Chapter 5: Arrays

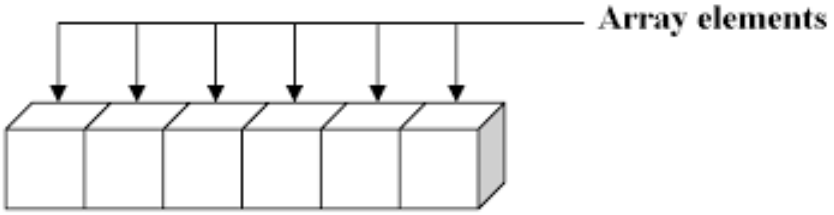
Yepang LIU

liuyp1@sustech.edu.cn

Objectives

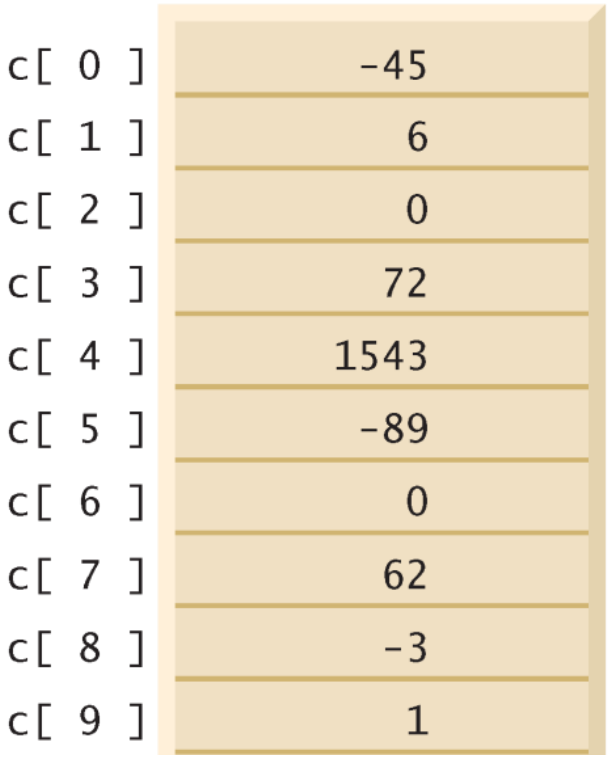
- Use arrays (数组) to store data in and retrieve data from lists and tables of values
- Declare arrays, initialize arrays and refer to individual elements of arrays
- Use the enhanced `for` statement to iterate through arrays
- Declare and manipulate multidimensional arrays

Arrays



- **Data structure (数据结构):** a data organization, management and storage format that enables efficient access and modification
- An **array** (a widely-used data structure) is **a group of elements** containing values of **the same type**.
- Arrays are objects, so they're considered reference types (we will talk about this more later)
- Array elements can be either primitive types or reference types.

Referring to Array Elements



array-access expression

c[5] refers to the 6th element

c is the reference to the array (or name of the array for simplicity)

5 is the position number of the element (index or subscript)

Referring to Array Elements

- The first element in every array has index zero.
- An index must be a nonnegative integer.
- A program can use an expression as an index (`c[ 1 + a ]`)
- The highest index in an array is *the number of elements - 1*.
- Array names follow the same conventions as other variable names (Lower Camel Case)
- Array-access expressions can be used on the left side of an assignment to place a new value into an array element (`c[1] = 2`)

Array Length

- Every array object knows its own length and stores it in a `length` instance variable (`c.length`)
- Even though the `length` instance variable of an array is `public`, it cannot be changed because it's a `final` variable (the keyword `final` creates constants).

Declaring and Creating Arrays

- Like other objects (recall the usage of `Scanner`), arrays are created with the keyword `new`.
- To create an array, you specify the type of the array elements and the number of elements as part of an **array-creation expression**:
  - `int[] c = new int[ 12 ];`
  - Returns a reference (representing the memory address of the array) that can be stored in an array variable.

Declaring and Creating Arrays

```
int[] c = new int[ 12 ];
```

- The square brackets following the type `int` indicate that the variable `c` will refer to an array
- When type of the array and the square brackets are combined at the beginning of the declaration, all the identifiers in the declaration are array variables.

```
int[] a, b = new int[10];
System.out.println(b.length);
```

Declaring and Creating Arrays

- A program can declare arrays of any type.
- Every element of a primitive-type array contains a value of the array's declared element type.
  - `int[] c = new int[ 12 ];`
- Similarly, in an array of a reference type, every element is a reference to an object of the array's declared element type.
  - `Car[] cars = new Car[ 3 ];`

Default Initialization

```
public class InitArray {
  public static void main(String[] args) {
    int[] array; // declare an array
    array = new int[10]; // create the array object
    System.out.printf("%s%8s\n", "Index", "Value");
    // output each array element's value
    for(int counter = 0; counter < array.length; counter++) {
      System.out.printf("%3d%8d\n", counter, array[counter]);
    }
  }
}
```

Default Initialization

```
public class InitArray {
  public static void main(String[] args) {
    int[] array; // declare an array
    array = new int[10]; // create the array object
    System.out.printf("%s%8s\n", "Index", "Value");
    // output each array element's value
    for(int counter = 0; counter < array.length; counter++) {
      System.out.printf("%3d%8d\n", counter, array[counter]);
    }
  }
}
```

Default Initialization

```
public class InitArray {
  public static void main(String[] args) {
    int[] array; // declare an array
    array = new int[10]; // create the array object
    System.out.printf("%s%8s\n", "Index", "Value");
    // output each array element's value
    for(int counter = 0; counter < array.length; counter++) {
      System.out.printf("%3d%8d\n", counter, array[counter]);
    }
  }
}
```

Be careful with array index, make sure it is within **[0, array.length – 1]**

Otherwise: `java.lang.ArrayIndexOutOfBoundsException`

Execution Result

Index	Value
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0

The `int` elements by default get the value of 0

Array Initialization

- You can create an array and initialize its elements with an array initializer—a comma-separated list of expressions enclosed in braces.

- `int[] n = { 10, 20, 30, 40, 50 };`
  - Compiler counts the number of values in the list to determine the size of the array, then sets up the appropriate `new` operation "behind the scenes".
  - Element `n[0]` is initialized to `10`, `n[1]` is initialized to `20`, and so on.

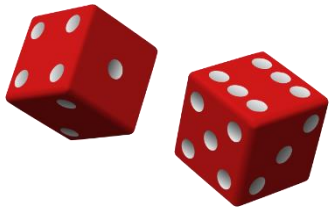
Initializing Elements One by One

```
public class InitArray2 {
  public static void main(String[] args) {
    int[] array = new int[10];
    //calculate value for each array element
    for(int counter = 0; counter < array.length; counter++) {
      array[counter] = 2 + 2 * counter;
    }
    System.out.printf("%s%8s\n", "Index", "Value");
    // output each array element's value
    for(int counter = 0; counter < array.length; counter++) {
      System.out.printf("%3d%8d\n", counter, array[counter]);
    }
  }
}
```

Execution Result

Index	Value
0	2
1	4
2	6
3	8
4	10
5	12
6	14
7	16
8	18
9	20

A Dice-Rolling Program



- Suppose we want to roll a dice 6000 times and count the frequency of each side
- We can use separate counters as below
  - `int faceOneFreq, faceTwoFreq, …`
- Now we have learned arrays. Is there a better design?

```
import java.util.Random;
public class DiceRolling {
  public static void main(String[] args) {
    Random generator = new Random();
    int[] frequency = new int[7];
    // roll 6000 times; use dice value as frequency index
    for(int roll = 1; roll <= 6000; roll++) {
      frequency[1 + generator.nextInt(6)]++;
    }
    System.out.printf("%s%10s\n", "Face", "Frequency");
    // output the frequency of each face
    for(int face = 1; face < frequency.length; face++) {
      System.out.printf("%4d%10d\n", face, frequency[face]);
    }
  }
}
```

Use an array to track frequency

```java
import java.util.Random;
public class DiceRolling {
  public static void main(String[] args) {
    Random generator = new Random();
    int[] frequency = new int[7];
    // roll 6000 times; use dice value as frequency index
    for(int roll = 1; roll <= 6000; roll++) {
      frequency[1 + generator.nextInt(6)]++;
    }
    System.out.printf("%s%10s\n", "Face", "Frequency");
    // output the frequency of each face
    for(int face = 1; face < frequency.length; face++) {
      System.out.printf("%4d%10d\n", face, frequency[face]);
    }
  }
}
```

`nextInt(6)` generates [0, 5]

```java
import java.util.Random;
public class DiceRolling {
  public static void main(String[] args) {
    Random generator = new Random();
    int[] frequency = new int[7];
    // roll 6000 times; use dice value as frequency index
    for(int roll = 1; roll <= 6000; roll++) {
      frequency[1 + generator.nextInt(6)]++;
    }
    System.out.printf("%s%10s\n", "Face", "Frequency");
    // output the frequency of each face
    for(int face = 1; face < frequency.length; face++) {
      System.out.printf("%4d%10d\n", face, frequency[face]);
    }
  }
}
```

Execution Result

Face	Frequency
1	1016
2	991
3	981
4	1011
5	988
6	1013

Enhanced for Statement

```
for ( parameter : arrayName ) {
   statement(s)
}
```

```
for ( int num : numbers ) {
   total += num;
}
```

- Iterates through the elements of an array without using a counter, thus avoiding the possibility of "stepping outside" the array.
  - *parameter* has a type and an identifier
  - *arrayName* is the array through which to iterate.
  - Parameter type must be consistent with the type of the elements in the array.

Enhanced for Statement

- Simple syntax compared to the normal `for` statement

```
for ( int num : numbers ) {
   // statements using num
}
```

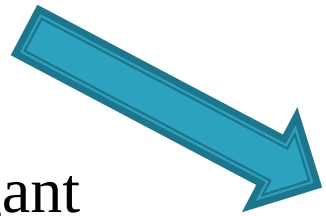
Semantically equivalent

```
for ( int i = 0; i < numbers.length; i++ ) {
   int num = numbers[i];
   // statements using num
}
```

Enhanced for Statement

- Often used to replace counter-controlled `for` statement when the code requires access only to element values.

```
for ( int i = 0; i < numbers.length; i++ ) {
   total += numbers[i];
}
```



Simpler and elegant ☺

```
for ( int num : numbers ) {
   total += num;
}
```

Enhanced for Statement

- Cannot be used to modify element values

```
for ( int num : numbers ) {
   num = 0;
}
```

Can this change the array element values?

```
for ( int i = 0; i < numbers.length; i++ ) {
   int num = numbers[i];
   num = 0;
}
```

Local variable num stores a copy of the array element value

Enhanced for Statement

- Cannot be used to modify element values

```
for ( int num : numbers ) {
   num = 0;
}
```

Can this change the array element values?

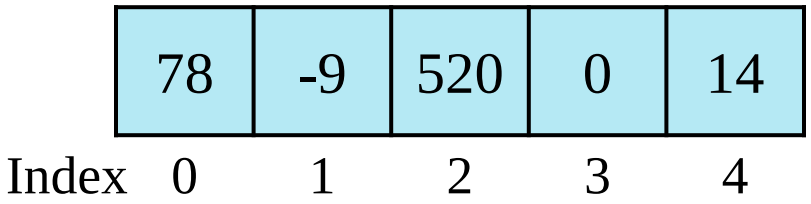
No! Only change the value of num

```
for ( int i = 0; i < numbers.length; i++ ) {
   int num = numbers[i];
   num = 0;
}
```

Local variable num stores a copy of the array element value

Two-Dimensional Arrays

- Arrays that we have considered up to now are one-dimensional arrays: a single line of elements.



Example: an array of five random numbers

Two-Dimensional Arrays

- Data in real life often come in the form of a table

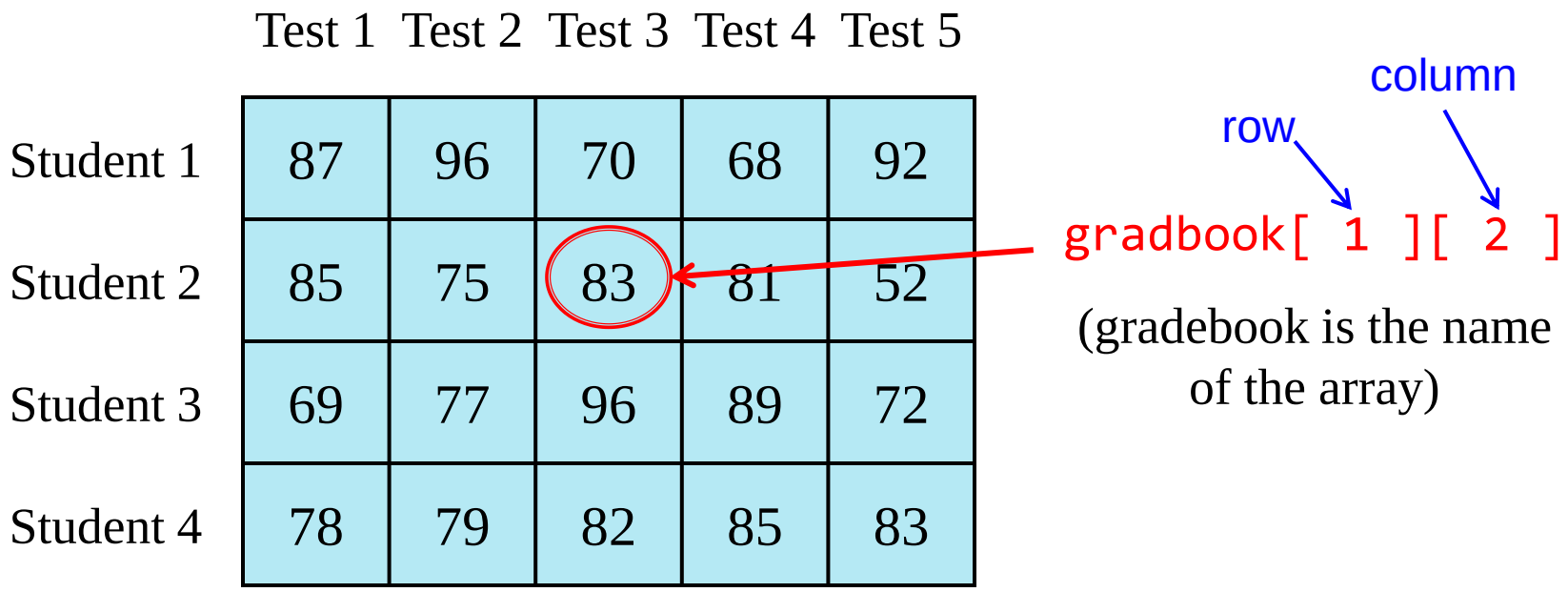
	Test 1	Test 2	Test 3	Test 4	Test 5
Student 1	87	96	70	68	92
Student 2	85	75	83	81	52
Student 3	69	77	96	89	72
Student 4	78	79	82	85	83

Example:
a gradebook

The table can be represented using a two-dimensional array in Java

Two-Dimensional (2D) Arrays

- 2D arrays are indexed by two subscripts: one for the row number, the other for the column number



2D Array Details (Similar to 1D Array)

- Similar to 1D array, each element in a 2D array should be of the same type: either primitive type or reference type
- Array access expression (subscripted variables) can be used just like a normal variable: `gradebook[1][2] = 77;`
- Array indices (subscripts) must be of type `int`, can be a literal, a variable, or an expression: `gradebook[1][j], gradebook[i+1][j+1]`
- If an array element does not exist, JVM will throw an exception `ArrayIndexOutOfBoundException`

Declaring and Creating 2D Arrays

```
int[][] gradebook;
```

- Declares a variable that references a 2D array of `int`

```
gradebook = new int[50][6];
```

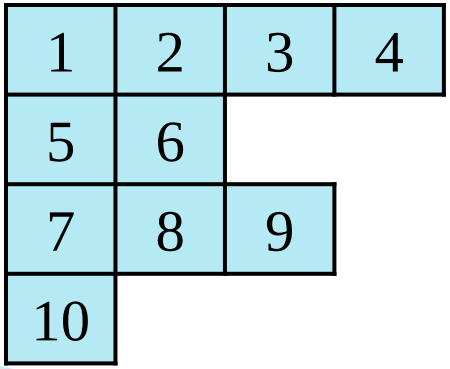
- Creates a 2D array **(50-by-6 array)** with **50 rows** (for 50 students) and **6 columns** (for 6 tests) and assign the reference to the new array to the variable `gradebook`

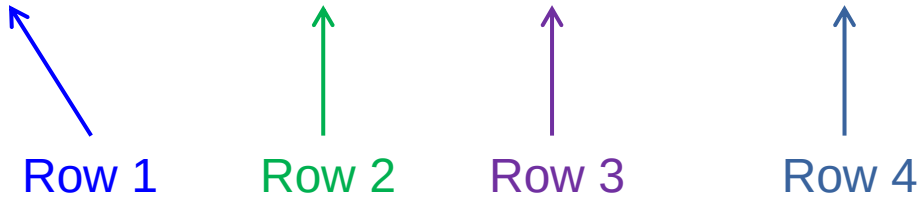
- Shortcut: `int[][] gradebook = new int[50][6];`

Array Initialization

- Similar to 1D array, we can create a 2D array and initialize its elements with nested array initializers as follows
  - `int[][] a = { { 1, 2 }, { 3, 4 } };`

- In 2D arrays, rows can have different lengths (ragged arrays)
  - `int[][] a = {{1, 2, 3, 4}, {5, 6}, {7, 8, 9}, {10}};`



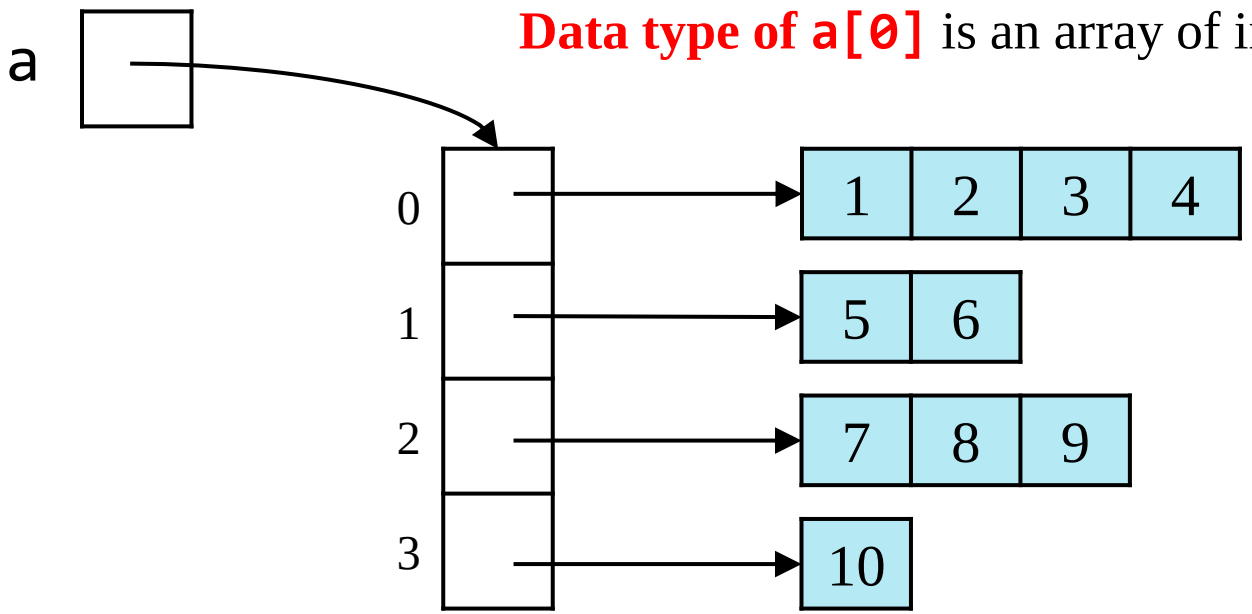


Note that the compiler will **"smartly"** determine the number of rows and columns

Under the Hood

- A 2D array is a 1D array of (references to) 1D arrays

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



Under the Hood

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

- What is the value of `a[0]`?
  - Answer: The reference (memory address) to the 1D array {1, 2, 3, 4}
- What is the value of `a.length`?
  - Answer: 4, the number of rows
- What the value of `a[1].length`?
  - Answer: 2, the second row only has 2 columns

Declaring and Creating 2D Arrays

- Since a 2D array is a 1D array of (references to) 1D arrays, a 2D array in which each row has a different number of columns can also be created as follows:

```
int[][] b = new int[ 2 ][ ];   // create 2 rows
b[ 0 ] = new int[ 5 ]; // create 5 columns for row 0
b[ 1 ] = new int[ 3 ]; // create 3 columns for row 1
```

Displaying Element values

```
public static void main(String[] args) {
    int[][] a = {{1, 2, 3, 4}, {5, 6}, {7, 8, 9}, {10}};
    // loop through rows
    for(int row = 0; row < a.length; row++) {
        // loop through columns
        for(int column = 0; column < a[row].length; column++) {
            System.out.printf("%d ", a[row][column]);
        }
        System.out.println();
    }
}
```

```
1 2 3 4
5 6
7 8 9
10
```

Computing Average Scores

```
public static void main(String[] args) {
    int[][] gradebook = {
        {87, 96, 70, 68, 92},
        {85, 75, 83, 81, 52},
        {69, 77, 96, 89, 72},
        {78, 79, 82, 85, 83}
    };
    for(int[] grades : gradebook) {
        int sum = 0;
        for(int grade : grades) {
            sum += grade;
        }
        System.out.printf("%.1f\n", ((double) sum)/grades.length);
    }
}
```

```
82.6
75.2
80.6
81.4
```

Multidimensional Arrays

- Arrays can have more than two dimensions.
  - `int[][][] a = new int[3][4][5];`
- Concepts for multidimensional arrays (2D above) can be generalized from 2D arrays
  - 3D array is an 1D array of (references to) 2D arrays, each of which is a 1D array of (references to) 1D arrays
- 1D array and 2D arrays are most commonly-used.