Chapter 5: Arrays

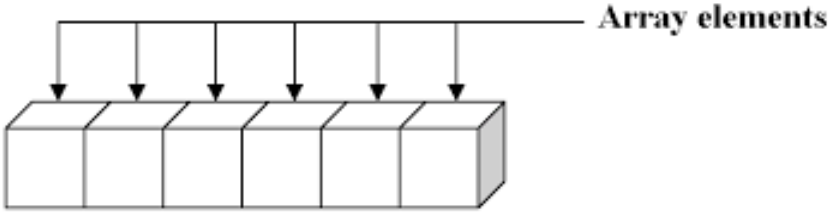
TAO Yida

taoyd@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 (also known as non-primitive types) (we will talk about this more later)
- Array elements can be either primitive types or reference types.

Declaring and Creating Arrays

- 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 ];
```

- Like other objects (recall the usage of `Scanner`), arrays are created with the keyword `new`.
- Returns a reference (representing the memory address of the array) that can be stored in an array variable.

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

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

Default: an int array of 12 zeros.
What's the println output?

[I@1b6d3586

Declaring and Creating Arrays

- 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 ];
```

- 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).

```
int[] a = new int[10];
a.length = 11;
```

Cannot assign a value to final variable 'length'

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);
System.out.println(a.length);
```

10

Compilation error: a has not been initialized

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 ];`

Where have we used String[]?

Referring to Array Elements

c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1

- The first element in every array has index zero.
- The highest index in an array is *the number of elements - 1*.

Referring to Array Elements

c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1

array-access expression

c[5] refers to the 6th element

- c is the reference to the array (or name of the array for simplicity)
  - Array names follow the same conventions as other variable names (Lower Camel Case)
- 5 is the position number of the element (index or subscript)

Referring to Array Elements

c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1

- A program can use an expression as an index (`c[ 1 + a ]`)
- An index must be a nonnegative integer (c[-2] causes error).
- If an index<0 or index>array.length-1, you'll get an ArrayIndexOutOfBounds Exception

Assigning to Array Elements

c[0]	-45	
c[1]	6	-> 2
c[2]	0	
c[3]	72	
c[4]	1543	
c[5]	-89	
c[6]	0	
c[7]	62	
c[8]	-3	
c[9]	1	

- 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`)

Default Initialization

```
public class InitArray {
  public static void main(String[] args) {
    int[] array; // declare an array
    array = new int[10]; // create the array object




  }
}
```

Default Initialization

```
public class InitArray {
  public static void main(String[] args) {
    int[] array; // declare an array
    array = new int[10]; // create the array object

  }
}
```

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

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]);
    }
  }
}
```

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

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

Otherwise: `java.lang.ArrayIndexOutOfBoundsException`

ERROR

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 = new int[]{ 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.

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 };`
  - Shortcut: initialize the array without using the new keyword
  - But, this shortcut is allowed only at the time of array declaration

```
int[] n;
n = { 10, 20, 30, 40, 50 };
```

Array initializer is not allowed here

Initializing Elements One by One

```
public class InitArray2 {
  public static void main(String[] args) {
    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]);
    }
  }
}
```

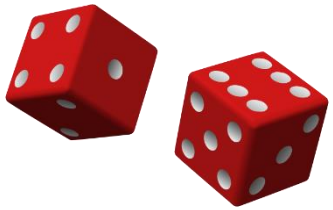
Index	Value
0	2
1	4
2	6
3	8
4	10
5	12
6	14
7	16
8	18
9	20

Initializing Elements One by One

```java
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]);
    }
  }
}
```

```
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?

```java
import java.util.Random;
public class DiceRolling {
  public static void main(String[] args) {
    Random generator = new Random();
    int[] frequency = new int[6];
    // roll 6000 times; use dice value as frequency index
    for(int roll = 1; roll <= 6000; roll++) {
      int face = generator.nextInt(6);
      frequency[face]++;
    }
    System.out.printf("%s%10s\n", "Face", "Frequency");
    // output the frequency of each face
    for(int face = 0; face < frequency.length; face++) {
      System.out.printf("%4d%10d\n", face+1, frequency[face]);
    }
  }
}
```

Use an array to track frequency

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

Execution Result

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

Q: Initialize an array by specifying its size, first element, and interval

```
java InitArray 5 0 4
Index   Value
    0       0
    1       4
    2       8
    3      12
    4      16
```

```
java InitArray 8 1 2
Index   Value
    0       1
    1       3
    2       5
    3       7
    4       9
    5      11
    6      13
    7      15
```

How to handle multiple parameters?

Fig. 6.15 | Initializing an array using command-line arguments. (Part 3 of 3.)

Using Command-Line Arguments

Lab 3

```java
public class Lab3_E1 {
    public static void main(String[] args) {
        String name = args[0];
        int age = Integer.parseInt(args[1]);
        float weight = Float.parseFloat(args[2]);
        char grade = args[3].charAt(0);

        System.out.printf("You are %s.\nYou are %d years old.\n", name, age);
        System.out.printf("You weigh %.1f KG.\n The highest grade you got is %c.\n", weight, grade);
    }
}
```

Command line arguments can be obtained in IDE or in command line:

(1) in command line

```
C:\workspace\javaLab\src\Lab3>javac Lab3_E1.java

C:\workspace\javaLab\src\Lab3>java Lab3_E1 zhangsan 20 60.1 A
You are zhangsan.
You are 20 years old.
You weigh 60.1 KG.
 The highest grade you got is A.
```

Using Command-Line Arguments

- It's possible to pass arguments from the command line to an application by including a parameter of type `String[]` in the parameter list of `main`.
  - `public static void main(String[] args)`
- By convention, this parameter is named `args`.
- When an application is executed using the `java` command, Java passes the command-line arguments that appear after the class name in the `java` command to the application's `main` method as `String`s in the array `args`.

```java
// Fig. 6.15: InitArray.java
// Initializing an array using command-line arguments.

public class InitArray
{
   public static void main( String[] args )
   {
      // check number of command-line arguments
      if ( args.length != 3 )
         System.out.println(
            "Error: Please re-enter the entire command, including\n" +
            "an array size, initial value and increment." );
      else
      {
         // get array size from first command-line argument
         int arrayLength = Integer.parseInt( args[ 0 ] );
         int[] array = new int[ arrayLength ]; // create array

         // get initial value and increment from command-line arguments
         int initialValue = Integer.parseInt( args[ 1 ] );
         int increment = Integer.parseInt( args[ 2 ] );

```

Fig. 6.15 | Initializing an array using command-line arguments. (Part 1 of 3.)

```java
            // calculate value for each array element
            for ( int counter = 0; counter < array.length; counter++ )
               array[ counter ] = initialValue + increment * counter;

            System.out.printf( "%s%8s\n", "Index", "Value" );

            // display array index and value
            for ( int counter = 0; counter < array.length; counter++ )
               System.out.printf( "%5d%8d\n", counter, array[ counter ] );
         } // end else
      } // end main
   } // end class InitArray
```

```
java InitArray
Error: Please re-enter the entire command, including
an array size, initial value and increment.
```

Fig. 6.15 | Initializing an array using command-line arguments. (Part 2 of 3.)

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

Enhanced for Statement

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

```
for ( parameterType identifier : arrayName ) {
   statement(s)
}
```

- 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 only **read access** 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?

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

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

Two-Dimensional Arrays

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

78	-9	520	0	14
Index 0	1	2	3	4

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

	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

column

row

```
gradbook[ 1 ][ 2 ]
```

(gradebook is the name of the array)

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}};`

1	2	3	4
5	6		
7	8	9	
10			

Row 1 Row 2 Row 3 Row 4

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

Why do we need ragged arrays?

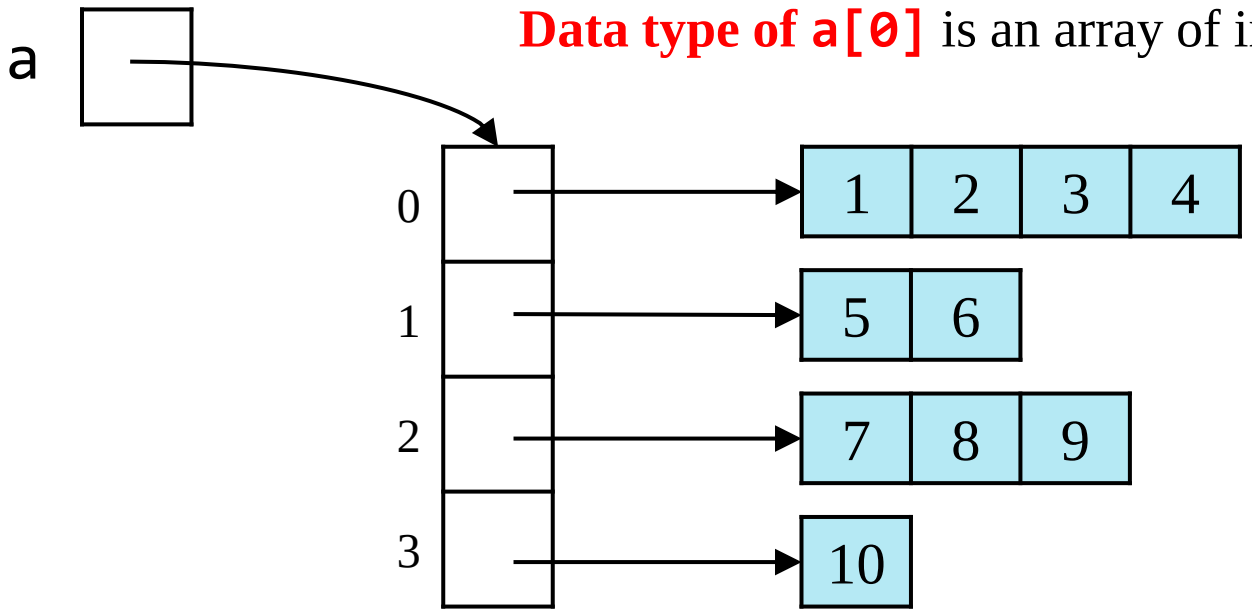
```
1 * 1 = 1
1 * 2 = 2  2 * 2 = 4
1 * 3 = 3  2 * 3 = 6  3 * 3 = 9
1 * 4 = 4  2 * 4 = 8  3 * 4 = 12 4 * 4 = 16
1 * 5 = 5  2 * 5 = 10 3 * 5 = 15 4 * 5 = 20 5 * 5 = 25
1 * 6 = 6  2 * 6 = 12 3 * 6 = 18 4 * 6 = 24 5 * 6 = 30 6 * 6 = 36
1 * 7 = 7  2 * 7 = 14 3 * 7 = 21 4 * 7 = 28 5 * 7 = 35 6 * 7 = 42 7 * 7 = 49
1 * 8 = 8  2 * 8 = 16 3 * 8 = 24 4 * 8 = 32 5 * 8 = 40 6 * 8 = 48 7 * 8 = 56 8 * 8 = 64
1 * 9 = 9  2 * 9 = 18 3 * 9 = 27 4 * 9 = 36 5 * 9 = 45 6 * 9 = 54 7 * 9 = 63 8 * 9 = 72 9 * 9 = 81
```

p =	0	1	2	3	4	5	6	7	8	9	10	11	12
n =0	1												
1	1	1											
2	1	2	1										
3	1	3	3	1									
4	1	4	6	4	1								
5	1	5	10	10	5	1							
6	1	6	15	20	15	6	1						
7	1	7	21	35	35	21	7	1					
8	1	8	28	56	70	56	28	8	1				
9	1	9	36	84	126	126	84	36	9	1			
10	1	10	45	120	210	252	210	120	45	10	1		
11	1	11	55	165	330	462	462	330	165	55	11	1	
12	1	12	66	220	495	792	924	792	495	220	66	12	1

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}};
```



Data type of `a[0]` is an array of integers

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
```

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[ 3 ][ ];   // create 2 rows
b[ 0 ] = new int[]{ 1, 2, 3, 4 }; // initialize row 0
b[ 1 ] = new int[]{ 5, 6 }; // initialize row 1
b[ 2 ] = { 7, 8, 9 };       // compilation error!
```

Displaying Element values

```
public static void main(String[] args) {
    int[][] a = {{1, 2, 3, 4}, {5, 6}, {7, 8, 9}, {10}};

}
```

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

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 <          ; row++) {
        // loop through columns
        for(int column = 0; column <              : column++) {
            System.out.printf("%d ",  _   --      _);
        }
        System.out.println();
    }
}
```

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

Computing Average Scores for each student (using the enhanced for loop)

```
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}
    };

}
```

```
82.6
75.2
80.6
81.4
```

Computing Average Scores for each student (using the enhanced for loop)

```
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(      grades : gradebook) {
        int sum = 0;



        ,
        System.out.printf("%.1f\n", ((double) sum)/grades.length);
    }
}
```

```
82.6
75.2
80.6
81.4
```

Computing Average Scores for each student (using the enhanced for loop)

```
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
```

Can we move int sum=0 before the for loop?

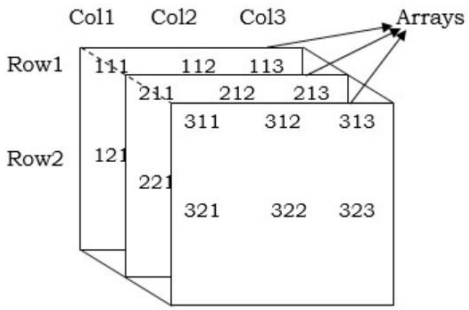
What is the value of matrix?

```java
int[][] matrix = new int[4][];
matrix[0] = new int[]{1,2,3};
matrix[1] = new int[]{1,2,3};
matrix[2] = new int[]{1,2,3};
matrix[3] = new int[]{1,2,3};

for(int col=0;col<3;col++){
    for(int row=1;row<4;row++){
        matrix[row][col] += matrix[row-1][col];
    }
}
```

1	2	3
1	2	3
1	2	3
1	2	3

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.

Multidimensional Arrays

An RGB image is stored as an m x n x 3 data array that defines red, green, and blue color components for each individual pixel

Multidimensional Arrays

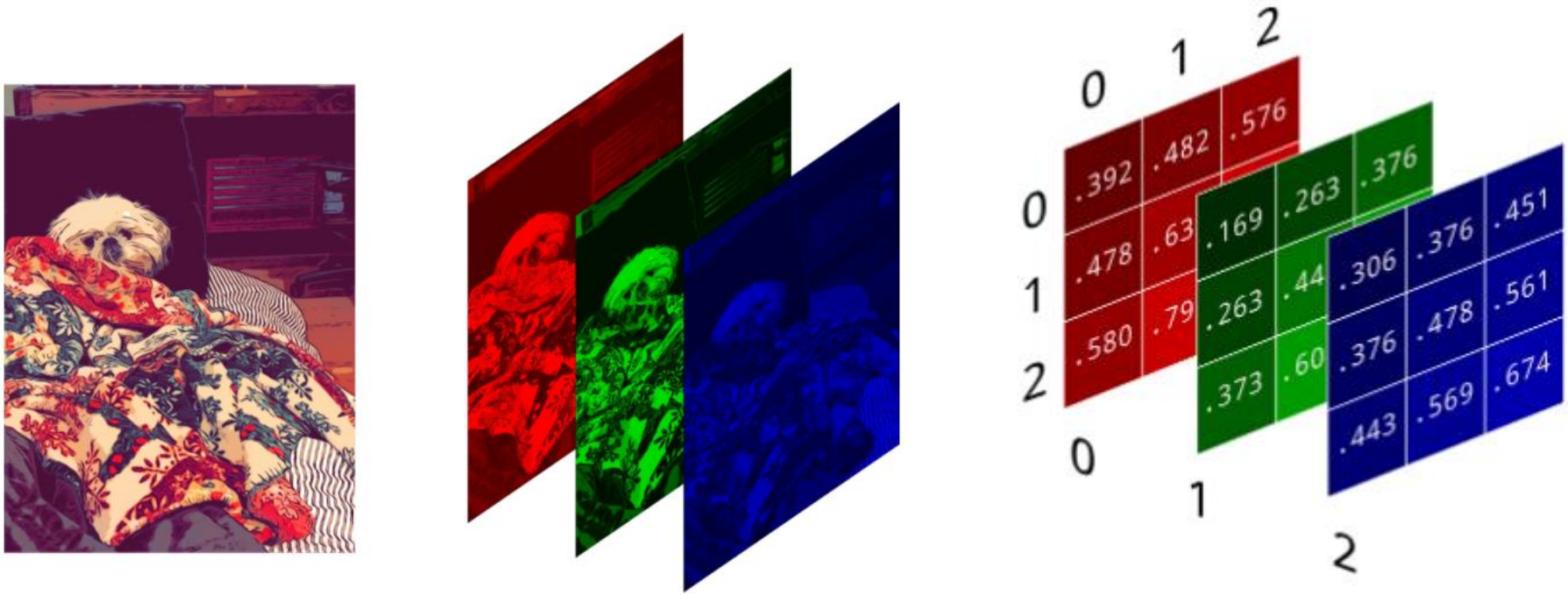


Image credit: Diane Rohrer

Image: https://www.kdnuggets.com/2019/12/convert-rgb-image-grayscale.html

6.12 Class `Arrays`

- Class Arrays helps you avoid reinventing the wheel by providing `static` methods for common array manipulations.
- These methods include sort for sorting an array (i.e., arranging elements into increasing order), binarySearch for searching an array (i.e., determining whether an array contains a specific value and, if so, where the value is located), equals for comparing arrays and fill for placing values into an array.
- These methods are overloaded for primitive-type arrays and for arrays of objects.
- You can copy arrays with class `System`'s `static` arraycopy method.
-

```java
// Fig. 6.16: ArrayManipulations.java
// Arrays class methods and System.arraycopy.
import java.util.Arrays;

public class ArrayManipulations
{
   public static void main( String[] args )
   {
      // sort doubleArray into ascending order
      double[] doubleArray = { 8.4, 9.3, 0.2, 7.9, 3.4 };
      Arrays.sort( doubleArray );
      System.out.printf( "\ndoubleArray: " );

      for ( double value : doubleArray )
         System.out.printf( "%.1f ", value );

      // fill 10-element array with 7s
      int[] filledIntArray = new int[ 10 ];
      Arrays.fill( filledIntArray, 7 );
      displayArray( filledIntArray, "filledIntArray" );

      // copy array intArray into array intArrayCopy
      int[] intArray = { 1, 2, 3, 4, 5, 6 };
      int[] intArrayCopy = new int[ intArray.length ];
```

Fig. 6.16 | Arrays class methods. (Part 1 of 4.)

```java
        System.arraycopy( intArray, 0, intArrayCopy, 0, intArray.length );
        displayArray( intArray, "intArray" );
        displayArray( intArrayCopy, "intArrayCopy" );

        // compare intArray and intArrayCopy for equality
        boolean b = Arrays.equals( intArray, intArrayCopy );
        System.out.printf( "\n\nintArray %s intArrayCopy\n",
           ( b ? "==" : "!=" ) );

        // compare intArray and filledIntArray for equality
        b = Arrays.equals( intArray, filledIntArray );
        System.out.printf( "intArray %s filledIntArray\n",
           ( b ? "==" : "!=" ) );

        // search intArray for the value 5
        int location = Arrays.binarySearch( intArray, 5 );

        if ( location >= 0 )
           System.out.printf(
              "Found 5 at element %d in intArray\n", location );
        else
           System.out.println( "5 not found in intArray" );

```

Fig. 6.16 | Arrays class methods. (Part 2 of 4.)

```java
      // search intArray for the value 8763
      location = Arrays.binarySearch( intArray, 8763 );

      if ( location >= 0 )
         System.out.printf(
            "Found 8763 at element %d in intArray\n", location );
      else
         System.out.println( "8763 not found in intArray" );
   } // end main

   // output values in each array
   public static void displayArray( int[] array, String description )
   {
      System.out.printf( "\n%s: ", description );

      for ( int value : array )
         System.out.printf( "%d ", value );
   } // end method displayArray
} // end class ArrayManipulations
```

Fig. 6.16 | `Arrays` class methods. (Part 3 of 4.)

```
doubleArray: 0.2 3.4 7.9 8.4 9.3
filledIntArray: 7 7 7 7 7 7 7 7 7 7
intArray: 1 2 3 4 5 6
intArrayCopy: 1 2 3 4 5 6

intArray == intArrayCopy
intArray != filledIntArray
Found 5 at element 4 in intArray
8763 not found in intArray
```

Fig. 6.16 | `Arrays` class methods. (Part 4 of 4.)