Exercises 4.73-4.78

The **LogAnalyzer** class is shown below:

```
public class LogAnalyzer
{
    // Where to calculate the hourly access counts.
    private int[] hourCounts;
    // Use a LogfileReader to access the data.
    private LogfileReader reader;

    /**
     * Create an object to analyze hourly web accesses.
     */
    public LogAnalyzer()
    {
        // Create the array object to hold the hourly
        // access counts.
        hourCounts = new int[24];
        // Create the reader to obtain the data.
        reader = new LogfileReader();
    }

    /**
     * Analyze the hourly access data from the log file.
     */
    public void analyzeHourlyData()
    {
        while(reader.hasNext()) {
            LogEntry entry = reader.next();
            int hour = entry.getHour();
            hourCounts[hour]++;
        }
    }
    /**
     * Return the number of  accesses recorded in the log file.
     */
    public int numberOfAccesses()
    {
        int total = 0;

        /*
         * Add the value in each element of hourCounts to
         * to total
         */

        for (int hours : hourCounts) {
            total += hours;
        }

        return total;
    }
    /**
     * Returns the busiest hour.
     */
    public int busiestHour()
    {
        // we assume that the smallest element is zero.
        // if this is not the case, use Integer.MIN_VALUE.
        int busiestHour = 0;

        for(int hour : hourCounts)
        {
            if(hour > busiestHour)
                busiestHour = hour;
        }

        return busiestHour;
    }

    /**
     * Returns the quietestHour.
     */
    public int quietestHour()
    {
        // we could possibly set this to busiestHour()
        // but that would be to much work.
        // also, for general case, use
        // hourCounts.length > 0 ? hourCounts[0] : 0
        int quietestHour =  hourCounts[0];

        for (int hour : hourCounts)
        {
            if(hour < quietestHour)
                quietestHour = hour;
        }

        return quietestHour;
    }
    /**
     * Returns the busiest two hour period.
     * @return The first hour of that interval.
     */
    public int busiestTwoHour()
    {
        int busiestTwoHour = 0;
        int busiestHour = 0;

        // the 23 could be replaced with
        // hourCounts.length - hourCounts.length % 2 -1
        // if the array size would not be known in advance.

        for(int i = 0; i < 23; i++)
        {
            if(hourCounts[i] + hourCounts[i+1] > busiestTwoHour)
            {
                busiestHour = hourCounts[i];
                busiestTwoHour = hourCounts[i]+ hourCounts[i+1];
            }

        }

```

```java
        return busiestHour;
    }

    /**
     * Print the hourly counts.
     * These should have been set with a prior
     * call to analyzeHourlyData.
     */
    public void printHourlyCounts()
    {
        System.out.println("Hr: Count");
        for(int hour = 0; hour < hourCounts.length; hour++) {
            System.out.println(hour + ": " + hourCounts[hour]);
        }
    }

    /**
     * Print the lines of data read by the LogfileReader
     */
    public void printData()
    {
        reader.printData();
    }
}
```

Exercise 'factorial'

The factorial algorithm based on iterations.

```java
public long factorial(int n)
{
    long result = 1;

    for (int i = 1; i <= n; i++)
    {
        if(i == 0)
        {
            result = 1;
        }
        else
        {
            result *=i;
        }
    }
    return result;
}
```

Exercise 'sum'

The summation algorithm based on iterations.

```java
public long sum (int[] v, int first , int last)
{
    if(first == last)
        return v[first];
    else
    {
        long result = 0;

        for (int i = first; i <= last; i++)
        {
            result += v[i];
        }

        return result;
    }
}
```

Exercise 'max vector'

Recursive and iterative ways of finding the maximum value of a vector. In order to use the recursive version, the *v* variable should be set to `vector.length-1` for this to work. Example implementation can be found in the end of this document.

```java
/**
 * Finds the maximum value in a vector recursively.
 * @param vector The vector.
 * @param v The length of the vector, i.e. vector.length-1.
 * @return The maximum value.
 */
public int maxValueInVectorRecursive(int[] vector, int v)
{
    if (v == 0)
    {
        return vector[0];
    }
    else
    {
        int temp = maxValueInVectorRecursive(vector, v-1);
        if(vector[v] > temp)
        {
            return vector[v];
        }
        else
        {
            return temp;
        }
    }
}

/**
 * Finds the maximum value in a vector using iteration.
 * @param vector The vector.
 * @return The maximum value.
 */
   public int maxValueInVector(int[] vector)
   {
    int maxValue = Integer.MIN_VALUE;

    for(int value : vector)
    {
        if(value > maxValue)
            maxValue = value;
    }

    return maxValue;
   }
```

The example implementation. This a part of a test class used to validate this algorithm.

```java
public void maxValueInVectorTest()
{
    int[] array = new int [50];
    for (int i = 0; i < 50; i++)
    {
        array[i] = randInt(0, 20);
    }

    int max = maxValueInVector(array);

    System.out.println(max);

    int _max = maxValueInVectorRecursive(array, array.length-1);

    if(max != _max)
        fail("Max values not equal. Test 1");

    int[] vec = {3,2,1,5};

    if(maxValueInVector(vec) != maxValueInVectorRecursive(vec, vec.length
        -1))
        fail("Max value not equal. Test 2");

    System.out.println(_max);
}

// Thanks to @Greg Case at
// http://stackoverflow.com/questions/363681/
// generating-random-integers-in-a-range-with-java

public int randInt(int min, int max) {
    // NOTE: Usually this should be a field rather than a method
    // variable so that it is not re-seeded every call.
    Random rand = new Random();
    // nextInt is normally exclusive of the top value,
    // so add 1 to make it inclusive
    int randomNum = rand.nextInt((max - min) + 1) + min;
    return randomNum;
}
```