DRAWING GRAPHICS IN JAVA USING THE **Graphics2D** CLASS

Swing is a package that allows programmers to create applications that use a Graphical User Interface (or GUI) instead of console-based programs. Graphical User Interfaces often include graphical components such as windows, command buttons, labels, text boxes, option buttons, etc. that users interact with. The Swing package includes numerous classes, but the first one we'll need to look at is the **JFrame** class.

JFrame

The top-level component for most Swing-based applications is called a **frame** and is created using the **JFrame** class. A frame is a window with a border, title, and buttons for minimizing, maximizing, and closing the frame.

The following are some of the more common constructors and methods used when creating, and working with, a **JFrame**:

CONSTRUCTOR

CONSTRUCTOR	DESCRIPTION
JFrame()	Creates a new frame with no title.
JFrame(String x)	Creates a new frame with a title.

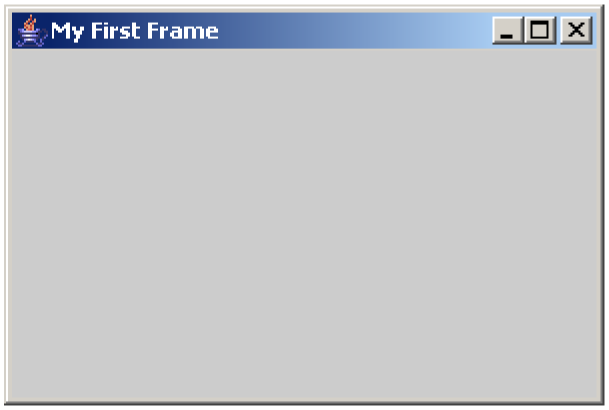
METHOD

METHOD	DESCRIPTION
void pack()	Adjusts the size of the frame to fit all the components.
void setBounds(int x, int y, int width, int height)	Sets the location (x and y co-ordinates) and size (width and height) of the frame.
void setDefaultCloseOperation()	Sets the action that will happen by default when the user clicks the Close button. The argument is usually **JFrame.EXIT_ON_CLOSE**.
void setLayout(LayoutManager x)	Sets the layout manager that is used to control how components are arranged when the frame is displayed. **BorderLayout** manager is the default, which
void setLocation(int x, int y)	Sets the x- and y-position of the top-left corner of the frame.

void setLocationRelativeTo (Component c)	Sets the frame's location relative to another component. If the parameter is **null**, the frame is centred on the screen.
void setResizable(boolean x)	Sets whether or not the size of the frame can be changed by the user; the default setting is **true**.
void setSize(int width, int height)	Sets the width and height of the frame.
void setTitle(String x)	Sets the title of the frame.
void setVisible(boolean x)	Shows this component if set to **true**; otherwise hides it (if set to **false**).

The following is an example of a frame where the **setTitle**, the **setSize**, **setResizeable**, **setDefaultCloseOperation** and the **setVisible** properties have been set:

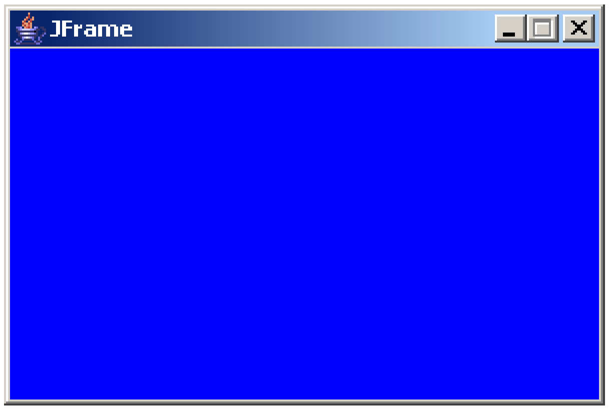
```
import javax.swing.*;

public class FirstFrame extends JFrame {

  public FirstFrame() {

    // Set properties of the frame
    setTitle("My FirstFrame");
    setSize(300, 200);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setResizable(false);
    setVisible(true);
  }

  public static void main(String[] args) {

    new FirstFrame();
  }
}
```



SETTING BACKGROUND COLOUR

You can change the background colour of your frame by using the **getContentPane()** and **setBackground()** methods included in the **JFrame** class. The following line of code will change the background colour of my frame to the default blue:

```
getContentPane().setBackground(Color.BLUE);
```



The content pane is a container that represents the main rectangle of the frame. In order to set anything within the content pane of the frame, you need to get a reference to the content pane using the **getContentPane()** method and then set the property you wish to set within the frame.

You can use any one of the following static fields included in the **Color** class to set the colour of your background:

Color.BLACK	Color.BLUE	Color.CYAN	Color.DARK_GRAY	Color.GRAY
Color.GREEN	Color.LIGHT_GRAY	Color.MAGENTA	Color.ORANGE	Color.PINK
	Color.RED	Color.WHITE	Color.YELLOW	

Alternatively, you can create your own custom colour by creating a **Color** object using the RGB colour model which is an additive colour model in which red, green and blue light is added together in various ways to reproduce a broad array of colours. RGB values are comprised of three numbers between 0 and 255. So, for example, if I want to set my background to a specific shade of purple, I would need a mixture of red and blue. The line of code to produce a purple background would look something like this:

```
getContentPane().setBackground(new Color(255, 0, 255));
```

USING THE **Graphics2D** CLASS

One way to print text, draw simple shapes, and position images on a component, such as a frame, is by using the **Graphics2D** class which is included in the **java.awt** package.

The following are some of the more common constructors and methods available in the **Graphics2D** class:

CONSTRUCTOR

CONSTRUCTOR	DESCRIPTION
Graphics2D()	Creates a new **Graphics2D** object.

METHODS

METHODS	DESCRIPTION
void draw(Shape s)	Draws the outline of a shape using the settings of the current Graphics2D context.
void draw3DRect(int x, int y, int width, int height, boolean raised)	Draws a 3-D highlighted outline of the specified rectangle.
void drawImage(Image img, int x, int y, ImageObserver observer)	Draws a specified image with its top-left corner indicated by the x- and y-coordinates. The **ImageObserver** can just be set to **null**.
void drawString(String str, int x, int y)	Draws the text given by the specified string, using this graphics context's current font and color.
void fill(Shape s)	Fills the interior of a shape.
void setColor(Color c)	Sets this graphics context's current color to the specified color.
void setFont(Font f)	Sets this graphics context's font to the specified font.
void setStroke(Stroke s)	Sets the thickness of the line used to draw a shape.

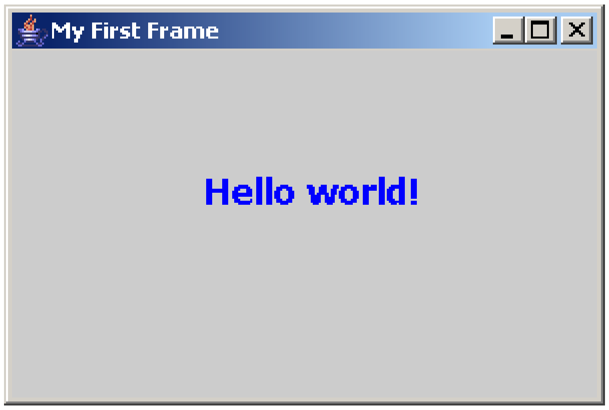
If we wanted to output the message "Hello world!" using the interface that we created above and set the text to a shade of blue and the font size and type to Tahoma, size 18, the following code would need to be added to our program:

Import java.awt.* $\longrightarrow$

```
import javax.swing.*;
import java.awt.*;

public class FirstFrame extends JFrame {

  public FirstFrame() {

    // Set properties of the frame
    setTitle("My FirstFrame");
    setSize(300, 200);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setResizable(false);
```

```
        setVisible(true);
    }

    public void paint(Graphics g) {

        // Repaints the frame and its components
        super.paint(g);

        // Declare and initialize a Graphics2D object
        Graphics2D g2 = (Graphics2D) g;

        // Declare and initialize Font and Color objects
        Font f = new Font("Tahoma", Font.BOLD, 18);
        Color blue = new Color(0, 0, 255);

        // Set color and font properties
        g2.setFont(f);
        g2.setColor(blue);

        // Output message
        g2.drawString("Hello world!", 100, 100);
    }

    public static void main(String[] args) {

        new FirstFrame();
    }
}
```

Add these lines



Hello world!

THE **FontMetrics** CLASS

You can use the **FontMetrics** class to get information about the geometry of the type of font that you use to draw strings. This class comes in handy when you want to center a string that you draw on the screen.

To centre text on the screen you will need to do the following:

1. Create and instantiate a **Font** object:

   ```
   Font f = new Font("Papyrus", Font.BOLD, 18);
   ```

2. Create and instantiate a **FontMetrics** object:

   ```
   FontMetrics fm = getFontMetrics(f);
   ```

3. Get the width of the string that you want to output to the screen:

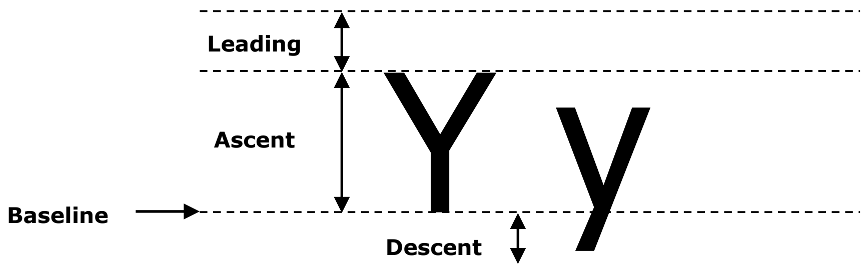
   ```
   int strWidth = fm.stringWidth("Hello world!");
   ```

4. Draw the string and make it centred horizontally on the screen:

   ```
   g2.drawString("Hello world!", getWidth() / 2 - strWidth /
      2, 100);
   ```

In the above example, the y-position of the string is set to 100, but if you wanted to center the text vertically on the screen as well, you would need to use the **getHeight()** method included in the **FontMetrics** class. The **getHeight()** method returns the standard height of a line of text in the specified font.

To centre text vertically is not as straightforward. This is how vertical space is allocated to text in Java:



LEADING:	The font's recommended space between lines of text.
ASCENT:	The distance from the font's baseline to the top of most alphanumeric characters.
DESCENT:	The distance from the font's baseline to the bottom of most alphanumeric characters.
BASELINE:	The position at the bottom of characters, but above their descenders. The y-coordinate of **drawString()** specifies the baseline position.

So, in order to centre text vertically on the string, you will need to set the y-position of the text at the halfway point of the **JFrame** + half of the ascent.

Here’s the code that would be used to center the string “Hello world!” both horizontally and vertically.

```
g2.drawString(“Hello world!”, getWidth() / 2 – strWidth /
   2, getHeight() / 2 - fm.getAscent() / 2);
```