Structured Data in Java
Arrays

Introduction to Arrays

- Arrays give us the ability to:
  - Store a (potentially large) collection of homogeneous data
  - Have direct access to any one element in the collection by its position

- An array is a special kind of object
  - Only has a small number of methods
  - Easiest to think of it as a collection of variables all of the same type
  - The position, or index, of an element uses zero-based indexing just like the characters in a string

Introduction to Arrays

- If we create an array called `score` that is an array of five elements, it is convenient to think of it as a collection of five variables:

`score[0]`, `score[1]`, `score[2]`, `score[3]`, and `score[4]`

- These five variables could be used anywhere a normal variable could be used:
  - We can access its current value
  - We can assign it a new value
  - We can pass it as a parameter to a method
  - Etc.

Array Terminology

- An *array* is the collection of values
- Any one value is an *array element*
- The position of an array element is specified by an *index*
- The kind of values stored is known as the *base type*
- The number of elements of the array is its *size* or *length*

Array Declaration

- Arrays are an object type
  - We must declare the variable *and* create the object
- Syntax:

```
type[] array_name = new type[length];
```

- Example:

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

- The length must be a non-negative integer value
  - Could be a compile-time constant or a run-time value
  - Once an array is created, its size cannot be changed

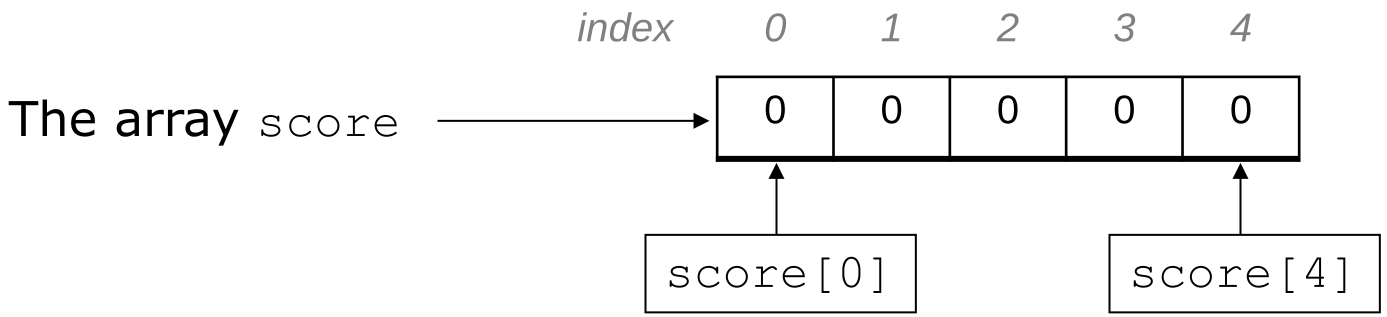
Array Declaration

- When created, each array element gets initialized to a "zero-equivalent" value

Type	**Default value**
`int`	`0`
`double`	`0.0`
`boolean`	`false`
any object type	`null` (no object yet)

Array Declaration

- We usually draw an array as a row or column of boxes
  - Example: the array `score` of five integers



Square Brackets with Arrays

- Square brackets are used in several places:
  1. When declaring an array
     ```
     int[] score;
     ```
  2. When creating an array object
     ```
     score = new int[5];
     ```
  3. When accessing an array element
     ```
     score[0] = 12;
     ```

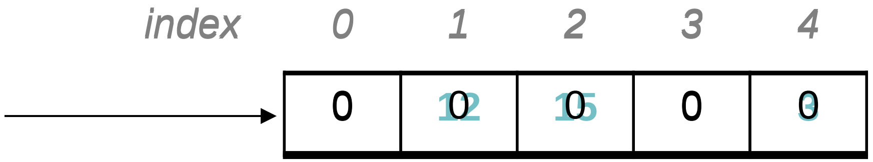
Accessing Array Elements

```
array_name[index]                 // to access
array_name[index] = value;        // to modify
```

– Example:

```
int[] score = new int[5];
number[1] = 12;
number[2] = 15;
if (number[1] < number[2]) {
    number[4] = number[2] - number[1];
}
```

The array `score`



Out-of-bounds

- Valid indices are between zero and the array's length - 1
  - Using an invalid index will result in an exception being thrown: `ArrayIndexOutOfBoundsException`
- Example:

```
int[] score = new int[5];
out.println(score[0]);          // okay
int i = 4;
out.println(score[i]);          // okay
out.println(score[-2]);         // throws exception
i = 5;
out.println(score[i]);          // throws exception
```

Array Processing

- It is easy to process all elements of an array with a `for` loop

```
for (int i = 0; i < 5; i++) {
    out.println(score[i]);
}
```

- Or to assign a new value to each element

```
for (int i = 0; i < 5; i++) {
    score[i] = 10 - i;
}
```

index	*0*	*1*	*2*	*3*	*4*
value	10	9	8	7	6

The Instance Variable `length`

- An array is an object, and that object has a field called `length` that stores the number of elements in the array
  - You access it using dot notation on the array name:
    name`.length`
  - It does not use parentheses like a String's `.length()`
    - Since it is a data field and not a method
  - It is read-only; you cannot change it
  - Using **name**`.length` typically produces cleaner code than using an integer literal

The Instance Variable `length`

- Here is an example of a `for` loop that uses the array's `length` field:

```
for (int i = 0; i < score.length; i++) {
    out.println(score[i]);
}
```

- This code will continue to work even if the array `score` is changed to hold more or fewer elements