Arrays of Objects

String Class

Java - String Class

- Strings are widely used in Java programming.
- In Java programming language, strings are treated as objects.
- A String object contains a collection of characters surrounded by double quotes:

```
String  greeting = "Hello world";
```

`String` key word	variable name	=	texts

- Whenever it encounters a string literal in your code, the compiler creates a String object with its value in this case, "Hello world!'.

Java - String Class

- As with any other object, you can create String objects by using the **new** keyword and a constructor. The String class has 11 constructors that allow you to provide the initial value of the string using different sources, such as an array of characters.

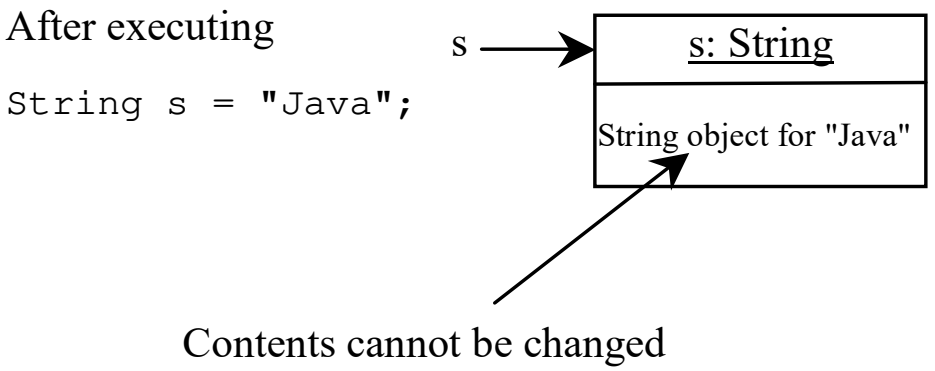
```
public class StringDemo {

        public static void main(String args[]) {
           char[] helloArray = { 'h', 'e', 'l', 'l', 'o', '.' };
           String helloString = new String(helloArray);
           System.out.println( helloString );
        }
    }
```

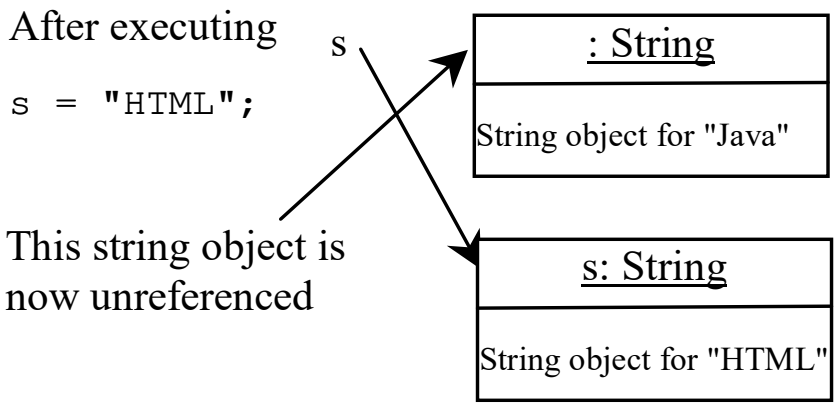
- Java Docs: String

Strings Are Immutable

A String object is immutable; its contents cannot be changed. Does the following code change the contents of the string?

```
String s = "Java";

s = "HTML";
```





String Methods: `length()`

- String Length

```
1  public class StringDemo {
2
3          public static void main(String args[]) {
4              String palindrome = "Dot saw I was Tod";
5              int len = palindrome.length();
6              System.out.println( "String Length is : " + len );
7          }
8      }
9
```

String Methods: `concat()`

- The String class includes a method for concatenating two strings
- The + operator can be used between strings to combine them. This is called concatenation:

```
String str1 = "hello";
String str2 = "Java";
String concatStr = str1.concat( str2 );
System.out.println( concatStr );
```

- Strings are more commonly concatenated with the + operator

```
String  str3 = "Hello" + "java" + "!";
```

String Conversions

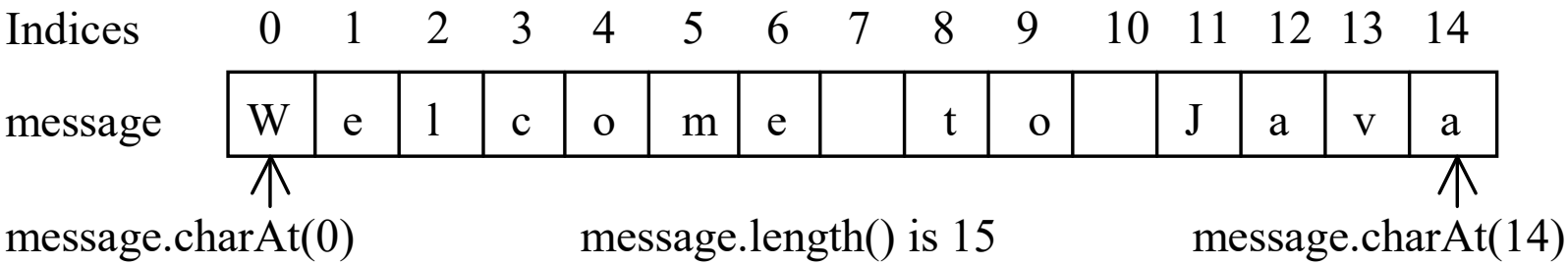
The contents of a string cannot be changed once the string is created. But you can convert a string to a new string using the following methods:

- `toLowerCase()`: returns a new string with all lowercase letters.
- `toUpperCase()`: returns a new string with all uppercase letters

```
"Welcome".toLowerCase(); // returns a new string welcome.
"Welcome".toUpperCase(); // returns a new string WELCOME.
```

Retrieving Individual Characters in a String

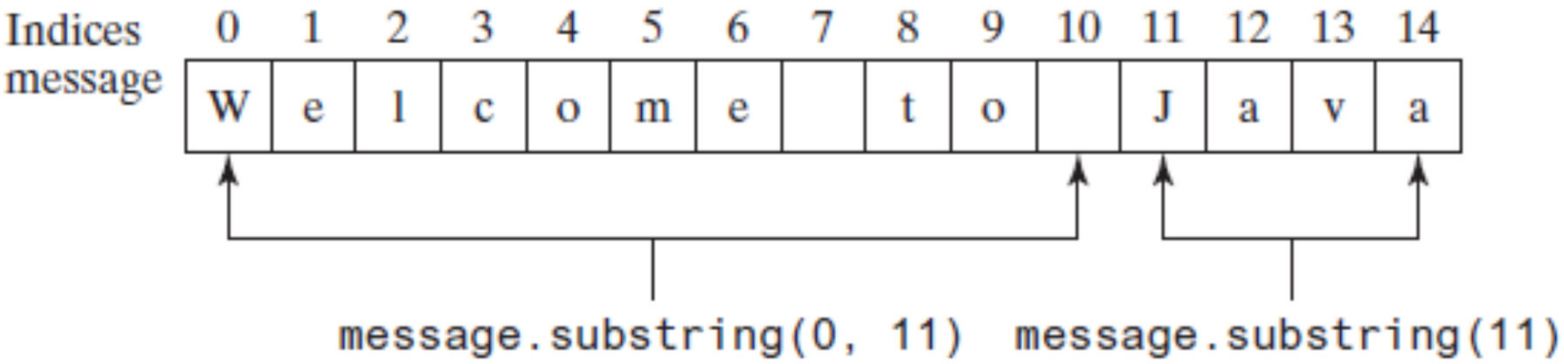
- Use `message.charAt(index)`
- Index starts from `0`
- Do not use an index beyond `s.length()-1`



Obtaining Substrings

- You can obtain a single character from a string using the `charAt()` method. You can also obtain a substring from a string using the `substring()` method.

```
String message1 = "Welcome to Java";
String message2 = message1.substring(0,11) + "HTML";
```



Conversion between Strings and Numbers

- You can convert a numeric string into a number.
- To convert a string into an int value, use the `Integer.parseInt()`
- To convert a string into an double value, use the `Double.parseDouble()`
- If the string is not a numeric string, the conversion would cause a runtime error.

```
int intValue = Integer.parseInt("123");
double doubleValue = Double.parseDouble("43.57");
```

Conversion between Strings and Numbers

- You can convert a number into a string; simply use the string concatenating operator as follows:

```
String s = 45.6 + "";
```

Finding a Character or a Substring in a String

```
"Welcome to Java".indexOf('W'); // returns 0.
"Welcome to Java".indexOf('x'); // returns -1.
"Welcome to Java".indexOf('o', 5); // returns 9.
"Welcome to Java".indexOf("come"); // returns 3.
"Welcome to Java".indexOf("Java", 5); // returns 11.
"Welcome to Java".indexOf("java", 5); // returns -1.
"Welcome to Java".lastIndexOf('a'); // returns 14.
```

More String Metho

- Java Docs: String

String	
+String()	Constructs an empty string
+String(value: String)	Constructs a string with the specified string literal value
+String(value: char[])	Constructs a string with the specified character array
+charAt(index: int): char	Returns the character at the specified index from this string
+compareTo(anotherString: String): int	Compares this string with another string
+compareToIgnoreCase(anotherString: String): int	Compares this string with another string ignoring case
+concat(anotherString: String): String	Concat this string with another string
+endsWith(suffix: String): boolean	Returns true if this string ends with the specified suffix
+equals(anotherString: String): boolean	Returns true if this string is equal to anther string
+equalsIgnoreCase(anotherString: String): boolean	Checks if this string equals anther string ignoring case
+getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin): void	Copies characters from this string into the destination character array
+indexOf(ch: int): int	Returns the index of the first occurrence of ch
+indexOf(ch: int, fromIndex: int): int	Returns the index of the first occurrence of ch after fromIndex
+indexOf(str: String): int	Returns the index of the first occurrence of str
+indexOf(str: String, fromIndex: int): int	Returns the index of the first occurrence of str after fromIndex
+lastIndexOf(ch: int): int	Returns the index of the last occurrence of ch
+lastIndexOf(ch: int, fromIndex: int): int	Returns the index of the last occurrence of ch before fromIndex
+lastIndexOf(str: String): int	Returns the index of the last occurrence of str
+lastIndexOf(str: String, fromIndex: int): int	Returns the index of the last occurrence of str before fromIndex
+regionMatches(toffset: int, other: String, offset: int, len: int): boolean	Returns true if the specified subregion of this string exactly matches the specified subregion of the string argument
+length(): int	Returns the number of characters in this string
+replace(oldChar: char, newChar: char): String	Returns a new string with oldChar replaced by newChar
+startsWith(prefix: String): boolean	Returns true if this string starts with the specified prefix
+subString(beginIndex: int): String	Returns the substring from beginIndex
+subString(beginIndex: int, endIndex: int): String	Returns the substring from beginIndex to endIndex
+toCharArray(): char[]	Returns a char array consisting characters from this string
+toLowerCase(): String	Returns a new string with all characters converted to lowercase
+toString(): String	Returns a new string with itself
+toUpperCase(): String	Returns a new string with all characters converted to uppercase
+trim(): String	Returns a string with blank characters trimmed on both sides
+copyValueOf(data: char[]): String	Returns a new string consisting of the char array data
+valueOf(c: char): String	Returns a string consisting of the character c
+valueOf(data: char[]): String	Same as copyValueOf(data: char[]): String
+valueOf(d: double): String	Returns a string representing the double value
+valueOf(f: float): String	Returns a string representing the float value
+valueOf(i: int): String	Returns a string representing the int value
+valueOf(l: long): String	Returns a string representing the long value

Arrays class

Java Arrays

- **Java array** is an object which contains elements of a similar data type.
- The elements of an array are stored in a contiguous memory location.
- We can store only a fixed set of elements in a Java array.
- Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

Declaring Array Variables

- `datatype[] arrayRefVar;`

  Example:

  ```
  double[] myList;
  ```

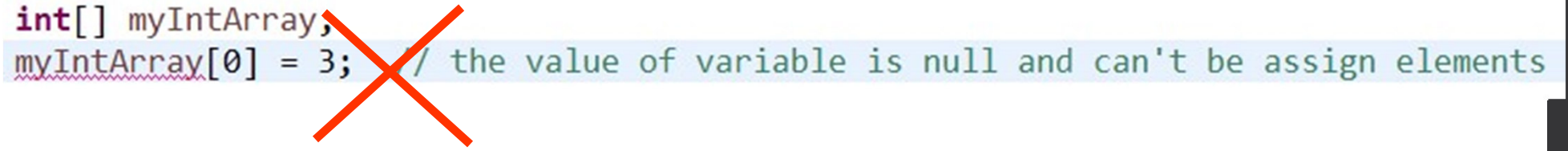
- `datatype arrayRefVar[];` // This style is correct, but not preferred

  Example:

  ```
  double myList[];
  ```

Creating Arrays

- Unlike declarations for primitive data type variables, the declaration of an array variable does not allocate any space in memory for the array
- If a variable does not contain a reference to an array, the value of the variable is **null**.



Creating Arrays

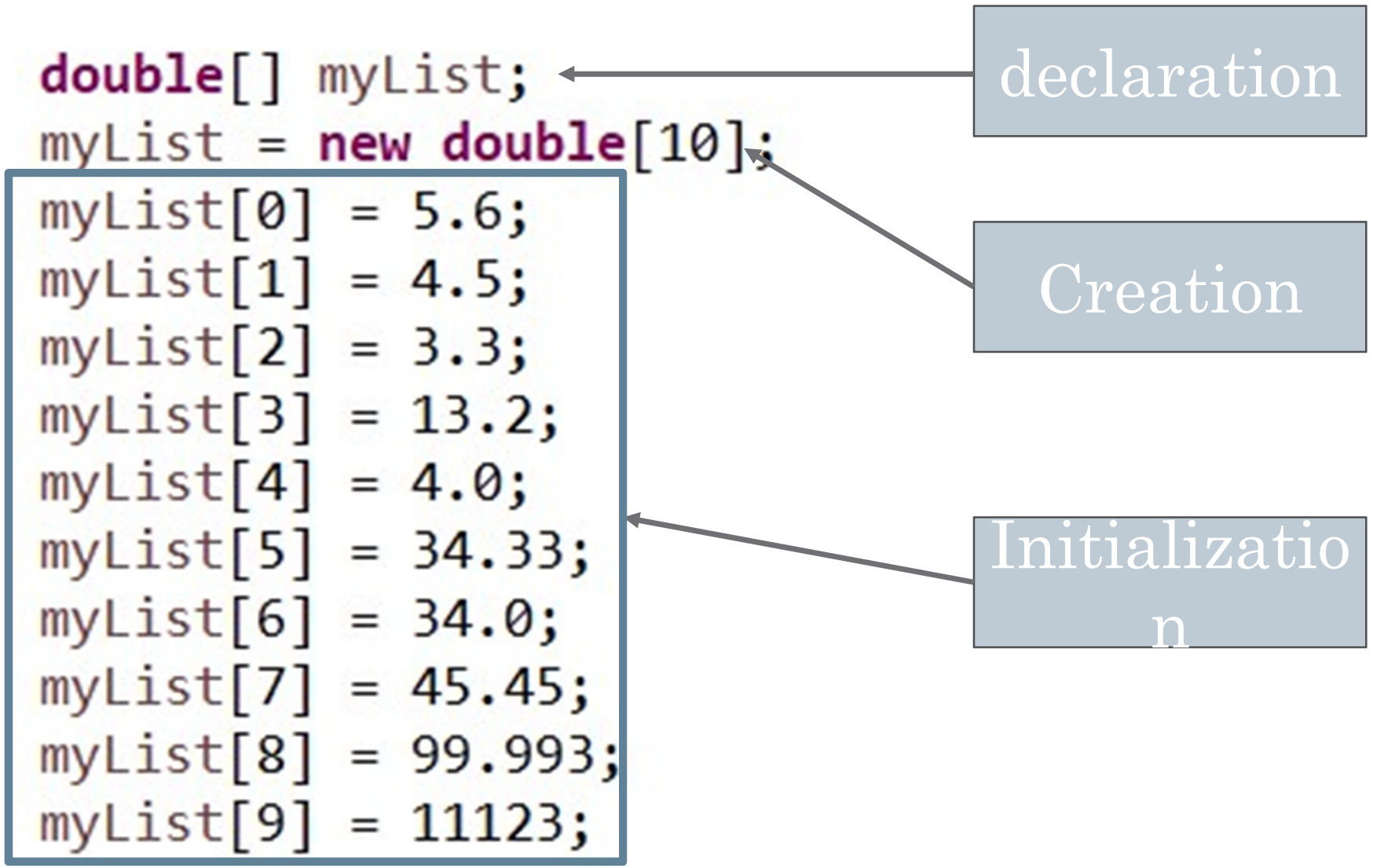
```
arrayRefVar = new datatype[arraySize];
```

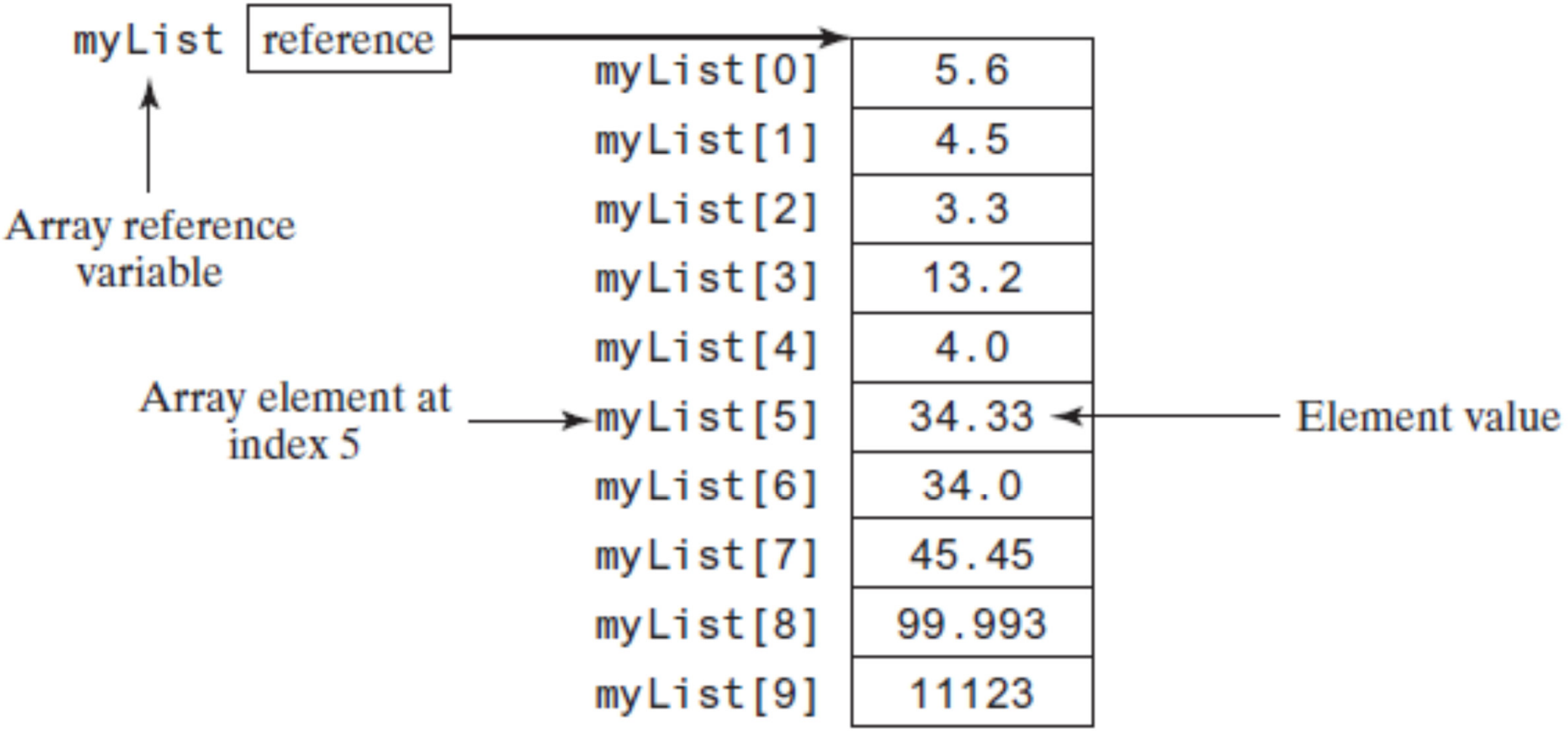
Example:

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

`myList[0]` references the first element in the array.

`myList[9]` references the last element in the array.


double[] myList;
myList = new double[10];
myList[0] = 5.6;
myList[1] = 4.5;
myList[2] = 3.3;
myList[3] = 13.2;
myList[4] = 4.0;
myList[5] = 34.33;
myList[6] = 34.0;
myList[7] = 45.45;
myList[8] = 99.993;
myList[9] = 11123;
declaration
Creation
Initialization


myList
reference
Array reference variable
myList[0]
5.6
myList[1]
4.5
myList[2]
3.3
myList[3]
13.2
myList[4]
4.0
Array element at index 5
myList[5]
34.33
Element value
myList[6]
34.0
myList[7]
45.45
myList[8]
99.993
myList[9]
11123

Declaring and Creating in One Step

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

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

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

```
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 lengthOfMyList = myList.length;
```

Default Values

When an array is created, its elements are assigned the default value of

- boolean : false
- int : 0
- double : 0.0
- String : null
- char: \u0000
- User Defined Type : null

Accessing Array Elements

- The array elements are accessed through the index.
- The index begins with 0 and ends at (total array size)-1.
- Each element in the array is represented using the following syntax, known as an *indexed variable*:

```
arrayRefVar[index];
```

- Example:

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

Array Initializers

- In a situation, where the size of the array and variables of array are already known, *array initializer* can be used, which combines the declaration, creation, and initialization of an array in one statement using the following syntax:

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

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

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;
```

Processing Arrays

1. (Initializing arrays)
2. (Printing arrays)
3. (Summing all elements)
4. (Finding the largest element)
5. (Finding the smallest index of the largest element)

Initializing arrays

```
public class ProcessingArray {
    public static void main(String[] args) {


        double[] myList; // declare an array
        myList = new double[5]; // create an array

        // Initializing arrays with input values
        java.util.Scanner input = new java.util.Scanner(System.in);
        System.out.print("Enter " + myList.length + " values: ");
        for (int i = 0; i < myList.length; i++) {
        myList[i] = input.nextDouble();
        }
```

Summing all elements

```
//Summing all elements
double total = 0;
for (double e : myList) {
    total += e;
}
System.out.println("the sum of all values is: "+ total);
```

Finding the largest element

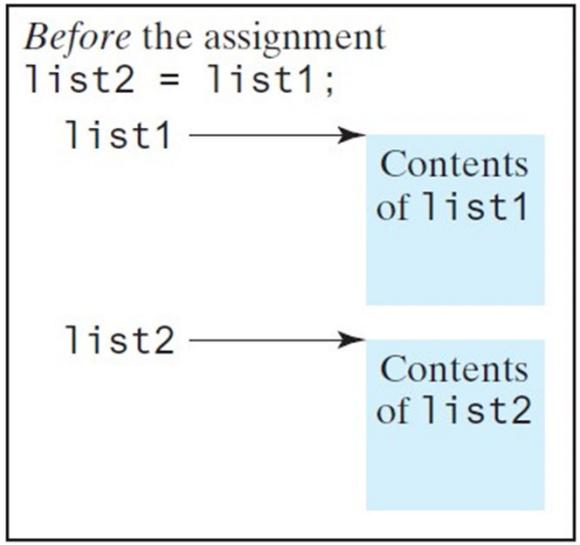
```
// Finding the largest element
double max = myList[0];
for (int i = 1; i < myList.length; i++) {
    if (myList[i] > max) max = myList[i];
}
System.out.println("The largest values is: "+ max);
```

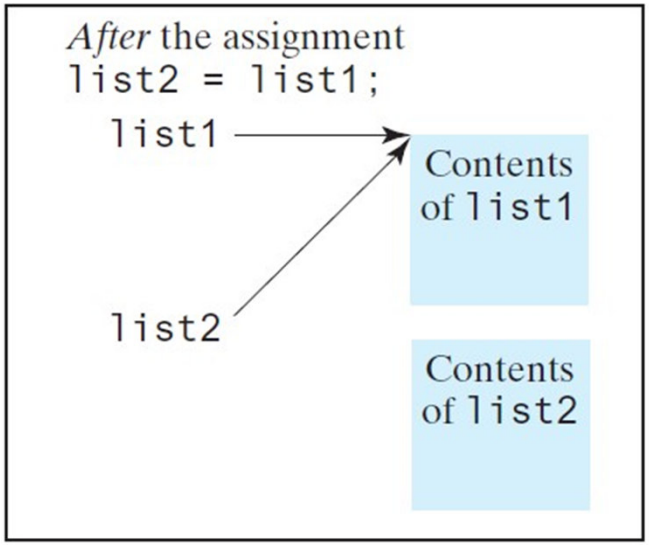
Copying Arrays

- 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;
```

copy





Copying Arrays

Using a loop:

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

for (int i = 0; i < sourceArray.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);
```

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

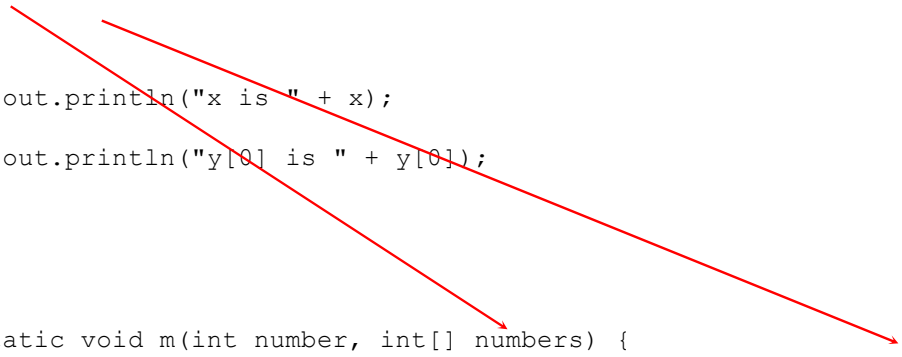
Java uses *pass by value* to pass parameters to a method. There are important differences between passing a value of variables of primitive data types and passing arrays.

- 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.

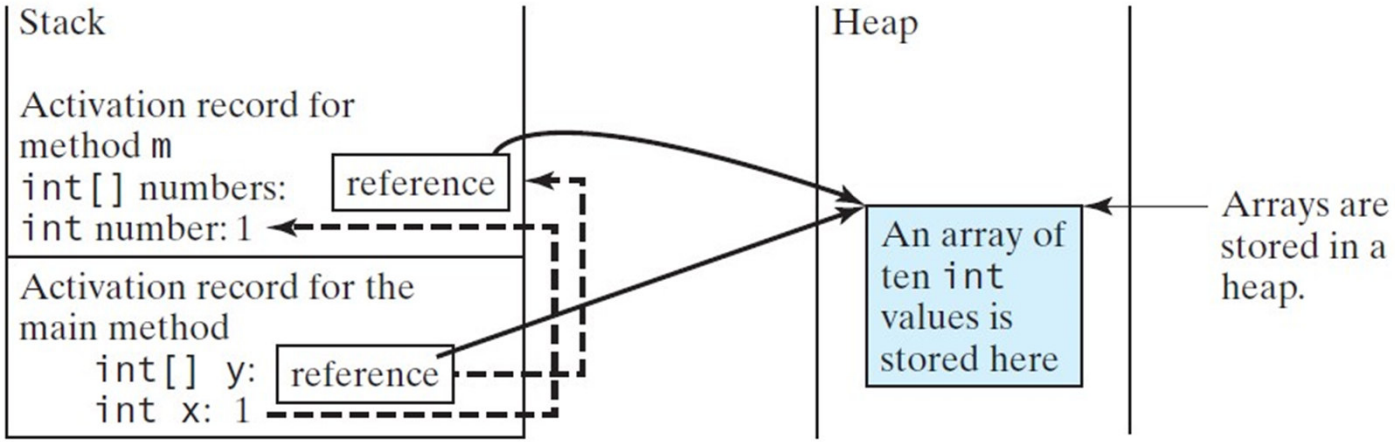
Simple Example

```
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]
  }
}
```



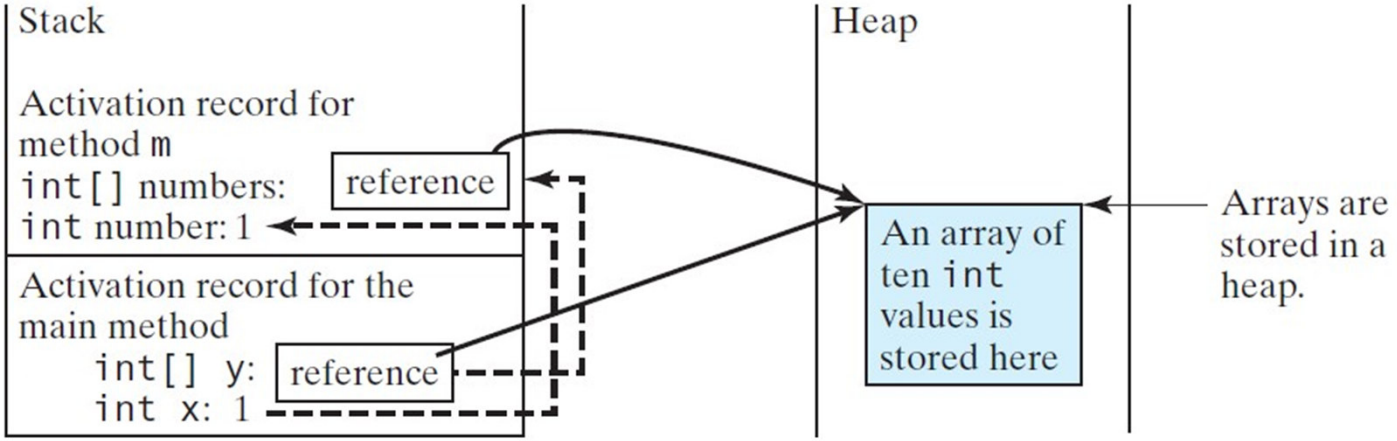
Call Stack

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 valu



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 arb



Passing Arrays as Arguments

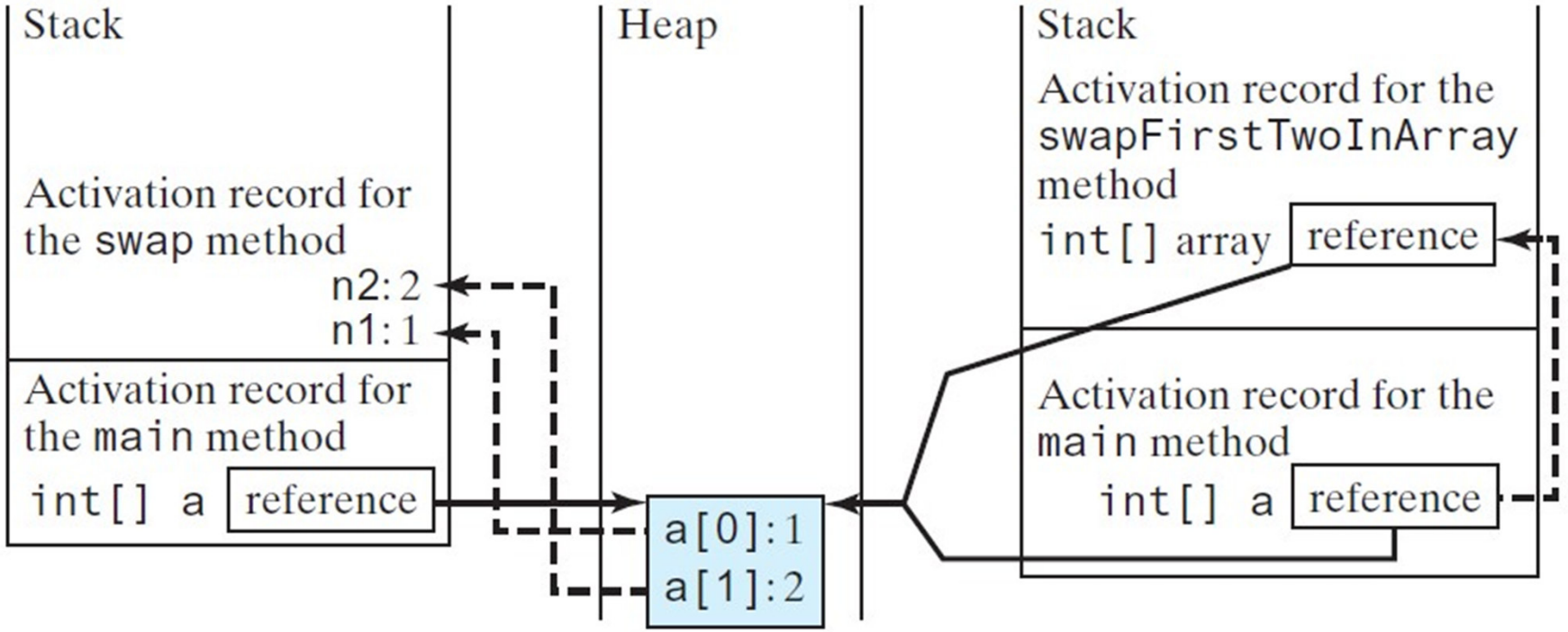
```java
public static void main(String[] args) {
    int[] a = {1, 2};

    // Swap elements using the swap method
    System.out.println("Before invoking swap");
    System.out.println("array is {" + a[0] + ", " + a[1] + "}");
    swap(a[0], a[1]);
    System.out.println("After invoking swap");
    System.out.println("array is {" + a[0] + ", " + a[1] + "}");
    // Swap elements using the swapFirstTwoInArray method
    System.out.println("Before invoking swapFirstTwoInArray");
    System.out.println("array is {" + a[0] + ", " + a[1] + "}");
    swapFirstTwoInArray(a);
    System.out.println("After invoking swapFirstTwoInArray");
    System.out.println("array is {" + a[0] + ", " + a[1] + "}");

}

/** Swap two variables */
public static void swap(int n1, int n2) {

/** Swap the first two elements in the array */
public static void swapFirstTwoInArray(int[] array) {
```

```java
/** Swap two variables */
public static void swap(int n1, int n2) {
    int temp = n1;
    n1 = n2;
    n2 = temp;
}

/** Swap the first two elements in the array */
public static void swapFirstTwoInArray(int[] array) {
    int temp = array[0];
    array[0] = array[1];
    array[1] = temp;
}
```

Passing Arrays as Arguments

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

```java
public class TestPassArray {

    /** Main method */
    public static void main(String[] args) {
        int[] a = {1, 2};

        // Swap elements using the swap method
        System.out.println("Before invoking swap");
        System.out.println("array is {" + a[0] + ", " + a[1] + "}");
        swap(a[0], a[1]);
        System.out.println("After invoking swap");
        System.out.println("array is {" + a[0] + ", " + a[1] + "}");
        // Swap elements using the swapFirstTwoInArray method
        System.out.println("Before invoking swapFirstTwoInArray");
        System.out.println("array is {" + a[0] + ", " + a[1] + "}");
        swapFirstTwoInArray(a);
        System.out.println("After invoking swapFirstTwoInArray");
        System.out.println("array is {" + a[0] + ", " + a[1] + "}");

    }

    /** Swap two variables */
    public static void swap(int n1, int n2) {

    /** Swap the first two elements in the array */
    public static void swapFirstTwoInArray(int[] array) {
```



Invoke swap(int n1, int n2).

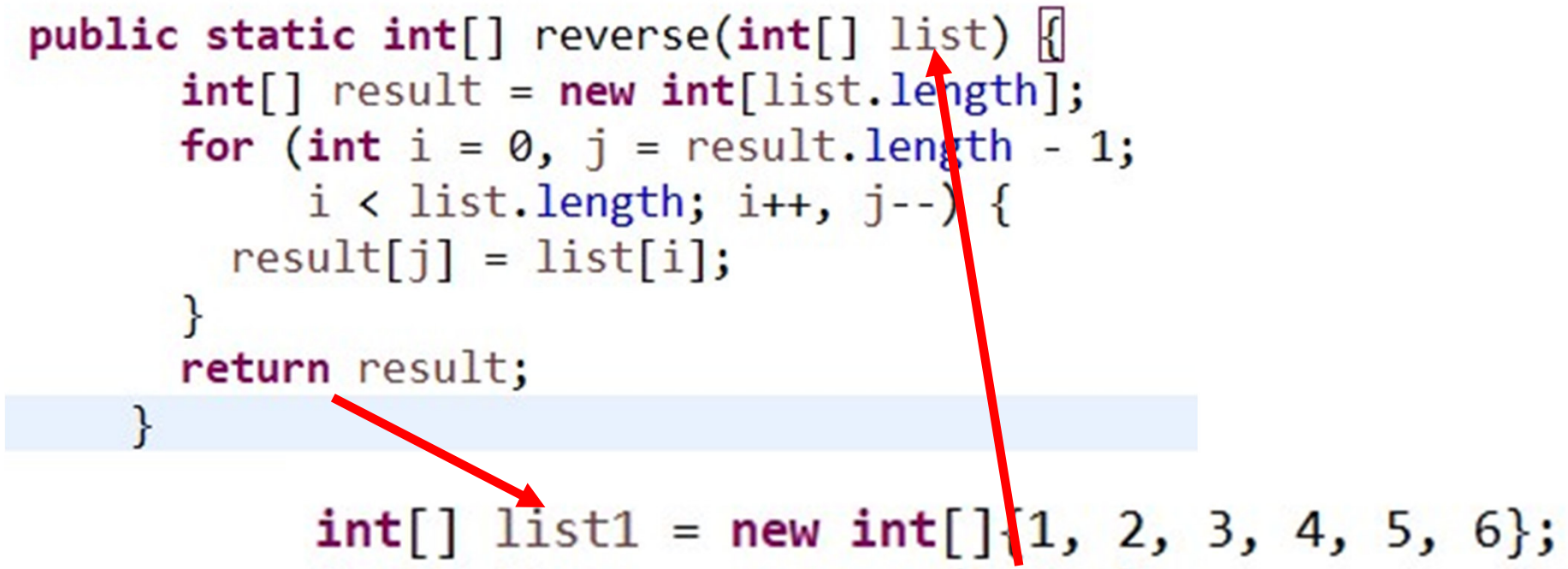
The arrays are stored in a heap

Invoke swapFirstTwoInArray(int[ ] array).

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



The Arrays Class

- The java.util.Arrays class contains various static methods for sorting and searching arrays, comparing arrays, and filling array elements.
  - `public static int binarySearch(Object[] a, Object key)`
  - `public static boolean equals(long[] a, long[] a2)`
  - `public static void fill(int[] a, int val)`
  - `public static void sort(Object[] a)`

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

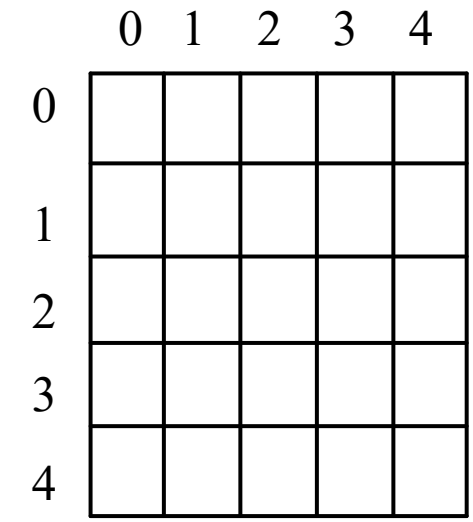
```
int matrix[][] = new int[10][10];

matrix[0][0] = 3;


for (int i = 0; i < matrix.length; i++)
  for (int j = 0; j < matrix[i].length; j++)
    matrix[i][j] = (int)(Math.random() * 1000);


double[][] x;
```

Two-dimensional Array Illustration

```
int[][] matrix = new int[3][3];
```

	Column 0	Column 1	Column 2
Row 0	x[0][0]	x[0][1]	x[0][2]
Row 1	x[1][0]	x[1][1]	x[1][2]
Row 2	x[2][0]	x[2][1]	x[2][2]

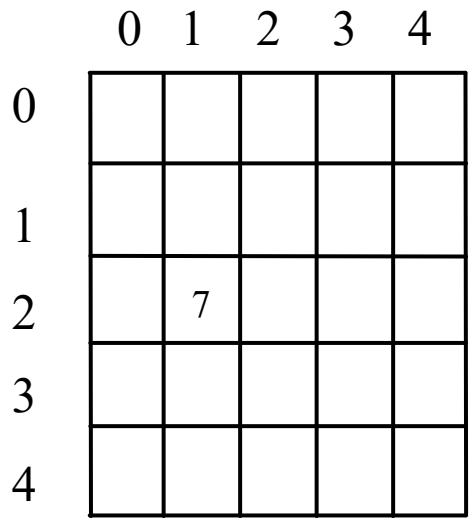
Two-dimensional Array Illustration



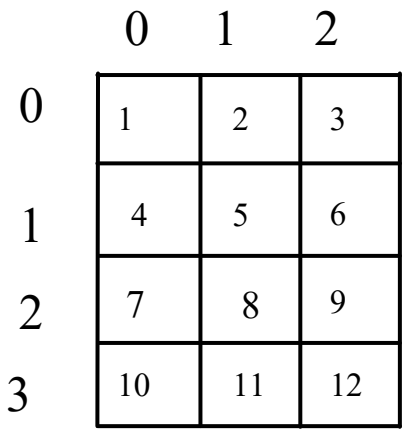

```
matrix = new int[5][5];
```

matrix.length? 5

matrix[0].length? 5




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




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

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}
};
```

(a)

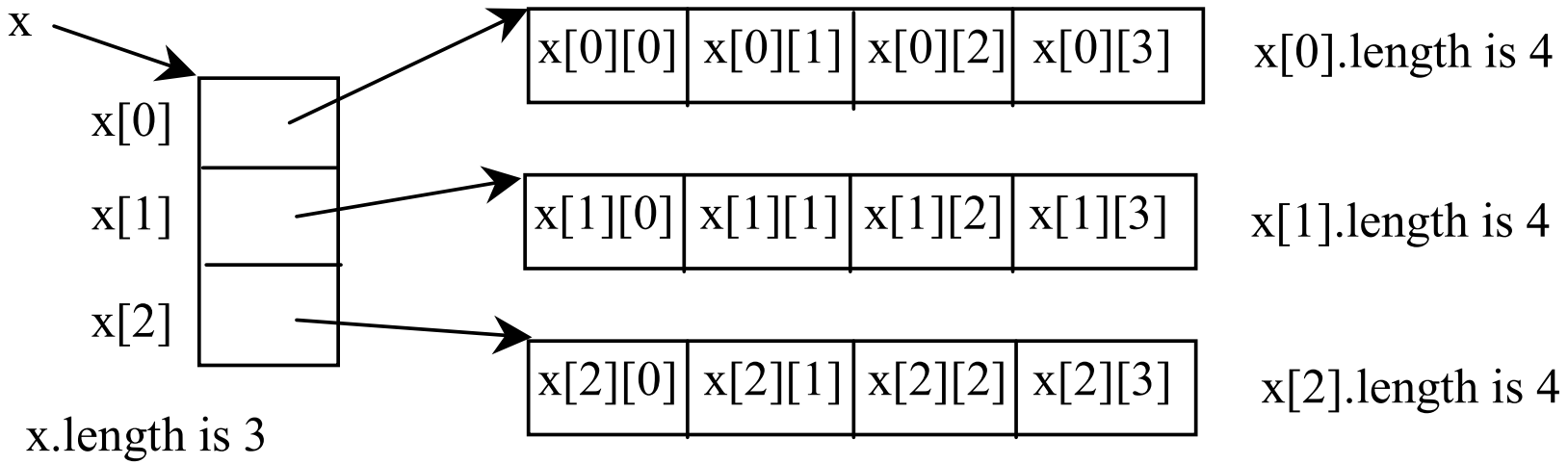
Equivalent

```
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;
```

(b)

Lengths of Two-dimensional Arrays

- A two-dimensional array is actually an array in which each element is a one-dimensional array.
- `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,

```
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

Grading Multiple-Choice Test

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

Students' Answers to the Questions:

	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

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

```java
public class GradeExam {
  /** Main method */
  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);
    }
  }
}
```

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. For example, the following syntax declares a three-dimensional array variable scores, creates an array, and assigns its reference to scores.

```
double[][][] scores = new double[10][5][2];
```

Array Of Objects

- Java is capable of storing objects as elements of the array along with other primitive and custom data types. Note that when you say ‘array of objects’, it is not the object itself that is stored in the array but the references of the object.

Declaring, Creating, and Initializing Array of Objects

```
Class_name[] objArray;
```

or

```
Class_name objArray[];
```

- Examples:

```
Employee[] empObjects;
```

or

```
Employee empObjects[];
```

```java
public class objectArray{
   public static void main(String args[]){
     //create array of employee object
    Employee[] obj = new Employee[2] ;

     //create & initialize actual employee objects using constructor
     obj[0] = new Employee(100,"ABC");
     obj[1] = new Employee(200,"XYZ");

     //display the employee object data
     System.out.println("Employee Object 1:");
     obj[0].showData();
     System.out.println("Employee Object 2:");
     obj[1].showData();
  }
}
```

```java
//Employee class with empId and name as attributes
    public class Employee{
      int empId;
      String name;
      //Employee class constructor
      Employee(int eid, String n){
         empId = eid;
         name = n;
      }
    public void showData(){
       System.out.print("EmpId = "+empId + "   " + " Employee Name = "+name);
       System.out.println();
     }
    }
```

Exercises 1

- (*The number of even numbers and odd numbers*) Write a program that reads ten integers, and then display the number of even numbers and odd numbers. Assume that the input ends with 0. Here is the sample run of the program.

  - Enter numbers: 1 2 3 2 1 6 3 4 5 2 3 6 8 9 9 0 (Enter)
  - The number of odd numbers: 8
  - The number of even numbers: 7

Exercises 2

$$r_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad \text{and} \quad r_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

- (*Algebra: quadratic equations*) Design a class named **QuadraticEquation** for a quadratic equation $ax2 + bx + c = 0$. The class contains:
  - Private data fields **a**, **b**, and **c** that represent three coefficients.
  - A constructor with the arguments for **a**, **b**, and **c**.
  - Three getter methods for **a**, **b**, and **c**.
  - A method named **getDiscriminant()** that returns the discriminant, which
  - is $b2 - 4ac$. The methods named **getRoot1()** and **getRoot2()** for returning two roots of the equation
- These methods are useful only if the discriminant is nonnegative. Let these methods return **0** if the discriminant is negative.

Exercises

(*Algebra: multiply two matrices*) Write a method to multiply two matrices. The header of the method is:

```
public static double[][]
    multiplyMatrix(double[][] a, double[][] b)
```

To multiply matrix a by matrix b, the number of columns in a must be the same as the number of rows in b, and the two matrices must have elements of the same or compatible types. Let c be the result of the multiplication. Assume the column size of matrix a is n. Each element c_{ij} is $a_{i1} \times b_{1j} + a_{i2} \times b_{2j} + \cdots + a_{in} \times b_{nj}$. For example, for two 3×3 matrices a and b, c is

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}$$

where $c_{ij} = a_{i1} \times b_{1j} + a_{i2} \times b_{2j} + a_{i3} \times b_{3j}$.

Write a test program that prompts the user to enter two 3×3 matrices and displays their product. Here is a sample run:

```
Enter matrix1: 1 2 3 4 5 6 7 8 9 Enter
Enter matrix2: 0 2 4 1 4.5 2.2 1.1 4.3 5.2 Enter
The multiplication of the matrices is
 1 2 3      0 2.0 4.0      5.3 23.9 24
 4 5 6   *  1 4.5 2.2  =   11.6 56.3 58.2
 7 8 9      1.1 4.3 5.2    17.9 88.7 92.4
```