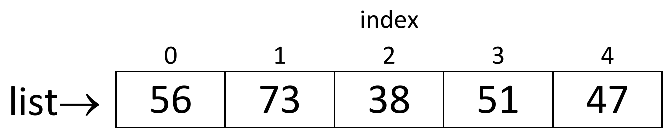
Chapter 6 - Arrays

Sections	Pages	Review Questions	Programming Exercises
6.1-6.11	180-218	1-9,11-14,16-28	2-28 even

Array Introduction

1. An *array* is a place to store things. It is like a filing cabinet turned on its side, or a row of drawers in a tackle box. Shown below is an array whose name is *list* and contains integers.



2. We store things in an array that belong together, e.g. an array of test scores. An array is composed of *elements* in numbered (starting at 0) positions. For instance, `list[0]` is the first element in the array, with value 56. Sometimes we refer to the *elements* in an array as *cells* or *items* or *values*.

3. We create an array by specifying its size and data type:

```
dataType[] arrayName = new dataType[arraySize];
```

For instance, the example above:

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

More on this later.

4. The total number of items that an array can hold is given by:

```
list.length
```

In the example above, *list.length* is 5. *length* is referred to as a *property* of the list array. In some ways, it is similar to a method, but one difference is that we don't use () at the end. *length* is, more specifically a *read-only property* which means that we can *read* (*access* or *get*) it's value, but we cannot *write* (*set* or *change*) it. Thus, the size of the array is *fixed*; it never changes.

Allowed	Not Allowed
`int x = list.length`	`list.length = 12`

5. We use an *index* to access individual *elements* in the array. In an array of length *n, t*he indices run from 0 to *n-1*. We say that arrays in Java are *zero-based*. When we speak, or write algorithms, however, it is convenient to refer to the *first* or the *last* element in the array. Thus, we must memorize the pattern shown below.

position	first	second	third	...	next to, next to last	next to last	last
reference	list[0]	list[1]	list[2]	...	list[list.length-1] - 2	list[list.length-1] - 1	list[list.length-1]
index	0	1	2	...	(n-1)-2	(n-1)-1	n-1
list→	334	482	145	...	873	479	562

In general, the *j*th item in the array has index *j-1*. Occasionally, people will use position when they are really referring to the index. You will learn to pick up the difference. For this class, however, I will stick to the interpretation above.

6. You can *read* and *write* element *values*.

Read	Write
`int x = list[2];`	`list[0] = 99;`

7. Some more examples:

index

	0	1	2	3
names→	“Dave”	“Anna”	“Paul”	“Amber”

index

	0	1	2	3	4	5
isVegetarian→	false	false	true	false	true	false

index

	0	1	2	3	4	5	6	7	8	9
maritalStatus→	‘M’	‘S’	‘M’	‘M’	‘D’	‘M’	‘S’	‘D’	‘M’	‘S’

index

	0	1	2
salaries→	42.85	55.23	22.78

Creating an Array

1. We *declare* an array like this:

```
        dataType[] arrayName;
```

For example:

```
        double[] salaries;
        boolean[] isVegetarian;
        int[] testScores;
        String[] names;
        char[] maritalStatus;
```

Arrays can be of *any type* and always contain the *same type of values*. The statements above do not *create* an array, they *declare* an array. They tell the java compiler that we will *refer to array* in our method. The array's name is *not the array itself*. It is a *reference* (*pointer*) to the array. The array is located at an *address* in memory. The *array name* contains the *value* of this address.

2. We create an array like this, in two lines of code:

```
        dataType[] arrayName;
        arrayName = new dataType[arraySize];
```

or in a single line of code:

```
        dataType[] arrayName = new dataType[arraySize];
```

Sometimes the first option is the right choice and other times, the second one is. Later, we will see examples that sort this out. Some examples of declaring and creating arrays:

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

        boolean[] isVegetarian = new boolean[18];

        int size = scanner.nextInt();
        int[] testScores = new int[size];
```

3. When an array is created, all elements have a *default value*: 0 for arrays with a numeric data type, *false* for boolean, and *null* for strings.

4. If you try to *print* an array with a statement as shown below, you get the *address in memory* of the beginning of the array in hexidecimal format.

```
System.out.println( myArray );          --> ...@10b62c9
```

Processing an Array

1. Frequently, we need to visit each element in an array to accomplish some task (*e.g.* sum all the elements). We say: *loop through the elements in the array* or *iterate over the elements in the array.* A *for* loop can be used to iterate over an array. These are some examples of very basic things you should be able to do with an array:

```
// Print all elements in array.
for( int i=0; i<list.length; i++)
        System.out.println( list[i] );

// Set array to random values
for( int i=0; i<list.length; i++)
        list[i] = Math.random() * 100;

// Sum the elements in an array
sum = 0;

for( int i=0; i<list.length; i++)
        sum += list[i];

// Count the elements that are less than 10
count = 0;

for( int i=0; i<list.length; i++)
     if ( list[i] < 10 )
        count++;

// Find the largest value in array.
max = list[0];

for( int i=0; i<list.length; i++)
        if ( max < list[i] )
              max = list[i];

// Find the index of the largest value in array.
max = list[0];
iMax = 0;

for( int i=0; i<list.length; i++)
{
        if ( max < list[i] )
        {
              max = list[i];
              iMax = i;
        }
}
System.out.prinln( iMax + “ “ + list[iMax] );

// Count the number of vegetarians.
for( int i=0; i<isVegetarian.length; i++)
        if ( isVegetarian[i] )
              sum++;

// Reduce all the values by 15%
for( int i=0; i<list.length; i++)
        list[i] *= 0.85;
```

Examples 1

1. Example: Suppose that we want to read 5 test scores from the user and store them in an array:

```
// Declare and create array.
     int[] scores = new int[5];

     Scanner in = new Scanner(System.in);

// Read scores from user.

//   for( int i=0; i<5; i++ )  // bad, why?

     for( int i=0; i<scores.length; i++ )
     {
          System.out.print( "Enter score " + (i+1) + ": " );

          scores[i] = in.nextInt();
     }
```

2. Example: Suppose that we want to read an arbitrary number of test scores from the user and store them in an array. One way is to first prompt the user for the number of scores they want to enter, then create the array, then read in the values:

```
// Declare array of scores.
     int[] scores;
     int numScores;

     Scanner in = new Scanner(System.in);

// Get number of scores from user.
     System.out.print( "How many scores do you want to enter? " );
     numScores = in.nextInt();

// Create scores array
     scores = new int[ numScores ];

// Read test scores from user.

     for( int i=0; i<scores.length; i++ )
     {
          System.out.print( "Enter score " + (i+1) + ": " );

          scores[i] = in.nextInt();
     }
```

3. Example: Sometimes, we want to allow the user to enter in an arbitrary number of elements, but we prefer not to prompt the user for the number of elements in the array. In this case, we must create an array of sufficient size. For instance, with test scores, we may assume that a class will never have more than 30 students. As we read in values (or add them some other way), we will have to keep track of the number of elements in the array ourselves, using a variable.

```
public static void main(String[] args)
{
    int[] scores = new int[ 30 ];
    int numScores = 0, score=0;

    Scanner in = new Scanner(System.in);

    while ( score != -1 )
    {
        System.out.print( "Enter score " + (numScores+1) +
                          " or -1 to quit: " );
        score = in.nextInt();

        if ( score != -1 ) scores[numScores++] = score;
    }
    System.out.println( "Num Scores: " + numScores );
    System.out.println( "Scores: " );

    for( int i=0; i<numScores; i++ ) System.out.println( scores[i] );
}
```

Initializing an Array

1. One way to initialize an array is to use an *array initializer*. For example, the statement below creates an array with the five elements shown:

```
int[] scores = {92,87,96,73,65};
```

Other examples:

```
double[] weights = {145.32, 223.81, 189.36};

boolean[] isSeniorCitizen = {true, true, false, true, true, false };

String[] names = {“Dave”, “Sue”, “Michelle”, “Lonny”, “Albert” };

char[] grades = {‘C’, ‘B’, ‘A’, ‘A’, ‘D’, ‘A’, ‘B’, ‘F’ };
```

foreach Loop

1. To loop through all the elements in an array, we can use the *for* loop as shown earlier:

```
for( int i=0; i<scores.length; i++ )
{
      // Do something with scores[i]
}
```

or, you can use a *foreach* loop:

```
for( int score : scores )
{
      // Do something with score.

      System.out.println( score )
}
```

We read this, “for each score in scores.” **Note, however, that you cannot change the elements in an array using the *foreach* loop, you can only read their values.**

Examples 2

1. Problem 6.6. A more efficient approach for determining if a number, n is prime is to check whether any of the prime numbers less than or equal to $\sqrt{n}$ are divisors of n. If not, n is prime. Write a program to find the first 50 prime numbers using this approach.

   a. Algorithm:

```
Create array to hold 50 primes
Initialize first prime = 2, and prime count = 1
Initialize n=2, the current candidate prime

Loop until 50 primes are found
    Increment n
    Loop over primes until current prime is less than or equal to $\sqrt{n}$
        If n is divisible by current prime
            break (i.e. n is not prime)
        Current prime = next prime
    If n is prime
        Increment count
        Save n in prime array.
Print primes
```

b. Code:

```
// Create array to hold 50 primes
int[] primes = new int[50];
// Initialize first prime = 2, and prime count = 1
primes[0] = 2;
int count=1;
// Initialize the current candidate prime
int curN=2;
boolean isPrime;

// Loop until 50 primes are found
while( count < primes.length )
{
   // Increment current candidate prime
   curN++;
   isPrime = true;
   // Set increment to first prime
   int j=0;

   // Loop over primes until current prime is
   // less than or equal to sqrt(currentPrime)
   while( primes[j] <= (int)Math.sqrt(curN) )
   {
      // If candidate prime is divisible by current prime..
      if ( curN % primes[j] == 0 )
      {
         // End loop and mark current candidate as not prime
         isPrime = false;
         break;
      }
      // Increment to next prime
      j++;
   }
   // If candidate prime is prime, save prime and
   // increment count of primes
   if( isPrime )primes[count++] = curN;
}
// Print primes
for( int val : primes )

   System.out.print( val + " " );
```

Passing an Array to a Method

1. We can pass arrays to methods similar to the way we pass variables to a method:

```
public static void main(String[] args)
{
   int[] scores = {82, 91, 45, 85};

   printArray( "Scores", scores );
}

public static void printArray( String title, int[] x )
{
   System.out.print( title + ": " );

   for( int val : x )

      System.out.print( val + " " );

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

2. Another example:

```
public static void main(String[] args)
{
   int[] scores = {82, 91, 45, 85};

   int min = findMinimum( scores );
}

public static int findMinimum( int[] x )
{
   int min = 1000;

   for( int val : x )
      if ( val < min ) min = val;

   return min;
}
```

Examples 3

1. Rewrite problem 6.6 (Examples 2 above) using a method to determine if a number is a prime.
   a. Analysis:

      The specification asks use to write a method that will, “determine if a number is a prime.” This implies that the method will accept an integer as input and return either *true* or *false*.

      Upon inspection of the previous code, we see that such a method will also need as input the *primes* array.

b. Algorithm for *main*:

```
Create array to hold 50 primes
Initialize first prime = 2, and prime count = 1
Initialize n=2, the current candidate prime

Loop until 50 primes are found
          Increment n

          IF ( isPrime(n,primes) )
                    Increment count
                    Save n in prime array.
Print primes
```

c. Algorithm for *isPrime()*

```
Loop over primes until current prime greater than $\sqrt{n}$
        If n is divisible by current prime
                Return false
        Increment current prime index
Return true
```

d. Code:

```
public static void main(String[] args)
{
   int[] primes = new int[50];
   primes[0] = 2;
   int count=1, curN=2;

   while( count < primes.length )
   {
      curN++;

      if ( isPrime( curN, primes ) ) primes[count++] = curN;
   }
   printArray( "Primes", primes );
}

public static boolean isPrime( int n, int[] primes )
{
      int j=0;

      while( primes[j] <= (int)Math.sqrt(n) )
      {
         if ( n % primes[j] == 0 ) return false;
         j++;
      }
      return true;
}
```

```
   public static void printArray( String title, int[] x )
   {
      System.out.print( title + ": " );

      for( int val : x )

         System.out.print( val + " " );

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

Stack and Heap

1. When an array is passed to a method, it is passed *by reference*. This means that the address of the array is passed to the method. Thus, when the method uses this address it refers to the same array as in the calling program. Thus, we can change the values in an array in a method and when control is returned to the calling method, it can see these changes:

```
   public static void main(String[] args)
   {
      int[] scores = {82, 91, 45, 85};

      reset( scores );

      printArray( "Reset Scores", scores );

   }

   public static void reset( int[] x )
   {
      for( int i=0; i<x.length; i++ ) x[i]=0;
   }
```

2. The *heap* is an area of memory used for *dynamic memory allocation*. An array resides in the heap. A *stack* is an area of memory used for storing information about the *active* methods. Draw stack and heap on the board and show output of program.

```
public static void main ( String[] args )
{
        int[] x = {5,9};

        changeX( x[0], x[1] );

        System.out.println( x[0] + ", " + x[1] );

        changeX( x );
        System.out.println( x[0] + ", " + x[1] );
}

public static void changeX( int a, int b );
{
        a=77;
        b=99;
}
public static void changeX( int[] z );
{
        z[0] = 77;
        z[1] = 99;
}
```

Examples 4

1. Problem 6.12. Write a method that reverses the elements in an array (without creating a new array)

   a. Analysis

	0	1	2	3	4	5
vals→	56	73	38	51	33	47

Swap first and last
Swap second and next to last
...

When do we stop this procedure?

How do we swap two values without losing one of them?

Will this idea work with an odd number of elements?

			index		
	0	1	2	3	4
vals→	56	73	38	51	47

   b. Code

```
public static void reverseArray( int[] vals )
{
   int first = vals[0];
   int tempVal;

   for( int i=0, j=vals.length-1; i<vals.length/2; i++, j-- )
   {
      tempVal = vals[i];
      vals[i] = vals[j];
      vals[j] = tempVal;
   }
}
```

Returning an Array from a Method

1. It is useful to be able to write methods that return an array. Usually, the array that is returned is **created in the method**. For example, the method below will accept an array as an argument and return a new array that has the elements reversed. This example shows a method that accepts an array of integers and returns a new array of integers where the positions are reversed from the original. Thus, we will have two arrays, one with the original values one with the reversed values:

```
      public static void main(String[] args)
      {
         int[] scores = {82, 91, 45, 85};

         int[] revScores;

         revScores = reverseOrder( scores );

         printArray( "Reverse Scores", revScores );
      }

      public static int[] reverseOrder( int[] x )
      {
        int[] y = new int[x.length];

        for( int i=0, j=x.length-1; i<x.length; i++, j-- )
         {
//             y[x.length-i-1] = x[i];

            y[j] = x[i];
         }
         return y;
     }
```

Examples 5

1. Write a method that creates an array of random integers. The user will specify the number of values to create and the range of the random integers by specifying the beginning and ending number. Write a driver that uses this method to generate 25 values between 50 and 100.

```java
public static void main(String[] args)
{
   int[] randAry;

   randAry = createArray( 5, 50, 100 );

   for( int val : randAry ) System.out.println( val );
}

public static int[] createArray( int n, int a, int b )
{
   int[] rand = new int[ n ];

   for( int i=0; i<rand.length; i++ )
   {
      rand[i] = (int)(a + (b-a+1)*Math.random());
   }

   return rand;

}
```

2. Suppose that we have an array where each three elements correspond to a person's first name, middle name, and last name. Write a method that accepts such an array and returns an array where each element corresponds to a person's complete name (first middle last).

```java
public static void main(String[] args)
{
   String[] nameParts = { "Ben", "Roe", "Wright", "Sam", "Len", "Hull" };
   String[] wholeNames;

   wholeNames = buildNames( nameParts );

   for( String name : wholeNames )

      System.out.println( name );
}

public static String[] buildNames( String[] names )
{
   String[] compNames = new String[ names.length / 3 ];

   for( int i=0, j=0; i<names.length/3; i++, j+=3 )
   {
      compNames[i] = names[j] + " " + names[j+1] + " " + names[j+2];
   }

   return compNames;
}
```

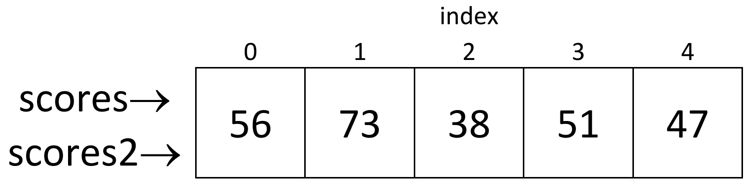
Copying an Array

1. Occasionally, we need to copy an array. It is usually a situation where we are about to do something with an array that will corrupt the elements and we need to keep a copy of the original array around. The code below is legal (and sometimes useful), but will *not* copy an array:

```
int[] scores = new int[5];
int[] scores2;

scores2 = scores;
```

Both *scores* and *scores2* point to the same physical array.



```
System.out.println( scores[3] );      --> 51
System.out.println( scores2[3] );     --> 51

scores[3] = 99;

System.out.println( scores[3] );      --> 99
System.out.println( scores2[3] );     --> 99
```

2. One way to copy an array:

```
int[] scores = new int[5];
int[] scores2 = new int[scores.length]
...
for( int i=0; i<scores.length; i++ )
{
       scores2[i] = scores[i];
}
```

Why might we prefer the *indexed* loop over the *foreach* loop in this case?

```
int i=0;
for( int score : scores )
{
       scores2[i++] = score;
}
```

3. Another, more versatile way to copy an array is to use the *arraycopy() method of the System class*.

```
System.arraycopy( sourceArray,          // array to copy from

                  srcPos,               // position (index) to begin copying from

                  targetArray,          // array to copy to

                  tarPos,               // position (index) to begin copying to in target

                  length )              // number of elements to copy
```

Note: the *targetArray* must already be created.

Example – to copy all of array *a* to array *b*:

```
System.arraycopy( a, 0, b, 0, a.length );
```

4. Example – what is the output of this code?

```
int[] a = new int[6];
int[] b = new int[10];

for( int i=0; i<a.length; i++ )

     a[i] = (i+1)*10;

System.arraycopy(a,2,b,4,3);

for( int val : b )

     System.out.print( val + " " );
```

5. Example – what is the output of this code?

```
int[] c = new int[6];
int[] d = new int[6];

for( int i=0; i<c.length; i++ )
     c[i] = i+1;

System.arraycopy(c,3,d,0,3);

for( int val : d )

System.out.print( val + " " );
```

Examples 6

1. Write a method that accepts an array of integers and returns another array that contains the differences between successive elements. For example: 2, 5, 1, 7 would produce: -3, 4, -6. Create an array with 10 elements with random values between 50 and 100 to test your method.

2. Write a method that accepts an array of characters and swaps elements in positions 1 & 2, 3 & 4, ... if the size of the array is even. Do not create a new array.

3. Write a method that accepts an array of doubles and rotates the elements one position to the right making the first element have the value of the last, the second the value of the first, *etc*. Do not create a new array.

4. Write a method that accepts an array of boolean values and returns the count of the longest string of consequtive *true*'s.

5. Write a method that accepts an array of boolean values and a boolean value, *searchFor*. The method returns the count of the longest string of consequtive *true* or *false*'s, depending on what the value of *searchFor* is.

6. Write a method that accepts an array of integers and returns the middle value. If the array's size is odd, then return the unique middle value. Otherwise, return the average of the two middle values.

7. A program will store the names of students and test scores in arrays. Write a method that accepts an array of test scores and returns the index of the score that is closest to the average. If there are two or more test scores that are equally close to the average, then return the first index. Finally, the program should print the test score and corresponding name.