LISTING 11.1  Recursive Countdown Method

public class RecursiveCountdown
{
    public static void main(String[] args)
    {
        countDown(3);
    }
    public static void countDown(int num)
    {
        if (num <= 0)
        {
            System.out.println();
        }
        else
        {
            System.out.print(num);
            countDown(num - 1);
        }
    }
}

Sample Screen Output

321
FIGURE 11.1 Recursive Calls for the `countDown` Method

```java
public static void countDown(int num) {
    if (num <= 0) {
        System.out.println();
    } else {
        System.out.print(num);
        countDown(num - 1);
    }
}
```

Console Output:

3 2 1
import java.util.Scanner;

public class RecursionDemo {
    public static void main(String[] args) {
        System.out.println("Enter an integer:");
        Scanner keyboard = new Scanner(System.in);
        int number = keyboard.nextInt();
        System.out.println("The digits in that number are:");
        displayAsWords(number);
        System.out.println();

        System.out.println("If you add ten to that number,"");
        System.out.println("the digits in the new number are:");
        number = number + 10;
        displayAsWords(number);
        System.out.println();
    }
}
/**
 * Precondition: number >= 0
 * Displays the digits in number as words.
 */
public static void displayAsWords(int number)
{
    if (number < 10)
        System.out.print(getWordFromDigit(number) + " ")
    else //number has two or more digits
    {
        displayAsWords(number / 10); //Recursive call
        System.out.print(getWordFromDigit(number) + " ")
    }
}
private static String getWordFromDigit(int digit) {
    String result = null;

    switch (digit) {
        case 0: result = "zero"; break;
        case 1: result = "one"; break;
        case 2: result = "two"; break;
        case 3: result = "three"; break;
        case 4: result = "four"; break;
        case 5: result = "five"; break;
        case 6: result = "six"; break;
        case 7: result = "seven"; break;
        case 8: result = "eight"; break;
        case 9: result = "nine"; break;
        default:
            System.out.println("Fatal Error.");
            System.exit(0);
    }
    return result;
}
Enter an integer:
987
The digits in that number are:
nine eight seven
If you add ten to that number,
the digits in the new number are:
nine nine seven
FIGURE 11.2 Executing a Recursive Call

```java
// Code for invocation of displayAsWords(987)
if (987 < 10)
    System.out.print(getWordFromDigit(987) + " ");
else // 987 has two or more digits
{
    displayAsWords(987 / 10);
    System.out.print(getWordFromDigit(987 % 10) + " ");
}
```

displayAsWords(987/10) is equivalent to displayAsWords(98), which is equivalent to executing:

```java
// Code for invocation of displayAsWords(98)
if (98 < 10)
    System.out.print(getWordFromDigit(98) + " ");
else // 98 has two or more digits
{
    displayAsWords(98 / 10);
    System.out.print(getWordFromDigit(98 % 10) + " ");
}
```
displayAsWords(98/10) is equivalent to displayAsWords(9), which is equivalent to executing:

```java
// Code for invocation of displayAsWords(9)
if (9 < 10)
    System.out.print(getWordFromDigit(9) + " ");
else  // 9 has two or more digits
{
    displayAsWords(9 / 10);
    System.out.print(getWordFromDigit(9 % 10) + " ");
}
```
LISTING 11.3  An Iterative Version of displayAsWords

```java
import java.util.Scanner;
public class IterativeDemo
{
    public static void main(String[] args)
    {
        // The rest of main is the same as Listing 11.2.>

        /**
         * Precondition: number >= 0
         * Displays the digits in number as words.
         */
        public static void displayAsWords(int number)
        {
            int divisor = getPowerOfTen(number);
            int next = number;
            while (divisor >= 10)
            {
                System.out.print(getWordFromDigit(next / divisor) + " ");
                next = next % divisor;
                divisor = divisor / 10;
            }
            System.out.print(getWordFromDigit(next / divisor) + " ");
        }
        // Precondition: n >= 0.
        // Returns 10 raised to the power n.
        private static int getPowerOfTen(int n)
        {
            int result = 1;
            while (n >= 10)
            {
                result = result * 10;
                n = n / 10;
            }
            return result;
        }
        private static String getWordFromDigit(int digit)
        {<The rest of getWordFromDigit is the same as in Listing 11.2.>
        }
    }
}
```

JAVA: An Introduction to Problem Solving & Programming, 6th Ed. By Walter Savitch
import java.util.Scanner;

public class RecursionDemo2
{
    public static void main(String[] args)
    {
        System.out.println("Enter a nonnegative number: ");
        Scanner keyboard = new Scanner(System.in);
        int number = keyboard.nextInt();
        System.out.println(number + " contains " +
                           getNumberOfZeros(number) + " zeros.");
    }
}
/**
   * Precondition: n >=0
   * Returns the number of zero digits in n.
   */
   
   public static int getNumberOfZeros(int n)
   {
       int result;
       if (n == 0)
           result = 1;
       else if (n < 10)
           result = 0; // n has one digit that is not 0
       else if (n % 10 == 0)
           result = getNumberOfZeros(n / 10) + 1;
       else // n % 10 != 0
           result = getNumberOfZeros(n / 10);
       return result;
   }
**Sample Screen Output**

Enter a nonnegative number:
2008
2008 contains 2 zeros.
LISTING 11.5  Recursion for Starting Over (part 1 of 2)

```java
import java.util.Scanner;
public class CountDown
{
    private int count;

    public static void main(String[] args)
    {
        CountDown countDowner = new CountDown();
        countDowner.getCount();
        countDowner.showCountDown();
    }

    public void getCount()
    {
        System.out.println("Enter a positive integer:");
        Scanner keyboard = new Scanner(System.in);
        count = keyboard.nextInt();
        if (count <= 0)
        {
            System.out.println("Input must be positive.");
            System.out.println("Try again.");
            getCount(); // start over
        }
    }

    public void showCountDown()
    {
        System.out.println("Counting down:");
        for (int left = count; left >= 0; left--)
        {
            System.out.print(left + ", ");
        }
        System.out.println("Blast Off!");
    }
}
```
Sample Screen Output

Enter a positive integer:
0
Input must be positive.
Try again.
Enter a positive integer:
3
Counting down:
3, 2, 1, 0, Blast Off!
FIGURE 11.3  A Binary Search Example

target is 33

Eliminate half of the array elements:

mid

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>7</td>
<td>9</td>
<td>13</td>
<td>32</td>
<td>33</td>
<td>42</td>
<td>54</td>
<td>56</td>
<td>88</td>
</tr>
</tbody>
</table>

1. mid = (0 + 9)/2 (which is 4).
2. 33 > a[mid] (that is, 33 > a[4]).
3. So if 33 is in the array, 33 is one of a[5], a[6], a[7], a[8], a[9].

Eliminate half of the remaining array elements:

mid

<table>
<thead>
<tr>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>33</td>
<td>42</td>
<td>54</td>
<td>56</td>
<td>88</td>
</tr>
</tbody>
</table>

1. mid = (5 + 9)/2 (which is 7).
2. 33 < a[mid] (that is, 33 < a[7]).
3. So if 33 is in the array, 33 is one of a[5], a[8].

Eliminate half of the remaining array elements:

mid

<table>
<thead>
<tr>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>33</td>
<td>42</td>
</tr>
</tbody>
</table>

1. mid = (5 + 6)/2 (which is 5).
2. 33 equals a[mid], so we found 33 at index 5.

33 found in a[5].
**LISTING 11.6  A Binary Search Class** (part 1 of 2)

```java
/**
 * Class for searching an already sorted array of integers.
 */
public class ArraySearcher
{
    private int[] a;

    /**
     * Precondition: theArray is full and is sorted from lowest to highest.
     */
    public ArraySearcher(int[] theArray)
    {
        a = theArray; // a is now another name for theArray.
    }

    /**
     * If target is in the array, returns the index of an occurrence of target. Returns -1 if target is not in the array.
     */
    public int find(int target)
    {
        return binarySearch(target, 0, a.length - 1);
    }
}
```
//Uses binary search to search for target in a[first] through
//a[last] inclusive. Returns the index of target if target
//is found. Returns -1 if target is not found.
private int binarySearch(int target, int first, int last)
{
    int result;
    if (first > last)
        result = -1;
    else
    {
        int mid = (first + last)/2;
        if (target == a[mid])
            result = mid;
        else if (target < a[mid])
            result = binarySearch(target, first, mid - 1);
        else // (target > a[mid])
            result = binarySearch(target, mid + 1, last);
    }
    return result;
}
LISTING 11.7  A Binary Search Demonstration (part 1 of 3)

```java
import java.util.Scanner;
public class ArraySearcherDemo
{
    public static void main(String[] args)
    {
        int[] anArray = new int[10];
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Enter 10 integers in increasing " +
                             "order, ");
        System.out.println("one per line.");
        for (int i = 0; i < 10; i++)
            anArray[i] = keyboard.nextInt();
        System.out.println();

        for (int i = 0; i < 10; i++)
            System.out.print("a[" + i + "]=" + anArray[i] + ");
        System.out.println();
        System.out.println();
    }
}
```

JAVA: An Introduction to Problem Solving & Programming, 6th Ed. By Walter Savitch
ArraySearcher finder = new ArraySearcher(anArray);
String ans;
do {
    System.out.println("Enter a value to search for:");
    int target = keyboard.nextInt();
    int result = finder.find(target);

    if (result < 0)
        System.out.println(target + " is not in the array.");
    else
        System.out.println(target + " is at index " + result);
    System.out.println("Again?");
    ans = keyboard.next();
} while (ans.equalsIgnoreCase("yes");

System.out.println("May you find what you're searching for.");
Sample Screen Output

Enter 10 integers in increasing order, one per line.

0
2
4
6
8
10
12
14
16
18

a[0]=0
Enter a value to search for:
14
14 is at index 7
Again?
yes
Enter a value to search for:
0
0 is at index 0
Again?
yes
Enter a value to search for:
2
2 is at index 1
Again?
yes

Enter a value to search for:
13
13 is not in the array.
Again?
no
May you find what you’re searching for.
LISTING 11.8  The MergeSort Class (part 1 of 3)

    /**
     * Class for sorting an array of integers from smallest to largest using the merge sort algorithm.
     */
    public class MergeSort
    {
      /**
       * Precondition: Every indexed variable of the array a has a value.
       * Postcondition: a[0] <= a[1] <= ... <= a[a.length - 1].
       */
      public static void sort(int[] a)
      {
        if (a.length >= 2)
        {
          int halfLength = a.length / 2;
          int[] firstHalf = new int[halfLength];
          int[] lastHalf = new int[a.length - halfLength];

          divide(a, firstHalf, lastHalf);
          sort(firstHalf);
          sort(lastHalf);
          merge(a, firstHalf, lastHalf);
        }
        //else do nothing. a.length == 1, so a is sorted.
      }
    }
// Precondition: a.length = firstHalf.length + lastHalf.length.
// Postcondition: All the elements of a are divided
// between the arrays firstHalf and lastHalf.
private static void divide(int[] a, int[] firstHalf,
                          int[] lastHalf)
{
    for (int i = 0; i < firstHalf.length; i++)
        firstHalf[i] = a[i];

    for (int i = 0; i < lastHalf.length; i++)
        lastHalf[i] = a[firstHalf.length + i];
}

// Precondition: Arrays firstHalf and lastHalf are sorted from
// smallest to largest; a.length = firstHalf.length +
// lastHalf.length.
// Postcondition: Array a contains all the values from firstHalf
// and lastHalf and is sorted from smallest to largest.
private static void merge(int[] a, int[] firstHalf,
                          int[] lastHalf)
{
    int firstHalfIndex = 0, lastHalfIndex = 0, aIndex = 0;
    while ((firstHalfIndex < firstHalf.length) &&
           (lastHalfIndex < lastHalf.length))
    {
        if (firstHalf[firstHalfIndex] < lastHalf[lastHalfIndex])
            {
                a[aIndex] = firstHalf[firstHalfIndex];
                firstHalfIndex++;
            }
        else
            {
                a[aIndex] = lastHalf[lastHalfIndex];
                lastHalfIndex++;
            }
        aIndex++;
    }
// At least one of firstHalf and lastHalf has been completely copied to a.

// Copy rest of firstHalf, if any.
while (firstHalfIndex < firstHalf.length)
{
    a[aIndex] = firstHalf[firstHalfIndex];
    aIndex++;
    firstHalfIndex++;
}

// Copy rest of lastHalf, if any.
while (lastHalfIndex < lastHalf.length)
{
    a[aIndex] = lastHalf[lastHalfIndex];
    aIndex++;
    lastHalfIndex++;
}
LISTING 11.9  Demonstration of the MergeSort Class

public class MergeSortDemo
{
    public static void main(String[] args)
    {
        int[] anArray = {7, 5, 11, 2, 16, 4, 18, 14, 12, 30};
        System.out.println("Array values before sorting:");
        for (int i = 0; i < anArray.length; i++)
            System.out.print(anArray[i] + " ");
        System.out.println();
        MergeSort.sort(anArray);
        System.out.println("Array values after sorting:");
        for (int i = 0; i < anArray.length; i++)
            System.out.print(anArray[i] + " ");
        System.out.println();
    }
}

Screen Output

Array values before sorting:
7 5 11 2 1 4 18 14 12 30
Array values after sorting:
2 4 5 7 11 12 14 16 18 30