CS 213 – Programming Language 2

Dr. Ahmed Hesham Mostafa

Lecture 2 – Java Basics

Keywords

Single Array

Multiple Dimensions array

Methods

Defining Methods

A method is a collection of statements that are grouped together to perform an operation.

Define a method

```
public static int max(int num1, int num2) {

   int result;

   if (num1 > num2)
     result = num1;
   else
     result = num2;

   return result;
}
```

Invoke a method

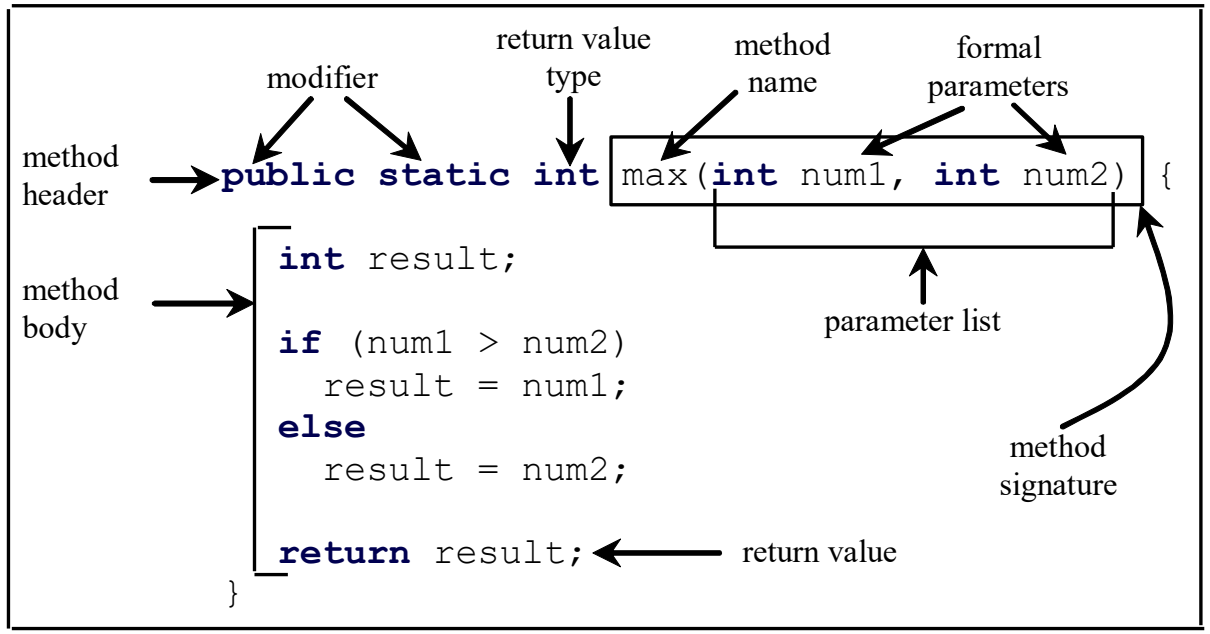
```
int z = max(x, y);
```

actual parameters
(arguments)

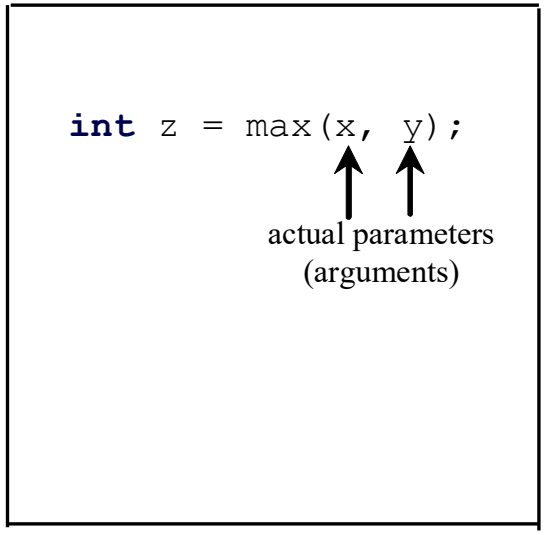
Defining Methods

A method is a collection of statements that are grouped together to perform an operation.

Define a method



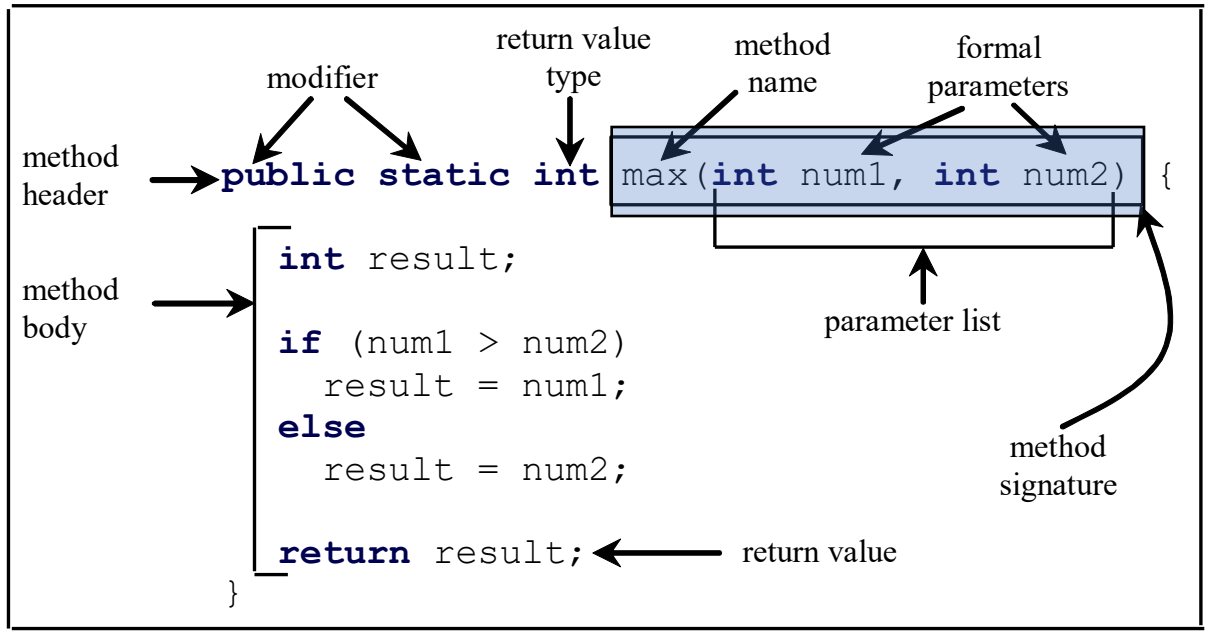
Invoke a method



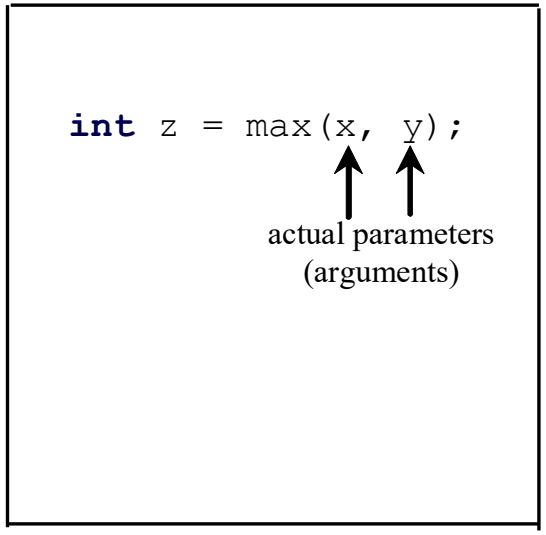
Method Signature

Method signature is the combination of the method name and the parameter list.

Define a method

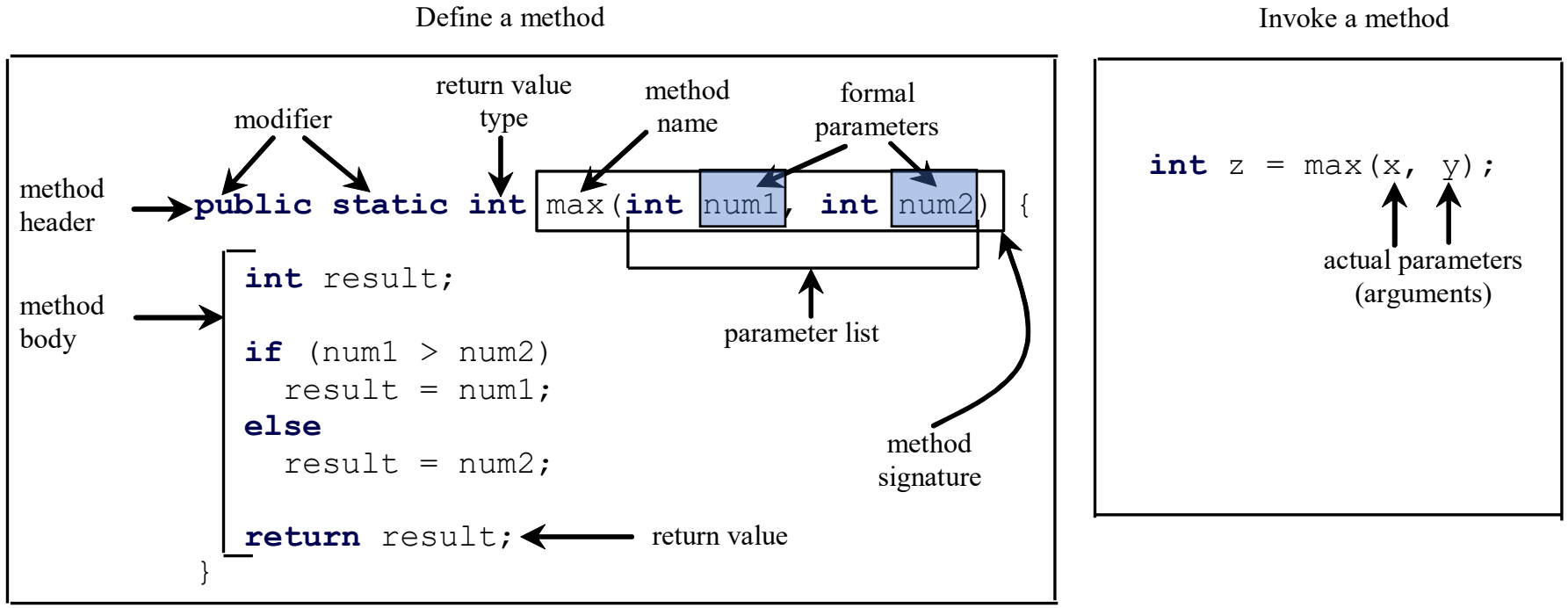


Invoke a method



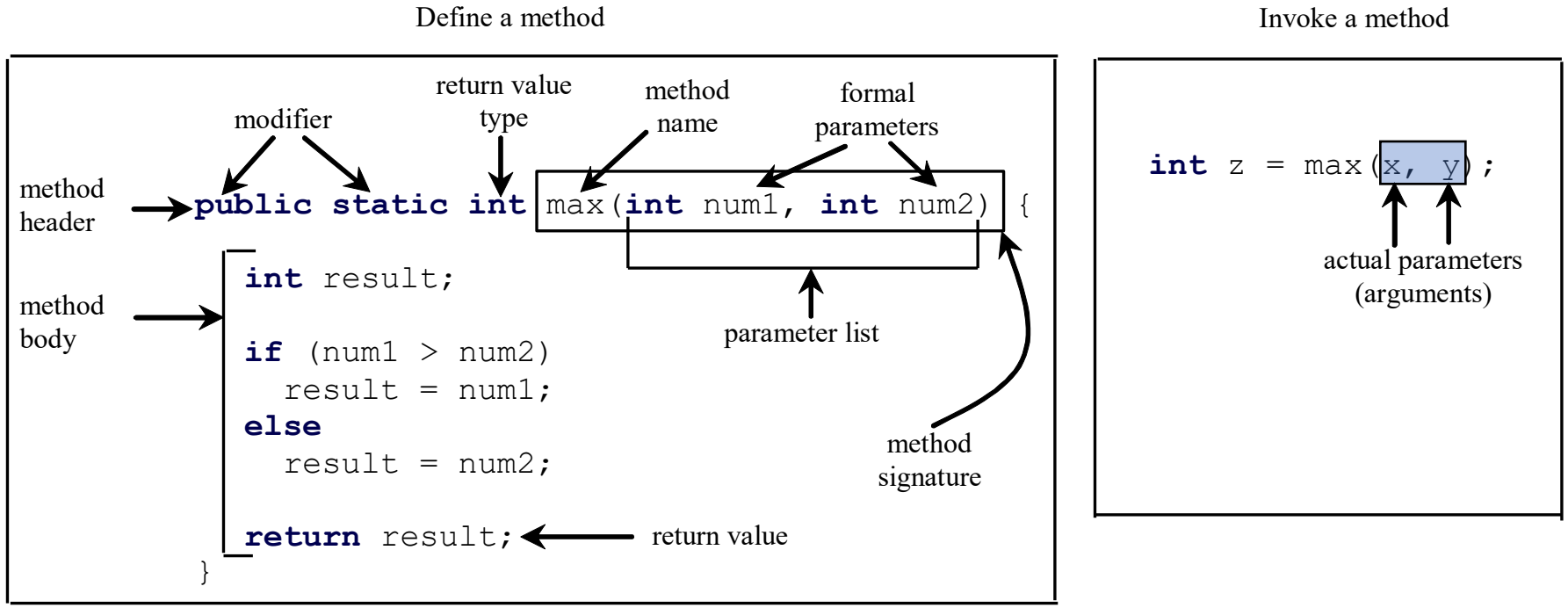
Formal Parameters

The variables defined in the method header are known as *formal parameters*.



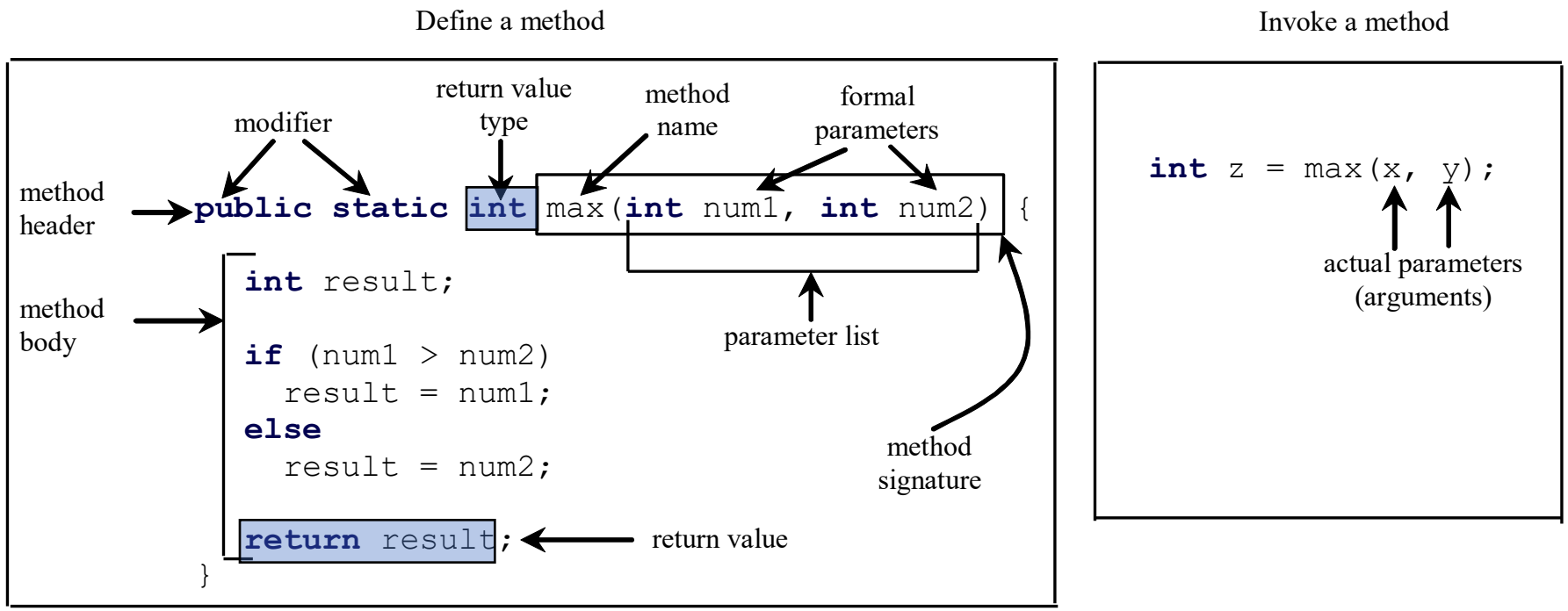
Actual Parameters

When a method is invoked, you pass a value to the parameter. This value is referred to as *actual parameter or argument*.

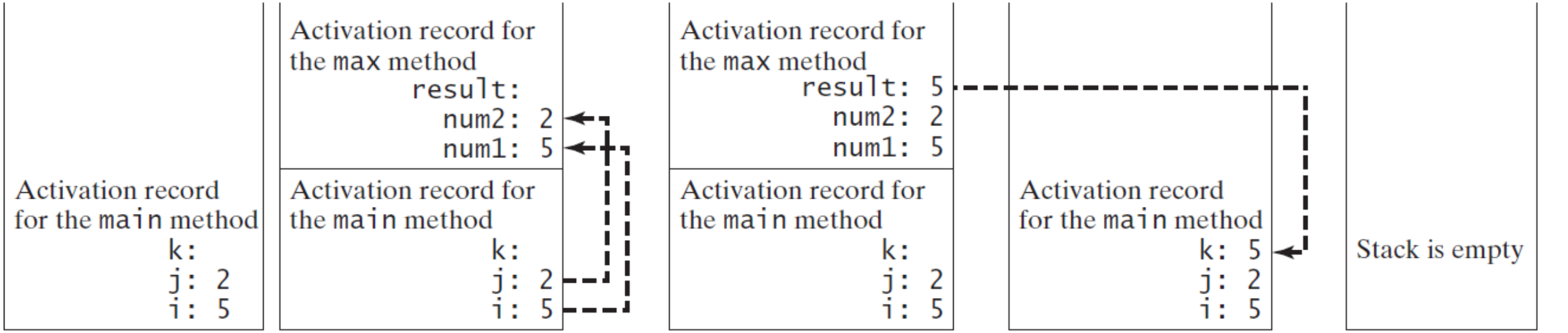


Return Value Type

A method may return a value. The returnValueType is the data type of the value the method returns. If the method does not return a value, the returnValueType is the keyword void. For example, the returnValueType in the main method is void.



Call Stacks



(a) The `main` method is invoked.

(b) The `max` method is invoked.

(c) The `max` method is being executed.

(d) The `max` method is finished and the return value is sent to `k`.

(e) The `main` method is finished.

Caution

A <u>return</u> statement is required for a value-returning method. The method shown below in (a) is logically correct, but it has a compilation error because the Java compiler thinks it possible that this method does not return any value.

```java
public static int sign(int n) {
  if (n > 0)
    return 1;
  else if (n == 0)
    return 0;
  else if (n < 0)
    return -1;
}
```

(a)

Should be →

```java
public static int sign(int n) {
  if (n > 0)
    return 1;
  else if (n == 0)
    return 0;
  else
    return -1;
}
```

(b)

To fix this problem, delete *<u>if (n < 0)</u>* in (a), so that the compiler will see a <u>return</u> statement to be reached regardless of how the <u>if</u> statement is evaluated.

Single Arrays

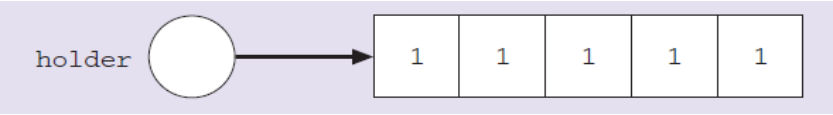
Arrays

- **way of creating an array is as follows:**


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

Here, we have only stated that there should be room for five integers to be stored. We can initialize these elements (they will default to the value 0) using assignment to individual locations:

```
holder[0]= 1;
holder[1]= 1;
holder[2]= 1;
holder[3]= 1;
holder[4]= 1;
```



- This is equivalent to the following:

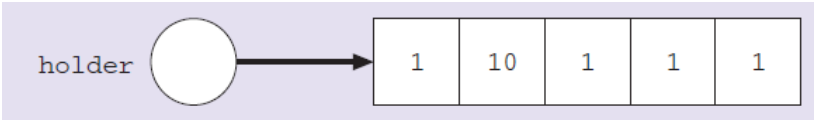
```
int[] myArray = {1,1,1,1,1};
```

Arrays

Assigning values to individual elements

- Example of changing the value stored in holder's **second** location to 10:

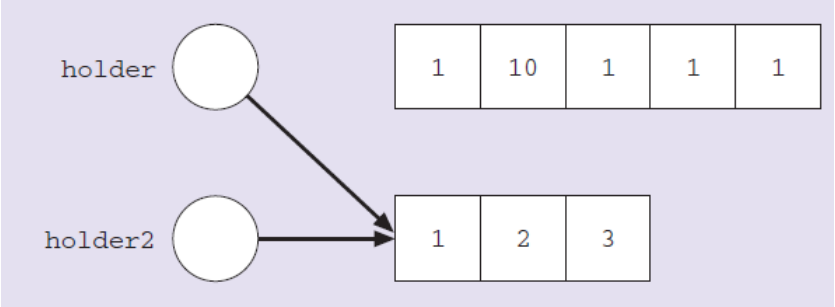
```
holder [1]= 10;
```



Changing an array reference

- As with strings (and it is true of all references), we can make an array reference variable refer to a different object.

```
int[] holder2 = {1, 2, 3};
holder = holder2;
```



Arrays

The array instance variable `length`

- Arrays have an instance variable `length`, which gives the capacity of the array. The expression:

  `holder.length`

  represents the number of locations in the array `holder`.

- Note that `length` is an **instance variable** associated with **arrays**, while `length()` is a **method** associated with **strings**.

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
- false for boolean types.

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.
- Each element in the array is represented using the following syntax, known as an *indexed variable*: arrayRefVar[index];

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

- 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};
```

Initializing arrays with input values

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {
      double[] myList = new double[3];
      Scanner input = new Scanner(System.in);
      System.out.println("Enter " + myList.length + " values: ");
      for (int i = 0; i < myList.length; i++) {
         myList[i] = input.nextDouble();
      }
      for (int i = 0; i < myList.length; i++) {
         System.out.println("mylist["+i+"]="+myList[i]);
      }
   }
}
```

```
Enter 3 values:
5
6
9
mylist[0]=5.0
mylist[1]=6.0
mylist[2]=9.0
```

Initializing arrays with random values

```
public class Main {
    public static void main(String[] args) {
        int[] myList = new int[3];
        for (int i = 0; i < myList.length; i++) {
            myList[i] = (int)(Math.random() * 100);
        }
        for (int i = 0; i < myList.length; i++) {
            System.out.println("mylist["+i+"]="+myList[i]);
        }
    }
}
```

```
mylist[0]=99
mylist[1]=71
mylist[2]=71
```

Note : The random Method

- Generates a random double value greater than or equal to 0.0 and less than 1.0
- (0 <= Math.random() < 1.0).

Examples:

```
(int)(Math.random() * 10)
```
→ Returns a random integer between 0 and 9.

```
50 + (int)(Math.random() * 50)
```
→ Returns a random integer between 50 and 99.

In general,

```
a + Math.random() * b
```
→ Returns a random number between a and a + b, excluding a + b.

Shifting Elements

```java
public class Main {
   public static void main(String[] args) {
      int[] myList = new int[4];
      for (int i = 0; i < myList.length; i++) {
         myList[i] = (int)(Math.random() * 100);
      }
      System.out.println("Before Shifting:");
      for (int i = 0; i < myList.length; i++) {
         System.out.println("mylist["+i+"]="+myList[i]);
      }
      int temp=myList[0];
      for (int i = 1; i < myList.length; i++) {
       myList[i-1]=myList[i];
      }
      myList[myList.length-1]=temp;
      System.out.println("After Shifting:");
      for (int i = 0; i < myList.length; i++) {
         System.out.println("mylist["+i+"]="+myList[i]);
      }
   }
}
```

```
Before Shifting:
mylist[0]=71
mylist[1]=88
mylist[2]=21
mylist[3]=11

After Shifting:
mylist[0]=88
mylist[1]=21
mylist[2]=11
mylist[3]=71
```

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.

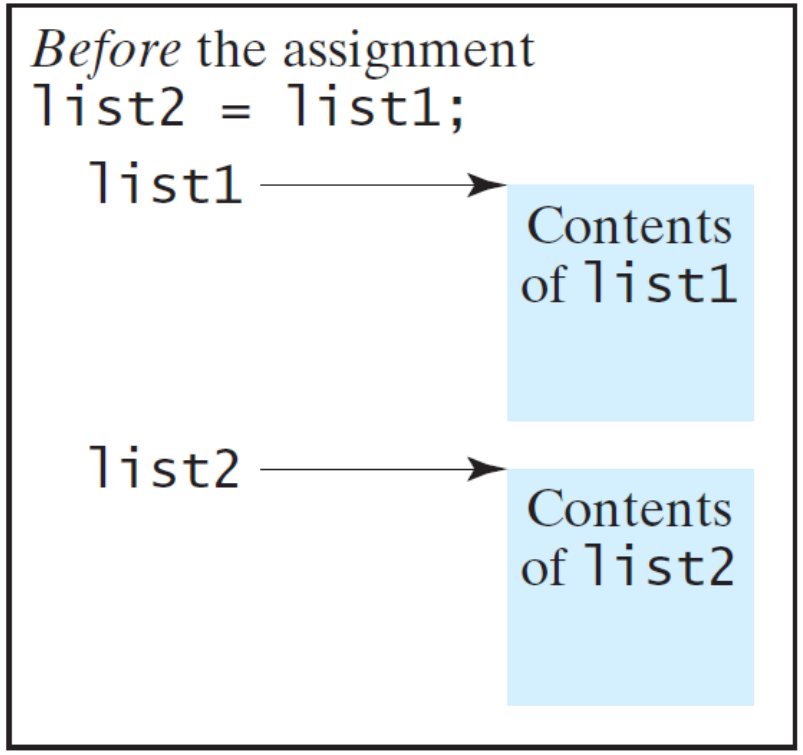
Enhanced for Loop (for-each loop)

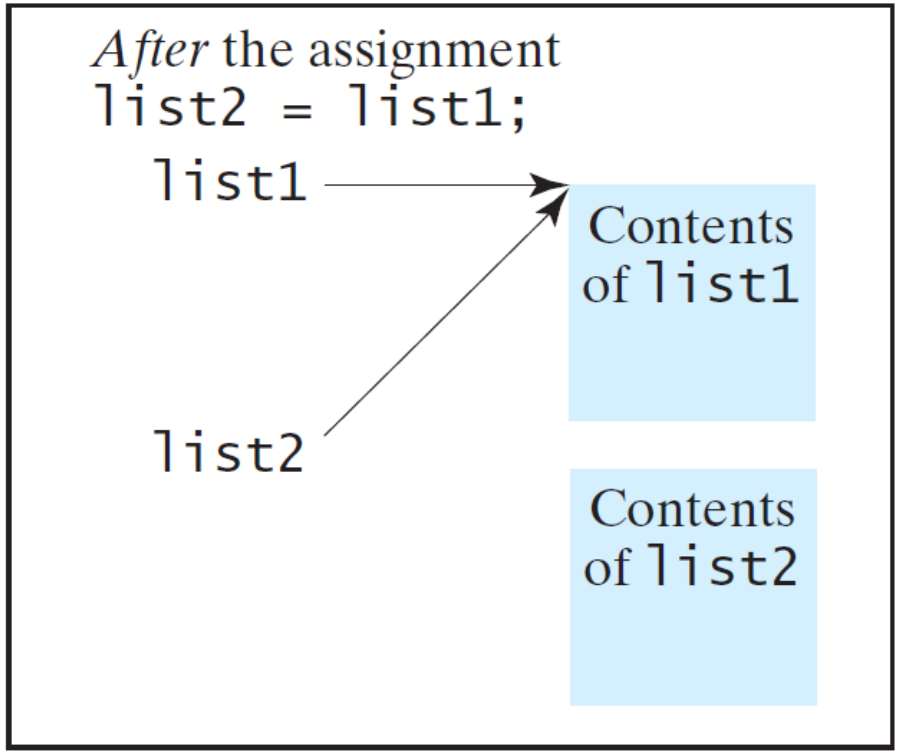
```
public class Main {
   public static void main(String[] args) {
   int[] marks = { 125, 132, 95, 116, 110 };
   int highest_marks = maximum(marks);
         System.out.println("The highest score is " + highest_marks);
}
public static int maximum(int[] numbers)
      {
      int maxSoFar = numbers[0];
      // for each loop
      for (int num : numbers)
      {
      if (num > maxSoFar)
      {
      maxSoFar = num;
      }
      }
      return maxSoFar;
      }}
```

```
The highest score is 132
```

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:

```
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];
```

The arraycopy Utility

```
arraycopy(sourceArray, src_pos, targetArray,
 tar_pos, length);
```

Example:

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

```java
public class Main {
    public static void main(String[] args) {
        int[] sourceArray = {2, 3, 1, 5, 10};
        int[] targetArray = {8,9,11,13};
        int[] temparray=new int[sourceArray.length];
        sourceArray=targetArray;
        System.out.println("source array");
        for (int i:sourceArray) {
            System.out.print(i+" ");
        }
        System.arraycopy(sourceArray, 0, temparray, 0, sourceArray.length);
        System.out.println();
        System.out.println("target array");
        for (int i:sourceArray) {
            System.out.print(i+" ");
        }
        System.out.println();
        System.out.println("address sourceArray = "+sourceArray);
        System.out.println("address targetArray = "+targetArray);
        System.out.println("address temparray = "+temparray);
    }
}
```

```
source array
8 9 11 13
target array
8 9 11 13
address sourceArray = [I@1d81eb93
address targetArray = [I@1d81eb93
address temparray = [I@7291c18f
```

Passing Arrays to Methods

```
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 Main {
   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
      y[0]=2222;
      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]
   }
}
```

```
x is 1
y[0] is 5555
```

Returning an Array from a Method

```
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;
}
```

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

Variable-Length Arguments

You can pass a variable number of arguments of the same type to a

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

```
The max value is 56.5
The max value is 3.0
```

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 Implementation

```
public class Main {
   public static int linearSearch(int[] list, int key) {
      for (int i = 0; i < list.length; i++)
         if (key == list[i])
            return i;
      return -1;
   }
   public static void main(String[] args) {
      int[] list = {1, 4, 4, 2, 5, -3, 6, 2};
      System.out.println(linearSearch(list, 4));
      System.out.println(linearSearch(list, -4));
      System.out.println(linearSearch(list, -3));
   }
}
```

```
1
-1
5
```

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);

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

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]);
  }
}
```

```java
class B {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++)
            System.out.println(args[i]);
    }
}
public class Main {
    public static void main(String[] args) {
        String[] strings = {"New York",
                "Boston", "Atlanta"};
        B.main(strings);
    }
}
```

```
New York
Boston
Atlanta
```

Two-dimensional Arrays

Declare/Create Two-dimensional Arrays

```
// Declare array ref var
dataType[][] refVar;

// Create array and assign its reference to variable
refVar = new dataType[10][10];

// Combine declaration and creation in one statement
dataType[][] refVar = new dataType[10][10];

// Alternative syntax
dataType refVar[][] = new dataType[10][10];
```

Declaring Variables of Two-dimensional Arrays and Creating Two-dimensional Arrays

- `int[][] matrix = new int[10][10];`
  or
- `int matrix[][] = new int[10][10];`
- `matrix[0][0] = 3;`

Two-dimensional Array Illustration

	[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];
```

(a)

	[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;
```

(b)

	[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}
};
```

(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,

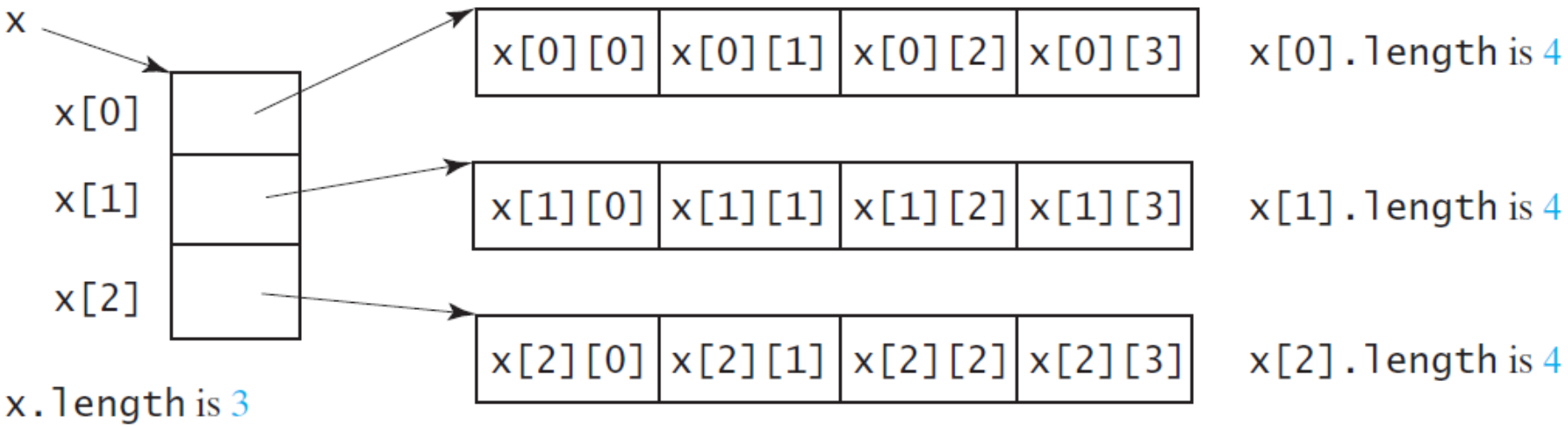
```
int[][] array = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

Same as

```
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];



Lengths of Two-dimensional Arrays, cont.

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

array.length

array[0].length

array[1].length

array[2].length

array[3].length

array[4].length ArrayIndexOutOfBoundsException

Ragged Arrays

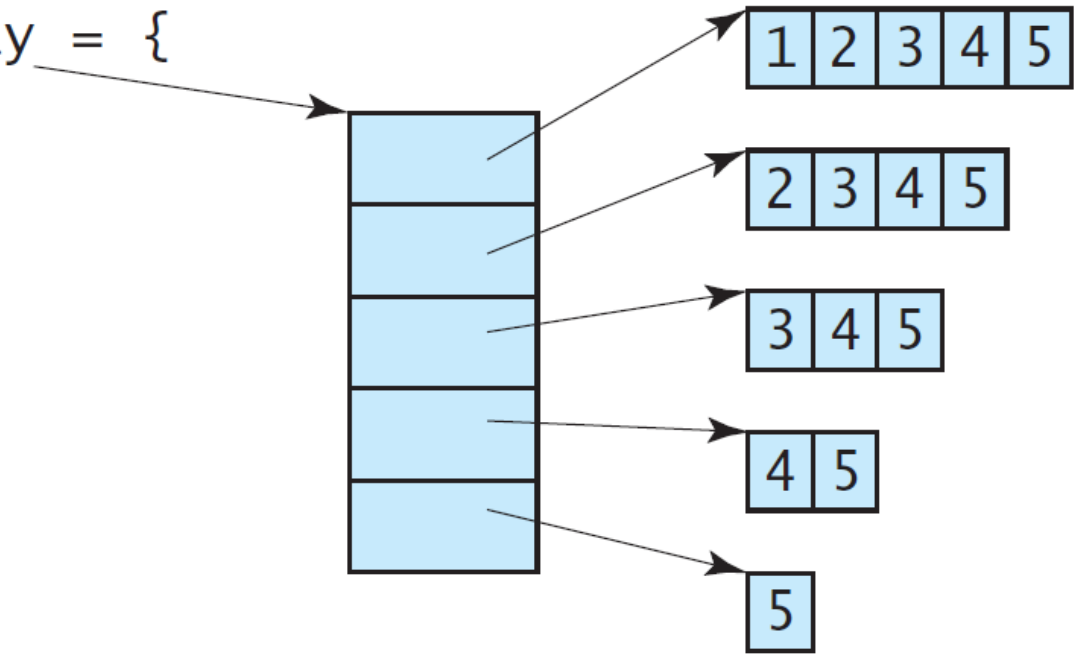
Each row in a two-dimensional array is itself an array. So, the rows can have different lengths. Such an array is known as *a ragged array*. For example,

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

```
triangleArray.length is 5
triangleArray[0].length is 5
triangleArray[1].length is 4
triangleArray[2].length is 3
triangleArray[3].length is 2
triangleArray[4].length is 1
```

Ragged Arrays, cont.

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



Printing arrays

```
public class Main {
   public static void main(String[] args) {
      int[][] triangleArray = {
            {1, 2, 3, 4, 5},
            {2, 3, 4, 5},
            {3, 4, 5},
            {4, 5},
            {5}
      };
      for (int row = 0; row < triangleArray.length; row++) {
         for (int column = 0; column < triangleArray[row].length; column++) {
            System.out.print(triangleArray[row][column] + " ");
         }
         System.out.println();
      }
   }
}
```

```
1 2 3 4 5
2 3 4 5
3 4 5
4 5
5
```

Summing all elements

```
public class Main {
   public static void main(String[] args) {
      int[][] triangleArray = {
            {1, 2, 3, 4, 5},
            {2, 3, 4, 5},
            {3, 4, 5},
            {4, 5},
            {5}
      };
      int total = 0;
      for (int row = 0; row < triangleArray.length; row++) {
         for (int column = 0; column < triangleArray[row].length; column++) {
            total += triangleArray[row][column];
         }
      }
      System.out.println("total = "+total);
   }
}
```

total = 55

Problem: Grading Multiple-Choice Test

Students' answer

	0	1	2	3	4	5	6	7	8	9
Student 0	A	B	A	C	C	D	E	E	A	D
Student 1	D	B	A	B	C	A	E	E	A	D
Student 2	E	D	D	A	C	B	E	E	A	D
Student 3	C	B	A	E	D	C	E	E	A	D
Student 4	A	B	D	C	C	D	E	E	A	D
Student 5	B	B	E	C	C	D	E	E	A	D
Student 6	B	B	A	C	C	D	E	E	A	D
Student 7	E	B	E	C	C	D	E	E	A	D

Objective: write a program that grades multiple-choice test.

Key to the Questions:

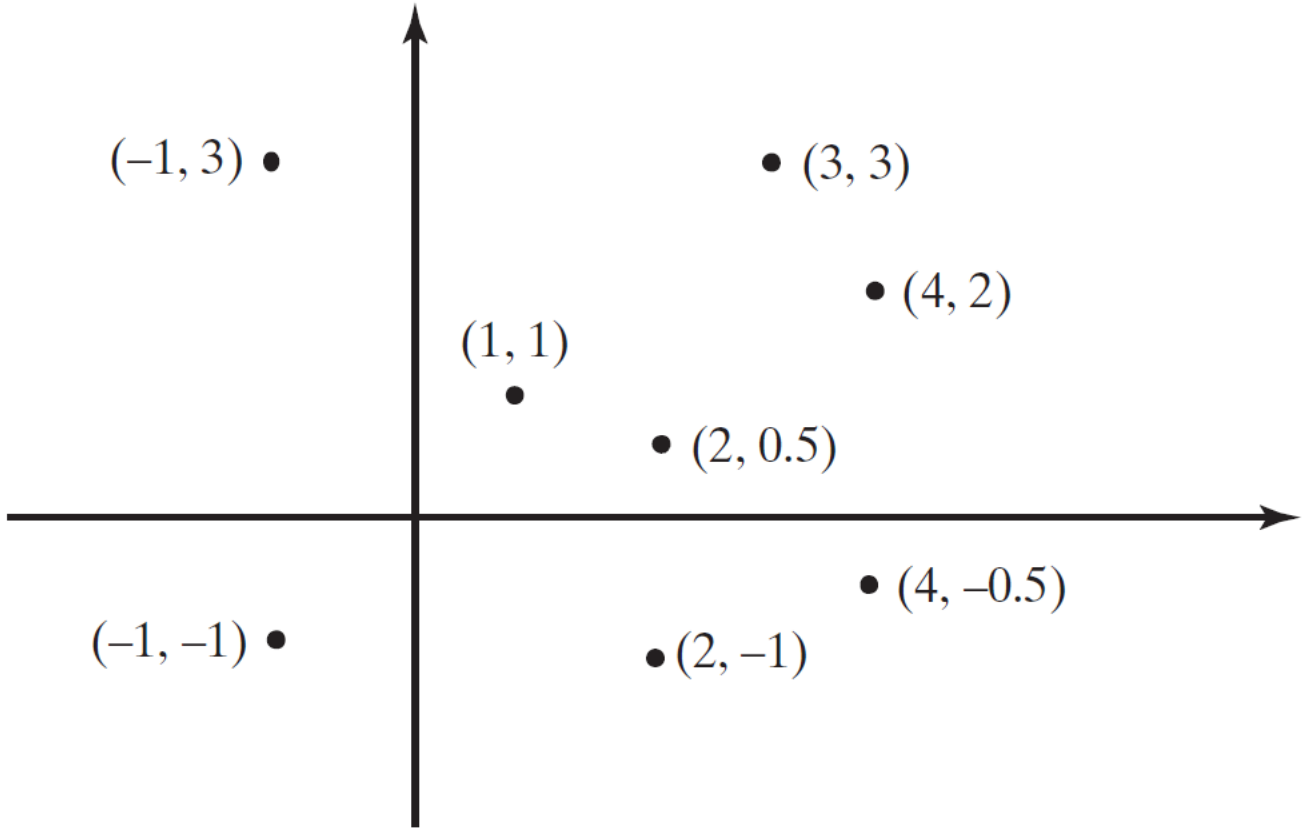
	0	1	2	3	4	5	6	7	8	9
Key	D	B	D	C	C	D	A	E	A	D

Problem: Grading Multiple-Choice Test

```java
public class Main {
  public static void main(String[] args) {
    // Students' answers to the questions
    char[][] answers = {
        {'A', 'B', 'A', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
        {'D', 'B', 'A', 'B', 'C', 'A', 'E', 'E', 'A', 'D'},
        {'E', 'D', 'D', 'A', 'C', 'B', 'E', 'E', 'A', 'D'},
        {'C', 'B', 'A', 'E', 'D', 'C', 'E', 'E', 'A', 'D'},
        {'A', 'B', 'D', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
        {'B', 'B', 'E', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
        {'B', 'B', 'A', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
        {'E', 'B', 'E', 'C', 'C', 'D', 'E', 'E', 'A', 'D'}};
    // Key to the questions
    char[] keys = {'D', 'B', 'D', 'C', 'C', 'D', 'A', 'E', 'A', 'D'};
    // Grade all answers
    for (int i = 0; i < answers.length; i++) {
      // Grade one student
      int correctCount = 0;
      for (int j = 0; j < answers[i].length; j++) {
        if (answers[i][j] == keys[j])
          correctCount++;
      }
      System.out.println("Student " + i + "'s correct count is " +correctCount);
    }  }}
```

```
Student 0's correct count is 7
Student 1's correct count is 6
Student 2's correct count is 5
Student 3's correct count is 4
Student 4's correct count is 8
Student 5's correct count is 7
Student 6's correct count is 7
Student 7's correct count is 7
```

Problem: Finding Two Points 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

PassTwoDimensionalArray

```java
import java.util.Scanner;
public class Main {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("Enter the number of points: ");
    int numberOfPoints = input.nextInt();
    // Create an array to store points
    double[][] points = new double[numberOfPoints][2];
    System.out.print("Enter " + numberOfPoints + " points: ");
    for (int i = 0; i < points.length; i++) {
      points[i][0] = input.nextDouble();
      points[i][1] = input.nextDouble();
    }
    // p1 and p2 are the indices in the points array
    int p1 = 0, p2 = 1; // Initial two points
    double shortestDistance = distance(points[p1][0], points[p1][1],
        points[p2][0], points[p2][1]); // Initialize shortestDistance
    // Compute distance for every two points
    for (int i = 0; i < points.length; i++) {
      for (int j = i + 1; j < points.length; j++) {
        double distance = distance(points[i][0], points[i][1],
            points[j][0], points[j][1]); // Find distance
        if (shortestDistance > distance) {
          p1 = i; // Update p1
          p2 = j; // Update p2
          shortestDistance = distance; // Update shortestDistance
        } }  }
```

```java
// Display result
    System.out.println("The closest two points are " +
        "(" + points[p1][0] + ", " + points[p1][1] + ") and (" +
        points[p2][0] + ", " + points[p2][1] + ")");
  }

  public static double distance(
      double x1, double y1, double x2, double y2) {
    return Math.sqrt((x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1));
  }
}
```

```
Enter the number of points: 4
Enter 4 points:
1 2
2 2
3 5
2 1
The closest two points are (1.0, 2.0) and (2.0, 2.0)
```

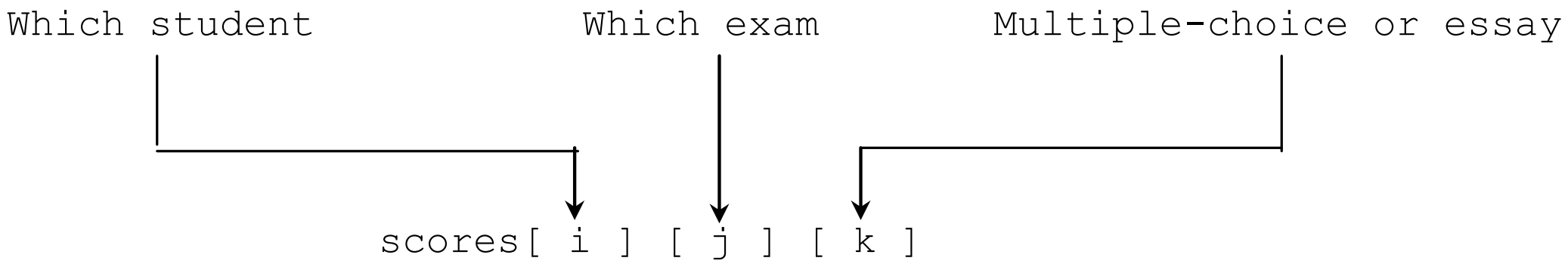
Multidimensional Arrays

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.

Multidimensional Arrays

```
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: Calculating Total Scores

- Objective: write a program that calculates the total score for students in a class. Suppose the scores are stored in a three-dimensional array named scores.
- The first index in scores refers to a student, the second refers to an exam, and the third refers to the part of the exam.
- Suppose there are 7 students, 5 exams, and each exam has two parts--the multiple-choice part and the programming part.
- So, scores[i][j][0] represents the score on the multiple-choice part for the i's student on the j's exam.
- Your program displays the total score for each student.

```java
public class Main {
  public static void main(String[] args) {
    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}},
        {{1.5, 29.5}, {6.4, 22.5}, {14, 30.5}, {10, 30.5}, {16, 6.0}}};
    // Calculate and display total score for each student
    for (int i = 0; i < scores.length; i++) {
      double totalScore = 0;
      for (int j = 0; j < scores[i].length; j++)
        for (int k = 0; k < scores[i][j].length; k++)
          totalScore += scores[i][j][k];

      System.out.println("Student " + i + "'s score is " +
          totalScore);
    }  }
}
```

```
Student 0's score is 160.0
Student 1's score is 163.0
Student 2's score is 148.4
Student 3's score is 174.4
Student 4's score is 202.4
Student 5's score is 181.4
Student 6's score is 166.9
```

Thanks

References

- Java: How To Program, Early Objects, 11th edition
- Cay S. Horstmann, Big Java: Late Objects
- Introduction to Java Programming and Data Structures, Comprehensive Version 12th Edition, by Y. Liang (Author), Y. Daniel Liang
- Java documentation https://docs.oracle.com/javase
- https://www.youtube.com/watch?v=LTgClBqDank&feature=youtu.be
- Listing 7.3 Animation by Y. Daniel Liang (pearsoncmg.com)
- Linear Search Animation by Y. Daniel Liang (pearsoncmg.com)
- Introduction to Java Programming and Data Structures, 12E, Y. Daniel Liang - VarArgsDemo.java (pearsoncmg.com)
- Introduction to Java Programming and Data Structures, 12E, Y. Daniel Liang - CountLettersInArray.java (pearsoncmg.com)

References

- Introduction to Java Programming and Data Structures, 12E, Y. Daniel Liang - TestPassArray.java (pearsoncmg.com)
- Introduction to Java Programming and Data Structures, 12E, Y. Daniel Liang - TestMax.java (pearsoncmg.com)
- Introduction to Java Programming and Data Structures, 12E, Y. Daniel Liang - Weather.java (pearsoncmg.com)
- Introduction to Java Programming and Data Structures, 12E, Y. Daniel Liang - TotalScore.java (pearsoncmg.com)
- Introduction to Java Programming and Data Structures, 12E, Y. Daniel Liang - GradeExam.java (pearsoncmg.com)
- Introduction to Java Programming and Data Structures, 12E, Y. Daniel Liang - CheckSudokuSolution.java (pearsoncmg.com)
- Binary Search Animation by Y. Daniel Liang (pearsoncmg.com)