Herbert Schildt's *Java: A Beginner's Guide*, 6th ed.

Chapter 5 Self Test Answers

1. **Show two ways to declare a one-dimensional array of 12 *double* values.**

   One dimensional arrays can be declared in multiple ways. The most common way to perform this declaration is:

   ```
   type arrayName[] = new type[size];
   ```

   A more readable example could be:

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

   which would create an array of 12 *int*s. Another way to declare this array on two separate lines would be:

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

   which would have the same result. If the values are known, the array could also be declared like this:

   ```
   int sample[] = {99, 5, 34, 67, 823, 342, 2, -245, 65, 77, 92, 23};
   ```

2. **Show how to initialize a one-dimensional array of integers to the values 1 through 5.**

   A one-dimensional array of integers with the values 1 through 5 could be declared:

   ```
   int sample[] = {1, 2, 3, 4, 5};
   ```

3. **Write a program that uses an array to find the average of 10 *double* values. Use any 10 values you like.**

   The following class meets the specified criteria:

   ```
   class ArrayAverager {

     public static void main(String args[]){
       double values[] = {12.47, 45.29, 88.33, 67.478565, 89.127845,
                          48.0, 3.14, 89.66, 45.8878549, 27.01};
       double sum = 0;

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

       System.out.println(sum/values.length);

     } // main
   } // ArrayAverager
   ```

4. **Change the sort in Try this 5-1 so that it sorts an array of strings. Demonstrate that it works.**

   The following program satisfies the designated criteria:

```
/*
* BubbleSortString - sorts the Strings of an array from
* least to greatest alphabetically.
* Adapted from BubbleSort (Page 141)
* Also consulted
* http://stackoverflow.com/questions/12986386/
*                sorting-an-array-of-strings-with-java
*/

public class BubbleSortString {
  public static void main (String args[]){
    String words[] = {"Live", "Long", "And", "Prosper"};

    int a, b;

    String t;

    System.out.println();
    // display the original array
    System.out.printf("Original array is:");
    for (int i=0; i < words.length; i++){
      System.out.printf(" " + words[i]);
    }
    System.out.println();

    // Bubble Sort
    for (a = 1; a <words.length; a++){
      for (b=words.length-1; b >= a; b--){
        if (words[b-1].compareTo(words[b])>0){ // if out of order
          // exchange elements
          t = words[b-1];
          words[b-1] = words[b];
          words[b] = t;
        } // end if
      } // end inner for
    }  // end outer for

    // display the sorted array
    System.out.printf("Sorted array is:");
    for (int i=0; i < words.length; i++){
      System.out.printf(" " + words[i]);
    }
    System.out.println();
    System.out.println();

  } // end main
} // end BubbleSortString
```

5. **What is the difference between the** *String* **methods** *indexOf()* **and** *lastIndexOf()***?**

   The difference between the two methods is minor but can be very useful in certain sitations. *indexOf()* returns the index of the first match of the parameter passed to the *indexOf()* method, while *lastIndexOf()* returns the index of the last match of the parameter. If there is no match found for the supplied parameter, then both will return the value of negative one (-1).

6. **Since all strings are objects of type *String*, show how you can call the *length()* and *charAt()* methods on this string literal: "I like Java".**

   The following program illustrates the use of the *length()* and *charAt()* methods, outputting the length of the String as well as identifying the 5th character in the String:

```
/*
* Demonstrates the use of String methods
* Adapted from example in text on Page 160
*/

class StrOps {
  public static void main(String[] args){
    String myString = "I Like Java!";

    System.out.println("The length of myString is " + myString.length());

    System.out.println("The fifth character in myString is "
                        + myString.charAt(5));
  } // end main
} // end StrOps
```

7. **Expanding on the *Encode* cipher class, modify it so that it uses an eight-character string as the key.**

   The following program meets the specifications outlined above:

```
/*
* Demonstrates cipher that uses 8-character string as key.
* Adapted from example in text on Page 624
*/

class Encode {
  public static void main(String[] args){
    String msg = "This is a cipher that uses a 8-character
                 string to encode itself.  See if you
                 can crack the cipher!";
    String encmsg = "";
    String decmsg = "";
    String key = "agrivate";
    int keyCounter = 0;

    System.out.println();
    System.out.println("Original message: " + msg);


    //encode the message
    for(int i = 0; i < msg.length(); i++){
      encmsg = encmsg + (char)(msg.charAt(i) ^ key.charAt(keyCounter));
      keyCounter++;
      if (keyCounter == 8){
        keyCounter = 0;
      }
    }

    System.out.println();
    System.out.println("Encoded message: " + encmsg);

    //decode the message
    keyCounter = 0;
    for(int i = 0; i < msg.length(); i++){
      decmsg = decmsg + (char)(encmsg.charAt(i) ^ key.charAt(keyCounter));
      keyCounter++;
      if (keyCounter == 8){
        keyCounter = 0;
      }
    }

    System.out.println();
    System.out.println("Decoded message: " + decmsg);
    System.out.println();

  } // end main
} // end Encode
```

8. **Can the bitwise operators be applied to the *double* type?**

   The bitwise operators manipulate individual bits and are therefore not suited to some variable types. Bitwise operators cannot be used on *boolean*, *float*, *double* or class types.

9. **Show how this sequence can be rewritten using the *?* operator.**

   ```
   if(x < 0) y = 10;
   else y = 20;
   ```

   Firstly, a rewriting of the code may be helpful and clarify the intent for beginners:

   ```
   if (x < 0){
     y = 10;
   } else {
     y= 20;
   }
   ```

   This code can be rewritten using the *?* in the following way:

   ```
   y = x < 0 ? 10: 20;
   ```

   In my (humble) opinion, using the *?* reduces the actual amount of typing required by the programmer, but does not make the code easier to read for other programmers (or the same programmer later). Referencing the arguments for good commenting, it is imperative that well-written code be descriptive and easy for others to understand. Just because you can name all your variables *x* does not mean that its a good idea to do so - using the *?* operator is possible, but doesn't make it easier to debug your code!

10. **In the following fragment, is the & a bitwise or a logical operator? Why?**

    ```
    boolean a, b;
    // ...
    if(a & b) ...
    ```

    By definition, bitwise operators do not operate on *boolean* variables. So it is impossible that the & operator in the code fragment above is a bitwise operator and must therefore be a logical operator.

11. **Is it an error to overrun the end of an array? Is it an error to index an array with a negative value?**

    Overrunning the end of an array will result in an *Out Of Bounds* exception being thrown. This is a very common error for beginning programmers to encounter. Arrays, by default, begin with the 0-index. It is not possible to index an array with a negative value.

12. **What is the unsigned right-shift operator?**

    The unsigned right shift operator in Java is >>>.

13. **Rewrite the *MinMax* class shown earlier in the chapter so that it uses a for-each style *for* loop.**

    Using a *for-each* style loop instead of a *for* loop simply requires rewriting the condition and replacing all the references to the *nums[i]* variable to simply refer to the *i* variable:

```
/*
* Determines the minimum and maximum values of an array.
* Adapted from example in text on Page 139
*/

class MinMax {
  public static void main(String[] args){
    int nums[] = {99, -10, 100123, 18, -978,
                  5623, 463, -9, 287, 49};
    int min = 0;
    int max = 0;

    min = max = nums[0];

    for (int i : nums){
      if(i < min){
        min = i;
      }

      if (i > max){
        max = i;
      }
    }

    System.out.println("The minimum number in the array was: " + min +
                         " and the maximum number was " + max + ".");
  } // end main
} // end MinMax
```

14. **Can the *for* loops that perform sorting in the *Bubble* class shown in Try This 5-1 be converted into for-each style loops? If not, why not?**

    As presented in the text, BubbleSort has two *for* loops, an outer loop and an inner loop. The first (outer) loop is completely standard and simply iterates over the array from start to finish. By definition, a *for-each* loop "cycles through a collection of objects, such as an array, in strictly sequential fashion, from start to finish." Initially, it would seem that the first (outer) loop can be replaced by a *for-each* loop, but this is incorrect because the value of the loop counter is needed by the second (inner) loop. The second (inner) *for* loop is clearly non-standard in that it starts with a value other than the 0 index of an array and actually decrements the value at which it started with each iteration. Neither the outer nor the inner loops of the BubbleSort algorithm can be replaced with a *for-each* loop.

15. **Can a *String* control a *switch* statement?**

    As of the release of JDK 7 (July 28, 2011), it is possible for a *switch* statement to be controlled by a *String* in Java. Using a *String* to control a *switch* statement can often result in more readable and maintainable code.