Richard G Baldwin (512) 223-4758, baldwin@austin.cc.tx.us,
http://www2.austin.cc.tx.us/baldwin/

The AWT Package, Graphics- Working with Fonts

Java Programming, Lecture Notes # 168, Revised 03/02/99.

Preface

Students in Prof. Baldwin's **Advanced Java Programming** classes at ACC are responsible for knowing and understanding all of the material in this lesson.

3/2/99 Note: The sample program in this lesson named Graphics08.java does not compile and run properly under JDK 1.2.

Introduction

An earlier lesson provided an overview of the **Graphics** class, and grouped the methods of that class into several different categories. This lesson will explore some of the methods in the category of **Drawing Text**.

To review, the following methods were put into this category in the earlier lesson.

- **drawString**(String, int, int) - Draws the text given by the specified string, using this graphics context's current font and color.
- **drawChars**(char[], int, int, int, int) - Draws the text given by the specified character array, using this graphics context's current font and color. Another version lets you pass an array of bytes to represent the characters to be drawn.
- **getFont**() - Gets the current font and returns an object of type **Font** which describes the context's current font.
- **getFontMetrics**() - Gets the font metrics of the current font. Returns an object of type **FontMetrics**. Methods of the **FontMetrics** class can be used to obtain metrics information (size, etc.) about the font to which the **getFontMetrics()** method is applied.
- **getFontMetrics**(Font) - Gets the font metrics for the specified font.

- **setFont**(Font) - Sets this graphics context's font to the specified font.

As is often the case when programming in Java, what you see in the above list represents only the controls in the cockpit. The real power lies "under the hood."

To understand what is "under the hood", you must examine the classes for the objects required as parameters by the above methods and the classes for the objects returned by the above methods.

A quick examination of the list indicates that you need to understand the following classes, as a minimum, in order to understand how to draw text on the screen. This list includes an additional item (**Toolkit)** that isn't obvious from examining the above list.

- String
- Font
- FontMetrics
- Toolkit

Hopefully, you already know about the **Sting** class. The following sections will provide a non-exhaustive discussion of the **Font, FontMetrics**, and **Toolkit** classes.

The Toolkit Class

We will begin with the toolkit class because it is needed to obtain some fundamental information about the available fonts.

The following description of the **Toolkit** class was extracted from the JavaSoft documentation for JDK 1.1.3.

This class is the abstract superclass of all actual implementations of the Abstract Window Toolkit. Subclasses of Toolkit are used to bind the various components to particular native toolkit implementations.

Most applications should not call any of the methods in this class directly. The methods defined by Toolkit are the "glue" that joins the platform-independent classes in the java.awt package with their counterparts in java.awt.peer. Some methods defined by Toolkit query the native operating system directly.

A complete discussion of the **Toolkit** class is beyond the scope of this lesson. However, there is one method of this class that is necessary for the effective use of fonts in Java. A description of that method follows:

public static synchronized Toolkit getDefaultToolkit()

Gets the default toolkit.

If there is a system property named "awt.toolkit", that property is treated as the name of a class that is a subclass of **Toolkit**.

If the system property does not exist, then the default toolkit used is the class named "sun.awt.motif.MToolkit", which is a motif implementation of the Abstract Window Toolkit.

Returns: the default toolkit.

Throws: AWTError if a toolkit could not be found, or if one could not be accessed or instantiated.

To bring this rather abstract discussion to a close, we will be executing the following statement in our sample program to obtain the list of fonts available on the platform that we are using.

```
String[] fonts = Toolkit.getDefaultToolkit().getFontList();
```

As of 11/26/97, the list of fonts on my system running JDK 1.1.3 under Win95, as reported by the **getFontList()** method was:

```
Dialog
SansSerif
Serif
Monospaced
Helvetica
TimesRoman
Courier
DialogInput
ZapfDingbats
```

I won't provide any further discussion about the availability of the different fonts. However, John Zukowski's book entitled Java AWT Reference contains an excellent discussion of how fonts are included on a particular system, where the names come from, and how it is possible to create an alias for a font name in the System Properties.

There are some important concepts described in John's book, including how to write Java programs in which the user has the ability to tailor the application by specifying fonts and making them persist from one invocation of the application to the next.

The Font Class

The **Font** class provides the following symbolic constants that are used to establish the *style* of an instantiated **Font** object.

- PLAIN
- BOLD
- ITALIC

The names of these constants should make it obvious what they are used for. They can be mixed where appropriate using the bitwise **or** operator in their specification. Note that *underline* is not available as one of the *style* constants. If you want your text to be underlined, you will have to take care of that yourself, probably using the **drawLine()** method of the **Graphics** class.

The **Font** class provides a single constructor as shown below.

Font(String, int, int) - Creates a new font object with the specified name, style and point size where the name is provided by the **String** parameter, and the style and point size are provided by the two **int** parameters.

The **Font** class contains many methods. The following list of methods was extracted from the JDK 1.1.3 documentation. We will use some of these methods in the sample program that we will present later. The purpose of most of these methods should be obvious from the name of the method. You may need to refer to the JavaSoft documentation for those that aren't obvious.

decode(String) - Gets the specified font using the name passed in.
equals(Object) - Compares this object to the specified object.
getFamily() - Gets the platform specific family name of the font.
getFont(String) - Gets a font from the system properties list.
getFont(String, Font) - Gets the specified font from the system properties list.
getName() - Gets the logical name of the font.
getPeer() - Gets the peer of the font.
getSize() - Gets the point size of the font.
getStyle() - Gets the style of the font.
hashCode() - Returns a hashcode for this font.
isBold() - Indicates whether the font's style is bold.
isItalic() - Indicates whether the font's style is italic.
isPlain() - Indicates whether the font's style is plain.
toString() Converts this object to a String representation.

.

The FontMetrics Class

A **FontMetrics** object provides information about the rendering of a particular font on a particular platform.

There is a lot involved in the design of a font and the placement of that font on the screen. A lot of information about a particular font can be obtained from a **FontMetrics** object instantiated for that particular font. This includes information about the *baseline*, *ascent*, *descent*, *advance width*, and *leading* which are all attributes of a particular font.

It is also possible to obtain information about the total size requirements of a string object when rendered in a given font.

Quite a lot of information about fonts in general, and information about these methods is provided in the JDK documentation for the **FontMetrics** class. You are referred to that documentation for a full understanding of what makes up a font, and what information does the **FontMetrics** class provide about a font.

In this lesson, we will confine our interest to the total size requirements (height and width) of a string rendered in a particular font, and leave the other attributes that describe a particular font for those persons interested in such matters.

Sample Program

The following program illustrates the drawing of text using methods primarily of the **Graphics**, **Font**, and **FontMetrics** classes. The **Toolkit** class is also used to obtain a list of fonts available on the machine at the time the program was run.

A variety of methods are illustrated along with a variety of font types, styles, sizes, and colors.

As mentioned earlier, the list of available fonts reported by the **getFontList()** method was as shown below.

- Dialog
- SansSerif
- Serif
- Monospaced
- Helvetica
- TimesRoman
- Courier
- DialogInput
- ZapfDingbats

An array of strings was created containing the names of each of these fonts, and subsequent code caused the name of each font to be displayed in the font identified by the name.

The size of each line of output (containing the name of a font) was increased as each successive font was displayed and the vertical spacing was made appropriate for each font. The horizontal size of each string was determined and used to center each string in the output.

According to John Zukowski in Java AWT Reference:

"The **ZapfDingbats** font name has been dropped completely because the characters in this font have official Unicode mappings in the range \u2700 to \u27ff."

As shown above, the **ZapfDingbats** font name still appears on my Win95 system as of JDK 1.1.3. However, an attempt to draw text using the **ZapfDingbats** font resulted in a series of identical rectangular boxes being drawn, one for each character in the text. Therefore, no attempt was made to demonstrate or draw text using the font associated with this name in this program.

This program was tested using JDK 1.1.3 under Win95.

If you will compile and run this program, and observe the output while walking through the source code (keeping your JDK documentation close at hand as you go), you should have no difficulty understanding how to use the text-drawing methods of the **Graphics** class, as well as a number of methods in the **Font** and **FontMetrics** classes. The extensive comments provided in the text should help you to develop this understanding.

```
/*File Graphics08.java
Copyright 1997, R.G.Baldwin

This program illustrates the drawing of text using methods
primarily of the Graphics, Font, and FontMetrics classes.

This program was tested using JDK 1.1.3 under Win95.

**************************************************/
import java.awt.*;
import java.awt.event.*;

class Graphics08 extends Frame{ //controlling class
  //Override the paint method
  public void paint(Graphics g){
    g.setColor(Color.red);//set the drawing color to red

    //Translate the 0,0 coordinate of the graphics context
    // to the upper left-hand corner of the client area of
    // the Frame object.
    g.translate(
               this.getInsets().left,this.getInsets().top);

    //Get list of fonts on the system
    String[] fonts =
                  Toolkit.getDefaultToolkit().getFontList();
```

```
    //Set the font to the fifth one in the list.  Set the
    // style to BOLD.  Set the size to 20-point.
    g.setFont(new Font(fonts[4],Font.BOLD,20));

    //Get the height attribute for the selected font
    int height = g.getFontMetrics().getHeight();

    //Set the initial y coordinate using the height
    // attribute.
    int y = height;

    //Display a title using the selected font at the
    // specified y-coordinate in red.  Center the title
    // around an x-ccordinate value of 200 pixels.
    String msg = "The list of available fonts follows:";
    g.drawString(
      msg,200 - (g.getFontMetrics().stringWidth(msg))/2,y);

    //Set drawing color to green
    g.setColor(Color.green);

    //Display the list of available fonts in green except
    // don't try to display ZapfDingbats because it is
    // used to represent characters beyond the first 256.
    // Display the name of each font in the font that
    // matches the name.  Center around a value of 200
    // pixels.  Increase the size of the font for each
    // line of output by adding the loop count to a base
    // size of 17 points.  Adjust the vertical spacing
    // between lines of output appropriately as the size
    // of the font increases. Set the style to BOLD ITALIC.
    for(int cnt = 0; cnt < fonts.length-1; cnt++){
      g.setFont(new Font(fonts[cnt],
                         Font.BOLD | Font.ITALIC,17+cnt));
      height = g.getFontMetrics().getHeight();
      g.drawString(fonts[cnt],
        200-(g.getFontMetrics().stringWidth(fonts[cnt]))/2,
                                                y+=height);
    }//end for loop

    //Set drawing color to blue
    g.setColor(Color.blue);

    //Set the font to the first one in the list of fonts.
    // Set the style to PLAIN.  Make it 17-point size.
    g.setFont(new Font(fonts[0],Font.PLAIN,17));
    height = g.getFontMetrics().getHeight();

    //Display another title
    y += height;
    g.drawString(
      "INFORMATION ABOUT THE SELECTED FONT FOLLOWS: ",5,y);

    //Display information about the current font
    y += height;
```

```
    g.drawString("" + g.getFont(),5,y);

    //Display another title
    y += 2*height;
    g.drawString(
                "PART OF THE AUTHOR'S NAME FOLLOWS: ",5,y);

    //Create a character array and display part of it
    char myData[] =
        {'D','i','c','k',' ','B','a','l','d','w','i','n',};
    y += height;
    //Display the data in the array, skipping the first
    // and last characters in the array
    g.drawChars(myData,1,10,5,y);

  }//end paint()

  public Graphics08(){//constructor
    this.setTitle(
              "Shape Samples,Copyright 1997, R.G.Baldwin");
    this.setSize(500,400);
    this.setVisible(true);

    //Anonymous inner-class listener to terminate program
    this.addWindowListener(
      new WindowAdapter(){//anonymous class definition
        public void windowClosing(WindowEvent e){
          System.exit(0);//terminate the program
        }//end windowClosing()
      }//end WindowAdapter
    );//end addWindowListener
  }//end constructor

  public static void main(String[] args){
    new Graphics08();//instantiate this object
  }//end main
}//end Graphics08 class
//=========================================================//
```

-end-