Project 6-3 The Quicksort

QSDemo.java

In Module 5 you were shown a simple sorting method called the Bubble sort. It was mentioned at the time that substantially better sorts exist. Here you will develop a version of one of the best: the Quicksort. The Quicksort, invented and named by C.A.R. Hoare, is the best general-purpose sorting algorithm currently available. The reason it could not be shown in Module 5 is that the best implementations of the Quicksort rely on recursion. The version we will develop sorts a character array, but the logic can be adapted to sort any type of object you like.

The Quicksort is built on the idea of partitions. The general procedure is to select a value, called the *comparand*, and then to partition the array into two sections. All elements greater than or equal to the partition value are put on one side, and those less than the value are put on the other. This process is then repeated for each remaining section until the array is sorted. For example, given the array **fedacb** and using the value **d** as the comparand, the first pass of the Quicksort would rearrange the array as follows:

Initial	f e d a c b
Pass1	b c a d e f

This process is then repeated for each section—that is, **bca** and **def**. As you can see, the process is essentially recursive in nature, and indeed, the cleanest implementation of Quicksort is recursive.

You can select the comparand value in two ways. You can either choose it at random, or you can select it by averaging a small set of values taken from the array. For optimal sorting, you should select a value that is precisely in the middle of the range of values. However, this is not easy to do for most sets of data. In the worst case, the value chosen is at one extremity. Even in this case, however, Quicksort still performs correctly. The version of Quicksort that we will develop selects the middle element of the array as the comparand.

Step by Step

1. Create a file called **QSDemo.java**.
2. First, create the **Quicksort** class shown here:

```
// Project 6-3: A simple version of the Quicksort.
class Quicksort {

  // Set up a call to the actual Quicksort method.
  static void qsort(char items[]) {
    qs(items, 0, items.length-1);
  }
```

(continued)

```
  // A recursive version of Quicksort for characters.
  private static void qs(char items[], int left, int right)
  {
    int i, j;
    char x, y;

    i = left; j = right;
    x = items[(left+right)/2];

    do {
      while((items[i] < x) && (i < right)) i++;
      while((x < items[j]) && (j > left)) j--;

      if(i <= j) {
        y = items[i];
        items[i] = items[j];
        items[j] = y;
        i++; j--;
      }
    } while(i <= j);

    if(left < j) qs(items, left, j);
    if(i < right) qs(items, i, right);
  }
}
```

To keep the interface to the Quicksort simple, the **Quicksort** class provides the **qsort( )** method, which sets up a call to the actual Quicksort method, **qs( )**. This enables the Quicksort to be called with just the name of the array to be sorted, without having to provide an initial partition. Since **qs( )** is only used internally, it is specified as **private**.

3. To use the **Quicksort**, simply call **Quicksort.qsort( )**. Since **qsort( )** is specified as **static**, it can be called through its class rather than on an object. Thus, there is no need to create a **Quicksort** object. After the call returns, the array will be sorted. Remember, this version works only for character arrays, but you can adapt the logic to sort any type of arrays you want.

4. Here is a program that demonstrates **Quicksort**:

```
// Project 6-3: A simple version of the Quicksort.
class Quicksort {

  // Set up a call to the actual Quicksort method.
  static void qsort(char items[]) {
    qs(items, 0, items.length-1);
  }
```

```
  // A recursive version of Quicksort for characters.
  private static void qs(char items[], int left, int right)
  {
    int i, j;
    char x, y;

    i = left; j = right;
    x = items[(left+right)/2];

    do {
      while((items[i] < x) && (i < right)) i++;
      while((x < items[j]) && (j > left)) j--;

      if(i <= j) {
        y = items[i];
        items[i] = items[j];
        items[j] = y;
        i++; j--;
      }
    } while(i <= j);

    if(left < j) qs(items, left, j);
    if(i < right) qs(items, i, right);
  }
}

class QSDemo {
  public static void main(String args[]) {
    char a[] = { 'd', 'x', 'a', 'r', 'p', 'j', 'i' };
    int i;

    System.out.print("Original array: ");
    for(i=0; i < a.length; i++)
      System.out.print(a[i]);

    System.out.println();

    // now, sort the array
    Quicksort.qsort(a);

    System.out.print("Sorted array: ");
    for(i=0; i < a.length; i++)
      System.out.print(a[i]);
  }
}
```