The Class `Arrays` and Resizing Arrays

13

13.1 The Class `Arrays`

`Arrays` is a class that provides static methods for manipulating and examining arrays. The class `Arrays` belongs to the package `java.util`, so the class must be imported, using one of the following two:

```
import java.util.Arrays;
import java.util.*;
```

Here are some methods of `Arrays`. Let `T` be any data type.

- `boolean Arrays.equals( T[] x, T[] y )`
  This method returns a `boolean` value representing whether or not the two arrays `x` and `y` have the same lengths, and the elements are equal between `x` and `y` at all positions. If `T` is a primitive data type, the equality between elements is tested with `==`. If `T` is an object data type, the result depends on how the equality method, `equals`, is executed in the class `T`.
- `void Arrays.fill( T[] x, T v )`
  This method fills the array `x` with the value `v`.
- `void Arrays.fill( T[] x, int fromIndex, int toIndex, T v )`
  The method stores `v` in the elements `x[ fromIndex ], ..., x[ toIndex - 1]`.
- `T[] Arrays.copyOf( T[] x, int copyLength )`
  This method returns an array consisting of the first `copyLength` elements of the array `x`.
  If the `copyLength` is greater than `x.length`, the method stores the default value of the data type `T` in the remaining `copyLength - x.length` elements of the returned array.
- `void Arrays.sort( T[] x )`

M. Ogihara, *Fundamentals of Java Programming*,
https://doi.org/10.1007/978-3-319-89491-1_13

This method rearranges the elements of x in increasing order. If T is a primitive data type, < is used for comparison, but a tie can be broken arbitrarily. If T is an object data type, T must admit comparison with a method named compareTo (see Chap. 17 for comparable data types).

- `void Arrays.sort( T[] x, int fromIndex, int toIndex )`
  This is a variant of sort. Sorting is applied only on the elements having indexes between fromIndex and toIndex - 1.

The next program demonstrates the use of the methods through a series of actions.

Step 1: The program instantiates a String array of ten elements.
Step 2: The program fills the array with a String literal "abc" using Arrays.fill.
Step 3: At each index of the array, the program generates a random four-letter String data and stores it at the index.
Step 4: The program creates a copy of the array having the same length as the original using Arrays.copyOf.
Step 5: The program compares the original and the copy using Arrays.equals.
Step 6: The program sorts the elements of the copy using Arrays.sort.
Step 7: The program compares the original and the copy using Arrays.copyOf.

After each action, the program announces the action that has been performed, and then prints the contents of the relevant arrays. To print the contents of an array, the program uses a method named print. The method receives an array as its formal parameter, and then prints all the elements in the array in just one line, with one white space attached in front of each element (Line 8).

```
1  import java.util.*;
2  public class ArraysMethods
3  {
4    public static void printArray( String[] data )
5    {
6      for ( int p = 0; p < data.length; p ++ )
7      {
8        System.out.print( " " + data[ p ] );
9      }
10     System.out.println();
11   }
12
```

Listing 13.1 A program that demonstrates the use of methods from class Arrays (part 1). The method for printing an array

To generate a random String in Step 3, the program uses a method named randomString. The method randomString receives an int value as its formal parameter, len, and returns a random String object having length len (Line 13). To accomplish this, the method generates len random lowercase letters and connects them into one String data. A for-loop is used to count from 1 to len (Line 16). To generate a single random letter, the method generates a random relative index between 0 and 25 with the formula appearing in Line 18:

```
    int diff = (int)( Math.random() * 26 );
```

The method then converts the relative index value, `diff`, to the absolute index in the ASCII table by adding it to `'a'`, and then casting `(char)` to the sum, as appearing in the right-hand side of Line 19:

```
    (char)( 'a' + diff )
```

The value of this `char` is appended directly to a `String` variable, `value`. The initial value of this variable is `""` (Line 15). The method at the end returns this `value` (Line 21). If the value of `len` is negative, the method does not execute the loop-body, so the return value of the method is an empty `String`.

```
13  public static String randomString( int len )
14  {
15    String value = "";
16    for ( int j = 1; j <= len; j ++ )
17    {
18      int diff = (int)( Math.random() * 26 );
19      value += (char)( 'a' + diff );
20    }
21    return value;
22  }
23
```

Listing 13.2 A program that demonstrates the use of methods from class `Arrays` (part 2). The method for randomly generating a (gibberish) character sequence of length `len`

The remainder of the code is the method `main`. The method `main` performs the following seven-step action.

1. After creating the initial array named `data` (Line 26), the program announces that it has created an array (Line 27), and then prints the contents of `data` (Line 28).
2. The program fills `data` with `"abc"` (Line 30), announces that it has filled `data` (Line 31), and then prints the contents of `data` (Line 32).
3. The program fills `data` with random elements (Lines 34–37), announces that it has stored random elements (Line 38), and then prints the contents of `data` (Line 39).
4. The program creates a copy of `data` having the same length, stores it in `copied` (Line 41), announces that it has created a copy (Line 42), and then prints the contents of `copied` (Line 43).
5. The program prints the result of comparing between `data` and `copied` (Line 45).
6. The program sorts `copied` (Line 47), announces that it has sorted `copied` (Line 48), and then prints the contents of `copied` (Line 49).
7. The program prints the result of comparing between `data` and `copied` (Line 50).

The probability that an 10-element array of random `String` data is already sorted is $10!/10^{10}$. The probability that the value of the equality test is still `true` in (7) is thus approximately 0.000363. This means that we can anticipate that the result of the equality test is very likely to change from `true` to `false`.

The code for the method main is presented next, with the calls to Arrays methods highlighted:

```
24    public static void main( String[] args )
25    {
26      String[] data = new String[ 10 ];
27      System.out.println( "Before assignment" );
28      printArray( data );
29
30      Arrays.fill( data, "abc" );
31      System.out.println( "Filled with \"abc\"" );
32      printArray( data );
33
34      for ( int pos = 0; pos < 10; pos ++ )
35      {
36        data[ pos ] = randomString( 4 );
37      }
38      System.out.println( "Filled with random Strings" );
39      printArray( data );
40
41      String[] copied = Arrays.copyOf( data, data.length );
42      System.out.println( "A copy has been generated" );
43      printArray( copied );
44
45      System.out.printf( "Equality:%s%n", Arrays.equals( data, copied ) );
46
47      Arrays.sort( copied );
48      System.out.println( "The copy has been sorted" );
49      printArray( copied );
50      System.out.printf( "Equality:%s%n", Arrays.equals( data, copied ) );
51    }
52  }
```

Listing 13.3 A program that demonstrates the use of methods from class Arrays (part 3). The method main

Here is an execution example of the code:

```
1  Before assignment
2   null null null null null null null null null null
3  Filled with "abc"
4   abc abc abc abc abc abc abc abc abc abc
5  Filled with random Strings
6   sbbo zgqz qzwa wgjj kvii nyff tcot ided hugg ndgj
7  A copy has been generated
8   sbbo zgqz qzwa wgjj kvii nyff tcot ided hugg ndgj
9  Equality:true
10 The copy has been sorted
11  hugg ided kvii ndgj nyff qzwa sbbo tcot wgjj zgqz
12 Equality:false
```

13.2 Reordering Elements in an Array

Programmers often encounter a situation where the order of some elements in an array must be rearranged. Such a task can be accomplished by:

- instantiating a new array with the same length as the original,
- storing the elements of the original in the new array at their respective new positions, and then,
- replacing the original array with the new array.

In addition, if either the number of elements that are affected by the rearrangement is small or the rearrangement is highly regular, the task can be accomplished without using another array. We call rearranging the order without using another array **reordering elements in place**. A key idea in reordering elements in place is the **element swap**. The element swap is exchanging values between two elements.

Suppose `x` and `y` are variables of some type `T`. To swap the values between `x` and `y`, we can use a temporary storage, say `temp` of the same type `T`, and execute:

```
1 T temp = x;
2 x = y;
3 y = temp;
```

or in the reverse order:

```
1 T temp = y;
2 y = x;
3 x = temp;
```

We can use this technique for exchanging values between two elements of an array. Suppose `myData` is an array of type `T`. Suppose, also, `p` and `q` are valid indexes of this array. In other words, the values of `p` and `q` are between 0 and `myData.length - 1`. We can swap values between `myData[ p ]` and `myData[ q ]` using a temporary storage, `temp`, of type `T` as follows:

```
1 T temp = myData[ p ];
2 myData[ p ] = myData[ q ];
3 myData[ q ] = temp;
```

The following program demonstrates the use of this technique for element swapping.

The program uses an array literal. The syntax for creating an array literal of elements of type `T` is:

```
new T[]{ ELEMENT_1, ..., ELEMENT_K }
```

where `ELEMENT_1, ..., ELEMENT_K` are the elements of the array. For example, the code:

```
1 String[] a;
2 int[] b;
3 a = new String[]{ "ABC", "DEF", "GHI" };
4 b = new int[]{ 10, 9, 8, 7, 6 };
```

instantiates an array of `String` data having length three and stores it in `a`, and then instantiates an array of `int` data having length five and stores it in `b`.

In the program, we use a 16-element array of String data, words. Initially, the elements of the array are the words from the title of a Pink Floyd tune: *Several Species of Small Furry Animals Gathered Together in a Cave and Grooving With a Pict* from their album *Ummagumma*.[1] We then swap elements between indexes 3 and 5.

The code demonstrates the modification by printing the elements of the array each time a change takes place on it. To print the contents, it uses a method named printArray (Line 3). The formal parameter of the method is an array of String data named words. The method uses a for-loop that iterates over the sequence 0, ..., words.length - 1 with a variable named i (Line 5), and prints each element with one white space preceding it. As we have done previously, the format to be used for inserting the white space is " " + words[index]. Since the length of words can be possibly large, the program prints the newline after printing eight elements successively in one line as well as after printing all the elements. Since the value of i starts from 0, after printing eight elements successively in one line, the value of i has the remainder 7 when divided by 8. Since the last valid index is words.length - 1, the condition for testing if the newline character must be printed is

```
i % 8 == 7 || i = words.length - 1
```

```
1  public class ArraySwap
2  {
3    public static void print( String[] words )
4    {
5      for ( int i = 0; i < words.length; i ++ )
6      {
7        System.out.print( " " + words[ i ] );
8        if ( i % 8 == 7 || i == words.length - 1 )
9        {
10         System.out.println();
11       }
12     }
13   }
```

Listing 13.4 A program that swaps elements between two positions in an array (part 1). The method print

The main part of the program goes as follows:

1. Create a String[] literal words that holds the words from the title of the song, using the syntax for creating an array literal (Lines 16–19), and then print the contents of the array (Line 20).
2. Copy the value at position 3 to a temporary variable named temp (Line 22).
3. Copy the value at position 5 to position 3 (Line 24), and then print the contents of the array (Line 24).
4. Finalize the swap by copying the value from temp to position 5 (Line 26), and then print the contents (Line 27).

[1] Pink Floyd is a British Rock band that was formed in 1968 and disbanded in 2012. It is one of the most successful rock bands in history. *Ummagumma* is their double vinyl album that was issued in 1969.

```
14    public static void main( String[] args )
15    {
16      String[] words = new String[]{
17        "Several", "Species", "of", "Small", "Furry", "Animals",
18        "Gathered", "Together", "in", "a", "Cave", "and",
19        "Grooving", "With", "a", "Pict" };
20      print( words );
21
22      String temp = words[ 3 ];
23      words[ 3 ] = words[ 5 ];
24      print( words );
25
26      words[ 5 ] = temp;
27      print( words );
28    }
29  }
```

Listing 13.5 A program that swaps elements between two positions in an array (part 2). The method `main`

Figure 13.1 illustrates the series of actions performed on the array elements.

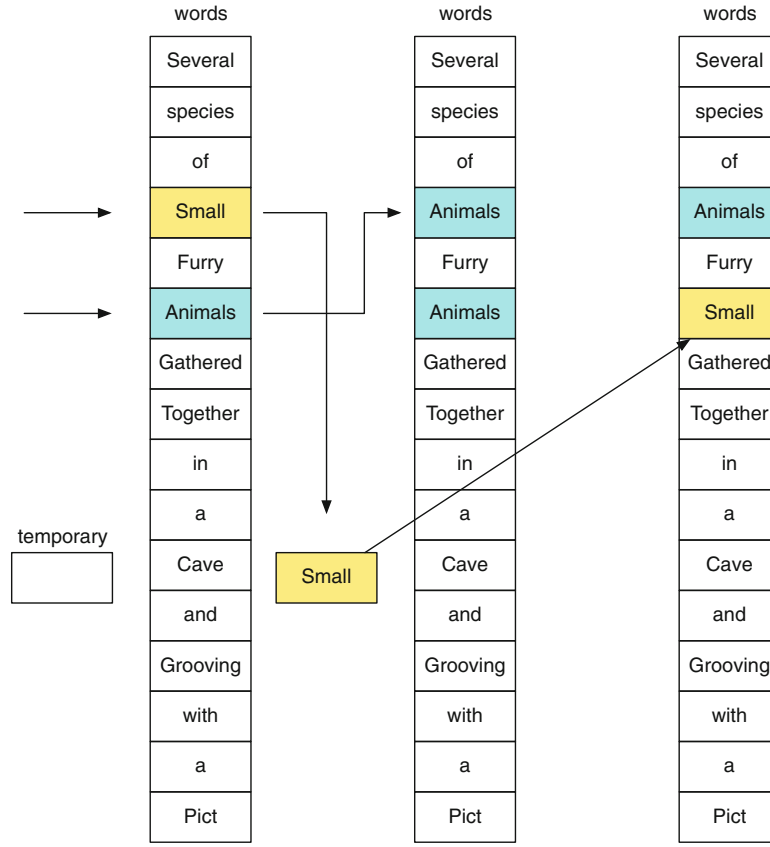

Fig. 13.1 Swapping values between two array elements

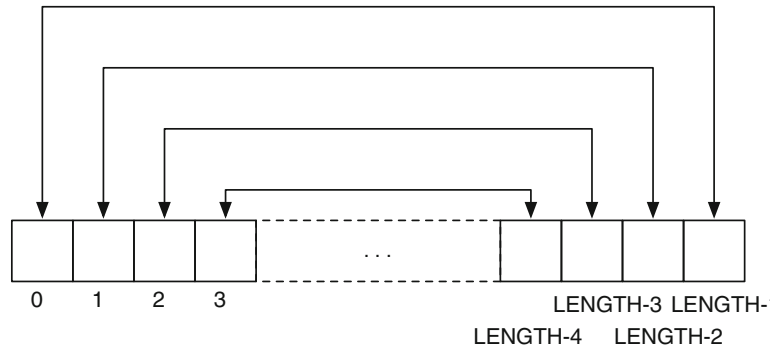

Fig. 13.2 Reversing the order of appearance of elements in an array

The result of executing the code is as follows:

```
Several Species of Small Furry Animals Gathered Together
in a Cave and Grooving With a Pict
Several Species of Animals Furry Animals Gathered Together
in a Cave and Grooving With a Pict
Several Species of Animals Furry Small Gathered Together
in a Cave and Grooving With a Pict
```

13.2.1 Reversing the Order of Elements

Suppose `data` is an array of type `T` and we want to reverse the order in which the elements appear in `data`. To accomplishing the task, we swap elements between indexes `i` and `data.length - 1 - i` for all values of `i` between 0 and `data.length / 2 - 1`.

```
T temp;
for ( int i= 0; i <= data.length / 2 - 1; i ++ )
{
  temp = data[ i ];
  data[ i ] = data[ data.length - 1 - i ];
  data[ data.length - 1 - i ] = temp;
}
```

Figure 13.2 visualizes the action of the reversal by exchange. The choice of the maximum value `data.length / 2 - 1` is important. In the case where `data.length` is an even number, say `2 * m`, the innermost exchange is between the indexes `m - 1` and `m`. In the case where `data.length` is an odd number, say `2 * m + 1`, the innermost exchange is between the indexes `m - 1` and `m + 1`, so the unique middle element, the one located at index `m`, is untouched.

13.2.2 Cyclic Shifting

Another application of element swapping is **cyclic shifting**. Cyclic shifting is to move all the elements in one direction and placing the displaced element at the other end. Depending on whether the elements move to lower indexes or higher indexes, we call the action the **left cyclic shift** and the **right cyclic shift**. Here is the more formal definition of cyclic shifting. Suppose we have an array named `x` having length `n` at hand.

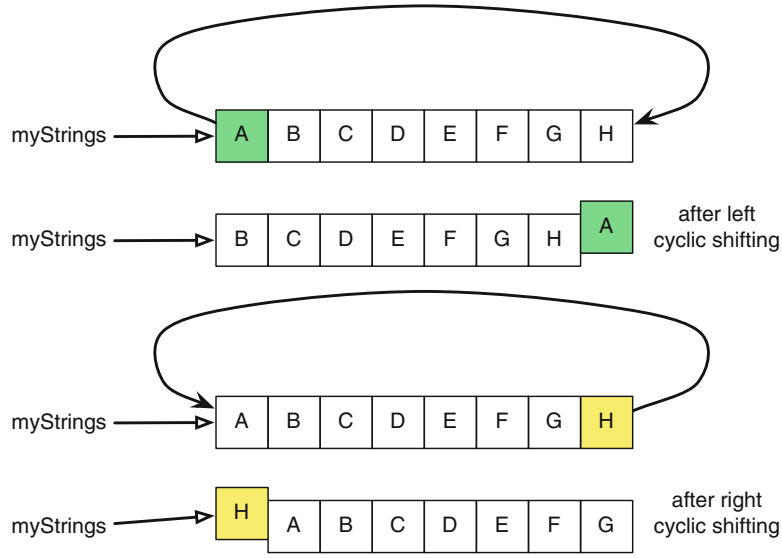

Fig. 13.3 The results obtained by executing cyclic shifts

- By **left cyclic shifting** of `x`, we mean to move, concurrently, `x[ 1 ], ..., x[ n - 1 ]` to `x[ 0 ], ..., x[ n - 2 ]` and `x[ 0 ]` to `x[ n - 1 ]`.
- By **right cyclic shifting** of `x`, we mean to move, concurrently, `x[ 0 ], ..., x[ n - 2 ]` to `x[ 1 ], ..., x[ n - 1 ]` and `x[ n - 1 ]` to `x[ 0 ]`.

Figure 13.3 visualizes the two cyclic shifts.

An in-place cyclic shift can be accomplished as follows:

1. Save the element that needs to rotate around (that is, the element that needs to move from one end to the other) in a temporary storage. This creates an unoccupied slot in the array.
2. While the destination of the element in the temporary storage becomes unoccupied, find the element that needs to move into the slot that is presently unoccupied, and move the element to its destination.
3. Moving the displaced element to its destination.

The program `ArrayCyclicShifting` demonstrates both left and right cyclic shifting using the above algorithm of successive relocation (Fig. 13.4). The program creates an array of random integers between 0 and 99, and then performs a series of cyclic shifts while printing the contents of the array at the beginning and after each cyclic shift. The first part of the source code is the method `printArray`. This method prints the elements of an array of `int` data that is given as the parameter (Line 3). The method prints all the elements of the array in one line by using the `printf` format of `" %2d"`. Since the elements of the array are between 0 and 99, exactly three character spaces will be used for an element.

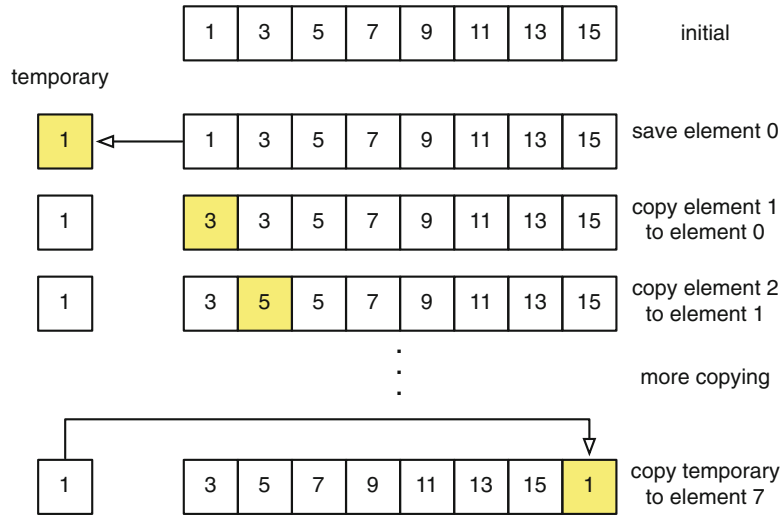

Fig. 13.4 An algorithm for left cyclic shift

```
1  public class ArrayCyclicShift
2  {
3    public static void printArray( int[] numbers )
4    {
5      for ( int index = 0; index < numbers.length; index ++ )
6      {
7        System.out.printf( " %2d", numbers[ index ] );
8      }
9      System.out.println();
10   }
11
```

Listing 13.6 A program that performs cyclic shifting (part 1). The method `printArray`

Next, we present the method for left cyclic shift, `leftCyclicShift`. The method has an array of `int` data, `numbers`, as its formal parameter (Line 12). The algorithm that method executes is as follows:

1. Store `numbers[ 0 ]` in an `int` variable named `temporary` (Line 15).
2. Using a for-loop that iterates over the sequence `1, ..., numbers.length - 1` with a variable named `index`, store the value of `numbers[ i ]` in `numbers[ i - 1 ]` (Lines 16–19).
3. Store the value in `temporary` in `numbers[ numbers.length - 1 ]` (Line 20).

After completing the shift, the method prints the contents of the array using the method `printArray` (Line 21).

```
12    public static void leftCyclicShift( int[] numbers )
13    {
14      System.out.println( "Left Cyclic Shift" );
15      int temporary = numbers[ 0 ];
16      for ( int index = 1; index < numbers.length; index ++ )
17      {
18        numbers[ index - 1 ] = numbers[ index ];
19      }
20      numbers[ numbers.length - 1 ] = temporary;
21      printArray( numbers );
22    }
23
```

Listing 13.7 A program that performs cyclic shifting (part 2). The method for left cyclic shifting

The method for right cyclic shifting, `rightCyclicShift`, moves the elements in the opposite direction (Line 24), by executing the following algorithm:

1. Store `numbers[ numbers.length - 1 ]` in an `int` variable named `temporary` (Line 27).
2. Using a for-loop that iterates over the sequence `numbers.length - 2, ..., 0` with a variable named `index`, store the value of `numbers[ i ]` in `numbers[ i + 1 ]` (Lines 28–31).
3. Store the value saved in `temporary` in `numbers[ 0 ]` (Line 32).

```
24    public static void rightCyclicShift( int[] numbers )
25    {
26      System.out.println( "Right Cyclic Shift" );
27      int temporary = numbers[ numbers.length - 1 ];
28      for ( int index = numbers.length - 1; index >= 1; index -- )
29      {
30        numbers[ index ] = numbers[ index - 1 ];
31      }
32      numbers[ 0 ] = temporary;
33      printArray( numbers );
34    }
35
```

Listing 13.8 A program that performs cyclic shifting (part 3). The method for right cyclic shifting

The last part of the code is the method `main`. The method `main` does the following:

1. Instantiate an array of 20 elements (Line 38) with random integers in the range 0..99 (Lines 39–42).
2. Print the contents of the array (Line 44).
3. Execute the right cyclic shift method twice (Lines 46 and 47). (The right cyclic method prints the contents of the array before returning.)
4. Execute the left cyclic shift once (Line 48). (The left cyclic method prints the contents of the array before returning.)

```
36    public static void main( String[] args )
37    {
38      int[] numbers = new int[ 20 ];
39      for ( int index = 0; index < numbers.length; index ++ )
40      {
41        numbers[ index ] = (int)( Math.random() * 100 );
42      }
43      System.out.println( "Original" );
44      printArray( numbers );
45
46      rightCyclicShift( numbers );
47      rightCyclicShift( numbers );
48      leftCyclicShift( numbers );
49    }
```

Listing 13.9 A program that performs cyclic shifting (part 4). The method `main`

Here is the result of executing the program:

```
1  Original
2    1 49 24 90 46 58 56  2 58 27 34 64 73 20 33 48 15  0 12 10
3  Right Cyclic Shift
4   10  1 49 24 90 46 58 56  2 58 27 34 64 73 20 33 48 15  0 12
5  Right Cyclic Shift
6   12 10  1 49 24 90 46 58 56  2 58 27 34 64 73 20 33 48 15  0
7  Left Cyclic Shift
8   10  1 49 24 90 46 58 56  2 58 27 34 64 73 20 33 48 15  0 12
```

13.3 Modifications of an Array That Require Resizing

13.3.1 Insertion and Deletion

Once an array has been instantiated, its length cannot be changed. To add an element to an existing array or to remove an element from an array, one must create a new array where the elements appear in their designated places, and then replace the array is currently used with the new one. Suppose the array that is currently used has the name `oldArray`. Here are specific steps to follow to insert an element, say `x`, to this array at some index `p`.

1. Instantiate a new temporary array, say `newArray`.
2. For all indexes `i` that are strictly smaller than `p`, copy `oldArray[ i ]` to `newArray[ i ]`.
3. Place `x` in `newArray[ p ]`.
4. For all indexes `i` that are strictly greater than `p`, copy `oldArray[ i ]` to `newArray[ i + 1 ]`.
5. Assign `newArray` to `oldArray`.

Figure 13.5 visualizes this action. In the middle three steps, there is no overlap among the destinations of the elements, so the orders of the three steps can be permuted.

To remove the element at some index `p`, we execute the following algorithm:

1. Instantiate a new temporary array with a different name, say `newArray`.
2. For all indexes `i` that are strictly smaller than `p`, copy `oldArray[ i ]` to `newArray[ i ]`.

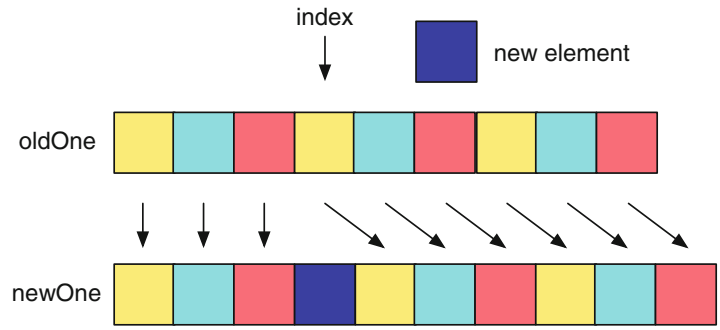

Fig. 13.5 Insertion of an element in an array

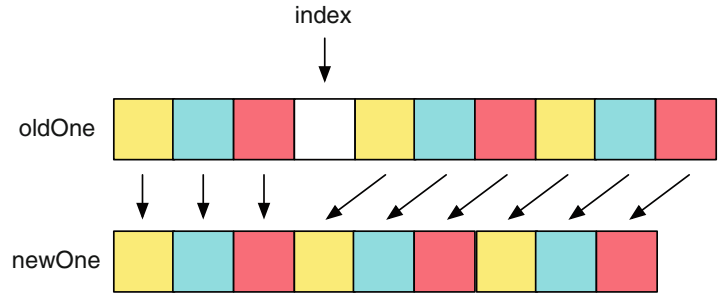

Fig. 13.6 Deleting an element from an array

3. For all indexes `i` that are strictly greater than `p`, copy `oldArray[ i ]` to `newArray[ i - 1 ]`.
4. Set `newArray` to `oldArray`.

We can visualize this action in a figure like Fig. 13.6. In the middle two steps, there is no overlap among the destinations of the elements, so the order of the two steps can be switched.

The following program, `ResizeArray`, demonstrates the methods for insertion and removal we have just discussed using arrays of `String` values. The program uses an array, `data`, for maintaining a list of `String` values. The program initializes the array by receiving its length and individual elements from the user. The program then repeats interactions with the user to (a) insert an element to the array, (b) remove an element from the array, or (c) present the elements of the array. This is repeated until the user instructs the program to terminate. The insertion and removal are carried out by methods, `insert` and `remove`. Both `insert` and `remove` return a new array constructed from the original by making the requested modifications. The method `main` calls these two methods, and replaces `data` with the array that the methods return.

The first part of the code is the method `insert`. The formal parameters of the method are a `String` array named `data`, an `int` value named `target`, and a `String` object `w`. As mentioned above, the return type of the method is an array of `String` data (Line 4). The task to be performed is to return an array created from `data` by inserting `w` at index `target`. The method executes the following algorithm to accomplish the task:

1. Create a new array named `newArray` whose length is `data.length + 1` (Line 6).
2. Copy each element `data[ pos ]` whose index, `pos`, is smaller than `target` to `newArray[ pos ]` (Lines 7–10).
3. Store `w` in `newArray[ target ]` (Line 11).
4. Copy each element `data[ pos ]` whose index, `pos`, is greater than `target` to `newArray[ pos + 1 ]` (Lines 12–15).
5. Return `newArray` (Line 16).

```
1  import java.util.*;
2  public class ResizeArray
3  {
4    public static String[] insert( String[] data, int target, String w )
5    {
6      String[] newArray = new String[ data.length + 1 ];
7      for ( int pos = 0; pos < target; pos ++ )
8      {
9        newArray[ pos ] = data[ pos ];
10     }
11     newArray[ target ] = w;
12     for ( int pos = target; pos < data.length; pos ++ )
13     {
14       newArray[ pos + 1 ] = data[ pos ];
15     }
16     return newArray;
17   }
18
```

Listing 13.10 A program that demonstrates resizing of an array (part 1). The class header and the method `insert`

The next part is the method, `remove`. The formal parameters of the method are a `String` array `data` and an `int` value `target` (Line 19). The task to be performed is to return an array created from `data` by removing the element at index `target`. To accomplish the required task, the method does the following:

1. Create a new array `newArray` whose length is `data.length - 1` (Line 21).
2. Copy each element `data[ pos ]` whose index, `pos`, is smaller than `target` to `newArray[ pos ]` (Lines 22–25).
3. Copy each element `data[ pos ]` whose index, `pos`, is greater than `target` to `newArray[ pos - 1 ]` (Lines 26–29).
4. Return `newArray` (Line 30).

```
19   public static String[] remove( String[] data, int target )
20   {
21     String[] newArray = new String[ data.length - 1 ];
22     for ( int pos = 0; pos < target; pos ++ )
23     {
24       newArray[ pos ] = data[ pos ];
25     }
26     for ( int pos = target + 1; pos < data.length; pos ++ )
27     {
28       newArray[ pos - 1 ] = data[ pos ];
29     }
30     return newArray;
31   }
32
```

Listing 13.11 A program that demonstrates resizing of an array (part 2). The method `remove`

In the method `main`, the program receives the initial length of the array from the user (Line 37), instantiates a new array with the number of elements specified by the user (Line 38), and then receives the initial elements that to be stored in the array from the user (Lines 40–44). The variable `pos` to specify a position in the array is declared outside the for-loop (Line 39). This variable is used in other places of the code.

```
33    public static void main( String[] args )
34    {
35      Scanner keyboard = new Scanner( System.in );
36      System.out.print( "Enter size: " );
37      int size = keyboard.nextInt();
38      String[] data = new String[ size ];
39      int pos;
40      for ( pos = 0; pos < data.length; pos ++ )
41      {
42        System.out.printf( "Enter element %d: ", pos );
43        data[ pos ] = keyboard.next();
44      }
45
```

Listing 13.12 A program that demonstrates resizing of an array (part 3). The part responsible creating the initial array

Two `String` variables, `value` and `answer`, are used for interaction (Line 46). The available actions are presented with the first letters of the names of the actions. The first letters of the action names are `'I'`, `'R'`, `'V'`, and `'Q'` (Lines 49). The user enters an action to be performed as a `String` value, and this is stored in `answer` (Line 50). To direct the execution, the program uses an if-else statement (Line 51)

In the case where the first character is `'I'` (Line 51), the action to be performed is insertion. The program receives the position of insertion, `pos`, and the element, `value`, to be inserted (Lines 53–56). The program then calls `insert( data, pos, value )`, and then substitutes `data` with the array that the method returns (Line 57).

In the case where the first character is `'R'` (Line 59), the action to be performed is removal. The program receives the position where the element to be removed is located, and stores it in the variable `pos` (Lines 61 and 62). The program then calls `remove( data, pos )`, and then substitutes `data` with the array that the method returns (Line 63).

In the case where the first character is `'V'` (Line 65), the action to be performed is showing the data. Using a for-loop, the program prints the elements, one element per line, with their indexes (Lines 67–70).

In all other cases, there will be no action to be performed, but in the case where the first character is `'Q'`, the program exits the loop and terminates (Line 72).

```
46      String value, answer;
47      do
48      {
49        System.out.print( "(I)nsert, (R)emove, (V)iew, (Q)uit? " );
50        answer = keyboard.next();
51        if ( answer.startsWith( "I" ) )
52        {
53          System.out.print( "Enter position: " );
54          pos = keyboard.nextInt();
55          System.out.print( "Enter value: " );
56          value = keyboard.next();
57          data = insert( data, pos, value );
58        }
59        else if ( answer.startsWith( "R" ) )
60        {
61          System.out.print( "Enter position: " );
62          pos = keyboard.nextInt();
63          data = remove( data, pos );
64        }
65        else if ( answer.startsWith( "V" ) )
66        {
67          for ( pos = 0; pos < data.length; pos ++ )
68          {
69            System.out.printf( "%3d:%s%n", pos, data[ pos ] );
70          }
71        }
72      } while ( !answer.startsWith( "Q" ) );
73    }
74  }
```

Listing 13.13 A program that demonstrates resizing of an array (part 4). The do-while-loop

In the following execution example, the user enters the last names of eight of his favorite operatic singers: George London, Dietrich Fischer-Dieskau, Astrid Varnay, Karita Mattila, Renee Fleming, Birgit Nilsson, Wolfgang Windgassen, and Ben Heppner. The user makes some changes by adding Christa Ludwig, removing Windgassen, and then adding Elisabeth Schwartzkopf.[2] Note that the array length is 8 at the start and the index position of the last element is 7, so to append the element at the end, the user enters 8.

[2]George London (May 30, 1920 to March 24, 1985) was a Canadian concert and operatic bass-baritone. Dietrich Fischer-Dieskau (May 28, 1925 to May 18, 2012) was a German concert and operatic baritone. Ibolyka Astrid Maria Varnay (April 25, 1918 to September 4, 2006) was a Swedish-born American operatic soprano. Karita Marjatta Mattila (born September 5, 1960) is a Finnish concert and operatic soprano. Renée Fleming (born February 14, 1959) is an American concert and operatic soprano. Birgit Nilsson (May 17, 1918 to December 25, 2005) was a Swedish concert and operatic soprano. Wolfgang Windgassen (June 26, 1914 to September 8, 1974) was a heldentenor. Thomas Bernard Heppner (born January 14, 1956) is a retired Canadian tenor. He is a Companion of the Orders of Canada. Christa Ludwig (March 16, 1928–) is a retired German concert and operatic mezzo-soprano. Dame Olga Maria Elisabeth Friederike Schwarzkopf (9 December 1915 to 3 August 2006) was a German soprano. She was a Dame Commander of the Most Excellent Order of the British Empire.

```
Enter size: 8
Enter element 0: London
Enter element 1: Fischer-Dieskau
Enter element 2: Varnay
Enter element 3: Mattila
Enter element 4: Fleming
Enter element 5: Nilsson
Enter element 6: Windgassen
Enter element 7: Heppner
(I)nsert, (R)emove, (V)iew, (Q)uit? I
Enter position: 8
Enter value: Ludwig
(I)nsert, (R)emove, (V)iew, (Q)uit? V
  0:London
  1:Fischer-Dieskau
  2:Varnay
  3:Mattila
  4:Fleming
  5:Nilsson
  6:Windgassen
  7:Heppner
  8:Ludwig
(I)nsert, (R)emove, (V)iew, (Q)uit? R
Enter position: 6
(I)nsert, (R)emove, (V)iew, (Q)uit? V
  0:London
  1:Fischer-Dieskau
  2:Varnay
  3:Mattila
  4:Fleming
  5:Nilsson
  6:Heppner
  7:Ludwig
(I)nsert, (R)emove, (V)iew, (Q)uit? I
Enter position: 2
Enter value: Schwartzkopf
(I)nsert, (R)emove, (V)iew, (Q)uit? V
  0:London
  1:Fischer-Dieskau
  2:Schwartzkopf
  3:Varnay
  4:Mattila
  5:Fleming
  6:Nilsson
  7:Heppner
  8:Ludwig
(I)nsert, (R)emove, (V)iew, (Q)uit? Q
```

13.3.2 Adjoining Two Arrays

Next, we consider adjoining two arrays. Suppose we have at hand two arrays, `list1` and `list2`, whose elements are of some data type `T`, and we want to create a new array `list3` by appending the elements of `list2` after the elements of `list1`. This task can be accomplished by instantiating `list3` as an array of length `list1.length + list2.length`, copying the elements from `list1` to the positions `0, ..., list1.length-1`, and then copying the elements of `list2` to the positions `list1.length, ..., list1.length+list2.length-1`.

```
  T[] list3 = new T[ list1.length + list2.length ];
  for ( int index = 0; index < list1.length; index ++ )
  {
    list3[ index ] = list1[ index ];
  }
  for ( int index = 0; index < list2.length; index ++ )
  {
    list3[ list1.length + index ] = list2[ index ];
  }
```

This code can be shorten slightly with the use of the method `copyOf` of the class `Arrays`. The method call `Arrays.copyOf( list1, list1.length + list2.length )` returns an array of length `list1.length + list2.length`, whose first `list1.length` elements form an exact copy of `list1`. We obtain the array returned by the method call, and then copy the elements of `list2` to their respective positions.

```
// java.util.Arrays must be imported
  T[] list3 = Arrays.copyOf( list1, list1.length +  list2.length );
  for ( int index = 0; index < list2.length; index ++ )
  {
    list3[ list1.length + index ] = list2[ index ];
  }
```

13.4 `args`

The formal parameter of the method `main`, `String[] args`, is an array that represents the tokens that appear after the command, when a Java program is executed from a command line interface. The array holds the tokens that follow the name of the Java program in the command line. For example, if `Foo` is a Java class name and `java Foo c1 c2` is typed as a command line, the array `args` of the method `main` of `Foo` has length two with elements, `"c1"` and `"c2"`. The character sequence that appears after the command are interpreted with the syntax of the command line interface. In the case of Unix-like environments, characters including `'|'`, `'>'`, `'<'`, and `';'`, serve as some type of punctuation. All the tokens appearing after any of these characters will be ignored when creating `args`.

Here is a program that simply prints the elements of the array one by one:

```
public class Args
{
  public static void main( String[] args )
  {
    System.out.printf( "args has length %d.%n", args.length );
    for ( int index = 0; index < args.length; index ++ )
    {
      System.out.printf( "args[%d] = %s%n", index, args[ index ] );
    }
  }
}
```

Listing 13.14 A program that prints the length and the elements of `args`

We present one execution example of the code. In the example, `"% "` is the prompt, `"java Args"` is the execution command, and `"10.0 -k, -n abc"` is the sequence that follows the command.

```
% java Args 10.0 -k, -n abc
args has length 4.
args[0] = 10.0
args[1] = -k,
args[2] = -n
args[3] = abc
```

If a semicolon appears, the part that follows is considered to be another command. In the following case, the semicolon splits the command into two separate commands, and so the program is executed twice.

```
% java Args 10.0 -k, -n abc; java Args -foobar -x
args has length 4.
args[0] = 10.0
args[1] = -k,
args[2] = -n
args[3] = abc
args has length 2.
args[0] = -foobar
args[1] = -x
```

13.5 Searching for an Element in an Array

13.5.1 Sequential Search

Searching in an array is the problem of checking, given some array `a` and a data `key`, if `key` is already an element of `a`. A simple solution to the problem is to examine the elements of the array in order. Such search is called the **sequential search**. Suppose that `a` is an array of `String` data and `key` is a `String` value. A sequential search can be executed as follows:

```
boolean found = false;
int i;
for ( i = 0; i < a.length; i ++ )
{
  if ( a[ i ].equals( key ) )
  {
    found = true;
    break;
  }
}
```

The value of `found` represents whether or not a match has been found. The value of `i` at the end is the location of the first match. If there is no match, the value of `i` at the end is `a.length`. We can convert the code fragment to a method that reports the outcome with an `int` return value. The conversion needs two `return` statements. One substitutes the action inside the if-statement and the other substitutes the action after the for-loop. Typically, the internal one returns the index at which a match is found, the external one returns a special value, e.g., `-1`.

```
1  for ( int i = 0; i < a.length; i ++ )
2  {
3    if ( a[ i ].equals( key ) )
4    {
5      return i;
6    }
7  }
8  return -1;
```

13.5.2 Binary Search

In the case where the data type of the elements of the array permits comparison and the array elements are already sorted, **binary search** can be used to speed up the search.

A typical binary search executes a loop with two indexes, say `low` and `high`, and maintains the following loop invariant:

(*) `0 <= low`, `low <= high`, `high <= names.length`, and if a key appears in the array, its index is between `low` and `high - 1`.

The loop goes as follows:

(A) Initialize `low` with `0` and `high` with `names.length`. The condition (*) holds.
(B) If `low == high`, because of (*), it can be guaranteed that the key does not appear in the array, so report that `key` does not appear in the array.
(C) If `low < high`, compute `(low + high)/2` and store the value in an `int` variable, `mid`.
  (i) If `names[ mid ].equals( key )`, the `key` has been found, so report the discovery and terminate the search.
  (ii) If `names[ mid ].compareTo( key ) < 0`, it means that all the elements at indexes `<= mid` are smaller than `key`, so update `low` with the value of `mid + 1`. The condition (*) still holds.
  (iii) Similarly, if `names[ mid ].compareTo( key ) > 0`, it means that all the elements at indexes `>= mid` are greater than `key`, so update `high` with the value of `mid`. The condition (*) still holds.

The number of elements in the search range (between the indexes `low` and `high - 1`) is `high - low`. Because `mid` is chosen to be a half-way point between `low` and `hight`, when either `low` or `high` is updated in Steps C-ii or C-iii, the number of elements in the search range decreases at least by one half. This means that for an initial range size of N, the number of times the loop-body is executed is at most $\lceil \log_2 N \rceil$. Even if there are one million elements in the array, it will require at most 20 probes to complete the search.

Suppose the result of comparing between the search key and the array element is stored in an `int` variable named `result`. We can use the following code for binary search. The loop terminates when the value of `result` becomes 0 (meaning a match has been found) or the value of `low` becomes `high` (meaning the search range has size 0). After the loop, the outcome of the search can be determined based on the value of `result`. Furthermore, if it is 0, the value of `mid` is the location of the match that has been found.

```
int mid, low = 0, high = names.length, result = -1;
while ( result != 0 && low < high )
{
  mid = ( low + high ) / 2;
  result = names[ mid ].compareTo( key );
  if ( result < 0 )
  {
    low = mid + 1;
  }
  else if ( result > 0 )
  {
    high = mid;
  }
}
if ( result == 0 )
{
  System.out.println ( key + " was found at " + mid );
}
else
{
  System.out.println ( key + " was not found" );
}
```

13.6 Arrays with Capacity and Size

Instead of resizing an array each time an element is inserted or removed, a fixed array can be used by remembering how many slots of the array are currently occupied. We call it an **array with capacity and size**. An array with capacity and size employs the following principles:

- We call the length of the array the **capacity**. This is the maximum number of elements you can store in the array.
- The length of the array is greater than or equal to the largest possible number of elements that need to be present in the array at the same time.
- An `int` variable is used to record the number of data being stored. We call it the **size** of the storage.
- The elements being stored in the array appear consecutively, at indexes between 0 and `size - 1`.
- A more data can be stored in the array if and only if the size is strictly less than the capacity.
- The element order in the array need not be preserved.

Figure 13.7 shows a drawing of such an array.[3] Since the element order need not be preserved, the following strategies can be used for implementing addition and removal.

- If `size < capacity`, a new element can be added to the array. To accomplish this, store the new element at index `size`, and then increase the value of `size` by 1.
- If `0 <= p` and `p < size`, the element at `p` can be eliminated. To accomplish this, copy the element at `size - 1` (the last element) to `p`, and then decrease the value of `size` by 1.

Both operations require just one element change in the array. Figure 13.8 shows these ideas.

[3] Joanne Brackeen (born July 26, 1938) is an American jazz pianist and composer. Terri Lyne Carrington (born August 4, 1965) is an American jazz drummer, composer, and producer. Carmen Mercedes McRae (April 8, 1922 to November 10, 1994) was an American jazz singer. Linda May Oh (born 1984 in Malaysia) is a Jazz bassist and composer. Esperanza Emily Spalding (born October 18, 1984) is an American jazz bassist and singer. Blossom Dearie (April 28, 1924 to February 7, 2009) was an American jazz singer, composer, and pianist.

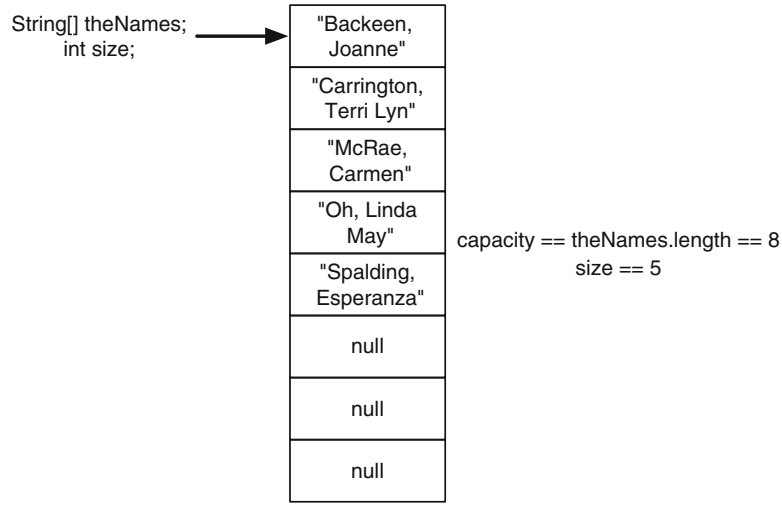

Fig. 13.7 The concept of an array with capacity 8 and size 5. The slots 0 through 4 are presently occupied

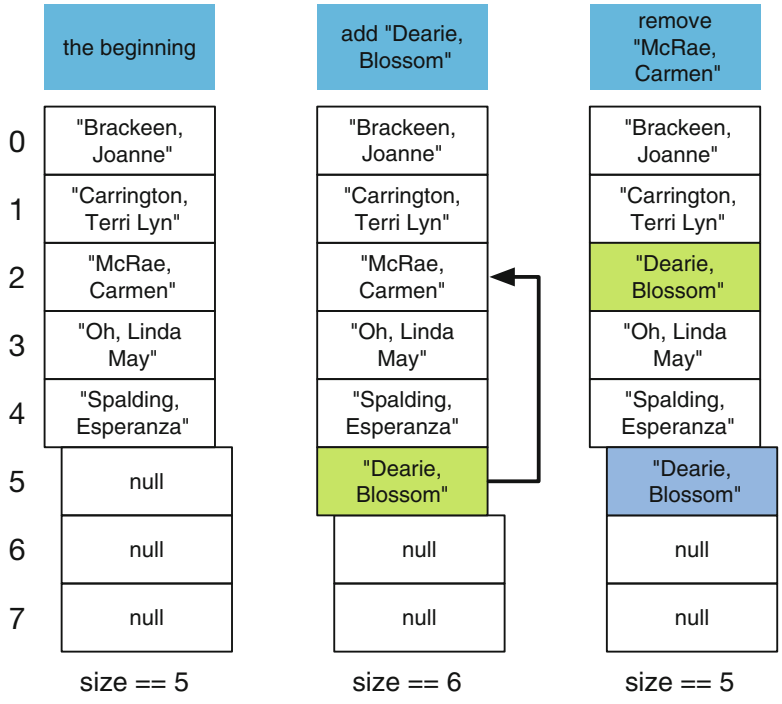

Fig. 13.8 The concept of an array with capacity and size. The figure shows the result of two consecutive actions: to insert `"Dearie, Blossom"` and then to remove `"McRae, Carmen"`. `"Dearie, Blossom"` appears in two position after the removal, but the one at position 5 will not be accessed because `size` is now equal to `5`

The program `FixedSizeUnsorted` is an example of arrays with capacity and size. The application receives an initial set of `String` data from the user, and then interacts with the user to make modifications and perform examinations on the data. The operations that are available on the data are adding elements in succession, removing one element, viewing the collection, and searching for all the elements matching a key specified by the user. The program uses four global variables, `String[] theNames`, `int capacity`, `int size`, and `Scanner keyboard` (Lines 6–10). The variables represent the array, the capacity of the array, the size of the array, and the keyboard with which to receive input from the user. The method `main` consists of two method calls: `setup` (Line 12), for setting up the initial array, and `action` (Line 13), for performing the interaction with the user.

```
1  import java.util.*;
2  // Using an array of fixed size
3  public class FixedSizeUnsorted
4  {
5    // global variables
6    public static int size, capacity;
7    public static String[] theNames;
8    public static Scanner  keyboard;
9    // main
10   public static void main( String[] args )
11   {
12     setup();
13     action();
14   }
```

Listing 13.15 A program that uses an array with capacity and size (part 1). The global variables and the method `main`

The method `setUp` instantiates the array after receiving the capacity from the user (Lines 19–21), and then sets `size` to 0 (Line 22). The method uses `Integer.parseInt` for converting a `String` data to an integer.

```
15   // initial set up
16   public static void setup()
17   {
18     keyboard = new Scanner( System.in );
19     System.out.print( "Enter capacity: " );
20     capacity = Integer.parseInt( keyboard.nextLine() );
21     theNames = new String[ capacity ];
22     size = 0;
23   }
```

Listing 13.16 A program that uses an array with capacity and size (part 2). The method for the initial set up

The method `action` is for interacting with the user. The user chooses one from five actions: `'A'` for adding multiple elements, `'R'` for removing an element, `'P'` for printing all the elements, `'S'` for searching for an element with a key, and `'Q'` for quitting the program. The method uses a `char` variable, `c`, to record the choice of action (Line 27). This is the first letter of the user input (Lines 30–32). By choosing `'Q'`, the user can terminate the loop that starts in Line 28. The flow is controlled with a switch statement.

- For the action of adding elements, the program receives an indefinite number of lines from the user, and calls `add` for each line entered (Lines 36–46). The user can stop the addition by entering an empty line (Line 47). The empty line is not added to the array.
- For the action of removing and element, the program receives the index of the element to be removed, and then calls the method `remove`. The input line that the user enters for the index is converted to an integer using `Integer.parseInt` (Lines 50–51).
- For the action of searching for elements with a key, the program receives a search key from the user, and then calls `search` (Lines 54–55).
- For the action of printing the array, the program calls `print` (Line 58).

After quitting the loop, the method prints a message stating that it is terminating the program (Line 61).

```
24  // action
25  public static void action()
26  {
27    char c;
28    do
29    {
30      System.out.print(
31          "A(dd), R(emove), P(rint), S(earch), Q(uit): " );
32      c = ( keyboard.nextLine() ).charAt( 0 );
33      switch ( c )
34      {
35        case 'A':
36          System.out.println(
37              "Enter new names, empty line to finish: " );
38          String name;
39          do
40          {
41            System.out.print( "> " );
42            name = keyboard.nextLine();
43            if ( name.length() != 0 )
44            {
45              add( name );
46            }
47          } while ( name.length() != 0 );
48          break;
49        case 'R':
50          System.out.print( "Enter an index for removal: " );
51          remove( Integer.parseInt( keyboard.nextLine() ) );
52          break;
53        case 'S':
54          System.out.print( "Enter a search key: " );
55          search( keyboard.nextLine() );
56          break;
57        case 'P':
58          print();
59      }
60    } while ( c != 'Q' );
61    System.out.println( "Closing..." );
62  }
```

Listing 13.17 A program that uses an array with capacity and size (part 3). The method that interacts with the user for modifications and examinations

The method `search` (Line 63) executes sequential search for elements containing a given search key. Like the example in Sect. 13.5.1, the program uses an indicator variable, `found`, to record whether or not any match has been found. The initial value of the variable is `false` (Line 54). The program uses a for-loop that iterates over all the valid indexes with a variable named `pos` (Line 67). In the loop-body, a match is tested with `theName[ pos ].indexOf( key ) >= 0` (Line 69). If this condition is `true`, the program stores `true` in the variable `found`, and then prints the element `theName[ pos ]` with the value of `pos` (Lines 72 and 73). The value change of `found` is only from `false` to `true`. After the loop, if the indicator variable `found` remains `false`, it means that no match has been found, so the program reports that no match has been found (Line 76–78).

In the method `print`, the programs uses a for-loop that iterates over the sequence of valid indexes, `0, ..., size - 1`, to print the elements with their indexes (Lines 84–87).

```
63   // method for searching
64   public static void search( String aName )
65   {
66     boolean found = false;
67     for ( int pos = 0; pos < size; pos ++ )
68     {
69       if ( theNames[ pos ].indexOf( aName ) >= 0 )
70       {
71         found = true;
72         System.out.printf(
73             "Found at %04d in %s\n", pos, theNames[ pos ] );
74       }
75     }
76     if ( !found )
77     {
78       System.out.println( "Not found" );
79     }
80   }
81   // method for printing data
82   public static void print()
83   {
84     for ( int pos = 0; pos < size; pos ++ )
85     {
86       System.out.printf( "%04d:%s%n", pos, theNames[ pos ] );
87     }
88   }
```

Listing 13.18 A program that uses an array with capacity and size (part 4). The methods for searching and printing

The method `add` (Line 90) is for adding a single element. The program first checks if there is space left in the array with the condition `size < capacity` (Line 92). If there is space left, the program places the new element in the array at index `size`, and then increases the size by 1 (Line 94). The two actions can be compressed into a single line:

```
theNames[ size ++ ] = aName
```

If there is no space left, the program prints an error message (Line 98).

```
89   // method for insertion
90   public static void add( String aName )
91   {
92     if ( size < capacity )
93     {
94       theNames[ size ++ ] = aName;
95     }
96     else
97     {
98       System.out.println( "The storage is full." );
99     }
100  }
```

Listing 13.19 A program that uses an array with capacity and size (part 5). The method for adding a single element

The method remove (Line 102) is for removing an element. The program checks if the value of index is valid with the condition index >= 0 && index < size (Line 103). If the value is valid, the program copies the last element to the suggested position, and then decreases size by 1. The two actions can be combined into a single statement (Line 105):

```
theNames[ index ] = theNames[ - size ].
```

Since the - appears before size, the decrement occurs before the assignment to theName[index].

```
101   // method for removal
102   public static void remove( int index ) {
103     if ( index >= 0 && index < size )
104     {
105       theNames[ index ] = theNames[ -- size ];
106     }
107     else
108     {
109       System.out.println( "The specified position does not exist." );
110     }
111   }
```

Listing 13.20 A program that uses an array with capacity and size (part 6). The method for removing an element

Here is an execution example of the code, where the names of several notable female jazz musicians appear as data to be stored.[4]

```
1  java FixedSizeUnsorted
2  Enter capacity: 100
3  A(dd), R(emove), P(rint), S(earch), Q(uit): A
4  Enter new names, empty line to finish:
5  > Joanne Brackeen
6  > Terri Lyn Carrington
7  > Blossom Dearie
8  > Diana Krall
9  > Carmen McRae
10 > Linda May Oh
11 > Sarah Vaughan
12 > Dianne Reeves
13 > Norma Winstone
14 > Nancy Wilson
15 >
16 A(dd), R(emove), P(rint), S(earch), Q(uit): P
17 0000:Joanne Brackeen
18 0001:Terri Lyn Carrington
19 0002:Blossom Dearie
20 0003:Diana Krall
21 0004:Carmen McRae
22 0005:Linda May Oh
```

[4]Diana Jean Krall (born November 16, 1964) is a Canadian jazz pianist and singer. She is an Officer of the Order of Canada and an officer of the Order of British Columbia, Canada. Dianne Reeves (born October 23, 1956) is an American jazz singer. Sarah Lois Vaughan (March 27, 1924 to April 3, 1990) was an American jazz singer. Cassandra Wilson (born December 4, 1955) is an American jazz musician, vocalist, songwriter, and producer. Nancy Wilson (born February 20, 1937) is an American singer, who has won three Grammy awards. Norma Ann Winstone (born 23 September 1941) is a British jazz singer and lyricist. She has the rank Most Excellent Order of the British Empire.

```
0006:Sarah Vaughan
0007:Dianne Reeves
0008:Norma Winstone
0009:Nancy Wilson
A(dd), R(emove), P(rint), S(earch), Q(uit): A
Enter new names, empty line to finish:
> Cassandra Wilson
> Norma Winstone
>
A(dd), R(emove), P(rint), S(earch), Q(uit): S
Enter a search key: Wilson
Found at 0009 in Nancy Wilson
Found at 0010 in Cassandra Wilson
A(dd), R(emove), P(rint), S(earch), Q(uit): R
Enter an index for removal: 0
A(dd), R(emove), P(rint), S(earch), Q(uit): P
0000:Norma Winstone
0001:Terri Lyn Carrington
0002:Blossom Dearie
0003:Diana Krall
0004:Carmen McRae
0005:Linda May Oh
0006:Sarah Vaughan
0007:Dianne Reeves
0008:Norma Winstone
0009:Nancy Wilson
0010:Cassandra Wilson
A(dd), R(emove), P(rint), S(earch), Q(uit): Q
Closing...
```

Summary

- The class `Arrays` provides a number of useful methods.
- Using a temporary variable, the values can be exchanged between two variables. Similarly, using a temporary variable, the values can be exchanged between two array elements.
- Using the idea of temporary variables, the elements of an array can be circularly moved and the appearance order of elements in an array can be reversed.
- Changing the length of an array requires an instantiation of a new array.
- Sequential search is the standard search method of an element in an array.
- If the elements are sorted, binary search can be used in place of sequential search.
- An array with capacity and size can be used to store an indefinite number of elements in an array.

Exercises

1. **A method that checks some property of an array, 1**

```
1  public static int matched( int[] a )
2  {
3    int count = 0;
4    for ( int pos = 0; pos < ( a.length + 1 ) / 2; pos ++ )
5    {
6      if ( a[ pos ] == a[ a.length - pos - 1 ] )
7      {
8        count ++;
9      }
10   }
11   return count;
12 }
```

   For each of the following `int` arrays, state the value returned by the method:
   (a) `[0, 1, 2, 3, 3, 2, 1, 0]`
   (b) `[0, 1, 2, 3, 0, 1, 2]`
   (c) `[0, 1, 2, 3, 4, 3, 2, 1, 0]`
2. **A method that checks some property of an array, 2** For each of the following `int` arrays, state the value returned by the method:

```
1  public static int matched( int[] a )
2  {
3    int count = 0;
4    int offset = a.length/2;
5    for ( int position = 0; position < offset; position ++ )
6    {
7      if ( a[ position ] == a[ offset + position ] )
8      {
9        count ++;
10     }
11   }
12   return count;
13 }
```

   (a) {`8, 7, 6, 8, 7, 6, 5`}
   (b) {`0, 1, 4, 7, 1, 4`}
   (c) {`0, 1, 2, 3, 4, 3, 2, 1, 0`}
3. **Copying and then sorting** Write a method named `copyAndSort( double[] data )` that returns the sorted version of `data`. Use methods from `Arrays` so that the source code does not have loops.
4. **Search for a key** Write a public static method, `searchForProbe`, that returns if an array contains a search key, where the method receives an array of `String` data and a `String` data as the search key. If the array contains the key, method must return one of the indexes at which the key appears; otherwise, the method must return `-1`.
5. **Count matches in an array** Write a public static method named `countMatches` that returns the number of elements in an array of `String` data that are equal to a given key. The method has two formal parameters. One is the array in which the key is searched for, and the other is the search key.

6. **Counting elements in an array whose values are in a range** Write a public static method named `searchInRange` that returns the number of elements in an array of `int` values that are strictly greater than a given `int` value and strictly less than another given `int` value. The formal parameters of the method are the array in which the numbers are sought, the lower bound, and the upper bound.
7. **Counting elements in an array whose values are outside a range** Write a public static method named `searchOutOfRange` that receives an array of `int` values and two additional `int` values as formal parameters, and returns the number of elements in the array that are either less than the first `int` value or greater than the second `int` value.
8. **Is a sequence increasing?** Write a public static method named `isIncreasing` that returns a `boolean` representing whether or not the elements in an array of `int` values are strictly increasing. The method receives the array as its formal parameter.
9. **Is a sequence decreasing?** Write a public static method named `isNondecreasing` that returns a `boolean` representing whether or not the elements of an array of `int` values are non-decreasing. The method receives the array as its formal parameter.
10. **Is a sequence pairwise distinct?** Write a public static method named `isDistinct` that returns a `boolean` representing whether or not two arrays of `int` values have no elements in common. The arrays are given as formal parameters. The elements in the array and their order of appearance must be preserved, so `Arrays.sort` cannot be performed on the parameters.
11. **Even number checking** Write a public static method named `evenParityCheck` that returns a `boolean` representing whether or not any even number appears in an array. The method receives the array as its formal parameter.
12. **lastIndexOf an element** Write a method named `lastIndexInArray` that computes the last position at which a given key appears in the array, where the type of the elements and the key is `char`. If the key does not appear in the array, the method should return `-1`. The method receives the array and the key as its formal parameters.
13. **secondToLastIndexOf an element** Write a method named `secondToLastIndexInArray` that computes the second to last position at which a given key appears, where the type of the elements and the key is `char`. If the key does not appear in the array more than once, the method should return `-1`. The method receives the array and the key as its formal parameters.
14. **Computing (max–min) of array elements** Write a method named `valueWidth` that returns the difference between the largest and the smallest numbers appearing in an array, where the array elements are `int` values. The method receives the array as its formal parameter.
15. **Characters occurring only once** Write a method named `singlyOccurringCharCount` that receives a `String` data, and then returns, of those characters appearing in the `String`, how many appear just once in it. Use the method `toCharArray` of `String` to obtain an array version of the class `String`, and then use the method `sort` of the class `Arrays` to obtain a sorted list of the characters in the array.
16. **Sorting** Write a program, `RandomArraySort`, that generates an array of random `int` data, and then sorts the array. The user specifies the length of the array as well as the value range of the elements in the array with its minimum and maximum. After generating the array randomly, the program prints the elements of the array, sorts the elements using `Arrays.sort`, and then prints the elements again. Design and use a method for printing the elements of an array of `int` values. The format is five elements per line, 12 character spaces per element, flush right, and without currency punctuation. Since the longest `int` without punctuation requires 11 character spaces, this should put at least once space between the elements in the same line.

The program may run as follows:

```
Enter size, the smallest, and the largest: 20 -1000000 1000000
********Before sorting
      817304     -680967      580358      698246     -552187
      401077     -481882     -967384      664908      839001
      185968     -462035      341899     -940863     -260153
     -832638      225017      136135     -310982     -391535
********Before sorting
     -967384     -940863     -832638     -680967     -552187
     -481882     -462035     -391535     -310982     -260153
      136135      185968      225017      341899      401077
      580358      664908      698246      817304      839001
```

17. **A simple merge of three arrays** Write a program, `SimpleThreeMerge`, that generates three arrays of random integers with identical lengths, and then merges the three into one. The user specifies the length as well as the range of the values of the array elements. The array generated by merging should be three times as long as the individual arrays. The array should contain the elements of the first array in the first third in the order they appear, the elements of the second array in the second third in the order they appear, and the elements of the third array in the last third in the order they appear. The program must print the elements of the individual arrays as well as the elements of the array that contains all three. Design and use a method for printing the elements of an array of `int` values. The format is five elements per line, 12 character spaces per element, flush right, and without currency punctuation. Since the longest `int` without punctuation requires 11 character spaces, this should put at least once space between the elements in the same line. The program may run as follows:

```
Enter size, the smallest, and the largest: 10 -100000000 100000000
********Data 1
   -33384206   -28969551    47024306   -33694198    -6858007
    84681217   -72689949   -60679845   -99988760    -1229077
********Data 2
   -21488327    66203194   -60898244    31539203   -45065541
    71798105     3208921   -12009976   -91486999    86835731
********Data 3
   -22300371   -18684131    46783358     6312420   -20064955
   -42988143    59528769    62786162   -43964318   -65677842
********Merged
   -33384206   -28969551    47024306   -33694198    -6858007
    84681217   -72689949   -60679845   -99988760    -1229077
   -21488327    66203194   -60898244    31539203   -45065541
    71798105     3208921   -12009976   -91486999    86835731
   -22300371   -18684131    46783358     6312420   -20064955
   -42988143    59528769    62786162   -43964318   -65677842
```

Programming Projects

18. **Merging three sorted arrays so that there are no duplicates** Write a method named `threeMerge` that receives three sorted arrays of `int` values, and then returns a new array, in which all the elements of the three arrays appear in nondecreasing order. Let `a`, `b`, and `c` be the three arrays and let `merged` be the array to be returned. Suppose we use three `int` variables `p`, `q`, and `r` as indexes to the elements of the three arrays. The initial value is 0 for each of the three index variables. We can write a source code for merging as follows:

```
for ( int i = 0; i < merged.length; i ++ )
{
  if ( X ) { merged[ i ] = a[ p ++ ]; }
  else if ( Y ) { merged[ i ] = b[ q ++ ]; }
  else { merged[ i ] = c[ r ++ ]; }
}
```

Assuming a tie can be broken arbitrary, figure out what conditions can be used where indicated with `X` and `Y`.

19. **Write-in election** Write a program, `Election`, that computes the tally in a write-in election, and announces the winner. Since the votes are write-in, there is no predetermined set of candidates. Whoever appears the most in the votes is the winner. The user enters the individual votes, one vote per line, and ends entering with either typing an empty line or pressing CTRL-D. To compute the tally, the program uses two arrays, a `String[]` variable, `names`, and an `int[]` variable, `counts`. Upon receiving a single vote, the program checks if the name on the vote appears in `names`, and if it does, the program adds 1 to the value of the element in `count` at the position corresponding to the name; if the name does not appear in `names`, the program extends both arrays by one element, stores the name in `names` at the last position and stores 1 in `counts` at the last position. In this manner, the two arrays will have the same lengths. The initial length is 0 for both arrays.

    Here is an example of how the program may run:

```
##########################################
# Enter the votes, one vote per line.    #
# End with either CTRL-D or an empty line.#
##########################################
Frodo
Sam
Pippin
Frodo
Frodo
Pippin
Pippin
Pippin
Sam
Sam
Pippin

Frodo received 3 votes.
Sam received 3 votes.
Pippin received 5 votes.
--------
The winner is Pippin!
```

20. **Sorting two arrays and merging them into one sorted array** Write a program, `TwoMerge`, that generates two arrays of an identical length that are filled with random integers, sorts them, and merges the sorted two arrays into one sorted array. The user specifies the length and the range of the numbers to be generated with its minimum and maximum. The program must print the elements of the individual and the merged arrays. Write a method for printing the elements of an array of `int` values. The format is five elements per line, 12 character spaces per element, flush right, and without currency punctuation. Also, write a method for merging two sorted arrays having the same lengths into one. Merging two sorted arrays can be accomplished using the

following idea: Concurrently examine the elements of the two arrays in the order they appear, one element at a time from each array, selecting whichever the smaller of the two as the next element in the merged array. The program may run as follows:

```
Enter size, the smallest, and the largest: 20 -100 100
********Data 1
         -94         -84         -82         -81         -73
         -68         -62         -50         -49         -47
         -31         -24         -20          11          21
          23          51          53          59          66
********Data 2
         -86         -84         -79         -75         -71
         -64         -53         -45         -30         -28
         -25         -24           3           6          42
          48          50          65          69          87
********Merged
         -94         -86         -84         -84         -82
         -81         -79         -75         -73         -71
         -68         -64         -62         -53         -50
         -49         -47         -45         -31         -30
         -28         -25         -24         -24         -20
           3           6          11          21          23
          42          48          50          51          53
          59          65          66          69          87
```

21. **Reading `double` tokens and quantifying changes** Write a program, `NumberTokens`, that receives a sequence of floating point numbers of an indefinite length from the user, and reporting the results of comparing the numbers after the first one to their immediate predecessors. The comparison has three outcomes: `'U'` to mean "significantly greater than the predecessor", `'D'` to mean "significantly smaller than the predecessor", and "neither". The "significance" of difference is measured by the absolute difference compared to a fixed positive threshold. In other words, the present value is significantly greater than its predecessor if the present value is greater than the predecessor plus the threshold, and the present value is significantly smaller than its predecessor if the present value is smaller than the predecessor minus the threshold. The threshold is entered after the input number sequence is entered, so the input sequence must be stored in an array.

    The program may run as follows:

```
Enter numbers, empty line to stop
> 3.4 5.3 9.0 -1.7 -24.5 -23 -22 -21 -19 0 18
> 17 19 20 23.7 8.5 7.5 2.5
>
Enter threshold (a negative to stop): 5
--DD----UU----D-D
Enter threshold (a negative to stop): 10
--DD----UU----D--
Enter threshold (a negative to stop): 20
---D-------------
Enter threshold (a negative to stop): -1
```