CEN 419
Introduction to Java Programming

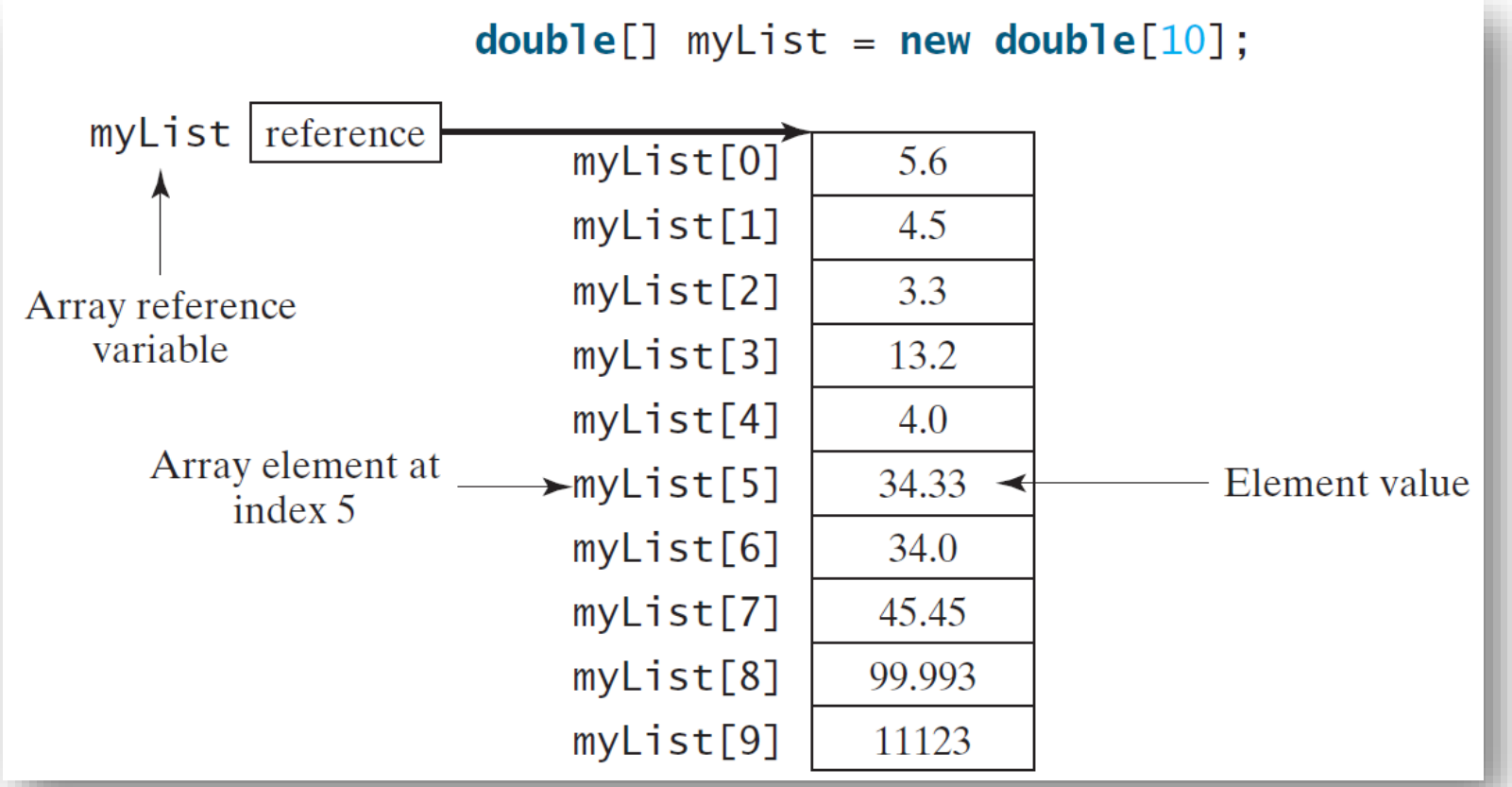
Dr. H. Esin ÜNAL

FALL 2021

Slides are modified from original slides of Y. Daniel Liang

Introducing Arrays

Array is a data structure that represents a collection of the same types of data.



Declaring Array Variables

- **`datatype[] arrayRefVar;`**

  Example:

  ```
  double[] myList;
  ```

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

  Example:

  ```
  double myList[];
  ```

Creating Arrays

- After an array variable is declared, you can **create an array** by using the **new** operator and assign its reference to the variable with the following syntax:

```
arrayRefVar = new datatype[arraySize];
```

- *Example:*

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

Default Values

- When an array is created, its elements are assigned the default value of:
  - ✓ **0** for the numeric primitive data types,
  - ✓ **'\u0000'** for char types, and
  - ✓ **false** for boolean types.

Indexed Variables

- The array elements are accessed through the index.
- The array indices are *0-based*:
  - ✓ **It starts from 0 to arrayRefVar.length-1.**
- Each element in the array is represented using the following syntax, known as an *indexed variable*:

```
arrayRefVar[index];
```

Example:

```
myList[0] //references the first element in the array.
myList[9] //references the last element in the array of size 10.
```

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];
```

Assigning Values

- To assign values to the elements, use the syntax:

```
arrayRefVar[index] = value;
```

Example:

```
myList[0] = 3.3
myList[9] = 99.998
```

Declaring and Creating 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,

```
double[] myList = new double[10];
myList.length //returns 10
```

Array Initializers

- Java has a shorthand notation, known as the *array initializer*, which combines the declaration, creation, and initialization of an array in one statement using the following syntax:

```
datatype[] arrayRefVar = {value0, value1, ..., valuek};
```

Declaring, creating, initializing Using the Shorthand Notation

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

- This shorthand notation is equivalent to the following statements:

```
double[] myList = new double[4];

myList[0] = 1.9;

myList[1] = 2.9;

myList[2] = 3.4;

myList[3] = 3.5;
```

CAUTION

- When you are using the shorthand notation, you have to declare, create, and initialize the array all in one statement.
- Splitting it would cause a syntax error.
- For example, the following is wrong:

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

Trace Program with Arrays

Declare array variable values, create an array, and assign its reference to values

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the array is created

0	0
1	0
2	0
3	0
4	0

Trace Program with Arrays

i becomes 1
i (=1) is less than 5

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the array is created

0	0
1	0
2	0
3	0
4	0

Trace Program with Arrays

After this line is executed, value[1] is 1

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the first iteration

0	0
1	1
2	0
3	0
4	0

Trace Program with Arrays

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After i++, i becomes 2

After the first iteration

0	0
1	1
2	0
3	0
4	0

Trace Program with Arrays

i (= 2) is less than 5

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the first iteration

0	0
1	1
2	0
3	0
4	0

Trace Program with Arrays

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After this line is executed, values[2] is 3 (2 + 1)

After the second iteration

0	0
1	1
2	3
3	0
4	0

Trace Program with Arrays

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After i++, i becomes 3

After the second iteration

0	0
1	1
2	3
3	0
4	0

Trace Program with Arrays

i (= 3) is less than 5

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the second iteration

0	0
1	1
2	3
3	0
4	0

Trace Program with Arrays

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After this line is executed, values[3] is 6 (3 + 3)

After the third iteration

0	0
1	1
2	3
3	6
4	0

Trace Program with Arrays

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After i++, i becomes 4

After the third iteration

0	0
1	1
2	3
3	6
4	0

Trace Program with Arrays

i (= 4) is less than 5

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the third iteration

0	0
1	1
2	3
3	6
4	0

Trace Program with Arrays

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After this line is executed, values[4] is 10 (4 + 6)

After the forth iteration

0	0
1	1
2	3
3	6
4	10

Trace Program with Arrays

After i++, i becomes 5

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the forth iteration

0	0
1	1
2	3
3	6
4	10

Trace Program with Arrays

i (=5) < 5 is false. Exit the loop

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After the forth iteration

0	0
1	1
2	3
3	6
4	10

Trace Program with Arrays

```
public class Test {
  public static void main(String[] args) {
    int[] values = new int[5];
    for (int i = 1; i < 5; i++) {
      values[i] = i + values[i-1];
    }
    values[0] = values[1] + values[4];
  }
}
```

After this line,
values[0] is 11 (1 + 10)

0	11
1	1
2	3
3	6
4	10

Processing Arrays

- ✓ Initializing arrays with input values
- ✓ Initializing arrays with random values
- ✓ Printing arrays
- ✓ Summing all elements
- ✓ Finding the largest element
- ✓ Random shuffling
- ✓ Shifting elements

Initializing arrays with input values

```
import java.util.Scanner;
public class Test {
  public static void main(String[] args) {
   Scanner input = new Scanner(System.in);
   double[] myList = new double[4];
   System.out.print("Enter " + myList.length + « values: ");
   for (int i = 0; i < myList.length; i++)
      myList[i] = input.nextDouble();
  }
}
```

Initializing arrays with random values

```
for (int i = 0; i < myList.length; i++) {
  myList[i] = Math.random() * 100;
}
```

Printing arrays

```
for (int i = 0; i < myList.length; i++) {
  System.out.print(myList[i] + " ");
}
```

Tip: For an array of the **char[]** type, it can be printed using one print statement.
For example, the following code displays **Dallas**:

```
char[] city = {'D', 'a', 'l', 'l', 'a', 's'};
System.out.println(city);
```

Summing all elements

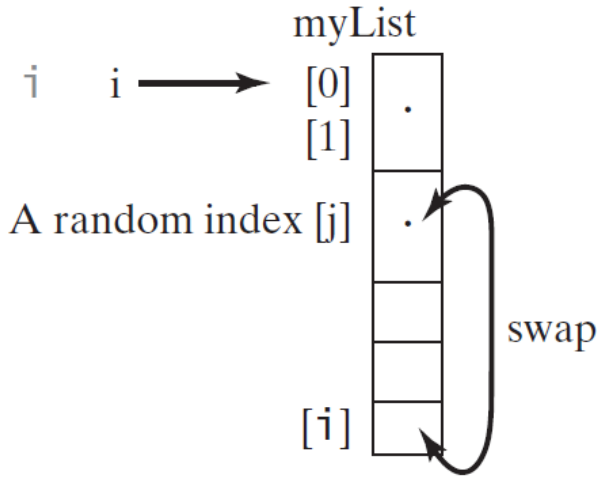
```
double total = 0;
for (int i = 0; i < myList.length; i++) {
  total += myList[i];
}
```

Finding the largest element

```
double max = myList[0];
for (int i = 1; i < myList.length; i++) {
  if (myList[i] > max)
      max = myList[i];
}
```

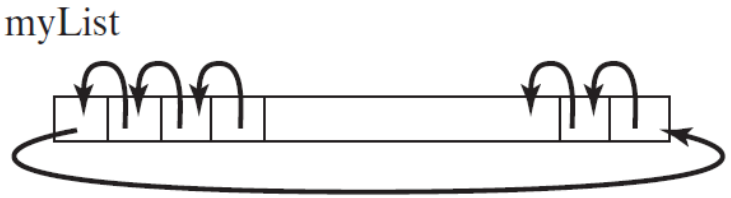
Random shuffling

```
for (int i = myList.length - 1; i > 0; i--) {
  // Generate an index j randomly with 0 <= j <= i
  int j = (int)(Math.random()
    * (i + 1));

  // Swap myList[i] with myList[j]
  double temp = myList[i];
  myList[i] = myList[j];
  myList[j] = temp;
}
```



Shifting Elements

```java
double temp = myList[0]; // Retain the first element

// Shift elements left
for (int i = 1; i < myList.length; i++) {
  myList[i - 1] = myList[i];
}

// Move the first element to fill in the last position
myList[myList.length - 1] = temp;
```



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.

Opening Problem

Read one hundred numbers, compute their average, and find out how many numbers are above the average.

Analyze Numbers

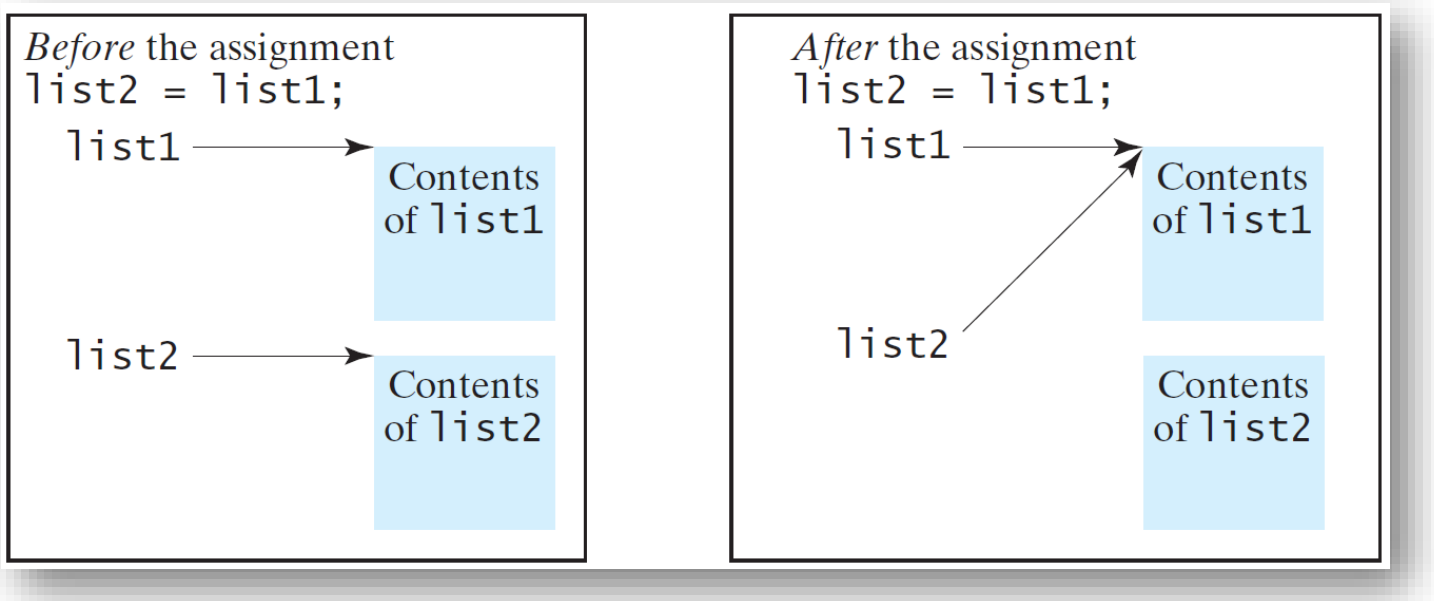
Intro to Java Programming, Y. Daniel Liang - AnalyzeNumbers.java (pearsoncmg.com)

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;

However, this statement does not copy the contents of the array referenced by list1 to list2.



Copying Arrays

You can ***use a loop to copy*** individual elements one by one:

```
int[] sourceArray = {2, 3, 1, 5, 10};
int[] targetArray = new int[sourceArray.length];

for (int i = 0; i < sourceArrays.length; i++)
   targetArray[i] = sourceArray[i];
```

Copying Arrays

You can ***use the static arraycopy method*** in the System class:

```
arraycopy(sourceArray, srcPos, targetArray, tarPos, length);
```

Example:

```
System.arraycopy(sourceArray, 0, targetArray, 0, sourceArray.length);
```

Passing Arrays to Methods

When passing an array to a method, the reference of the array is passed to the method.

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

Invoke the method:

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

Invoke the method:

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

Anonymous array

Anonymous Array

The statement

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

creates an array using the following syntax:

```
new dataType[]{literal0, literal1, ..., literalk};
```

There is no explicit reference variable for the array.

Such array is called an *anonymous array*.

Pass By Value

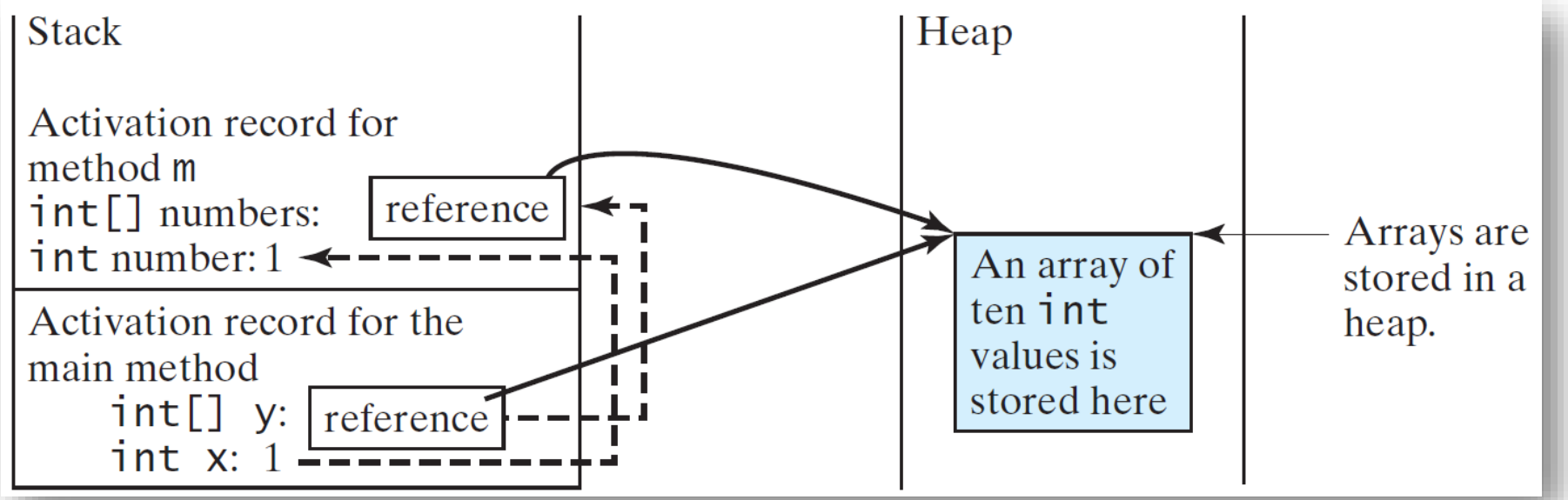
- Java uses ***pass by value*** to pass arguments to a method. There are important differences between passing a value of variables of primitive data types and passing arrays.
  - *For a parameter of a primitive type value*, **the actual value is passed**. Changing the value of the local parameter inside 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

```
public class Test {
  public static void main(String[] args) {
    int x = 1; // x represents an int value
    int[] y = new int[10]; // y represents an array of int values

    m(x, y); // Invoke m with arguments x and y

    System.out.println("x is " + x);
    System.out.println("y[0] is " + y[0]);
  }

  public static void m(int number, int[] numbers) {
    number = 1001; // Assign a new value to number
    numbers[0] = 5555; // Assign a new value to numbers[0]
  }
}
```

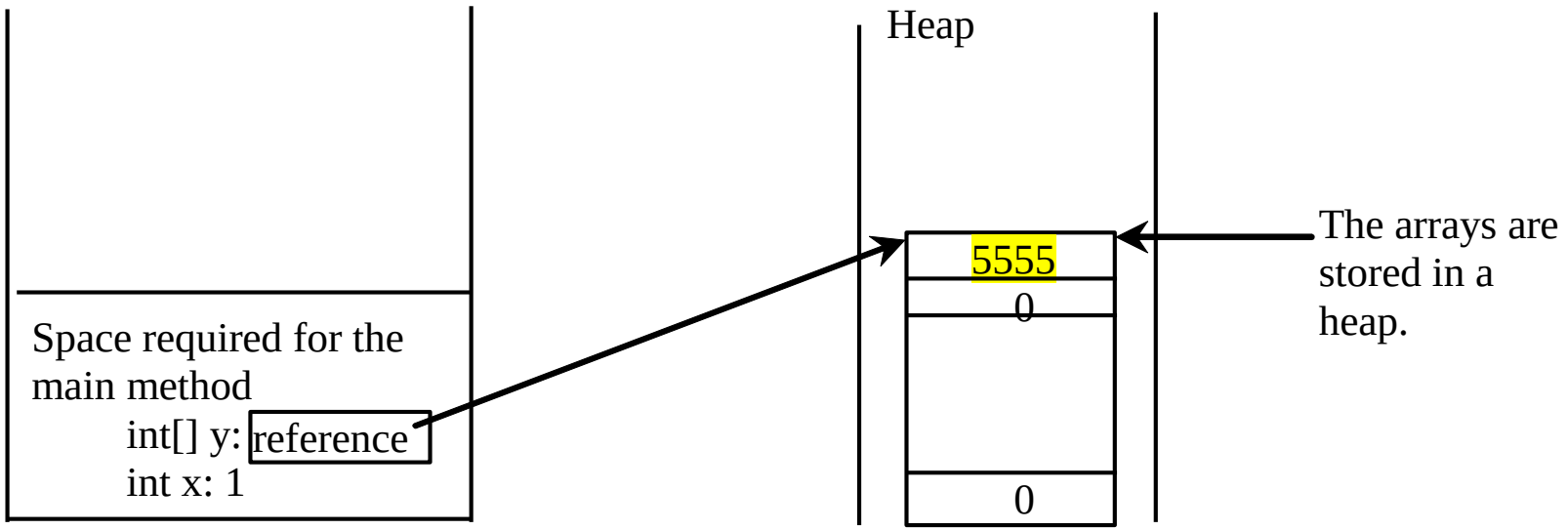
Call Stack

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 <u>dynamic memory allocation</u> where blocks of memory are allocated and freed in an arbitrary order.



Passing Arrays as Arguments

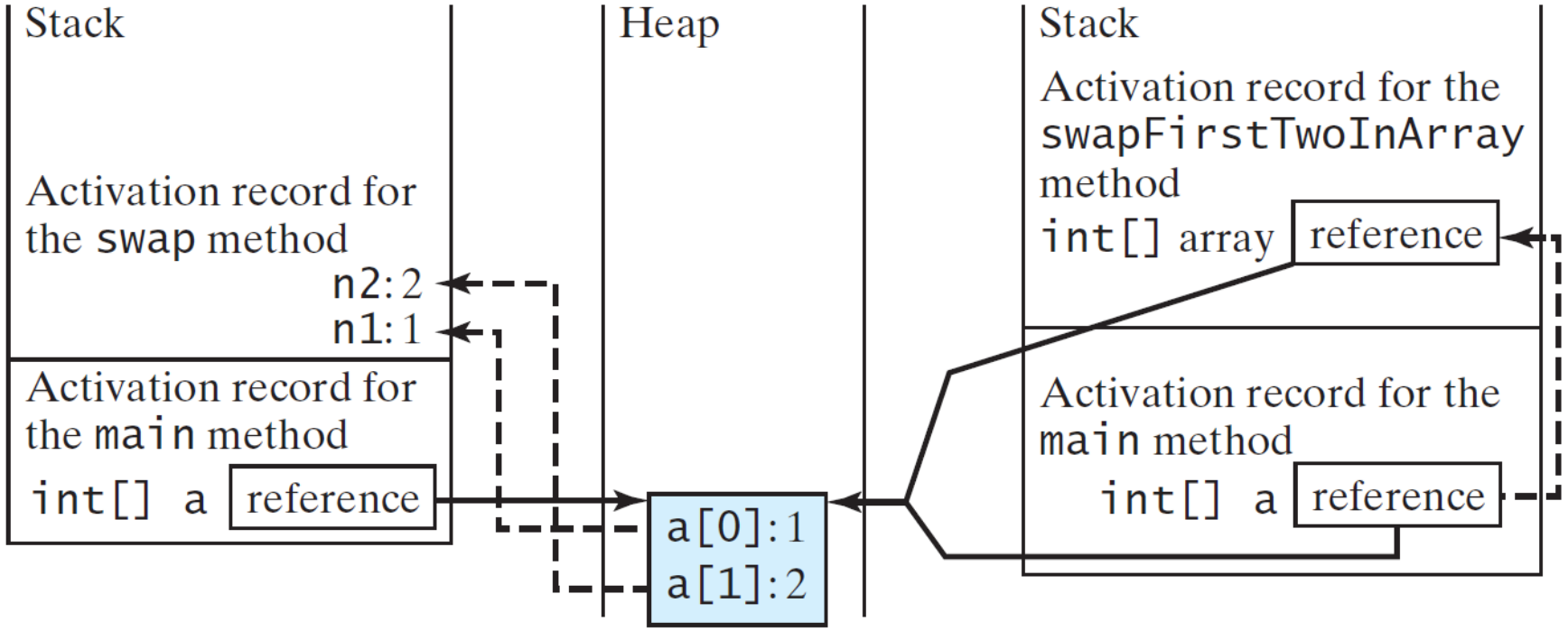
- Objective: Demonstrate differences of passing primitive data type variables and array variables.

TestPassArray

Intro to Java Programming, Y. Daniel Liang - TestPassArray.java (pearsoncmg.com)

Call Stack

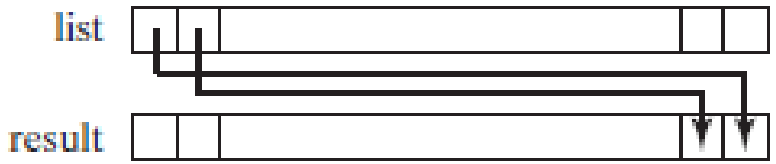
When invoking m(x, y), the values of x and y are passed to number and numbers. Since y contains the reference value to the array, numbers now contains the same reference value to the same array.



Returning an Array from a Method

When a method returns an array, the reference of the array is returned.

```
public static int[] reverse(int[] list) {
 int[] result = new int[list.length];

 for (int i = 0, j = result.length - 1; i < list.length;
      i++, j--) {
    result[j] = list[i];
  }

  return result;
}
```



While invoking this method:

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

Problem: Counting Occurrence of Each Letter

- Generate 100 lowercase letters randomly and assign to an array of characters.
- Count the occurrence of each letter in the array.

Count Letters

Intro to Java Programming, Y. Daniel Liang - CountLettersInArray.java (pearsoncmg.com)

Variable-Length Argument Lists

- A variable number of arguments of the same type can be passed to a method and treated as an array.
- The parameter in the method is declared as follows:

```
typeName... parameterName
```

- In the method declaration, you specify the type followed by an ellipsis (...).
- Only one variable-length parameter may be specified in a method, and this parameter must be the last parameter.

Variable-Length Argument Lists

- What is wrong in the following method headers?

```
public static void print(String... strings, double... numbers)
public static void print(double... numbers, String name)
public static double... print(double d1, double d2)
```

- A valid method header:

```
public static void print (double... numbers)
```

Variable-Length Argument Lists

```
public class VarArgsDemo {
public static void main(String[] args) {
printMax(34, 3, 3, 2, 56.5);
printMax(new double[]{1, 2, 3});
}

public static void printMax(double... numbers) {
if (numbers.length == 0) {
System.out.println("No argument passed");
return;
}

double result = numbers[0];

for (int i = 1; i < numbers.length; i++)
if (numbers[i] > result)
result = numbers[i];

System.out.println("The max value is " + result);
}
}
```

If you invoke the printMax as:

`printMax(new int[]{1, 2, 3});`

Error occurs since int cannot be converted into double implicitly.

Searching Arrays

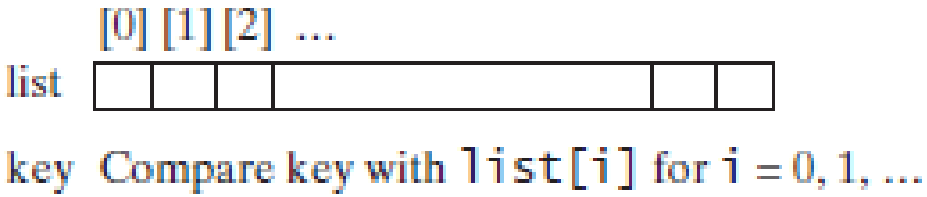
- Searching is the process of looking for a specific element in an array.
  - For example, discovering whether a certain score is included in a list of scores.
- There are many algorithms and data structures devoted to searching.
- Two commonly used approaches are, *linear search* and *binary search*.

Linear Search

- The linear search approach compares the key element, **key**, *sequentially* with each element in the array **list**.
- The method continues to do so until the key matches an element in the list or the list is exhausted without a match being found.
- If a match is made, the linear search returns the **index of the element** in the array that matches the key. If no match is found, the search returns **-1**.

Linear Search

```
public class LinearSearch {
  /** The method for finding a key in the list */
  public static int linearSearch(int[] list, int key){
    for (int i = 0; i < list.length; i++)
      if (key == list[i])
        return i;
    return -1;
  }
}
```



Linear Search Animation

Intro to Java Programming, Y. Daniel Liang - LinearSearch.java (pearsoncmg.com)

Binary Search

- For binary search to work, **the elements in the array must already be ordered**.
- Without loss of generality, assume that the array is in ascending order.

  e.g., 2 4 7 10 11 45 50 59 60 66 69 70 79
- The binary search first compares the key with the element in the middle of the array.

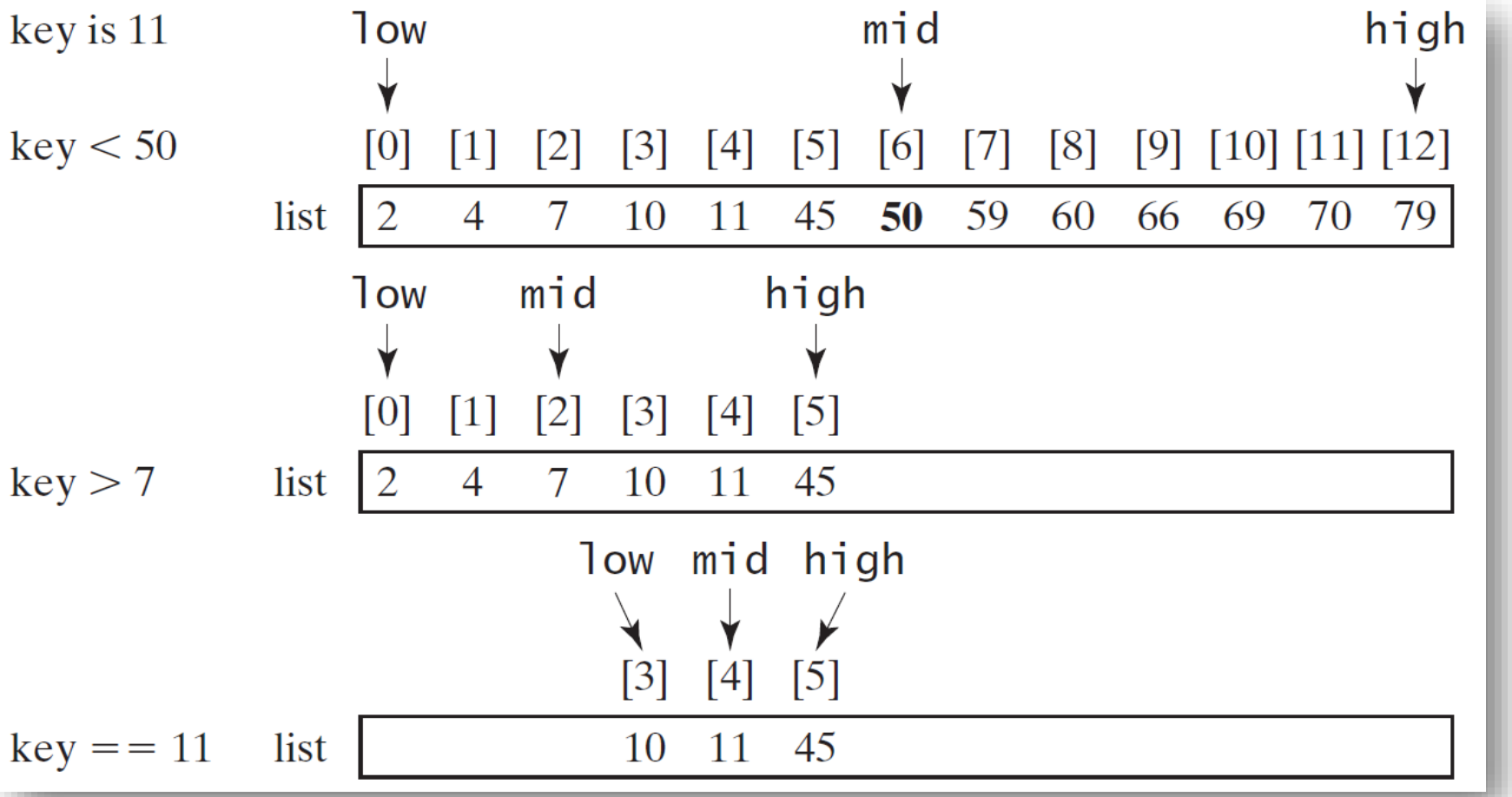
Binary Search

- Consider the following three cases:
  - If the key is less than the middle element, you only need to search the key in the first half of the array.
  - If the key is equal to the middle element, the search ends with a match.
  - If the key is greater than the middle element, you only need to search the key in the second half of the array.

Binary Search Animation

Intro to Java Programming, Y. Daniel Liang - BinarySearch.java (pearsoncmg.com)

Binary Search



Binary Search

The binarySearch method returns the index of the element in the list that matches the search key if it is contained in the list. Otherwise, it returns

-insertion point - 1

The insertion point is the point at which the key would be inserted into the list.

Binary Search

key is 54

key > 50

	low						mid						high
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
list	2	4	7	10	11	45	**50**	59	60	66	69	70	79

key < 66

							low		mid			high
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
list								59	60	66	69	70	79

key < 59

	low	mid	high
	[7]	[8]	
list	59	60	

	low	high
	[6]	[7]	[8]
	59	60	

Binary Search

```
/** Use binary search to find the key in the list */
public static int binarySearch(int[] list, int key) {
  int low = 0;
  int high = list.length - 1;

  while (high >= low) {
    int mid = (low + high) / 2;
    if (key < list[mid])
      high = mid - 1;
    else if (key == list[mid])
      return mid;
    else
      low = mid + 1;
  }
  return -1 - low;
}
```

Note - I

- In the worst case when using the binary search approach, you need **$\log_2 n+1$** comparisons to find an element in the sorted array.
- In the worst case when using the linear search approach, you need **n-1** comparisons to find an elemet in the array

Note - II

- Linear search is useful for finding an element in a small array or an unsorted array, but it is inefficient for large arrays.
- Binary search is more efficient, but it requires that the array be presorted.

The Arrays.binarySearch Method

- Since binary search is frequently used in programming, Java provides several overloaded binarySearch methods for searching a key in an array of int, double, char, short, long, and float in the java.util.Arrays class.
- For example, the following code searches the keys in an array of numbers and an array of characters.

```
int[] list = {2, 4, 7, 10, 11, 45, 50, 59, 60, 66, 69, 70, 79};
System.out.println("Index is " +  java.util.Arrays.binarySearch(list, 11));

char[] chars = {'a', 'c', 'g', 'x', 'y', 'z'};
System.out.println("Index is " +  java.util.Arrays.binarySearch(chars, 't'));
```

Return is 4

Return is –4 (insertion point is 3, so return is -3-1)

- For the binarySearch method to work, the array must be pre-sorted in increasing order.

The Arrays.sort Method

- Since sorting is frequently used in programming, Java provides several overloaded sort methods for sorting an array of int, double, char, short, long, and float in the java.util.Arrays class.
- For example, the following code sorts an array of numbers and an array of characters.

```
double[] numbers = {6.0, 4.4, 1.9, 2.9, 3.4, 3.5};
java.util.Arrays.sort(numbers);
```

Sort the whole array

```
char[] chars = {'a', 'A', '4', 'F', 'D', 'P'};
java.util.Arrays.sort(chars,1,3);
```

Sort the part of the array

- Java 8 now provides Arrays.parallelSort(list) that utilizes the multicore for fast sorting.

The Arrays.toString Method

- The Arrays.toString method can be used to return a string representation for the list.
- For example, the following code:

```
int[] list = {2, 4, 7, 10};
System.out.println(Arrays.toString(list));
```

displays [2, 4, 7, 10].

The Arrays.equals Method

- The Arrays.equals method can be used to check whether two arrays are strictly equal.
- For example:

```
int[] list1 = {2, 4, 7, 10};
int[] list2 = {2, 4, 7, 10};
int[] list3 = {4, 2, 7, 10};
System.out.println(java.util.Arrays.equals(list1, list2));
// true
System.out.println(java.util.Arrays.equals(list2, list3));
// false
```

The Arrays.fill Method

- The Arrays.fill method can be used to fill in all or part of the array.
- For example:

```
int[] list1 = {2, 4, 7, 10};
int[] list2 = {2, 4, 7, 7, 7, 10};
java.util.Arrays.fill(list1, 5); // Fill 5 to the whole array
java.util.Arrays.fill(list2, 1, 5, 8); // Fill 8 to a partial
                                       // array
```

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 TestMain is invoked by a method in A, as shown below:

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

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

Passing Strings to the **main** method

You can pass strings to a **main** method from the command line when you run the program.

```
class TestMain {
  public static void main(String[] args) {
  ...
  }
}

java TestMain arg0 arg1 arg2 ... argn
```

Problem: Calculator

- Objective: Write a program that will perform binary operations on integers. The program receives three parameters: an operator and two integers.

```
java Calculator 2 + 3
java Calculator 2 - 3
java Calculator 2 / 3
java Calculator 2 . 3
```

Calculator

Intro to Java Programming, Y. Daniel Liang - Calculator.java (pearsoncmg.com)

Two-Dimensional Arrays

- Data in a table or a matrix can be represented using a two-dimensional array.
- An element in a two-dimensional array is accessed through a row and column index.

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

```
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

- The syntax for declaring a two-dimensional array is:

```
dataType[][] refVar;
```

Or

```
datatype arrayRefVar[][];//Allowed,but not preferred
```

- The syntax for creating a two-dimensional array and assigning its reference to a variable is:

```
refVar = new dataType[10][10];
```

- Combine declaration and creation in one statement:

```
dataType[][] refVar = new dataType[10][10];
```

Declare/Create Two-Dimensional Arrays

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

- Two subscripts are used in a two-dimensional array, one for the row and the other for the column. As in a one-dimensional array, the index for each subscript is of the **int** type and starts from **0**.

	[0]	[1]	[2]	[3]	[4]
[0]	0	0	0	0	0
[1]	0	0	0	0	0
[2]	0	0	0	0	0
[3]	0	0	0	0	0
[4]	0	0	0	0	0

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

	[0]	[1]	[2]	[3]	[4]
[0]	0	0	0	0	0
[1]	0	0	0	0	0
[2]	0	7	0	0	0
[3]	0	0	0	0	0
[4]	0	0	0	0	0

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

	[0]	[1]	[2]
[0]	1	2	3
[1]	4	5	6
[2]	7	8	9
[3]	10	11	12

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

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,

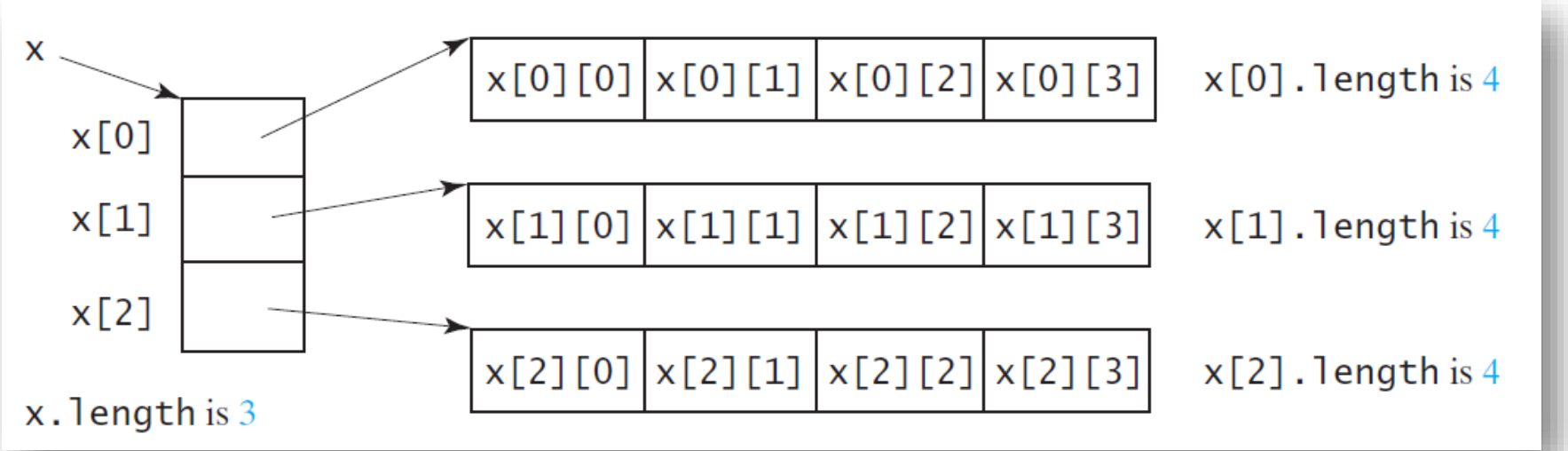
```
int[][] array = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

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];
```



`x[3].length` is ArrayIndexOutOfBoundsException

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,

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

matrix.length is 5

matrix[0].length is 5

matrix[1].length is 4

matrix[2].length is 3

matrix[3].length is 2

matrix[4].length is 1

Processing Two-Dimensional Arrays

- ✓ Initializing arrays with input values
- ✓ Initializing arrays with random values
- ✓ Printing arrays
- ✓ Summing all elements
- ✓ Summing elements by column
- ✓ Which row has the largest sum
- ✓ Random shuffling

Initializing arrays with input values

```
import java.util.Scanner;
public class Test {
  public static void main(String[] args) {
   Scanner input = new Scanner(System.in);
   System.out.println("Enter " + matrix.length + " rows and " +
matrix[0].length + " columns: ");
   for (int row = 0; row < matrix.length; row++) {
     for (int column = 0; column < matrix[row].length; column++) {
    matrix[row][column] = input.nextInt();
     }
   }
  }
}
```

Initializing arrays with random values

```
for (int row = 0; row < matrix.length; row++) {
 for (int column = 0; column < matrix[row].length; column++)
{
  matrix[row][column] = (int)(Math.random() * 100);
 }
}
```

Printing arrays

```
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column < matrix[row].length; column++)
{
  System.out.print(matrix[row][column] + " ");
  }
  System.out.println();
}
```

Summing all elements

```
int total = 0;
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column < matrix[row].length; column++)
{
  total += matrix[row][column];
  }
}
```

Summing elements by column

```
for (int column = 0; column < matrix[0].length; column++) {
  int total = 0;
  for (int row = 0; row < matrix.length; row++)
  total += matrix[row][column];
  System.out.println("Sum for column "+column+" is "+total);
}
```

Which row has the largest sum?

```
int maxRow = 0;
int indexOfMaxRow = 0;
// Get sum of the first row in maxRow
for (int column = 0; column < matrix[0].length; column++) {
  maxRow += matrix[0][column];
}
for (int row = 1; row < matrix.length; row++) {
  int totalOfThisRow = 0;
  for (int column = 0; column < matrix[row].length; column++)
   totalOfThisRow += matrix[row][column];
  if (totalOfThisRow > maxRow) {
   maxRow = totalOfThisRow;
   indexOfMaxRow = row;
  }
}
System.out.println("Row " + indexOfMaxRow+ " has the maximum sum of " +
maxRow);
```

Random shuffling

```
for (int i = 0; i < matrix.length; i++) {
  for (int j = 0; j < matrix[i].length; j++) {
     int i1 = (int)(Math.random() * matrix.length);
     int j1 = (int)(Math.random() * matrix[i].length);
     // Swap matrix[i][j] with matrix[i1][j1]
     int temp = matrix[i][j];
     matrix[i][j] = matrix[i1][j1];
     matrix[i1][j1] = temp;
  }
}
```

Passing Two-Dimensional Arrays to Methods

- When passing a two-dimensional array to a method, the reference of the array is passed to the method.

Pass Two-Dimensional Arrays

Intro to Java Programming, Y. Daniel Liang - PassTwoDimensionalArray.java (pearsoncmg.com)

Problem: Finding Two Points Nearest to Each Other

- Given a set of points, the closest-pair problem is to find the two points that are nearest to each other.

	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

Find Nearest Points

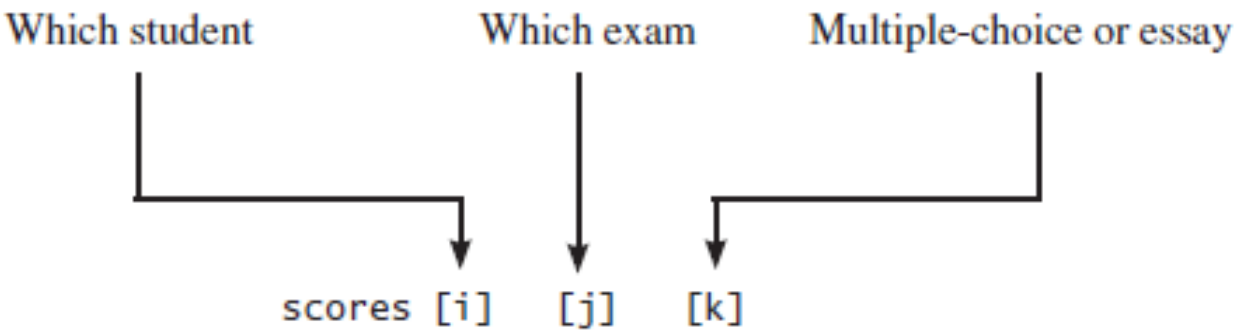
Intro to Java Programming, Y. Daniel Liang - FindNearestPoints.java (pearsoncmg.com)

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.
- A two-dimensional array consists of an array of one-dimensional arrays and a three-dimensional array consists of an array of two-dimensional arrays.

Multidimensional Arrays

```
double[][][] scores = new double[6][5][2];

double[][][] scores = {
{{7.5, 20.5}, {9.0, 22.5}, {15, 33.5}, {13, 21.5}, {15, 2.5}},
{{4.5, 21.5}, {9.0, 22.5}, {15, 34.5}, {12, 20.5}, {14, 9.5}},
{{6.5, 30.5}, {9.4, 10.5}, {11, 33.5}, {11, 23.5}, {10, 2.5}},
{{6.5, 23.5}, {9.4, 32.5}, {13, 34.5}, {11, 20.5}, {16, 7.5}},
{{8.5, 26.5}, {9.4, 52.5}, {13, 36.5}, {13, 24.5}, {16, 2.5}},
{{9.5, 20.5}, {9.4, 42.5}, {13, 31.5}, {12, 20.5}, {16, 6.5}}};
```



Problem: Guessing Birthday

- Remember our previous program that guesses a birthday. The program can be simplified by storing the numbers in five sets in a three-dimensional array, and it prompts the user for the answers using a loop.

Guess Birthday

Intro to Java Programming, Y. Daniel Liang - GuessBirthdayUsingArray.java (pearsoncmg.com)