Arrays

One-Dimensional, Multidimensional &

Command-Line Arguments

Arrays

- An array is an **indexed list** of values.
- All elements of an array must have the **same type**.
- An array is a group of like-typed variables that are referred to by a **common name**.
- A specific element in an array is accessed by its index.
- The index starts at **zero** and ends at **length-1**.
- Arrays of any type can be created

  `Example: int, double, String, etc.`

- Arrays can have **one** or **more dimensions**.

One-Dimensional Arrays

- To create an array, you first must create an **array variable** of the desired type.

```
type var-name[ ];     or
type[ ] var-name;
```

- Here, **type** declares the element type (also called the base type) of the array. The element type for the array determines what type of data the array will hold.
- For example, the following declares an array named **ar** with the type "array of int":

```
int ar[];  or
int[] ar;
```

- Actually **ar** is an array variable, no array actually exists.
- To link **ar** with an actual, physical array of integers, you must allocate one using **new** and assign it to **ar**.

One-Dimensional Arrays

- **new** is a special operator that allocates memory.

  array-var = new type [size];

- **type** specifies the type of data being allocated
- **size** specifies the number of elements in the array
- **array-var** is the array variable that is linked to the array.

  ar=new int[10];

- To use **new** to allocate an array, you must specify the **type** and **number of elements** to allocate.
- The elements in the array allocated by **new** will automatically be initialized to **zero** (for numeric types), **false** (for boolean), or **null** (for reference types).

One-Dimensional Arrays

- Obtaining an array is a two-step process.
- First, you must **declare** a variable of the desired array type.
- Second, you must **allocate** the **memory** that will hold the array, using **new**, and assign it to the array variable.
- In Java, all arrays are dynamically allocated.
- It is possible to combine the **declaration** of the array variable with the **allocation** of the array itself, as shown here:

```
type var-name[ ]= new type [size];
```

Example:

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

One-Dimensional Arrays

- Arrays can be initialized when they are declared.

  type var-name [] = { array initialize list };

- An array initializer is a list of comma-separated expressions surrounded by curly braces.
- The commas separate the values of the array elements.
- It can only be used when you declare the variable.
- The array will automatically be created large enough to hold the number of elements you specify in the array initializer.
- There is no need to use **new**.
- Example:

```
int ar[]={10,20,30,40,50,60,70,80,90,100};
```

- The array **ar** is constructed to hold 10 elements (equal to the length of the list of elements in the block), where the first element is initialized to the value of the first expression (10), the second element to the value of the second expression (20), and so on.

One-Dimensional Arrays

- Java strictly checks to make sure you do not accidentally try to store or reference values outside of the range of the array.
- The **Java run-time system** will check to be sure that all array indexes are in the correct range.
- For example, the run-time system will check the value of each index into **ar** to make sure that it is between 0 and 9 inclusive.
- If you try to access elements outside the range of the array (negative numbers or numbers greater than the length of the array), you will cause a **run-time error**.
- If the index value is less than 0, or greater than or equal to array length, an java.lang.ArrayIndexOutOfBoundsException is thrown.
- To access the elements of an array, use the [] operator:

```
var-name[index],   eg. ar[0]
```

One-Dimensional Arrays

- For example:

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

ar[0] = 10;
ar[1] = 20;
ar[2] = 30;
ar[3] = 40;
ar[4] = 50;
ar[5] = 60;
ar[6] = 70;
ar[7] = 80;
ar[8] = 90;
ar[9] = 100;
for(int i:ar)
System.out.print(i+" ");
```

- For example:

```
int  ar[]={10,20,30,40,50,60,70,80,90,100};
for(int i:ar)
System.out.print(i+" ");
```

One-Dimensional Arrays

- For example:

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

ar[0] = 10;
ar[1] = 20;
ar[2] = 30;
ar[3] = 40;
ar[4] = 50;
ar[5] = 60;
ar[6] = 70;
ar[7] = 80;
ar[8] = 90;
ar[9] = 100;
for(int i:ar)
System.out.print(i+" ");
```

- For example:

```
int  ar[]={10,20,30,40,50,60,70,80,90,100};
for(int i:ar)
System.out.print(i+" ");
```

Output:

10 20 30 40 50 60 70 80 90 100

One-Dimensional Arrays

- Each array has a **length** variable built-in that contains the length of the array.

```
int ar[];
ar = new int[10];
int size = ar.length;
int ar2[] = {1,2,3,4,5};
int size2 = ar2.length;
```

One-Dimensional Arrays

- For example:

```
int ar[];
ar = new int[10];
ar[0] = 10;
ar[1] = 20;
ar[2] = 30;
ar[3] = 40;
ar[4] = 50;
ar[5] = 60;
ar[6] = 70;
ar[7] = 80;
ar[8] = 90;
ar[9] = 100;
for(int i:ar)
System.out.print(i+" ");
```

- For example:

```
int ar[]={10,20,30,40,50,60,70,80,90,100};
for(int i=0; i<ar.length;i++)
System.out.print(ar[i]+" ");
```

Output:

10 20 30 40 50 60 70 80 90 100

Multi-dimensional Arrays

- In Java, multidimensional arrays are actually **arrays of arrays**.
- To declare a multidimensional array variable, specify each additional index using another set of square brackets.

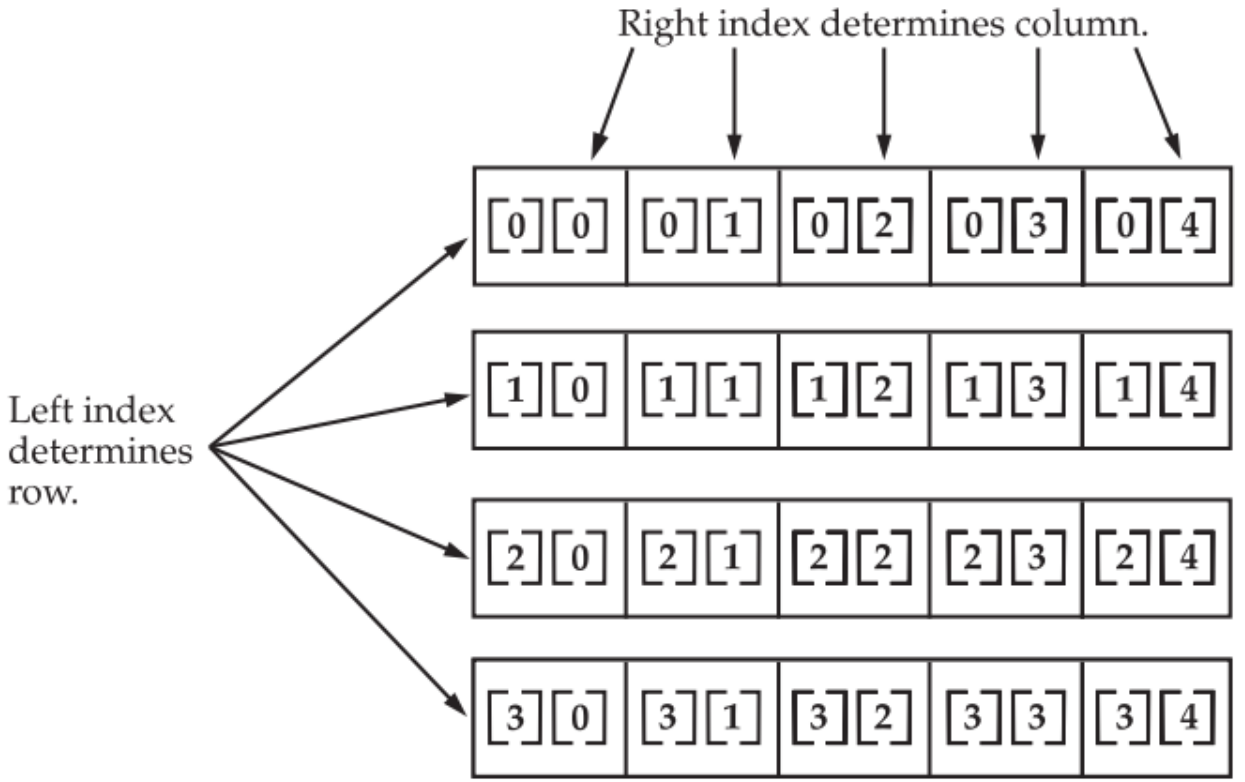
<element type>[][]...[] <array name>; or
<element type> <array name>[][]...[];

- Example:

```
int twoD[][] = new int[4][5];
```

- These declarations are all equivalent:

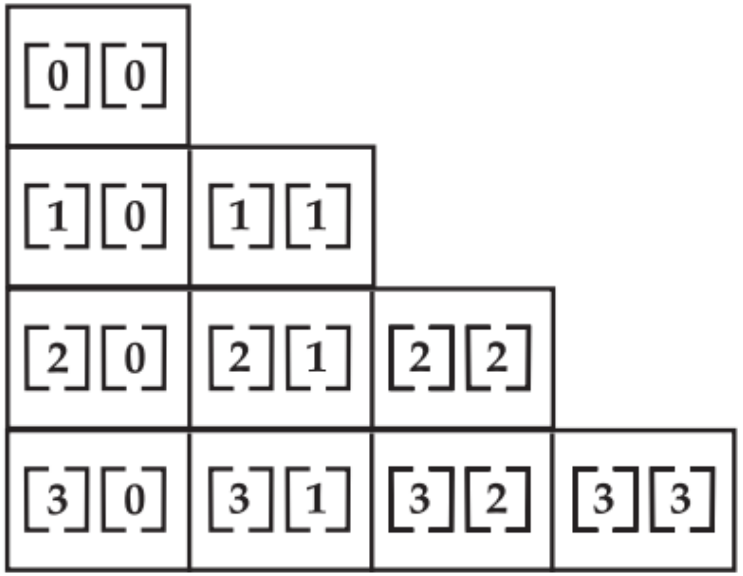
```
int[][] twoD;
int[] twoD[];
int twoD[][];
```



Multi-dimensional Arrays

- When you allocate memory for a multidimensional array, you need only specify the memory for the first (leftmost) dimension.
- You can allocate the remaining dimensions separately.
- Example:

```
int twoD[][] = new int[4][];
twoD[0] = new int[1];
twoD[1] = new int[2];
twoD[2] = new int[3];
twoD[3] = new int[4];
```



- When you allocate dimensions manually, you do not need to allocate the same number of elements for each dimension.
- As stated earlier, since multidimensional arrays are actually arrays of arrays, the length of each array is under your control.

Multi-dimensional Arrays

```
int twoD[][] = new int[4][];
twoD[0] = new int[1];
twoD[1] = new int[2];
twoD[2] = new int[3];
twoD[3] = new int[4];
int i, j, k = 0;
for(i=0; i<4; i++)
      for(j=0; j<i+1; j++) {
            twoD[i][j] = k;
            k++;
      }
for(i=0; i<4; i++) {
      for(j=0; j<i+1; j++)
            System.out.print(twoD[i][j] + " ");
      System.out.println();
}
```

Multi-dimensional Arrays

```
int twoD[][] = new int[4][];
twoD[0] = new int[1];
twoD[1] = new int[2];
twoD[2] = new int[3];
twoD[3] = new int[4];
int i, j, k = 0;
for(i=0; i<4; i++)
      for(j=0; j<i+1; j++) {
            twoD[i][j] = k;
            k++;
      }
for(i=0; i<4; i++) {
      for(j=0; j<i+1; j++)
            System.out.print(twoD[i][j] + " ");
      System.out.println();
}
```

Output:

```
0
1 2
3 4 5
6 7 8 9
```

Multi-dimensional Arrays

- Array initialization:

```
int twoD [][] = {
{0},                // 1. row
{1, 2},             // 2. row
{3, 4, 5}           // 3. row
{6, 7, 8, 9}        // 4. row
}
```

Multi-dimensional Arrays

- Multiply two Matrices:

```
int a[][]={{3,4,5},{1,2,3},{2,3,4}};
int b[][]={{1,2,1},{2,3,2},{3,4,3}};
int c[][]=new int[3][3];
for(int i=0;i<3;i++){
      for(int j=0;j<3;j++){
            c[i][j]=0;
            for(int k=0;k<3;k++) {
                  c[i][j]+=a[i][k]*b[k][j];
            }
      System.out.print(c[i][j]+" ");
      }
System.out.println();
}
```

(a11 x b11) + (a12 x b21) + (a13 x b31) (a11 x b12) + (a12 x b22) + (a13 x b32)…
(a21 x b11) + (a22 x b21) + (a23 x b31) (a21 x b12) + (a22 x b22) + (a23 x b32) ….

…

….

Multi-dimensional Arrays

- Multiply two Matrices:

```
int a[][]={{3,4,5},{1,2,3},{2,3,4}};
int b[][]={{1,2,1},{2,3,2},{3,4,3}};
int c[][]=new int[3][3];
for(int i=0;i<3;i++){
      for(int j=0;j<3;j++){
            c[i][j]=0;
            for(int k=0;k<3;k++) {
                  c[i][j]+=a[i][k]*b[k][j];
            }
      System.out.print(c[i][j]+" ");
      }
System.out.println();
}
```

Output:

26 38 26

14 20 14

20 29 20

Multi-dimensional Arrays

- Each array has a **length** variable built-in that contains the length of the array.

```
int twoD[][] = new int[10][5];

int length1 = twoD.length;

int length2 = twoD[0].length;

System.out.println(length1);

System.out.println(length2);
```

Multi-dimensional Arrays

- Each array has a **length** variable built-in that contains the length of the array.

```
int twoD[][] = new int[10][5];

int length1 = twoD.length;

int length2 = twoD[0].length;

System.out.println(length1);

System.out.println(length2);
```

Output:

10

5

Command-Line Arguments

- Sometimes you will want to pass information into a program when you run it. This is accomplished by passing command-line arguments to main( ).
- A command-line argument is the information that directly follows the program's name on the command line when it is executed.
- To access the command-line arguments inside a Java program is quite easy. They are stored as strings in a String array passed to the **args** parameter of main( ).
- The first command-line argument is stored at args[0], the second at args[1], and so on.

Command-Line Arguments

- Example:

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

- Executing this program:

  java abc this is my first program

Command-Line Arguments

- Example:

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

- Executing this program:

  java abc this is my first program

Output:

args[0]: this
args[1]: is
args[2]: my
args[3]: first
args[4]: program