CS3231
Object Oriented Programming I

Name: ________________________________ (   ) Date: ________________

Chapter 10: Arrays

Lesson Objectives

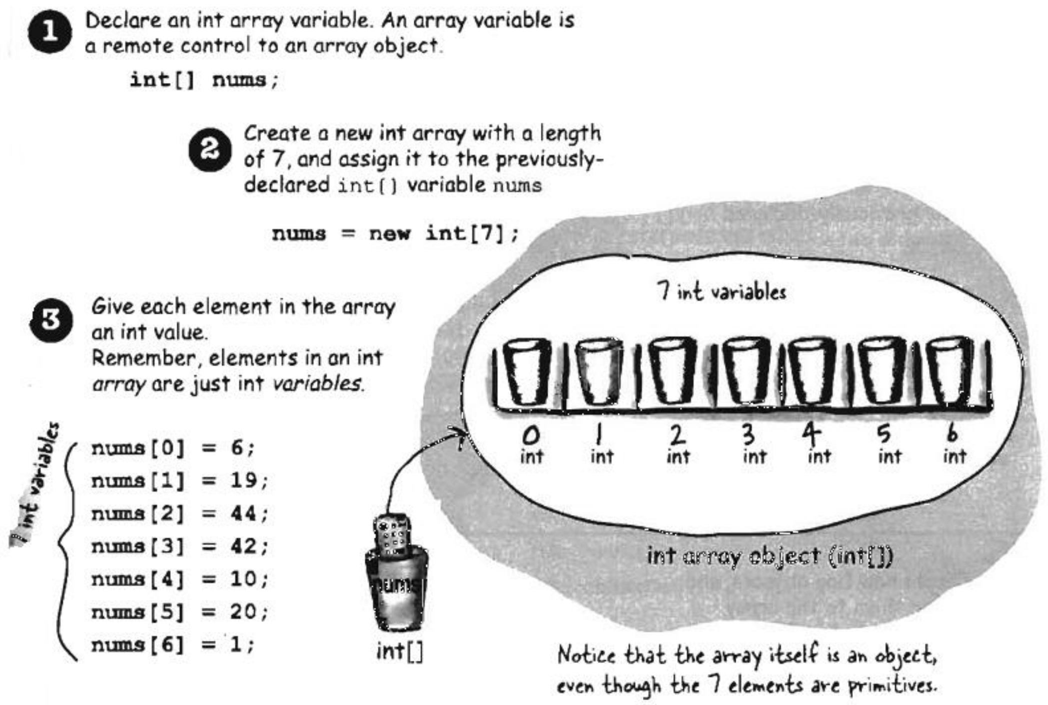
After completing the lesson, the student will be able to

- *describe why arrays are necessary in programming*
- *declare array reference variables and create arrays*
- *obtain array size using arrayRefVar.length and know default values in an array*
- *access array elements using indexes*
- *declare, create, and initialize an array using an array initializer*
- *simplify programming using the for-each loops*
- *copy arrays and process arrays with methods*
- *store and process objects in arrays*
- *create and use 2D arrays*

10.1 Array Basics

Often in problem solving will have to store a large number of values during the execution of a program. For instance, if you need to read 100 numbers, compute their average, and find out the maximum or minimum of the values. Your program first reads the numbers and computes their average, then compares each number to look for the maximum or minimum values. In order to accomplish this task, the numbers must all be stored in variables. You have to declare 100 variables and repeatedly write almost identical code 100 times. Writing a program this way would be impractical and an efficient, organized approach is needed.

Java and most other high-level languages provide a data structure, the **array, which stores a fixed-size sequential collection of elements of the same type. A single array variable can reference a large collection of data. Once an array is created, its size is fixed. An array reference variable is used to access the elements in an array using an index.**



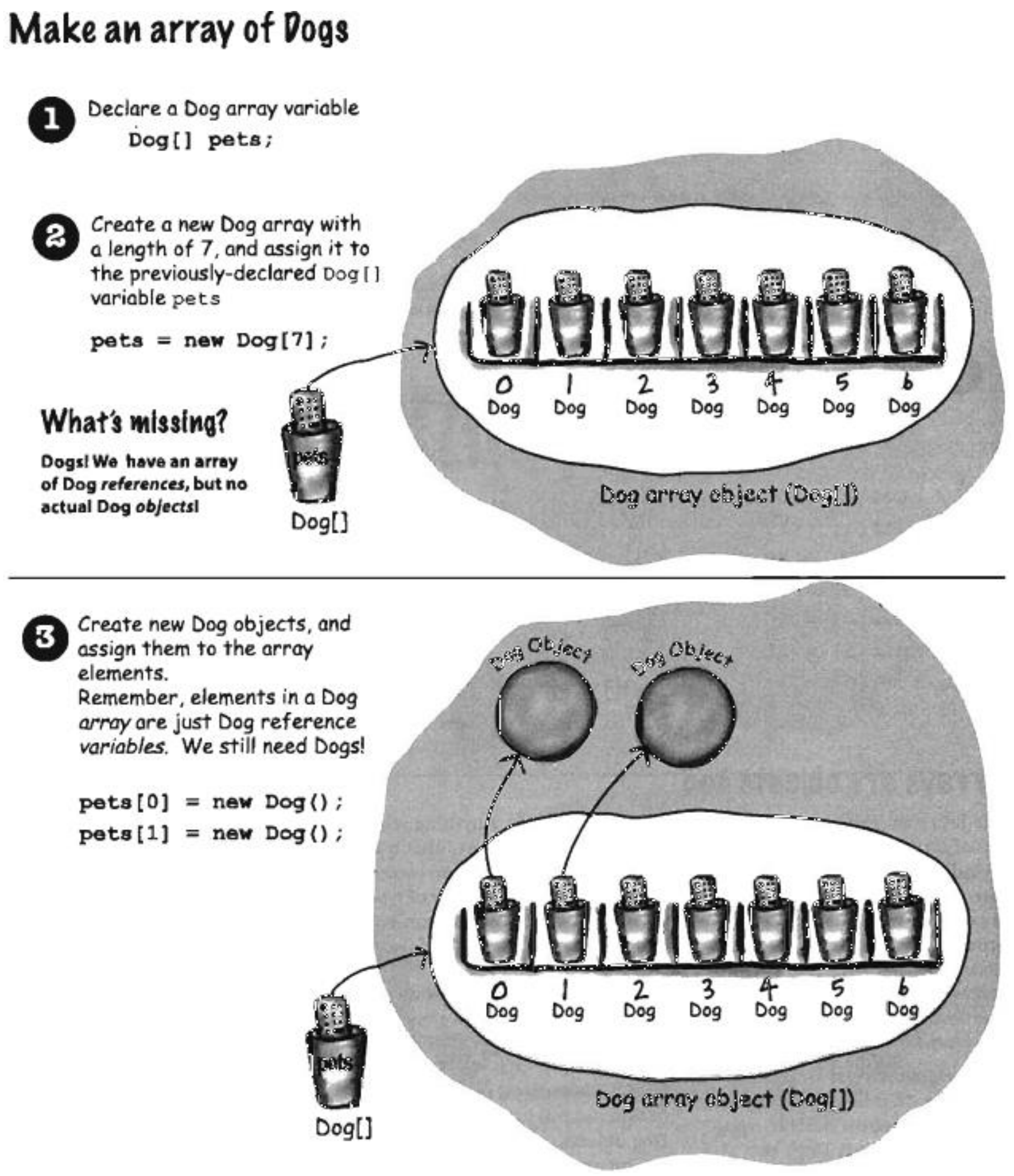
1 Declare an int array variable. An array variable is a remote control to an array object.
int[] nums;
2 Create a new int array with a length of 7, and assign it to the previously-declared int[] variable nums
nums = new int[7];
3 Give each element in the array an int value. Remember, elements in an int array are just int variables.
int variables
nums[0] = 6;
nums[1] = 19;
nums[2] = 44;
nums[3] = 42;
nums[4] = 10;
nums[5] = 20;
nums[6] = 1;
7 int variables
0 int
1 int
2 int
3 int
4 int
5 int
6 int
nums
int[]
int array object (int[])
Notice that the array itself is an object, even though the 7 elements are primitives.

Figure 10.1 Steps to create an array of `int` primitives

An array is used to store a collection of data, but often we find it more useful to think of an array as a **collection of variables of the same type**. Instead of declaring individual variables, such as number0, number1, . . . , and number99, you declare one array variable such as numbers and use numbers[0] , numbers[1] , . . . , and numbers[99] to represent individual variables.

10.2 Array Are Objects

The Java API includes lots of sophisticated data structures including maps, trees and sets, but arrays are great when you just want a quick, ordered efficient list of things. Arrays give you fast random access by letting you use an index position to get to any element in the array.



Make an array of Dogs
1 Declare a Dog array variable
Dog[] pets;
2 Create a new Dog array with a length of 7, and assign it to the previously-declared Dog[] variable pets
pets = new Dog[7];
What's missing?
Dogs! We have an array of Dog references, but no actual Dog objects!
pets
Dog[]
0 1 2 3 4 5 6
Dog Dog Dog Dog Dog Dog Dog
Dog array object (Dog[])
3 Create new Dog objects, and assign them to the array elements.
Remember, elements in a Dog array are just Dog reference variables. We still need Dogs!
pets[0] = new Dog();
pets[1] = new Dog();
Dog Object
Dog Object
pets
Dog[]
0 1 2 3 4 5 6
Dog Dog Dog Dog Dog Dog Dog
Dog array object (Dog[])

Figure 10.2 Steps to create an array of Dog object reference

Every element in an array is just a variable. In other words, one of the eight primitive variable types or a reference variable. Anything you would put in a variable of that type can be assigned to an array element of that type. So in an array of type int (`int []`), each element can hold an int. In a Dog array (`Dog[]`) each element can hold .. a Dog? No, remember that **a reference**

variable just holds a reference (a remote control), not the object itself. So in a Dog array, each element can hold a remote control to a Dog. Of course, we still have to make the Dog objects.

An important concept to take note is that an array is an object even though it may be an array of primitives (Figure 10.1). **Arrays are always objects, whether they are declared to hold primitives or object references**. In other words, the array object can have elements which are primitives, but the array itself is never a primitive. Regardless of what the array holds, the array itself is always an object.

10.3 Declaring Array Variables

To use an array in a program, you must declare a variable to reference the array and specify the array's element type. Here is the syntax for declaring an array variable:

```
elementType[] arrayRefVar;
```

The `elementType` can be any data type, and all elements in the array will have the same data type. For example, the following code declares a variable `myList` that references an array of double elements.

```
double[] myList;
```

10.4 Creating Arrays

Unlike declarations for primitive data type variables, **the declaration of an array variable does not allocate any space in memory for the array. It creates only a storage location for the reference to an array**. **If a variable does not contain a reference to an array, the value of the variable is null.** You cannot assign elements to an array unless it has already been created. After an array variable is declared, you can create an array by using the `new` operator and assign its reference to the variable with the following syntax:

```
arrayRefVar = new elementType[arraySize];
```

This statement does two things:

- it creates an array using `new elementType[arraySize];`
- it assigns the reference of the newly created array to the variable `arrayRefVar`.

Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement as:

```
elementType[] arrayRefVar = new elementType[arraySize];
```

or

```
elementType arrayRefVar[] = new elementType[arraySize];
```

Below is an example of such a statement:

```
double[] myList = new double[10];
```

This statement declares an array variable, `myList`, creates an array of ten elements of double type, and assigns its reference to `myList`. To assign values to the elements, use the syntax:

```
arrayRefVar[index] = value;
```

For example, the following code initializes the array. Figure 10.3 shows the effect after the initialization.

```
myList[0] = 5.6;
myList[1] = 4.5;
myList[2] = 3.3;
myList[3] = 13.2;
myList[4] = 4.0;
myList[5] = 34.33;
myList[6] = 34.0;
myList[7] = 45.45;
myList[8] = 99.993;
myList[9] = 11123
```

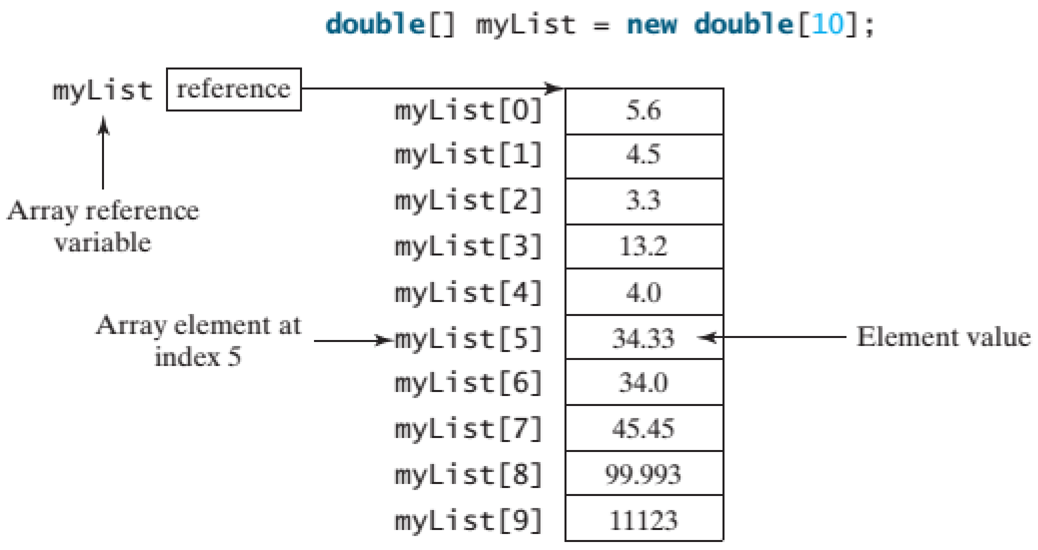


Figure 10.3 The array `myList` has ten elements of double type and int indices from 0 to 9.

Note: An array variable that appears to hold an array actually contains a reference to that array. Strictly speaking, an array variable and an array are different, but most of the time the distinction can be ignored. Thus it is all right to say, for simplicity, that `myList` is an array, instead of stating, at greater length, that `myList` is a variable that contains a reference to an array of ten double elements.

10.4.1 Array Size and Default Values

When space for an array is allocated, the array size must be given, specifying the number of elements that can be stored in it. The size of an array cannot be changed after the array is created. Size can be obtained using `arrayRefVar.length`. For example, `myList.length` is 10.

When an array is created, its elements are assigned the default value of 0 for the numeric primitive data types, \u0000 for char types, and false for boolean types.

10.4.2 Accessing Array Elements

The array elements are accessed through the index. **Array indices are 0 based; that is, they range from 0 to arrayRefVar.length - 1**. In Figure 10.3, `myList` holds ten double values, and the indices are from 0 to 9. Each element in the array is represented using the following syntax, known as an **indexed variable**:

```
arrayRefVar[index];
```

For example, `myList[9]` represents the last element in the array `myList`.

An indexed variable can be used in the same way as a regular variable. For example, the following code adds the values in `myList[0]` and `myList[1]` to `myList[2]`.

```
myList[2] = myList[0] + myList[1];
```

The following loop assigns 0 to `myList[0]`, 1 to `myList[1]`, . . . , and 9 to `myList[9]`:

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

10.4.3 Initializing Arrays

There are a few ways you can initialize or fill your arrays. Let's look at them.

<u>Using a loop</u>

```
int[] array = new int[10];
for (int i = 0; i < array.length; i++) {
    array[i] = i + 2;
}
```

<u>Using array initializers</u>

Java has a shorthand notation, known as the array initializer, which combines the declaration, creation, and initialization of an array in one statement using the following syntax:

```
elementType[] arrayRefVar = {value0, value1, ..., valuek};
```

For example, the statement

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

declares, creates, and initializes the array `myList` with four elements, which is equivalent to the following statements:

```
double[] myList = new double[4];
myList[0] = 1.9;
myList[1] = 2.9;
myList[2] = 3.4;
myList[3] = 3.5;
```

Note: The `new` operator is not used in the array-initializer syntax. **Using an array initializer, you have to declare, create, and initialize the array all in one statement.** Splitting it would cause a syntax error. Thus, the next statement is wrong:

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

<u>Using Arrays.fill()</u>

The *java.util.Arrays* class has several methods named *fill()* which accept different types of arguments and fill the whole array with the same value:

```
long array[] = new long[5];
Arrays.fill(array, 30);
```

The method also has several alternatives which set a range of an array to a particular value:

```
int array[] = new int[5];
Arrays.fill(array, 0, 3, -50);
```

Note that the method accepts the array, the start index to fill the array, the end index to fill the array and the value.

Using Arrays.setAll()
The method *Arrays.setAll()* sets all elements of an array using a generator function:

```
int[] array = new int[20];
Arrays.setAll(array, p -> p > 9 ? 0 : p);

// [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

If the generator function is null, then a *NullPointerException* is thrown.

10.5 Processing Arrays

When processing array elements, you will often use a `for` loop—for two reasons:

- All of the elements in an array are of the same type. They are evenly processed in the same fashion repeatedly using a loop.
- Since the size of the array is known, it is natural to use a `for` loop.

PROGRAM 10-1 demonstrates a series of operations on how to process arrays using mutiple `for` loops.

PROGRAM 10-1

```
public class processingArray {
 public static void main(String[] args){
    double[] myList = new double[10];
    // generate 10 random double values into myList
    for (int i = 0; i < myList.length; i++){
       myList[i] = Math.random() * 100;
    }
    // print out the individual element values in myList
    for (int i = 0; i < myList.length; i++){
      System.out.println("myList[" + i + "]" + " is " + myList[i]);
    }
    // sum and print out the sum in myList
    double total = 0;
    for (int i = 0; i < myList.length; i++){
      total += myList[i];
    }
    System.out.println("The total in myList is " + total);
    // find the maximum and print out the max in myList
    double max = myList[0];
    for (int i = 1; i < myList.length; i++) {
      if (myList[i] > max)
         max = myList[i];
    }
    System.out.println("The maximum in myList is " + max);
    }
}
```

PROGRAM OUTPUT

```
myList[0] is 81.60082313254128
myList[1] is 22.0316236793085
myList[2] is 87.19894818450447
myList[3] is 27.857169597765242
myList[4] is 9.62874596499691
myList[5] is 92.91479290102103
myList[6] is 93.63627187041918
myList[7] is 39.669502843151236
myList[8] is 56.34560547925288
myList[9] is 66.84590653176853
The total in myList is 577.7293901847293
The maximum in myList is 93.63627187041918
```

Arrays can be used to greatly simplify coding for certain tasks. For example, suppose you wish to obtain the English name of a given month by its number. If the month names are stored in an array, the month name for a given month can be accessed simply via the index. The following code prompts the user to enter a month number and displays its month name:

PROGRAM 10-2

```
import java.util.Scanner;
public class processingArray {
 public static void main(String[] args){
    Scanner input = new Scanner(System.in);
    String[] months = {"January","February","March","April",
                        "May", "June", "July", "August",
                        "September", "October","November", "December"};
    System.out.print("Enter a month number (1 to 12): ");
    int monthNumber = input.nextInt();
    System.out.println("The month is " + months[monthNumber - 1]);
 }
}
```

PROGRAM OUTPUT

```
Enter a month number (1 to 12): 6
The month is June
```

If you did not use the months array, you would have to determine the month name using a lengthy multi-way `if-else` statement as follows:

```
if (monthNumber == 1)
      System.out.println("The month is January");
else if (monthNumber == 2)
      System.out.println("The month is February");
      ...
else
      System.out.println("The month is December");
```

10.6 Array of Objects

You can also create arrays of objects. For example, the following statement declares and creates an array of ten Circle objects:

```
Circle[] cArray = new Circle[10];
```

IMPORTANT: `new Circle[10]` only create 10 Circle references. No Circle objects has been physically created yet. If you try to use `cArray[0].getRadius()`,you will get java.lang.NullPointerException error during run time.

To initialize `circleArray` with Circle objects, you can use a for loop like this one:

```
for (int i = 0; i < cArray.length; i++) {
 cArray[i] = new Circle(); // call constructor to create objects
}
```

An array of objects is actually an array of reference variables. So, invoking `circleArray[1].getArea()` involves two levels of referencing, as shown in Figure 10.4.

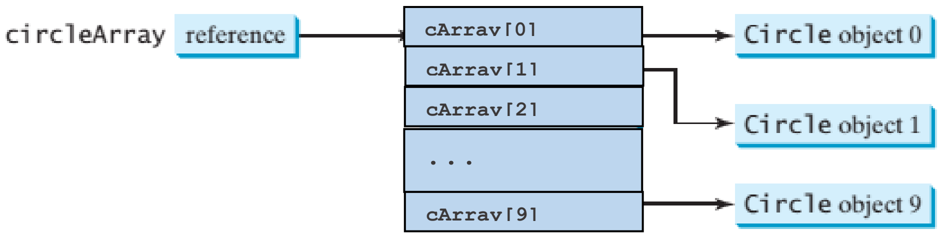


Figure 10.4 In an array of objects, an element of the array contains a reference to an object.

When an array of objects is created using the `new` operator, each element in the array is a reference variable with a default value of null .

PROGRAM 10-3A and 10-3B gives an example that demonstrates how to use an array of objects. The program summarizes the areas of an array of circles. The program creates circleArray, an array composed of five Circle objects; it then initializes circle radii with random values and displays the total area of the circles in the array.

PROGRAM 10-3A

```
public class Circle {
        private double radius = 1;
        public Circle() {}
        public Circle(double newRadius) {
            radius = newRadius;
        }
        public double getRadius() {
            return radius;
        }
        public double getArea() {
            return radius * radius * Math.PI;
        }
        public void setRadius(double r) {
            radius = r;
        }
        public String toString() {
              return "Circle radius " + radius;
        }
}
```

PROGRAM 10-3B

```
public class TotalArea {
  public static void main(String[] args) {
    Circle[] cArray = createCircleArray();  // Create circleArray
    printCircleArray(cArray);
  }

  public static Circle[] createCircleArray() {
    Circle[] cArray =  new Circle[5];
    for (int i = 0; i < cArray.length; i++)
      cArray[i] = new Circle(Math.random() * 100);

    return cArray;
  }
  /** Print an array of circles and their total area */
  public static void printCircleArray( Circle[] cArray) {
    System.out.printf("%-30s%-15s\n", "Radius", "Area");
    System.out.println("────────────────────────────────────────-");
    for (int i = 0; i < cArray.length; i++) {
     System.out.printf("%-30f%-15f\n", cArray[i].getRadius(),
     cArray[i].getArea());
    }

   System.out.printf("%-30s%-15f\n", "\nTotal area of circles is",
   sum(cArray) );
  }

 public static double sum(Circle[] circleArray) {
   double sum = 0;
   for (int i = 0; i < circleArray.length; i++)
      sum += circleArray[i].getArea();

   return sum;
 }
}
```

PROGRAM OUTPUT

```
Radius                        Area
──────────────────────────────────────────-
60.882509                     11644.878220
57.962632                     10554.704175
21.525097                     1455.593560
18.878039                     1119.601851
89.060473                     24918.383665

Total area of circles is     49693.161471
```

10.7 For-each Loops (Enhanced For Loop)

Java supports a convenient for loop, known as a **for-each loop**, which enables you to traverse the array sequentially without using an index variable. For example, the following code displays all the elements in the array myList:

```
for (double e: myList) {
 System.out.println(e);
}
```

You can read the code as "for each element `e` in `myList`, do the following." Note that the variable, `e`, must be declared as the same type as the elements in `myList`. In general, the syntax for a `foreach` loop is

```
for (elementType element: arrayRefVar) {
// Process the element
}
```

From JDK 9, Java introduced a new feature: “var” to define and initialise the local variables as shown below:

```
var array = new int[20];
```

Here Java 9 Compiler will infer the type of array reference as new int[].

However, for easy code readibility, it is still recommended to initialize your data types using

```
int[] array = new int[20];
```

The use of var, however, is useful when you use the for-each loop.

```
for(var i: array) {
      System.out.print(i + “,” );
}
```

Here, instead of using int i: array, we can use var i. This code can be reused for printing arrays of different data type.

While the for-each loop is quick to code for printing of arrays, you still have to use an index variable if you wish to traverse the array in a different order or change the elements in the array. Accessing an array out of bounds is a common programming error that throws a runtime ArrayIndexOutOfBoundsException. To avoid it, make sure that you do not use an index beyond `arrayRefVar.length – 1`.

10.8 Copying Arrays

To copy the contents of one array into another, you have to copy the array’s individual elements into the other array. Often, in a program, you need to duplicate an array or a part of an array. In such cases you could attempt to use the assignment statement (=), as follows:

```
list2 = list1;
```

However, this statement does not copy the contents of the array referenced by `list1` to `list2`, but instead merely copies the reference value from `list1` to `list2`. After this statement, `list1` and `list2` reference the same array, as shown in Figure 10.5. The array previously referenced by `list2` is no longer referenced; it becomes garbage, which will be automatically collected by the Java Virtual Machine (this process is called garbage collection).

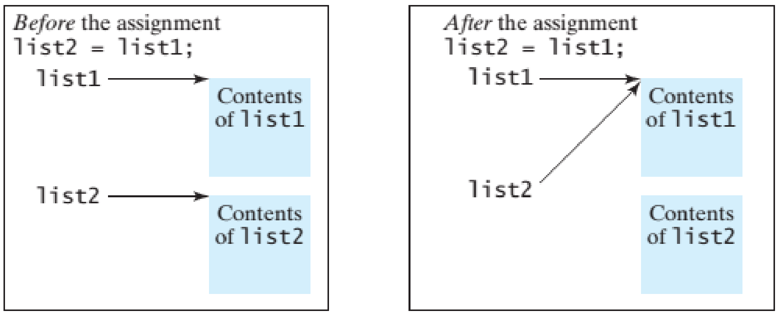


Figure 10.5 Effect of Array Assignment

In Java, you can use assignment statements to copy primitive data type variables, but not arrays. Assigning one array variable to another array variable actually copies one reference to another and makes both variables point to the same memory location. See Figure 10.4.

There are two ways to copy arrays:

- Use a loop to copy individual elements one by one.
- Use Arrays.copyOf() method

Using a loop

You can write a loop to copy every element from the source array to the corresponding element in the target array. The following code, for instance, copies `sourceArray` to `targetArray` using a `for` loop.

```
int[] sourceArray = {2, 3, 1, 5, 10};
int[] targetArray = new int[sourceArray.length];
for (int i = 0; i < sourceArray.length; i++) {
 targetArray[i] = sourceArray[i];
}
```

Using Arrays.copyOf()

Another approach is to use the `Arrays.copyOf()` method to copy arrays instead of using a loop. For example:

```
int[] array = {1,2,3,4,5};
int[] copy = Arrays.copyOf(array, 5);

array[0] = 99;

for(var i: array)
      System.out.print(i + ",");

System.out.println();
for(var i: copy)
      System.out.print(i + ",");
```

Output:

```
99,2,3,4,5,
1,2,3,4,5,
```

However, do note that using the Arrays.copyOf() on objects will result in a **SHALLOW COPY** of the array objects.
For example:

PROGRAM 10-4

```
1   Circle[] list = new Circle[3];
2   list[0] = new Circle(1);
3   list[1] = new Circle(2);
4   list[2] = new Circle(3);
5
6   Circle[] duplicate = Arrays.copyOf(list,list.length);
7
8   list[0].setRadius(99);
9
10  System.out.println();
11  for(var i: list)
12        System.out.print(i + ",");
13
14  System.out.println();
15  for(var i: duplicate)
16        System.out.print(i + ",");
```

Output:

```
Circle radius 99.0,Circle radius 2.0,Circle radius 3.0,
Circle radius 99.0,Circle radius 2.0,Circle radius 3.0,
```

Note that in line 8, although only the radius of the first circle object in list is changed to 99, the change also affects the duplicate array as they are in fact pointing to the same circle object.

This will result in the undesirable effect of editing one array, and affecting the content of the other array.

To create a DEEP COPY of the array of circles, you need to use a for loop and create a new instance of Circle object for each array index.
For example:

PROGRAM 10-5

```
1   Circle[] list = new Circle[3];
2   list[0] = new Circle(1);
3   list[1] = new Circle(2);
4   list[2] = new Circle(3);
5
6   for(int i=0; i<list.length; i++) {
7         duplicate[i] = new Circle(list[i].getRadius());
8   }
9
10  list[0].setRadius(99);
11
12  System.out.println();
13  for(var i: list)
14        System.out.print(i + ",");
15
16  System.out.println();
17  for(var i: duplicate)
18        System.out.print(i + ",");
```

Output:

```
Circle radius 99.0,Circle radius 2.0,Circle radius 3.0,
Circle radius 1.0,Circle radius 2.0,Circle radius 3.0,
```

In line 7, a new Circle object is created with the same radius value of the original Circle object in list. Now the 2 arrays are pointing to two set of Circle objects, each with the same radius. Hence, when the radius of first Circle object is edited in line 10, the change will not affect the Circle object in duplicate.

10.9 Sorting Array Elements

The Arrays class has the method sort() which allows us to quickly sort an array of primitive data types.

For example:

```
int[] array = {2, 45, 23, 7, 1, 78};
Arrays.sort(array);
for(var i: array)
      System.out.print(i + ",");
```

Output:

```
1,2,7,23,45,78,
```

For example:

```
char[] array = {'c', 'a', 'z', 'e'};
Arrays.sort(array);
for(var i: array)
      System.out.print(i + ",");
```

Output:

```
a,c,e,z,
```

10.10 Passing Array and Array Element to Methods

PROGRAM 10-6

```java
1  public class TestDoWhile {
2      public static void main(String[] args) {
3          int[] array = { 1, 2, 3, 4, 5 };
4          System.out.printf( "Effects of passing reference to entire array:%n" +
5                  "The values of the original array are:%n");
6          for (int value : array) // output original array elements
7              System.out.printf(" %d", value);
8  
9          modifyArray(array);
10         System.out.printf("%n%nThe values of the modified array are:%n");
11         for (int value : array) // output modified array elements
12             System.out.printf(" %d", value);
13 
14         System.out.printf("%n%nEffects of passing array element value:%n" +
15                 "array[3] before modifyElement: %d%n", array[3]);
16         modifyElement(array[3]);
17 
18         System.out.printf("array[3] after modifyElement: %d%n", array[3]);
19     }
20     public static void modifyArray(int[] array2){
21         for (int counter = 0; counter < array2.length; counter++)
22             array2[counter] *= 2;
23     }
24     public static void modifyElement(int element){
25         element *= 2;
26         System.out.printf("Value of element in modifyElement: %d%n", element);
27     }
28 }
29 
```

PROGRAM OUTPUT

```
Effects of passing reference to entire array:
The values of the original array are:
 1 2 3 4 5

The values of the modified array are:
 2 4 6 8 10

Effects of passing array element value:
array[3] before modifyElement: 8
Value of element in modifyElement: 16
array[3] after modifyElement: 8
```

PROGRAM 10-6 demonstrates the difference between passing an entire array and passing a primitive-type array element to a method. Notice that `main` invokes static methods `modifyArray()` (line 26) and `modifyElement()`(line 31) directly. **A static method of a class can invoke other static methods of the same class directly**. Line 11 invokes method `modifyArray()`, passing array as an argument. The method (lines 26–29) receives a copy of array's reference and uses it to multiply each of array's elements by 2. To prove that array's elements were modified, lines 15-16 output array's elements again. As the output shows, method `modifyArray()` doubled the value of each element. **When a copy of an individual primitive-type array element is passed to a method, modifying the copy in the called method does not affect the original value of that element in the calling method's array**.

Lines 18–19 output the value of `array[3]` before invoking method `modifyElement()`. Remember that the value of this element is now 8 after it was modified in the call to `modifyArray()`. Line 20 calls method `modifyElement()` and passes `array[3]` as an argument.

Remember that `array[3]` is actually one int value (8) in array. Therefore, **the program passes a copy of the value of `array[3]` .**

Method `modifyElement()` (lines 31–33) multiplies the value received as an argument by 2, stores the result in its parameter element, then outputs the value of element (16). Since **method parameters, like local variables, cease to exist when the method in which they are declared completes execution, the method parameter element is destroyed when method `modifyElement()` terminates.** When the program returns control to `main`, lines 31 output the unmodified value of `array[3]` (i.e., 8).

10.11 Methods That Return an Array

In Java, a method may return an array. You specify the return type for a method that returns an array in the same way that you specify a type for an array parameter. PROGRAM 10-7 show how a method returns an array.

PROGRAM 10-7

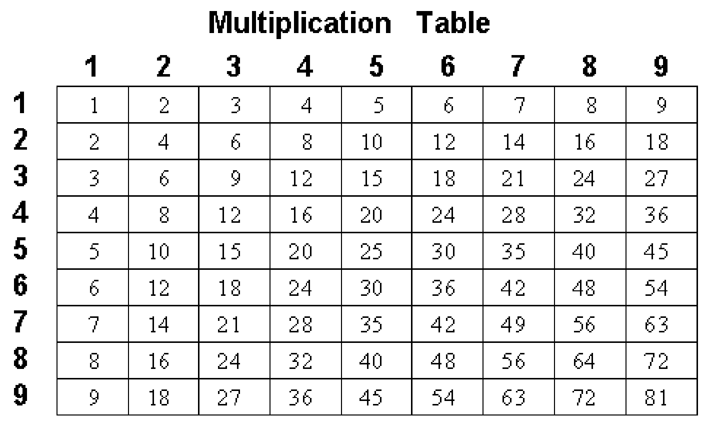
```
public class TestDoWhile {
 public static void main(String[] args) {
  int[] num = {1,2,3,4,5};
  int[] revNum;
  revNum = reverse(num);
  for(int i: revNum)
      System.out.println(i);
 }
 public static int[] reverse(int[] list) {
   int[] result = new int[list.length];
   for (int i = 0, j = result.length - 1; i < list.length; i++, j--)
        result[j] = list[i];

   return result;
 }
}
```

The array result stored the elements in descending order which is 5, 4, 3, 2, 1. The reference from result is assigned to `revNum`. Thus, when the program prints out the elements in `revNum`, it prints out the elements in reverse order.

10.12 Create 2 Dimensional (2D) Arrays

Two-dimensional arrays are useful in representing tabular information as shown in Figure



Multiplication Table

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

Figure 10.6 Tabular information

The syntax to declare and instantiate a 2D array as such:

```
<data type>[][] <array name> = new <data type>[<rowSize>][<colSize>];
```

The following statements create a logical table of cell. See Figure 10.7.

```
double[][] payScaleTable = new double[4][5];
```

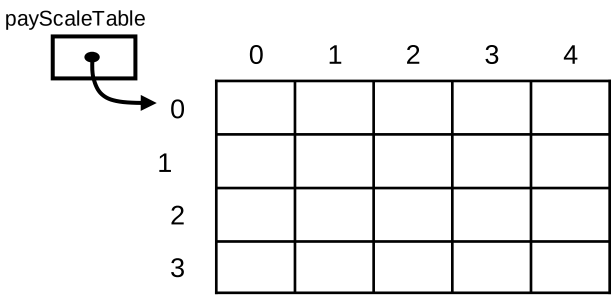


Figure 10.7 Creating a Logical Table of Cells

An element in a two-dimensional array is accessed by its row (pri) and column (sec) index. See Figure 10.8.

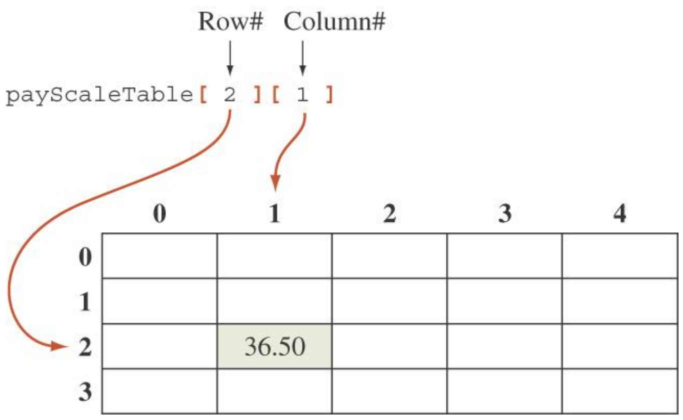


Figure 10.8 Accessing a Cell in a 2D array

In Java, a 2D array such as `payScaleTable` is actually an array of arrays. The above `payScaleTable` array is actually a 1D array of size 4, whose base type is a double 1D array of size 5. See Figure 10.9 for illustration.

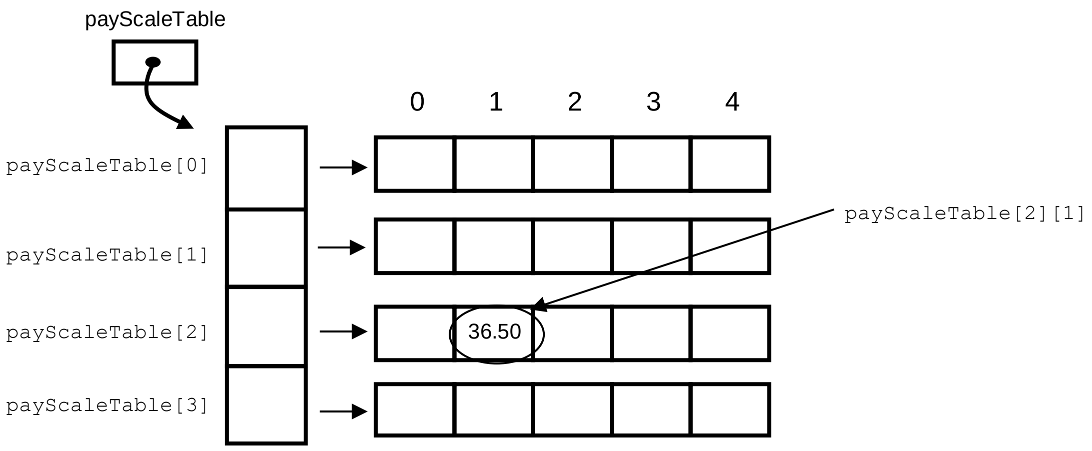


Figure 10.9 Implementation of a 2D Array

The sample array creation

```
double[][] payScaleTable = new double[4][5];
```

is really a shorthand for

```
double[][] payScaleTable = new double [4][];

payScaleTable[0] = new double[5];
payScaleTable[1] = new double[5];
payScaleTable[2] = new double[5];
payScaleTable[3] = new double[5];
payScaleTable[4] = new double[5];
```

Commonly, 2D arrays are accessed using a nested for loop:

```
double[] average = { 0.0, 0.0, 0.0, 0.0 };
for (int i = 0; i < payScaleTable.length; i++) {
   for (int j = 0; j < payScaleTable[i].length; j++) {
      average[i] += payScaleTable[i][j];
   }
   average[i] = average[i] / payScaleTable[i].length;
   System.out.println(average[i]);
}
```

Note that `payScaleTable.length` will give the number of ROWS in the 2D array.
While `payScaleTable[i].length` will give the number of COLUMNS in the 2D array.

10.13 Ragged Arrays

There is no need for each row in a 2D array to have the same number of entries. Different rows can have different number of columns. Such arrays are known as **ragged arrays**. See Figure 10.10 for declaring a ragged array.

```
String[][] s = new String[4][];
s[0] = new String[2];
s[1] = new String[2];
s[2] = new String[4];
s[3] = new String[3];
```

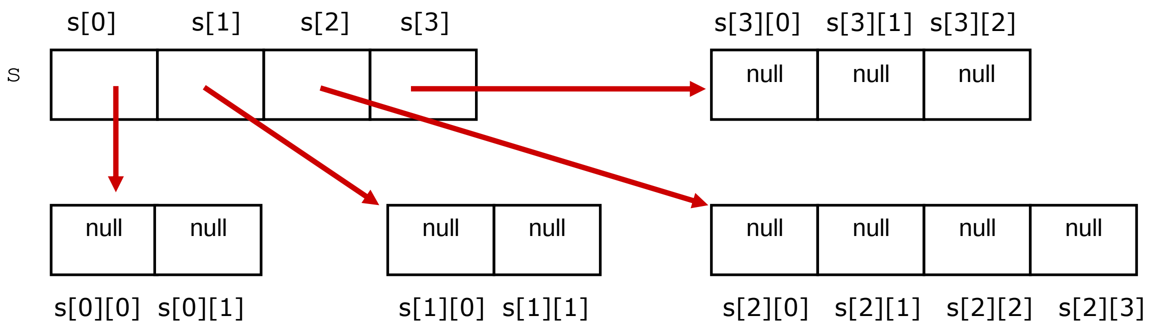


Figure 10.10 Declaration of A Ragged Array

The following shows how to declare and initialize a ragged array. See Figure 10.11.

```
int[][] c = { {1, 2}, {8, 9},{3, 8}, {12, 1, 7} };
```

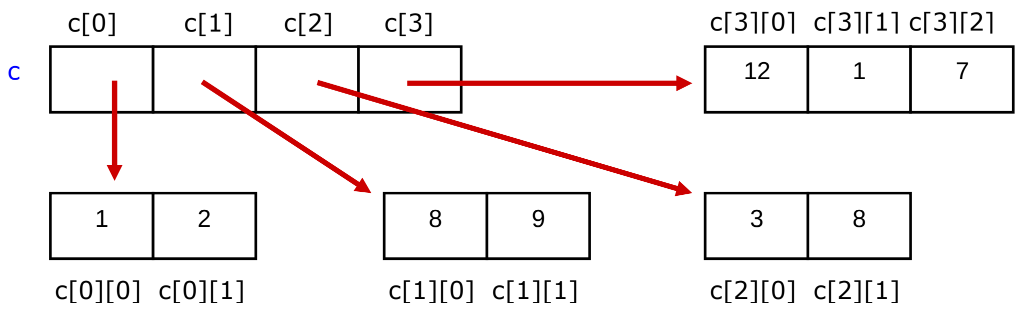


Figure 10.11 Initialization of A Ragged Array

Suppose we would like to create a triangular array (See Figure 10.12). The following code illustrates how the creation is carry out.

```
triangularArray = new double[4][ ];
for (int i = 0; i < 4; i++)
   triangularArray[i] = new double [i + 1];
```

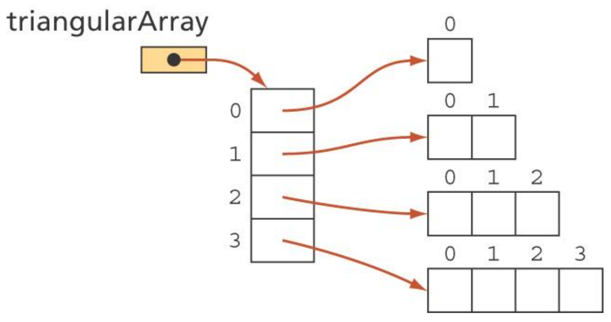


Figure 10.12 Creating a Triangular Array

10.14 Conversion between Strings and Arrays (Self-Reading)

Strings are not arrays, but a string can be converted into an array, and vice versa. To convert a string into an array of characters, use the `toCharArray()` method. For example, the following statement converts the string Java to an array.

```
char[] chars = "Java".toCharArray();
```

Thus, chars[0] is J, chars[1] is a, chars[2] is v, and chars[3] is a.

You can also use the `getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)` method to copy a substring of the string from index `srcBegin` to index `srcEnd-1` into a character array `dst` starting from index `dstBegin`.

For example, the following code copies a substring "3720" in "CS3720" from index 2 to index 6-1 into the character array `dst` starting from index 4.

```
char[] dst = {'J' , 'A' , 'V' , 'A' , '1' , '3' , '0' , '1' };
"CS3720".getChars(2, 6, dst, 4);
```

Thus, dst becomes {'J', 'A', 'V', 'A', '3', '7', '2', '0'}.

To convert an array of characters into a string, use the `String(char[])` constructor or the `valueOf(char[])` method.

For example, the following statement constructs a string from an array using the String constructor.

```
String str = new String(new char[]{'J' , 'a' , 'v' , 'a' });
```

The next statement constructs a string from an array using the valueOf method.

```
String str = String.valueOf(new char[]{'J' , 'a' , 'v' , 'a' });
```

[Reference]

[1] Introduction to Java Programming Comprehensive Version 10th Ed, Daniel Liang, 2016.
[2] Java How To Program 10th Ed, Paul Deitel, Harvey Deitel, 2016

.