# Java & JEE Training

## Day 4 – Handling Arrays in Java

## MindsMapped Consulting

## Agenda

- ✓ Autoboxing and Unboxing in Java
- ✓ Handling Arrays

# Autoboxing and Unboxing in Java

MindsMapped Consulting

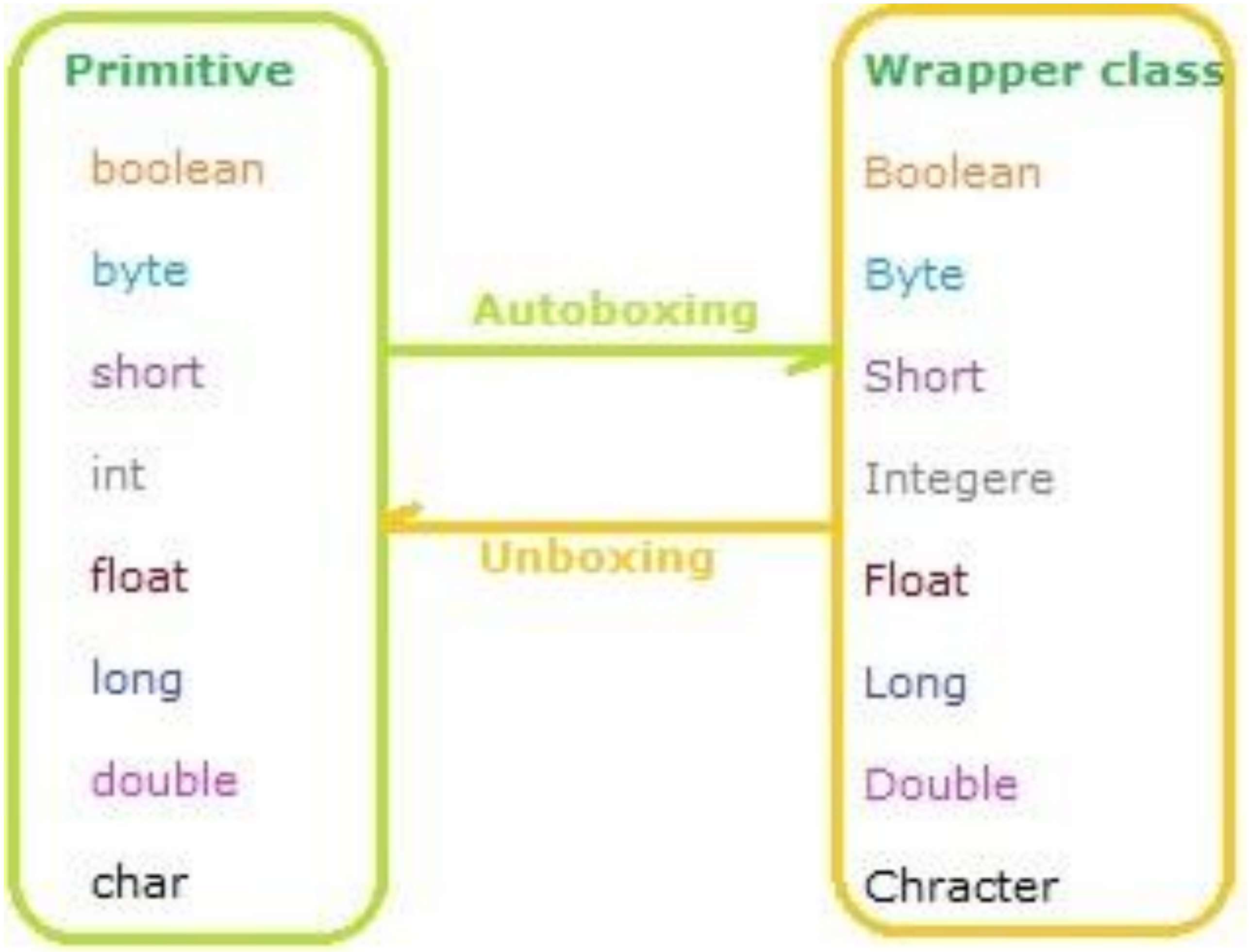
## Autoboxing and Unboxing in Java

- ✓Autoboxing is the automatic conversion that the Java compiler makes between the primitive types and their corresponding object wrapper classes. For example, converting an int to an Integer, a double to a Double, and so on.

- ✓If the conversion goes the other way, this is called unboxing.

- ✓Introduced in JDK 5

## Autoboxing and Unboxing in Java



## Agenda

- ✓ Autoboxing and Unboxing in Java
- ➢ **Handling Arrays**

## Handling Arrays

MindsMapped Consulting

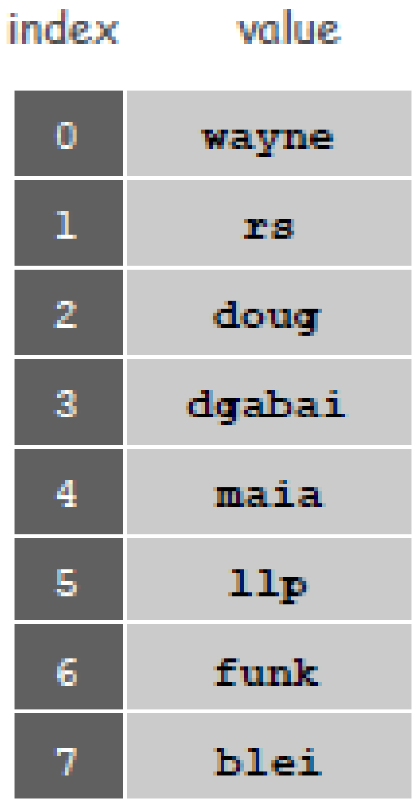
## Arrays

- We need a way to store and manipulate huge quantities of data.

- ***"An array is an <u>indexed sequence</u> of values of the <u>same type</u>."***

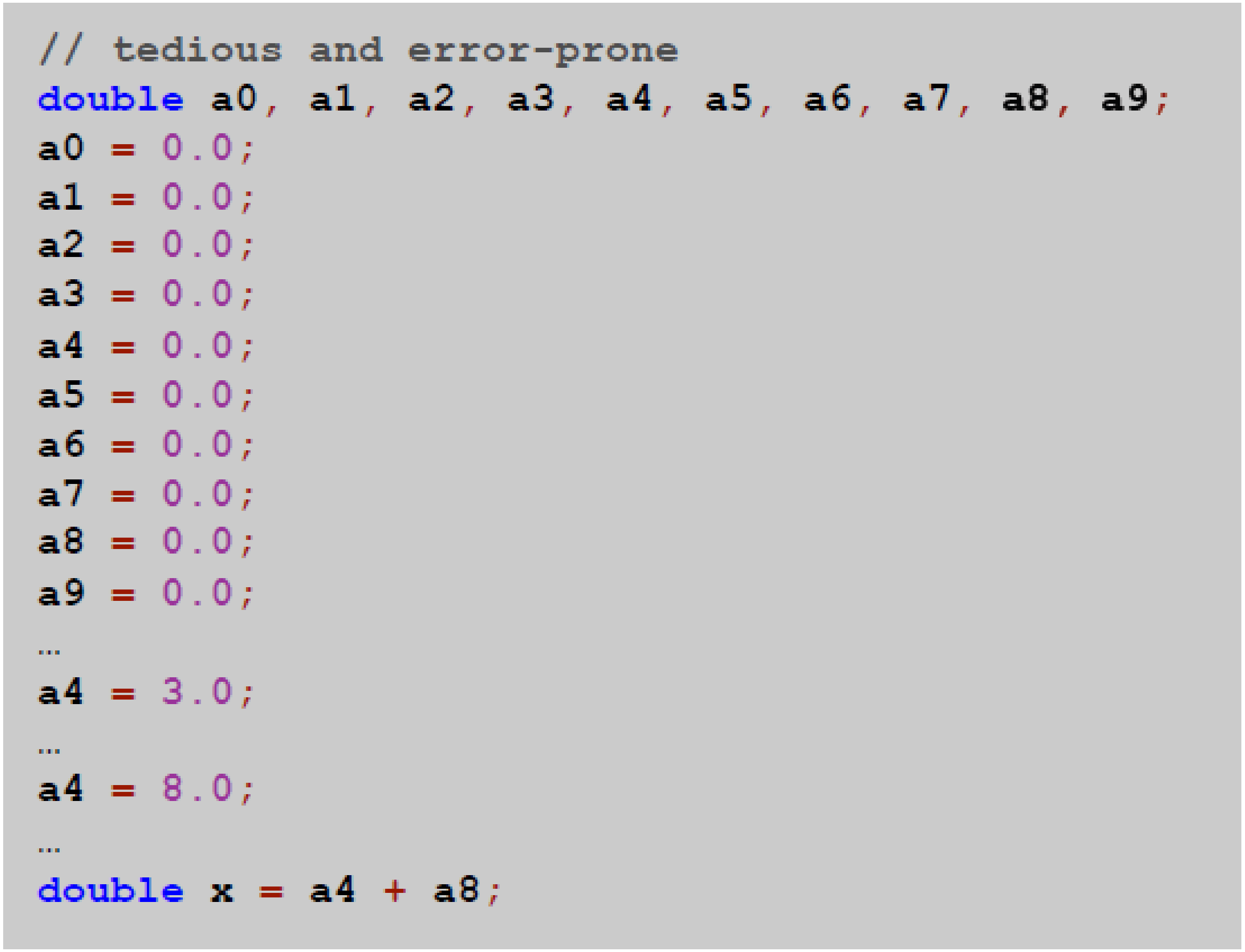
Examples.

- 52 playing cards in a deck.
- 5 thousand undergrads at Princeton.
- 1 million characters in a book.
- 10 million audio samples in an MP3 file.
- 4 billion nucleotides in a DNA strand.
- 73 billion Google queries per year.
- 50 trillion cells in the human body.



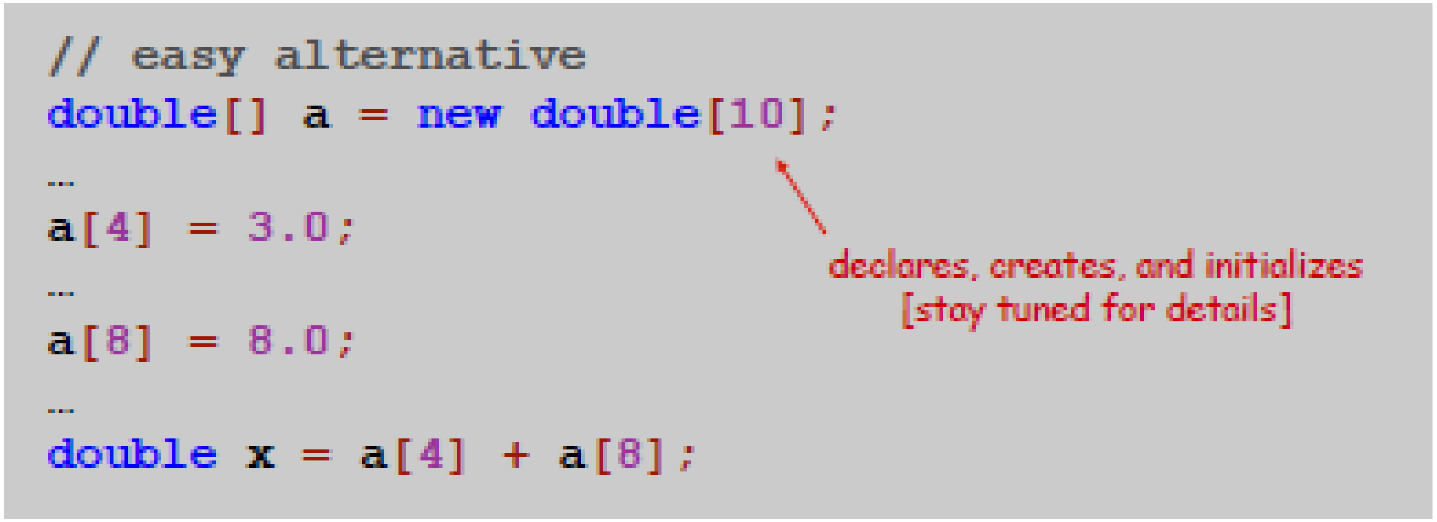
| index | value |
|---|---|
| 0 | wayne |
| 1 | rs |
| 2 | doug |
| 3 | dgabai |
| 4 | maia |
| 5 | llp |
| 6 | funk |
| 7 | blei |

## Goal: Many variables of the same type



```
// tedious and error-prone
double a0, a1, a2, a3, a4, a5, a6, a7, a8, a9;
a0 = 0.0;
a1 = 0.0;
a2 = 0.0;
a3 = 0.0;
a4 = 0.0;
a5 = 0.0;
a6 = 0.0;
a7 = 0.0;
a8 = 0.0;
a9 = 0.0;
…
a4 = 3.0;
…
a4 = 8.0;
…
double x = a4 + a8;
```

## Goal: Many variables of the same type



```
// easy alternative
double[] a = new double[10];
...
a[4] = 3.0;
...
a[8] = 8.0;
...
double x = a[4] + a[8];
```

declares, creates, and initializes
[stay tuned for details]

## Arrays in Java

Java has special language support for arrays.

- To make an array: declare, create, and initialize it.
- To access element i of array named a, use a[i].
- Array indices start at 0.

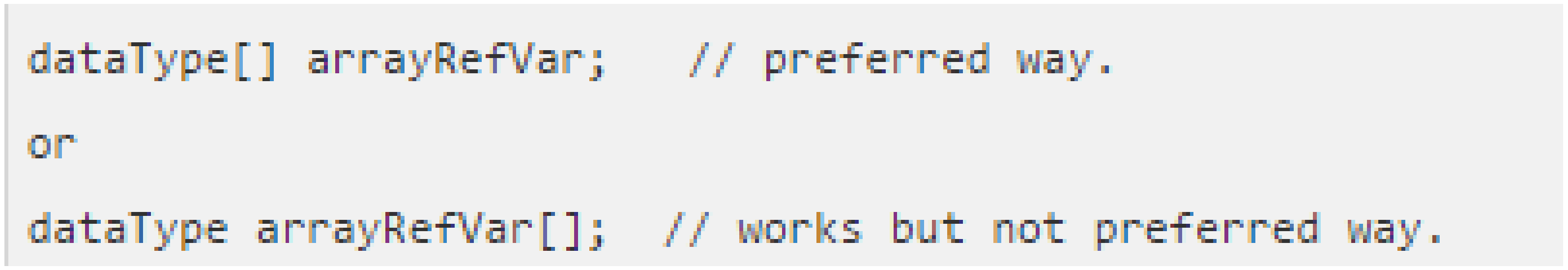
```
int N = 10;                  // size of array
double[] a;                  // declare the array
a = new double[N];           // create the array
for (int i = 0; i < N; i++)  // initialize the array
   a[i] = 0.0;               // all to 0.0
```

**Compact alternative.**

- Declare, create, and initialize in one statement.
- Default initialization: all numbers automatically set to zero.

```
int N = 10;                 // size of array
double[] a = new double[N]; // declare, create, init
```

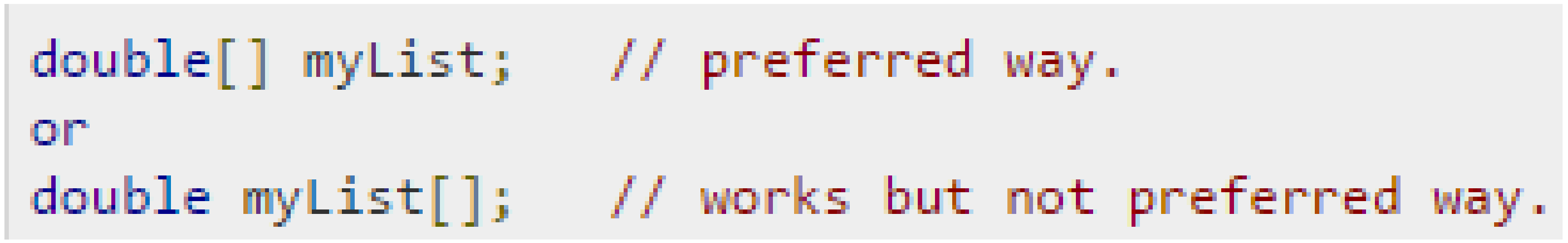
## Declaring Arrays – Two ways



```
dataType[] arrayRefVar;   // preferred way.

or

dataType arrayRefVar[];  // works but not preferred way.
```



```
double[] myList;    // preferred way.
or
double myList[];    // works but not preferred way.
```

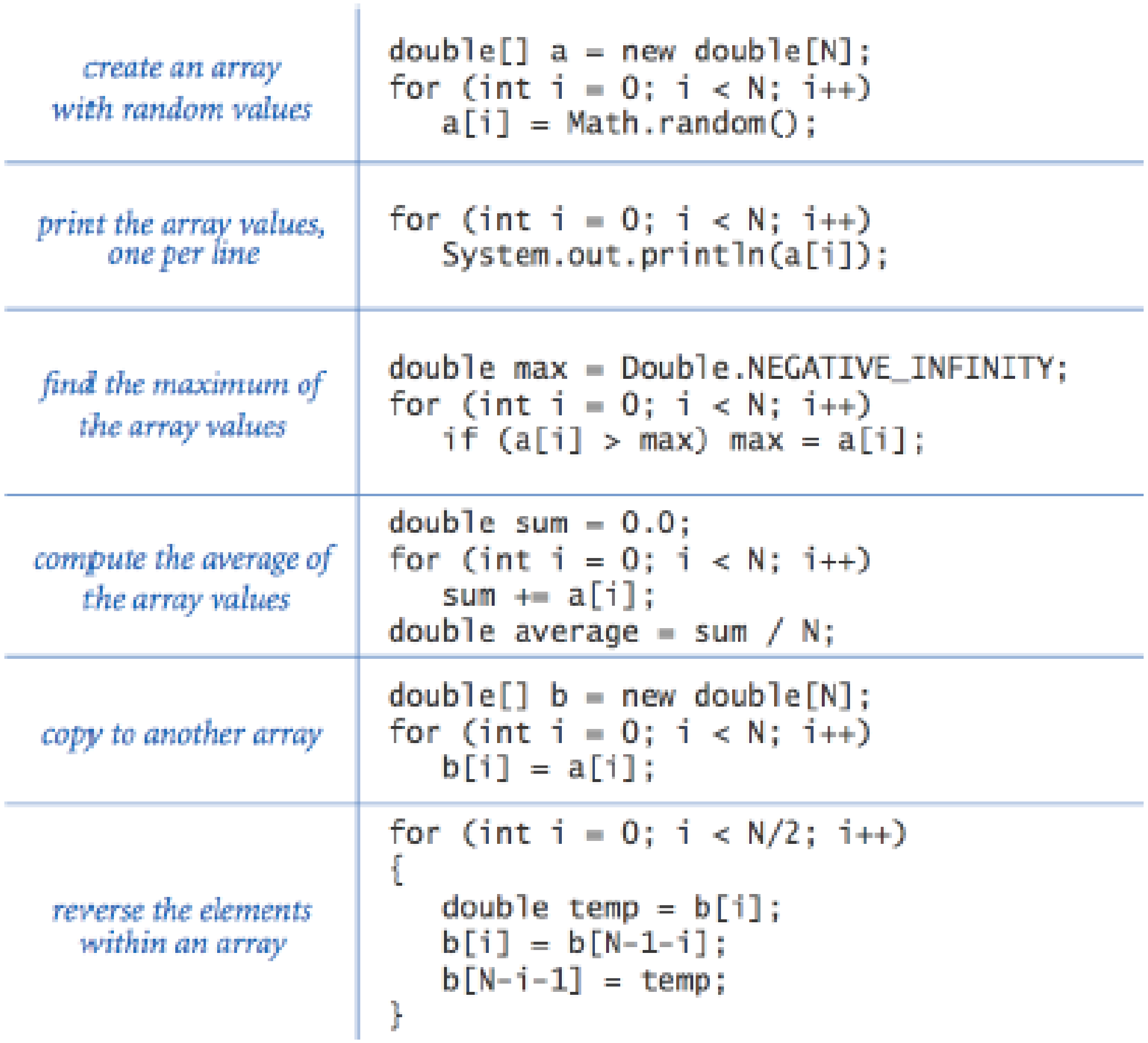
## Exercise: Vector dot product

**Dot product.** Given two vectors x[] and y[] of length N, their dot product is the sum of the products of their corresponding components.

```
double[] x = { 0.3, 0.6, 0.1 };
double[] y = { 0.5, 0.1, 0.4 };
int N = x.length;
double sum = 0.0;
for (int i = 0; i < N; i++) {
   sum = sum + x[i]*y[i];
}
```

| i | x[i] | y[i] | x[i]*y[i] | sum |
|---|---|---|---|---|
| | | | | 0 |
| 0 | .30 | .50 | .15 | .15 |
| 1 | .60 | .10 | .06 | .21 |
| 2 | .10 | .40 | .04 | .25 |
| | | | | .25 |

## Array Processing Examples

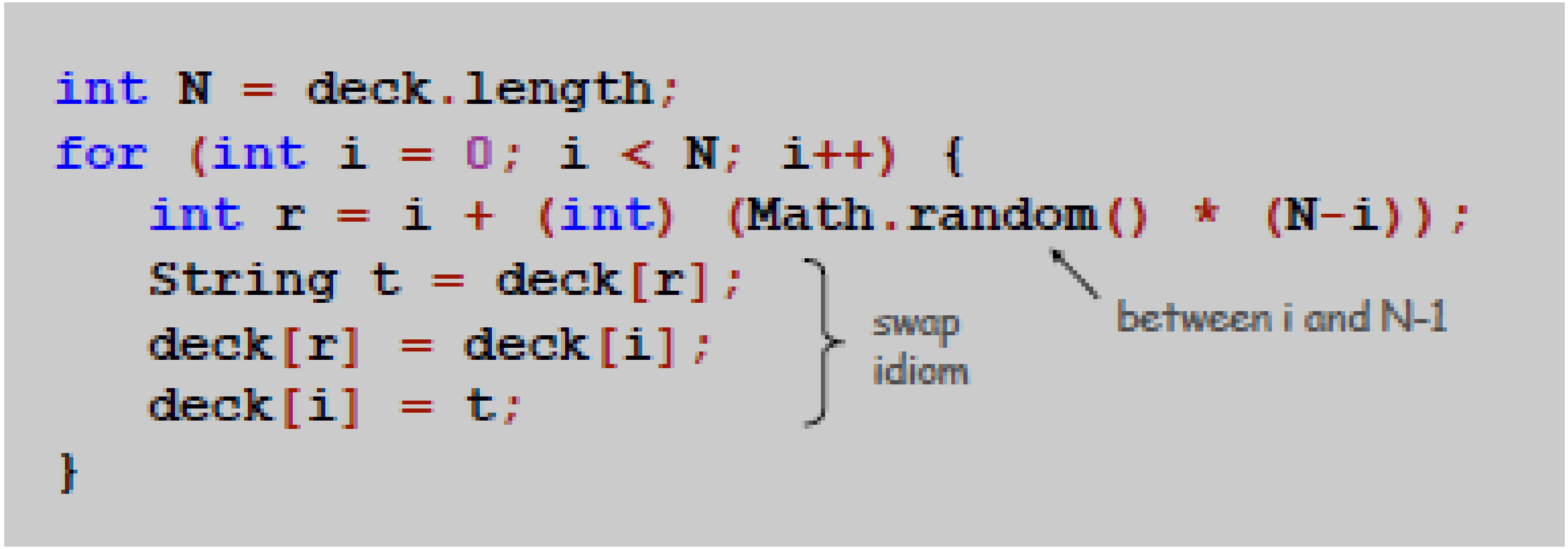


| | |
|---|---|
| *create an array with random values* | `double[] a = new double[N];`<br>`for (int i = 0; i < N; i++)`<br>`   a[i] = Math.random();` |
| *print the array values, one per line* | `for (int i = 0; i < N; i++)`<br>`   System.out.println(a[i]);` |
| *find the maximum of the array values* | `double max = Double.NEGATIVE_INFINITY;`<br>`for (int i = 0; i < N; i++)`<br>`   if (a[i] > max) max = a[i];` |
| *compute the average of the array values* | `double sum = 0.0;`<br>`for (int i = 0; i < N; i++)`<br>`   sum += a[i];`<br>`double average = sum / N;` |
| *copy to another array* | `double[] b = new double[N];`<br>`for (int i = 0; i < N; i++)`<br>`   b[i] = a[i];` |
| *reverse the elements within an array* | `for (int i = 0; i < N/2; i++)`<br>`{`<br>`   double temp = b[i];`<br>`   b[i] = b[N-1-i];`<br>`   b[N-i-1] = temp;`<br>`}` |

## Example: Card shuffling algorithm

**Goal.** Given an array, rearrange its elements in random order.

- Shuffling algorithm.
- In iteration i, pick random card from deck[i] through deck[N-1], with each card equally likely.
- Exchange it with deck[i].



```
int N = deck.length;
for (int i = 0; i < N; i++) {
    int r = i + (int) (Math.random() * (N-i));
    String t = deck[r];
    deck[r] = deck[i];
    deck[i] = t;
}
```

## Two Dimensional Arrays

**Two-dimensional arrays.**

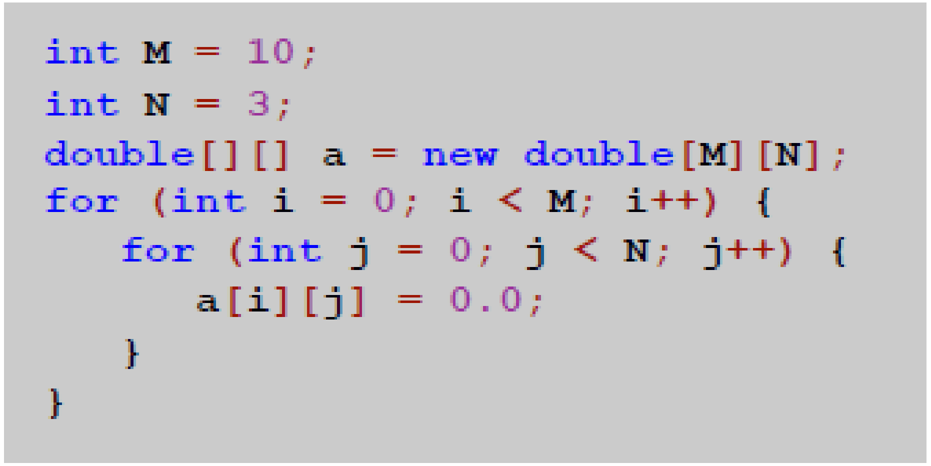
- Table of data for each experiment and outcome.
- Table of grades for each student and assignments.
- Table of grayscale values for each pixel in a 2D image.

**Mathematical abstraction**. Matrix.

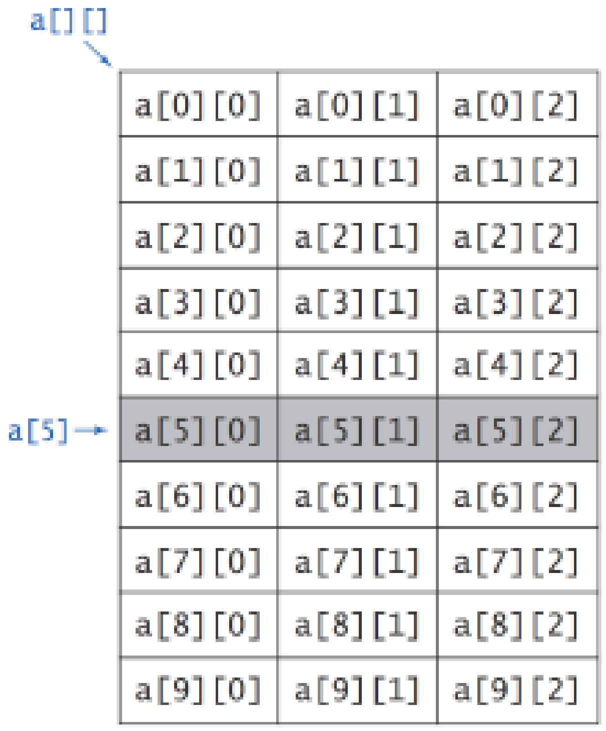
**Java abstraction**. 2D array.

## Two Dimensional Arrays in Java

- **Array access.** Use a[i][j] to access element in row i and column j.
- **Zero-based indexing**. Row and column indices start at 0.
- a[i] is the i'th row, which is a one dimensional array here.



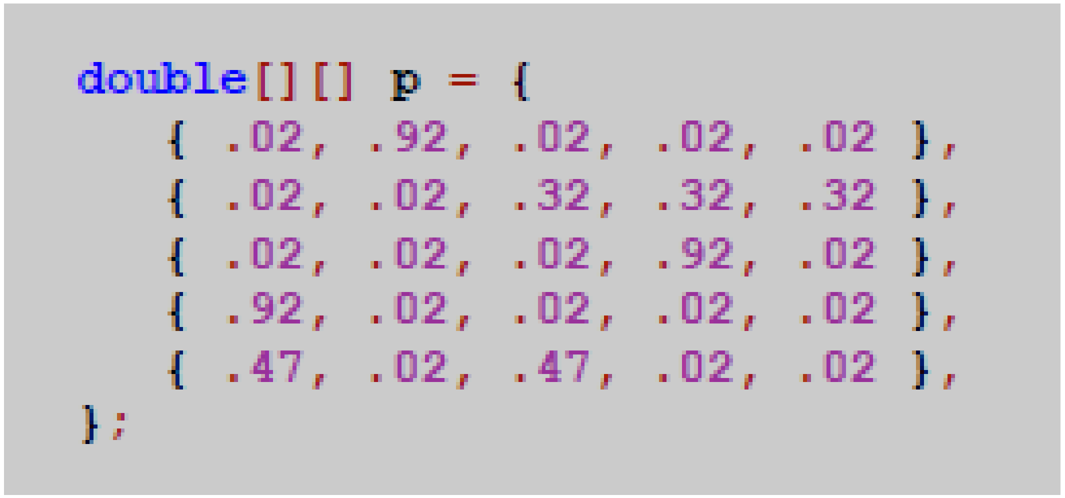
```
int M = 10;
int N = 3;
double[][] a = new double[M][N];
for (int i = 0; i < M; i++) {
    for (int j = 0; j < N; j++) {
        a[i][j] = 0.0;
    }
}
```



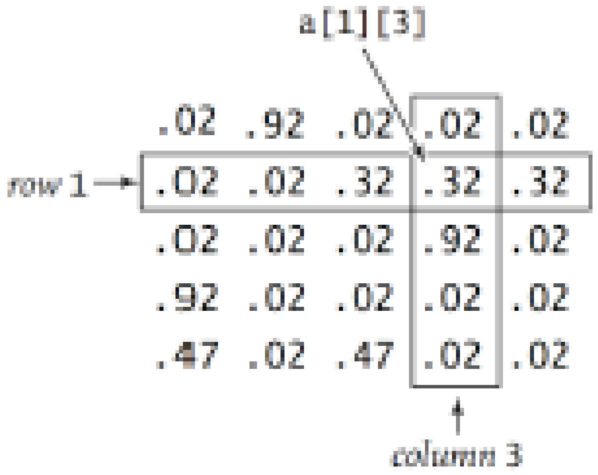
|  |  |  |
|---|---|---|
| a[0][0] | a[0][1] | a[0][2] |
| a[1][0] | a[1][1] | a[1][2] |
| a[2][0] | a[2][1] | a[2][2] |
| a[3][0] | a[3][1] | a[3][2] |
| a[4][0] | a[4][1] | a[4][2] |
| a[5][0] | a[5][1] | a[5][2] |
| a[6][0] | a[6][1] | a[6][2] |
| a[7][0] | a[7][1] | a[7][2] |
| a[8][0] | a[8][1] | a[8][2] |
| a[9][0] | a[9][1] | a[9][2] |

*A 10-by-3 array*

## Initialize 2D array : Inline initialization



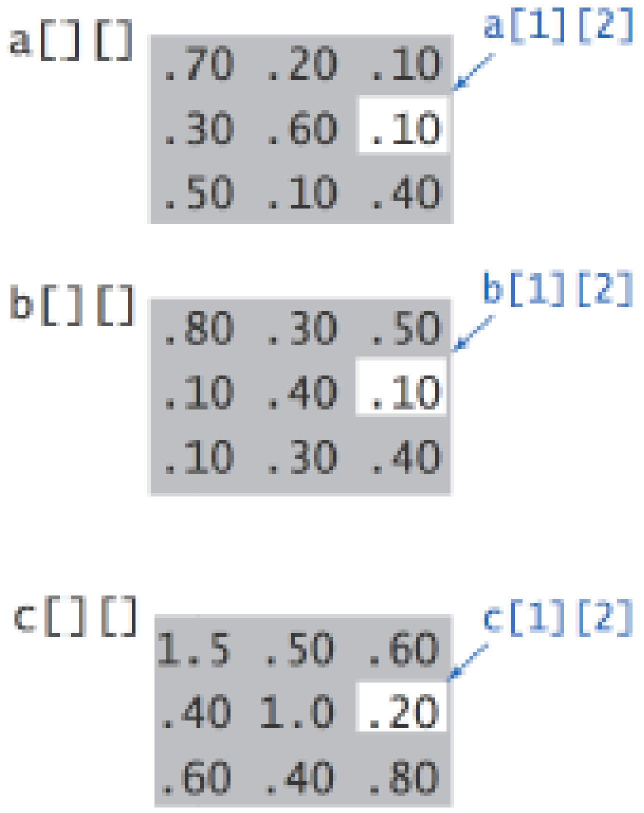
```
double[][] p = {
    { .02, .92, .02, .02, .02 },
    { .02, .02, .32, .32, .32 },
    { .02, .02, .02, .92, .02 },
    { .92, .02, .02, .02, .02 },
    { .47, .02, .47, .02, .02 },
};
```



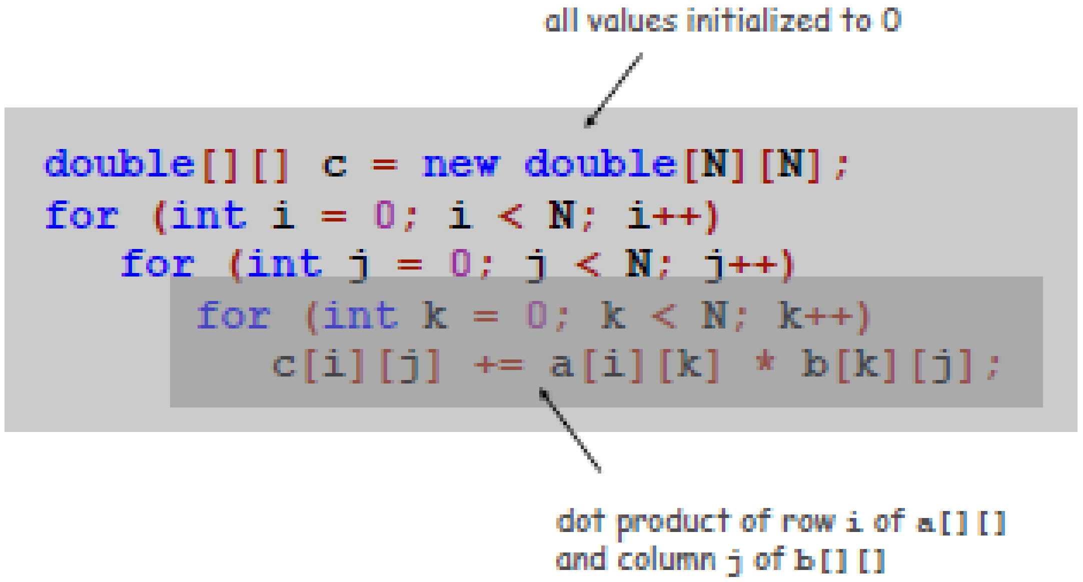
|  |  |  |  |  |
|---|---|---|---|---|
| .02 | .92 | .02 | .02 | .02 |
| .02 | .02 | .32 | .32 | .32 |
| .02 | .02 | .02 | .92 | .02 |
| .92 | .02 | .02 | .02 | .02 |
| .47 | .02 | .47 | .02 | .02 |

## Example: Matrix Addition

```
double[][] c = new double[N][N];
for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
        c[i][j] = a[i][j] + b[i][j];
```

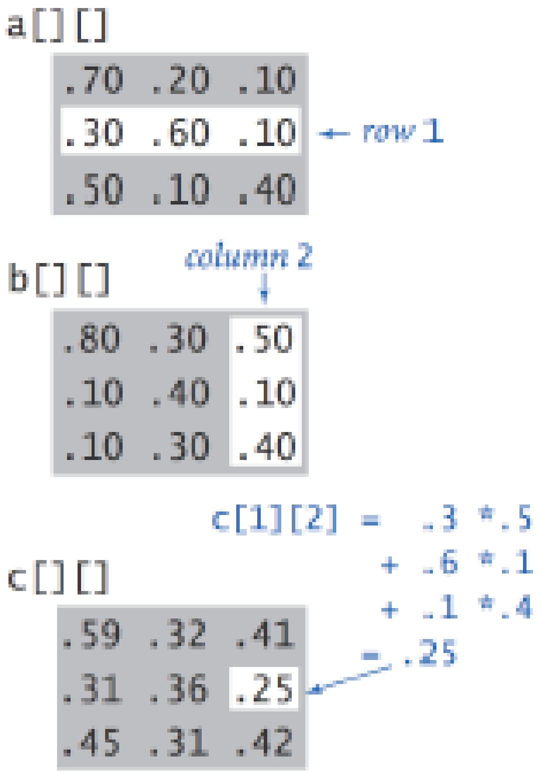


# Example: Matrix Multiplication

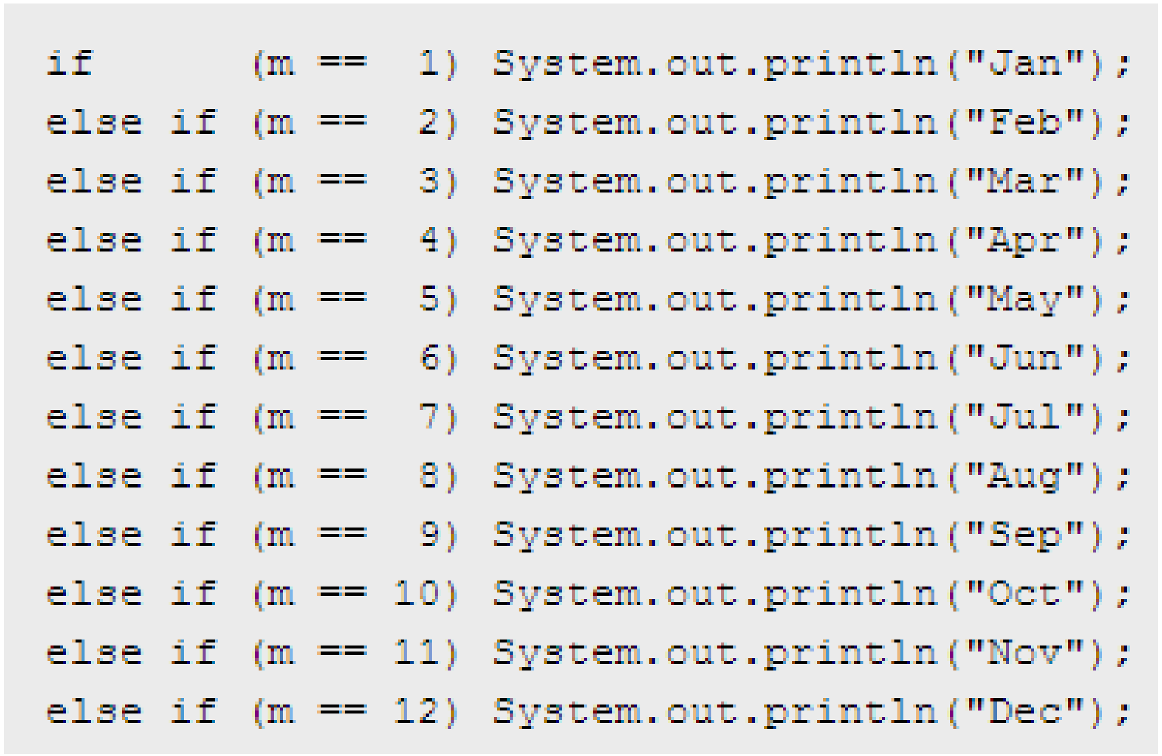


```
double[][] c = new double[N][N];
for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++)
      for (int k = 0; k < N; k++)
         c[i][j] += a[i][k] * b[k][j];
```

all values initialized to 0
dot product of row i of a[][] and column j of b[][]



## Example: Using arrays to simplify repetitive code



```
if      (m ==  1) System.out.println("Jan");
else if (m ==  2) System.out.println("Feb");
else if (m ==  3) System.out.println("Mar");
else if (m ==  4) System.out.println("Apr");
else if (m ==  5) System.out.println("May");
else if (m ==  6) System.out.println("Jun");
else if (m ==  7) System.out.println("Jul");
else if (m ==  8) System.out.println("Aug");
else if (m ==  9) System.out.println("Sep");
else if (m == 10) System.out.println("Oct");
else if (m == 11) System.out.println("Nov");
else if (m == 12) System.out.println("Dec");
```

**How to simplify writing the above code?**

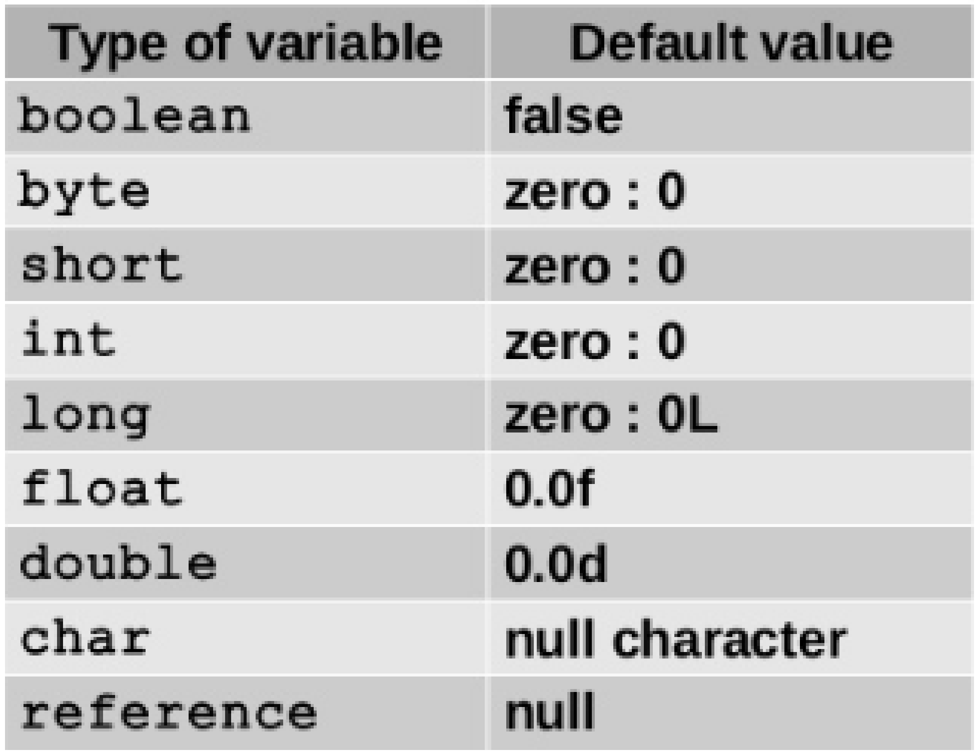
## Example: Using arrays to simplify repetitive code

```
if      (m ==  1) System.out.println("Jan");
else if (m ==  2) System.out.println("Feb");
else if (m ==  3) System.out.println("Mar");
else if (m ==  4) System.out.println("Apr");
else if (m ==  5) System.out.println("May");
else if (m ==  6) System.out.println("Jun");
else if (m ==  7) System.out.println("Jul");
else if (m ==  8) System.out.println("Aug");
else if (m ==  9) System.out.println("Sep");
else if (m == 10) System.out.println("Oct");
else if (m == 11) System.out.println("Nov");
else if (m == 12) System.out.println("Dec");
```

```
String[] MONTHS = {
    "", "Jan", "Feb", "Mar", "Apr", "May", "Jun",
    "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
};
...
System.out.println(MONTHS[m]);
```

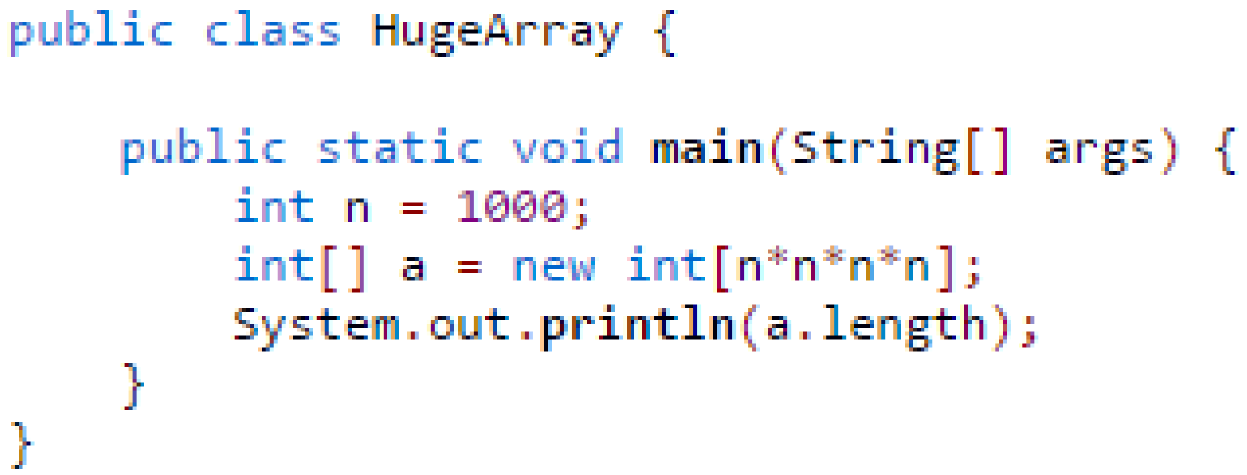
## Default Initialization: Class / instance variable & Arrays

Each class variable, instance variable, or array component is initialized with a default value when it is created:



| Type of variable | Default value |
|---|---|
| boolean | false |
| byte | zero : 0 |
| short | zero : 0 |
| int | zero : 0 |
| long | zero : 0L |
| float | 0.0f |
| double | 0.0d |
| char | null character |
| reference | null |

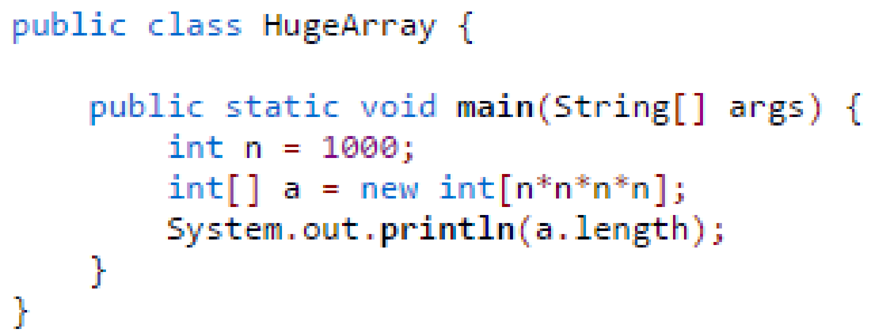
## Exercise: What is the output? Why?



```java
public class HugeArray {

    public static void main(String[] args) {
        int n = 1000;
        int[] a = new int[n*n*n*n];
        System.out.println(a.length);
    }
}
```

## Exercise: What is the output? Why?



```
public class HugeArray {

    public static void main(String[] args) {
        int n = 1000;
        int[] a = new int[n*n*n*n];
        System.out.println(a.length);
    }
}
```

Attempts to create an array of size n^4 for n = 1000.

This program compiles cleans.

When n is 1000, it leads to the following error

java.lang.NegativeArraySizeException

because 1000^4 overflows an int and results in a negative integer.

When n is 200, it leads to the following error

java.lang.OutOfMemoryError: Requested array size exceeds VM limit

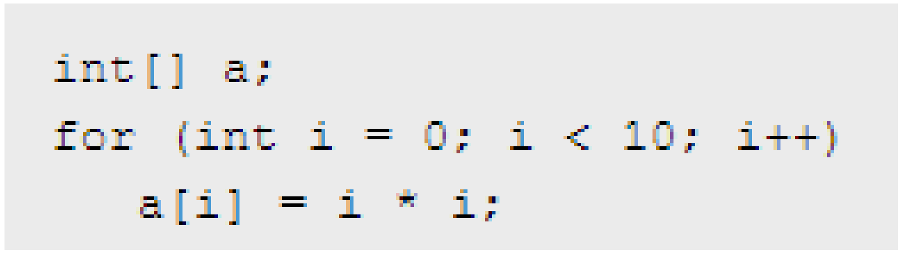
## Exercise: What is the output?

What is wrong with the following code fragment?

```
int[] a;
for (int i = 0; i < 10; i++)
   a[i] = i * i;
```

## Exercise: What is the output?

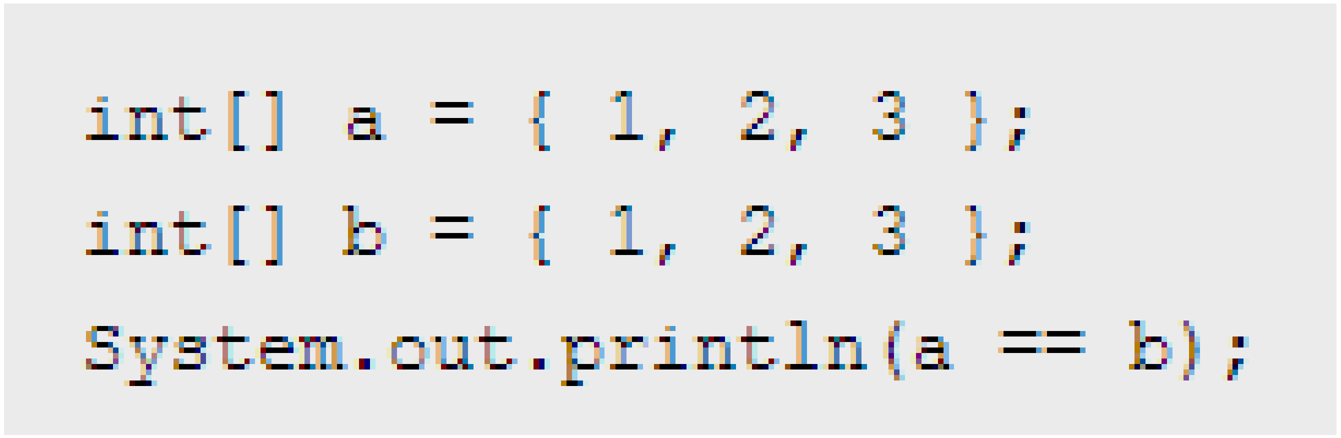
What is wrong with the following code fragment?



```
int[] a;
for (int i = 0; i < 10; i++)
    a[i] = i * i;
```

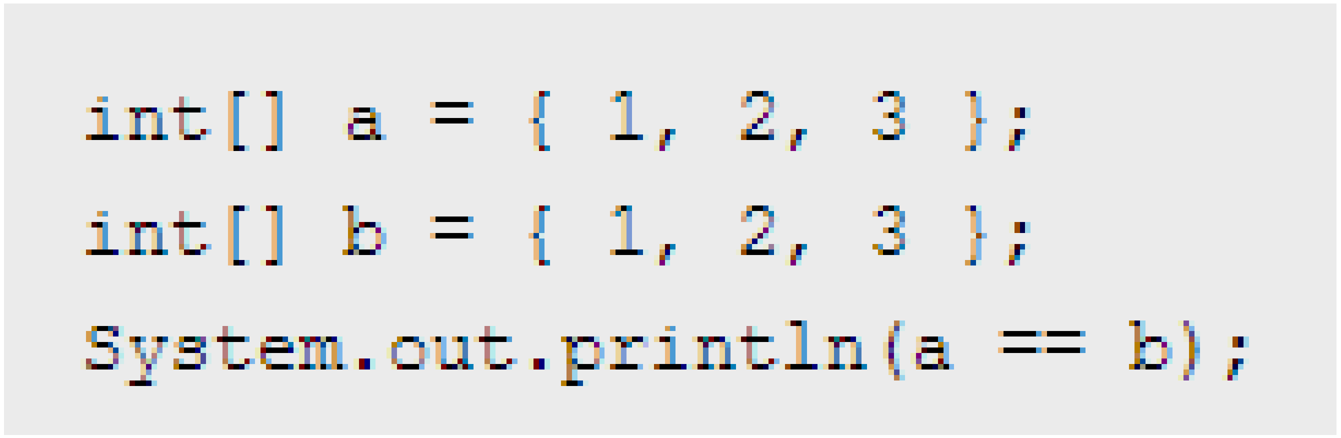
✓It does not allocate memory for a[] with new. The code results in a variable might not have been initialized compile-time error.

## Exercise: What is the output?



```
int[] a = { 1, 2, 3 };
int[] b = { 1, 2, 3 };
System.out.println(a == b);
```

## Exercise: What is the output?



```
int[] a = { 1, 2, 3 };
int[] b = { 1, 2, 3 };
System.out.println(a == b);
```

✓ It prints false. The == operator compares whether the (memory addresses of the) two arrays are identical, not whether their corresponding values are equal.

## What is the output?

- What happens when you try to compile a program with the following statement?

```
int[] a = new int[-17];
```

## What is the output?

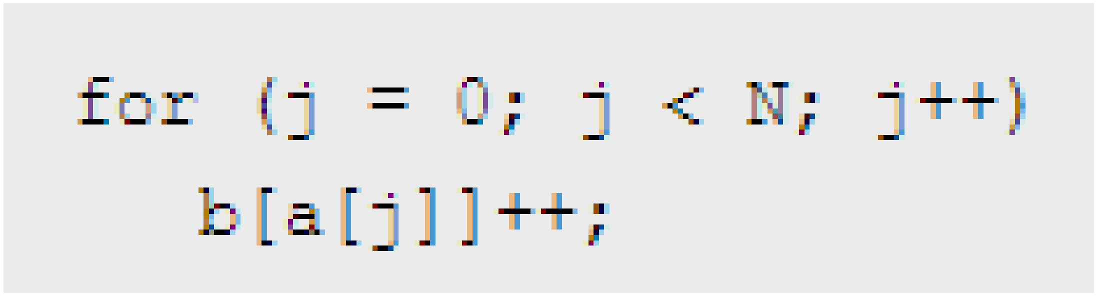
- What happens when you try to compile a program with the following statement?

```
int[] a = new int[-17];
```

✓ It compiles cleanly, but throws a java.lang.NegativeArraySizeException when you execute it.

## What is the output?

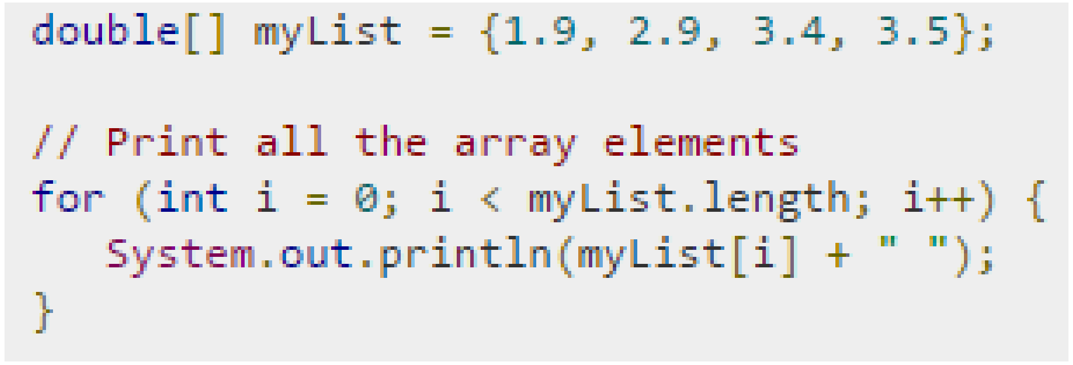
- Suppose that b[] is an array of 100 elements, with all entries initialized to 0, and that a[] is an array of N elements, each of which is an integer between 0 and 99. What is the effect of the following loop?



```
for (j = 0; j < N; j++)
    b[a[j]]++;
```

## Iterating through elements in an Array – Two ways

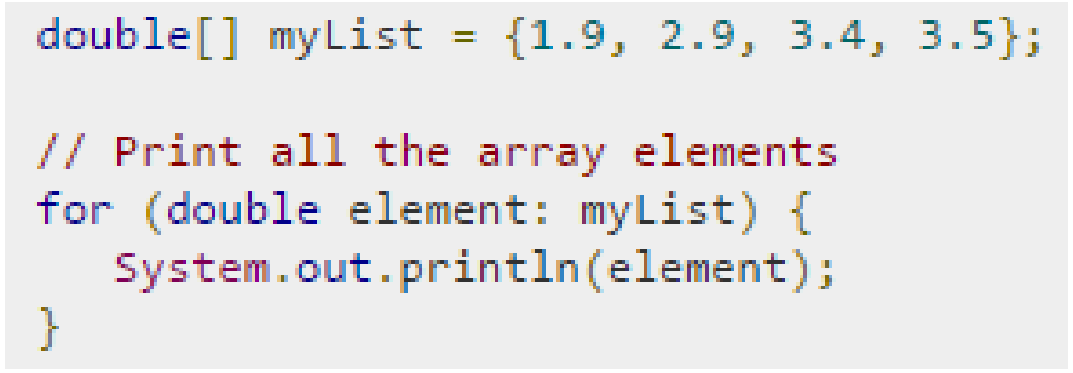
- for Loop



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

// Print all the array elements
for (int i = 0; i < myList.length; i++) {
   System.out.println(myList[i] + " ");
}
```

- foreach Loop (introduced in Java 1.5)



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

// Print all the array elements
for (double element: myList) {
   System.out.println(element);
}
```

## Arrays – Sorting and Searching

- java.util package has API for search and sort
  - Arrays.sort() –
    - Default sorting is ascending order
    - But this can be used with sorting order Collections.reverseOrder() which can be used for sorting in descending order, but can be used only with Object type and not primitive type
  - Arrays.binarySearch()

## Exercise

- Given an array of sorted elements, search for the element requested.
- Given a 2D array that is sorted column wise and row wise, write an algorithm to find if an element exists in the matrix or not. Keep the temporal complexity of the algorithm in mind.