

# ***For LAB10 and Homework8 describe literally what each line of code is doing.***

## ***Java Notes: Example - WordFrequency***

This program reads files of words and lists their frequencies. A file of words to ignore can also be supplied. This example illustrates the use of generic data structures (Sets, Maps, and ArrayList), regular expressions (as used by Scanner), and Comparators.

**Exercises.** A command-line interface is provided, but an interesting exercise is to write a GUI interface.

### **A text command-line interface**

```
1 // File   : data-collections/wordfreq2/WordFrequencyCmd.java
2 // Purpose: Main program to test WordCounter
3 //       Prints word frequency in source file. Ignores words in ignore file.
4 //       Uses Sets, Maps, ArrayList, regular expressions, BufferedReader.
5 // Author  : Fred Swartz - April 2007 - Placed in public domain.
6
7 import java.io.*;
8 import java.util.*;
9
10 //////////////////////////////////////////////////// WordFrequencyCmd
11 class WordFrequencyCmd {
12     //===== main
13     public static void main(String[] unused) {
14         Scanner in = new Scanner(System.in);
15
16         try {
17             //... Read two file names from the input.
18             System.out.println("Name of file containing text to analyze:");
19             File inputFile = new File(in.nextLine());
20
21             System.out.println("Name of file containing words to ignore:");
22             File ignoreFile = new File(in.nextLine());
23
24             //... Supply two files to WordCounter.
25             WordCounter counter = new WordCounter();
26             counter.ignore(ignoreFile);
27             counter.countWords(inputFile);
28
29             //... Get the results.
30             String[] wrds = counter.getWords(WordCounter.SortOrder.BY_FREQUENCY);
31             int[] frequency = counter.getFrequencies(WordCounter.SortOrder.BY_FREQUENCY);
32
33             //... Display the results.
34             int n = counter.getEntryCount();
35             for (int i=0; i<n; i++) {
36                 System.out.println(frequency[i] + " " + wrds[i]);
37             }
38
39             System.out.println("\nNumber of input words: " + counter.getWordCount());
40             System.out.println("Number of unique words: " + n);
41
42         } catch (IOException iox) {
43             System.out.println(iox);
44         }
45     }
46 }
```

## The "Model" -- Word frequency without any user interface

The *model* (the logic of the program without any user interface) is implemented primarily in the `WordCounter` class, which uses utility classes: `ComparatorFrequency` and `ComparatorAlphabetic` to sort, and `Int` to record the frequency counts.

```
1 // File : data-collections/wordfreq2/WordCounter.java
2 // Purpose: Provides methods to read ignore file, input file, get results.
3 // Computes the frequency for each word.
4 // Author : Fred Swartz - April 2007 - Placed in public domain.
5
6 import java.io.*;
7 import java.util.*;
8 import java.util.regex.*;
9
10 /** Computes word frequency in source file; ignores words in ignore file.
11 * Uses generic Sets, Maps, ArrayLists, regular expressions, Scanner.
12 * @author Fred Swartz
13 * @version 2007-05-06
14 */
15 public class WordCounter {
16     //===== constants
17     private static final Comparator<Map.Entry<String, Int>> SORT_BY_FREQUENCY =
18         new ComparatorFrequency();
19     private static final Comparator<Map.Entry<String, Int>> SORT_ALPHABETICALLY =
20         new ComparatorAlphabetic();
21     public enum SortOrder {ALPHABETICALLY, BY_FREQUENCY}
22
23     //===== fields
24     Set<String> _ignoreWords; // Words to ignore.
25     Map<String, Int> _wordFrequency; // Words -> frequency
26     int _totalWords; // Total source words.
27
28     //===== constructor
29     /** Constructor */
30     public WordCounter() {
31         _ignoreWords = new HashSet<String>();
32         _wordFrequency = new HashMap<String, Int>();
33         _totalWords = 0;
34     }
35
36     //===== ignore
37     /**
38     * Reads file of words to ignore. Ignore words are added to a Set.
39     * The IOException is passed to caller because we certainly don't
40     * know what the user interface issue is.
41     *
42     * @param ignoreFile File of words to ignore.
43     */
44     public void ignore(File ignoreFile) throws IOException {
45         Scanner ignoreScanner = new Scanner(ignoreFile);
46         ignoreScanner.useDelimiter("[^A-Za-z]+");
47
48         while (ignoreScanner.hasNext()) {
49             _ignoreWords.add(ignoreScanner.next());
50         }
51         ignoreScanner.close(); // Close underlying file.
52     }
53
54     //===== ignore
55     /**
56     * Takes String of words to ignore. Ignore words are added to a Set.
57     *
58     * @param ignore String of words to ignore.
59     */
60     public void ignore(String ignoreStr) {
61         Scanner ignoreScanner = new Scanner(ignoreStr);
62         ignoreScanner.useDelimiter("[^A-Za-z]+");
63
64         while (ignoreScanner.hasNext()) {
65             _ignoreWords.add(ignoreScanner.next());
66         }
67     }
68
69     //===== countWords
70     /** Record the frequency of words in the source file.
71     * May be called more than once.
72     */
```

```

73     * IOException is passed to caller, who might know what to do with it.
74     * @param File of words to process.
75     */
76     public void countWords(File sourceFile) throws IOException {
77         Scanner wordScanner = new Scanner(sourceFile);
78         wordScanner.useDelimiter("[^A-Za-z]+");
79
80         while (wordScanner.hasNext()) {
81             String word = wordScanner.next();
82             _totalWords++;
83
84             //... Add word if not in map, otherwise increment count.
85             if (!_ignoreWords.contains(word)) {
86                 Int count = _wordFrequency.get(word);
87                 if (count == null) { // Create new entry with count of 1.
88                     _wordFrequency.put(word, new Int(1));
89                 } else { // Increment existing count by 1.
90                     count.value++;
91                 }
92             }
93         }
94         wordScanner.close(); // Close underlying file.
95     }
96
97
98     //===== countWords
99     /** Record the frequency of words in a String.
100     * May be called more than once.
101     * @param String of words to process.
102     */
103     public void countWords(String source) {
104         Scanner wordScanner = new Scanner(source);
105         wordScanner.useDelimiter("[^A-Za-z]+");
106
107         while (wordScanner.hasNext()) {
108             String word = wordScanner.next();
109             _totalWords++;
110
111             //... Add word if not in map, otherwise increment count.
112             if (!_ignoreWords.contains(word)) {
113                 Int count = _wordFrequency.get(word);
114                 if (count == null) { // Create new entry with count of 1.
115                     _wordFrequency.put(word, new Int(1));
116                 } else { // Increment existing count by 1.
117                     count.value++;
118                 }
119             }
120         }
121     }
122
123     //===== getWordCount
124     /** Returns number of words in all source file(s).
125     * @return Total number of words processed in all source files.
126     */
127     public int getWordCount() {
128         return _totalWords;
129     }
130
131     //===== getEntryCount
132     /** Returns the number of unique, non-ignored words, in the source file(s).
133     * This number should be used to for the size of the arrays that are
134     * passed to getWordFrequency.
135     * @return Number of unique non-ignored source words.
136     */
137     public int getEntryCount() {
138         return _wordFrequency.size();
139     }
140
141     //===== getWordFrequency
142     /** Stores words and their corresponding frequencies in parallel array lists
143     * parameters. The frequencies are sorted from low to high.
144     * @param Unique words that were found in the source file(s).
145     * @param counts Frequency of words at corresponding index in words array.
146     */
147     public void getWordFrequency(ArrayList<String> out_words,
148         ArrayList<Integer> out_counts) {
149         //... Put in ArrayList so sort entries by frequency
150         ArrayList<Map.Entry<String, Int>> entries =
151             new ArrayList<Map.Entry<String, Int>>(_wordFrequency.entrySet());
152         Collections.sort(entries, new ComparatorFrequency());

```

```

153
154 //... Add word and frequency to parallel output ArrayLists.
155 for (Map.Entry<String, Int> ent : entries) {
156     out_words.add(ent.getKey());
157     out_counts.add(ent.getValue().value);
158 }
159 }
160
161 //===== getWords
162 /** Return array of unique words, in the order specified.
163  * @return An array of the words in the currently selected order.
164  */
165 public String[] getWords(SortOrder sortBy) {
166     String[] result = new String[_wordFrequency.size()];
167     ArrayList<Map.Entry<String, Int>> entries =
168         new ArrayList<Map.Entry<String, Int>>(_wordFrequency.entrySet());
169     if (sortBy == SortOrder.ALPHABETICALLY) {
170         Collections.sort(entries, SORT_ALPHABETICALLY);
171     } else {
172         Collections.sort(entries, SORT_BY_FREQUENCY);
173     }
174
175     //... Add words to the String array.
176     int i = 0;
177     for (Map.Entry<String, Int> ent : entries) {
178         result[i++] = ent.getKey();
179     }
180     return result;
181 }
182
183 //===== getFrequencies
184 /** Return array of frequencies, in the order specified.
185  * @return An array of the frequencies in the specified order.
186  */
187 public int[] getFrequencies(SortOrder sortBy) {
188     int[] result = new int[_wordFrequency.size()];
189     ArrayList<Map.Entry<String, Int>> entries =
190         new ArrayList<Map.Entry<String, Int>>(_wordFrequency.entrySet());
191     if (sortBy == SortOrder.ALPHABETICALLY) {
192         Collections.sort(entries, SORT_ALPHABETICALLY);
193     } else {
194         Collections.sort(entries, SORT_BY_FREQUENCY);
195     }
196
197     //... Add words to the String array.
198     int i = 0;
199     for (Map.Entry<String, Int> ent : entries) {
200         result[i++] = ent.getValue().value;
201     }
202     return result;
203 }
204 }

```

## Comparators

The key-value (`Map.Entry`) pairs in a `HashMap` are not sorted, but users would certainly like to see them sorted, either alphabetically by word, or by frequency.

Because sorts are usually based on comparison of two values at a time, all that is needed is a way to compare two values. That's what a `Comparator` does -- it defines a `compare()` method that tells how two values compare.

```
1 // File : data-collections/wordfreq2/ComparatorAlphabetic.java
2 // Purpose: A comparator to sort Map.Entries alphabetically.
3 // Author : Fred Swartz - March 2005 - Placed in public domain.
4
5 import java.util.*;
6
7 ////////////////////////////////////////////////// class ComparatorAlphabetic
8 /** Order words alphabetically. */
9 class ComparatorAlphabetic implements Comparator<Map.Entry<String, Int>> {
10     public int compare(Map.Entry<String, Int> entry1
11         , Map.Entry<String, Int> entry2) {
12         return entry1.getKey().compareTo(entry2.getKey());
13     }
14 }

1 // File : data-collections/wordfreq2/ComparatorFrequency.java
2 // Purpose: A comparator to sort Map.Entries by frequency.
3 // Author : Fred Swartz - April 2007 - Placed in public domain.
4
5 import java.util.*;
6
7 ////////////////////////////////////////////////// class ComparatorFrequency
8 /** Order words from least to most frequent, put ties in alphabetical order. */
9 class ComparatorFrequency implements Comparator<Map.Entry<String, Int>> {
10     public int compare(Map.Entry<String, Int> obj1
11         , Map.Entry<String, Int> obj2) {
12         int result;
13         int count1 = obj1.getValue().value;
14         int count2 = obj2.getValue().value;
15         if (count1 < count2) {
16             result = -1;
17         } else if (count1 > count2) {
18             result = 1;
19         } else {
20             //... If counts are equal, compare keys alphabetically.
21             result = obj1.getKey().compareTo(obj2.getKey());
22         }
23         return result;
24     }
25 }
26 }
27 }
```

## Mutable Integers

Because only objects (not primitives) can be stored in Collections data structures, the integer count must be an object, not just an `int`. The natural choice would be the `Integer` wrapper type, but it is *immutable*, which means that once an `Integer` object is created, its value can never be changed. But here we need to update the value, so here's a class that stores an `int` that can be changed.

```
1 // File : data-collections/wordfreq2/Int.java
2 // Purpose: Simple value class to hold a mutable int.
3 // Author : Fred Swartz - March 2005 - Placed in public domain.
4 ////////////////////////////////////////////////// value class Int
5 /** Utility class to keep int as Object but allow changes (unlike Integer).
6  * Java collections hold only Objects, not primitives, but need to update value.
7  * The intention is that the public field should be used directly.
8  * For a simple value class this is appropriate. */
9 class Int {
10     //===== fields
11     public int value; // Just a simple PUBLIC int.
12
13     //===== constructor
14     /** Constructor
15      * @param value Initial value. */
16     public Int(int value) {
17         this.value = value;
18     }
19 }
```

# Second Program

```
1 // GCS Exercise 10.1 Solution: MyShape.java
2 // Declaration of class MyShape.
3 import java.awt.Color;
4 import java.awt.Graphics;
5
6 public abstract class MyShape
7 {
8     private int x1; // x coordinate of first endpoint
9     private int y1; // y coordinate of first endpoint
10    private int x2; // x coordinate of second endpoint
11    private int y2; // y coordinate of second endpoint
12    private Color myColor; // color of this shape
13
14    // default constructor initializes values with 0
15    public MyShape()
16    {
17        this( 0, 0, 0, 0, Color.BLACK ); // call constructor to set values
18    } // end MyShape no-argument constructor
19
20    // constructor
21    public MyShape( int x1, int y1, int x2, int y2, Color color )
22    {
23        setX1( x1 ); // set x coordinate of first endpoint
24        setY1( y1 ); // set y coordinate of first endpoint
25        setX2( x2 ); // set x coordinate of second endpoint
26        setY2( y2 ); // set y coordinate of second endpoint
27        setColor( color ); // set the color
28    } // end MyShape constructor
29
30    // set the x-coordinate of the first point
31    public void setX1( int x1 )
32    {
33        this.x1 = ( x1 >= 0 ? x1 : 0 );
34    } // end method setX1
35
36    // get the x-coordinate of the first point
37    public int getX1()
38    {
39        return x1;
40    } // end method getX1
41
42    // set the x-coordinate of the second point
43    public void setX2( int x2 )
44    {
45        this.x2 = ( x2 >= 0 ? x2 : 0 );
46    } // end method setX2
47
48    // get the x-coordinate of the second point
49    public int getX2()
50    {
51        return x2;
52    } // end method getX2
53
54    // set the y-coordinate of the first point
55    public void setY1( int y1 )
56    {
57        this.y1 = ( y1 >= 0 ? y1 : 0 );
58    } // end method setY1
59
60    // get the y-coordinate of the first point
61    public int getY1()
62    {
63        return y1;
64    } // end method getY1
65
66    // set the y-coordinate of the second point
67    public void setY2( int y2 )
68    {
69        this.y2 = ( y2 >= 0 ? y2 : 0 );
70    } // end method setY2
71
72    // get the y-coordinate of the second point
```

<pre> 73 public int getY2() 74 { 75     return y2; 76 } // end method getY2 77 78 // set the color 79 public void setColor( Color color ) 80 { 81     myColor = color; 82 } // end method setColor 83 84 // get the color 85 public Color getColor() 86 { 87     return myColor; 88 } // end method getColor 89 90 // abstract draw method 91 public abstract void draw( Graphics g ); 92 } // end class MyShape </pre>	
<pre> 1 // GCS Exercise 10.1 Solution: MyOval.java 2 // Declaration of class MyOval. 3 import java.awt.Color; 4 import java.awt.Graphics; 5 6 public class MyOval extends MyShape 7 { 8     private boolean filled; // whether this shape is filled 9 10 // call default superclass constructor 11 public MyOval() 12 { 13     super(); 14     setFilled( false ); 15 } // end MyOval no-argument constructor 16 17 // call superclass constructor passing parameters 18 public MyOval( int x1, int y1, int x2, int y2, 19     Color color, boolean isFilled ) 20 { 21     super( x1, y1, x2, y2, color ); 22     setFilled( isFilled ); 23 } // end MyOval constructor 24 25 // get upper left x coordinate 26 public int getUpperLeftX() 27 { 28     return Math.min( getX1(), getX2() ); 29 } // end method getUpperLeftX 30 31 // get upper left y coordinate 32 public int getUpperLeftY() 33 { 34     return Math.min( getY1(), getY2() ); 35 } // end method getUpperLeftY 36 37 // get shape width 38 public int getWidth() 39 { 40     return Math.abs( getX2() - getX1() ); 41 } // end method getWidth 42 43 // get shape height 44 public int getHeight() 45 { 46     return Math.abs( getY2() - getY1() ); 47 } // end method getHeight 48 49 // determines whether this shape is filled 50 public boolean isFilled() 51 { 52     return filled; 53 } // end method isFilled 54 55 // sets whether this shape is filled 56 public void setFilled( boolean isFilled ) </pre>	

<pre> 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 </pre>	<pre> {     filled = isFilled; } // end method setFilled  // draw oval public void draw( Graphics g ) {     g.setColor( getColor() );      if ( isFilled() )         g.fillOval( getUpperLeftX(), getUpperLeftY(),             getWidth(), getHeight() );     else         g.drawOval( getUpperLeftX(), getUpperLeftY(),             getWidth(), getHeight() ); } // end method draw } // end class MyOval </pre>	
<pre> 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 </pre>	<pre> // GCS Exercise 10.1 Solution: MyRect.java // Declaration of class MyRect. import java.awt.Color; import java.awt.Graphics;  public class MyRect extends MyShape {     private boolean filled; // whether this shape is filled      // call default superclass constructor     public MyRect()     {         super();         setFilled( false );     } // end MyRect no-argument constructor      // call superclass constructor passing parameters     public MyRect( int x1, int y1, int x2, int y2,         Color color, boolean isFilled )     {         super( x1, y1, x2, y2, color );         setFilled( isFilled );     } // end MyRect constructor      // get upper left x coordinate     public int getUpperLeftX()     {         return Math.min( getX1(), getX2() );     } // end method getUpperLeftX      // get upper left y coordinate     public int getUpperLeftY()     {         return Math.min( getY1(), getY2() );     } // end method getUpperLeftY      // get shape width     public int getWidth()     {         return Math.abs( getX2() - getX1() );     } // end method getWidth      // get shape height     public int getHeight()     {         return Math.abs( getY2() - getY1() );     } // end method getHeight      // determines whether this shape is filled     public boolean isFilled()     {         return filled;     } // end method is filled      // sets whether this shape is filled     public void setFilled( boolean isFilled )     {         filled = isFilled;     } // end method setFilled </pre>	

<pre> 60 61 62 63 64 65 66 67 68 69 70 71 72 73 </pre>	<pre> // draw rectangle public void draw( Graphics g ) {     g.setColor( getColor() );     if ( isFilled() )         g.fillRect( getUpperLeftX(), getUpperLeftY(),             getWidth(), getHeight() );     else         g.drawRect( getUpperLeftX(), getUpperLeftY(),             getWidth(), getHeight() ); } // end method draw } // end class MyRect </pre>	
<pre> 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 </pre>	<pre> // GCS Exercise 10.1 Solution: MyLine.java // Declaration of class MyLine. import java.awt.Color; import java.awt.Graphics;  public class MyLine extends MyShape {     // call default superclass constructor     public MyLine()     {         super();     } // end MyLine no-argument constructor      // call superclass constructor passing parameters     public MyLine( int x1, int y1, int x2, int y2, Color color )     {         super( x1, y1, x2, y2, color );     } // end MyLine constructor      // draw line in specified color     public void draw( Graphics g )     {         g.setColor( getColor() );         g.drawLine( getX1(), getY1(), getX2(), getY2() );     } // end method draw } // end class MyLine </pre>	
<pre> 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 </pre>	<pre> // GCS Exercise 10.1 Solution: DrawPanel.java // Program randomly draws shapes. import java.awt.Color; import java.awt.Graphics; import java.util.Random; import javax.swing.JPanel;  public class DrawPanel extends JPanel {     private Random randomNumbers = new Random();     private MyShape shapes[]; // array containing all the shapes     private int shapeCount[]; // statistic on the number of each shape      // String containing shape statistic information     private String statusText;      // constructor that takes in the number of shapes     public DrawPanel( int numberShapes )     {         setBackground( Color.WHITE );          shapes = new MyShape[ numberShapes ];         shapeCount = new int[ 3 ];          for ( int i = 0; i &lt; shapes.length; i++ )         {             // generate random coordinates             int x1 = randomNumbers.nextInt( 450 );             int x2 = randomNumbers.nextInt( 450 );             int y1 = randomNumbers.nextInt( 450 );             int y2 = randomNumbers.nextInt( 450 ); </pre>	

<pre> 33 // generate a random color 34 Color color = new Color ( randomNumbers.nextInt( 256 ), 35 randomNumbers.nextInt( 256 ), randomNumbers.nextInt( 256 ) ); 36 37 // get filled property 38 boolean filled = randomNumbers.nextBoolean(); 39 40 int shapeType = randomNumbers.nextInt( 3 ); 41 ++shapeCount[ shapeType ]; 42 43 switch ( shapeType ) 44 { 45 case 0: // line 46 shapes[ i ] = new MyLine( x1, y1, x2, y2, color ); 47 break; 48 case 1: // oval 49 shapes[ i ] = new MyOval( x1, y1, x2, y2, color, filled ); 50 break; 51 case 2: // rectangle 52 shapes[ i ] = new MyRect( x1, y1, x2, y2, color, filled ); 53 break; 54 } // end switch 55 } // end for 56 57 // create string for the status bar label 58 statusText = String.format( " %s: %d, %s: %d, %s: %d", 59 "Lines", shapeCount[ 0 ], "Ovals", shapeCount[ 1 ], 60 "Rectangles", shapeCount[ 2 ] ); 61 } // end DrawPanel constructor 62 63 // returns a label containing shape statistics for this panel 64 public String getLabelText() 65 { 66 return statusText; 67 } // end method getStatusLabel 68 69 // draw shapes using polymorphism 70 public void paintComponent( Graphics g ) 71 { 72 super.paintComponent( g ); 73 74 for ( MyShape shape : shapes ) 75 shape.draw( g ); 76 } // end method paintComponent 77 } // end class DrawPanel </pre>	
<pre> 1 // GCS Exercise 10.1 Solution: TestDraw.java 2 // Test application to display a DrawPanel 3 import java.awt.BorderLayout; 4 import javax.swing.JLabel; 5 import javax.swing.JOptionPane; 6 import javax.swing.JFrame; 7 8 public class TestDraw 9 { 10 public static void main( String args[] ) 11 { 12 // Ask the user to enter the number of shapes 13 int shapes = Integer.parseInt( 14 JOptionPane.showInputDialog( "Number of shapes" ) ); 15 16 DrawPanel panel = new DrawPanel( shapes ); 17 JFrame application = new JFrame(); 18 19 // Create a label containing the shape information 20 JLabel statusLabel = new JLabel( panel.getLabelText() ); 21 22 application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE ); 23 24 application.add( panel ); // add drawing to CENTER by default 25 // Add the status label to the SOUTH (bottom) of the frame 26 application.add( statusLabel, BorderLayout.SOUTH ); 27 28 application.setSize( 500, 500 ); 29 application.setVisible( true ); 30 } // end main 31 } // end class TestDraw </pre>	

