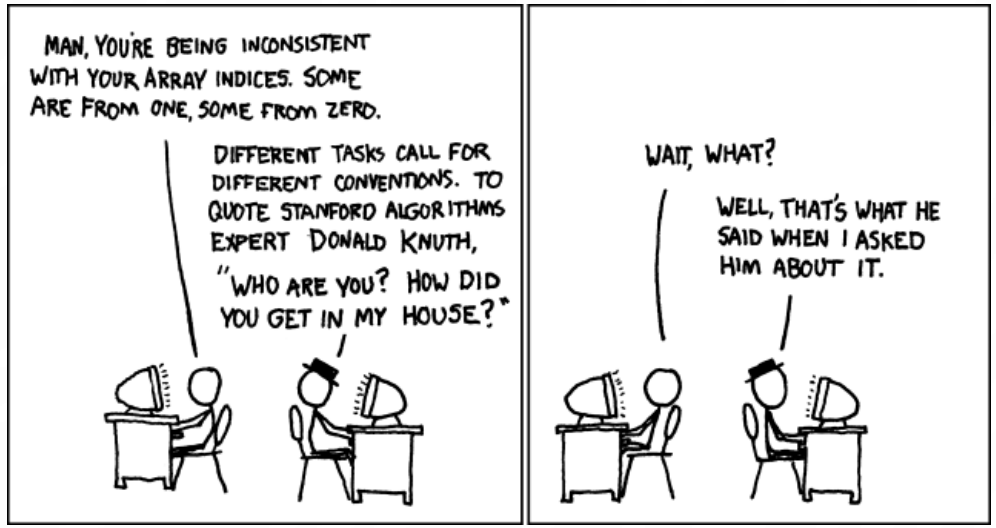
	x	y
0	–1	3
1	–1	–1
2	1	1
3	2	0.5
4	2	–1
5	3	3
6	4	2
7	4	–0.5

MULTIDIMENSIONAL ARRAYS

- Occasionally, you will need to represent n-dimensional data structures. In Java, you can create n-dimensional arrays for any integer n.
- The way to declare two-dimensional array variables and create two-dimensional arrays can be generalized to declare n-dimensional array variables and create n-dimensional arrays for n >= 3.

SUMMARY

- Arrays.
  - Organized way to store huge quantities of data.
  - Almost as easy to use as primitive types.
  - Can directly access an element given its index.



http://imgs.xkcd.com/comics/donald_knuth.png