Painting a Swing Component

Three methods are at the heart of painting a swing component like JPanel etc. For instance, *paint()* gets called when it's time to render -- then Swing further factors the *paint()* call into three separate methods, which are invoked in the following order:

— protected void *paintComponent(Graphics g)*
— protected void *paintBorder(Graphics g)*
— protected void *paintChildren(Graphics g)*

Lets look at these methods in order in which they get executed

paintComponet( )

— it is a main method for painting
— By default, it first paints the background
— After that, it performs custom painting (drawing circle, rectangles etc.)

paintBorder( )

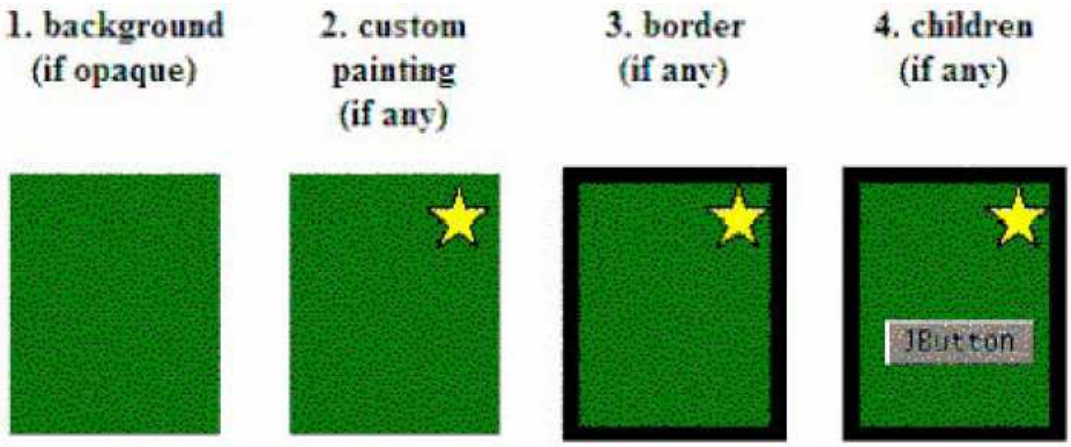
— Tells the components border (if any) to paint.
— It is suggested that you do not override or invoke this method

paintChildern( )

— Tells any components contained by this component to paint themselves
— It is suggested that you do not override or invoke this method too.

Example: Understanding methods calls

Consider the following figure



The figure above illustrates the order in which each component that inherits from *JComponent* paint itself. Figure 1 to 2 --painting the background and performing custom painting is performed by the *paintComponent* method

In Figure 3 – *paintBorder* is get called And finally in figure 4 – *paintChildern* is called that causes the *JButton* to render itself. Note: The important thing to note here is for *JButton* (since it is a *JComponent*), all these methods are also called in the same order.

Your Painting Strategy
You must follow the three steps in order to perform painting.

Subclass JPanel
– class MyPanel extends JPanel
– Doing so MyPanel also becomes a JPanle due to inheritance

Override the paintComponent(Graphics g) method
– Inside method using graphics object, do whatever drawing you want to do

Install that JPanel inside a JFrame
– When frame becomes visible through the *paintChildren()* method your panel become visible
– To become visible your panel will call *paintComponent()* method which will do your custom drawing

Example # 01 (Basic Example of create drawing)

Step 1: Create a general purpose component

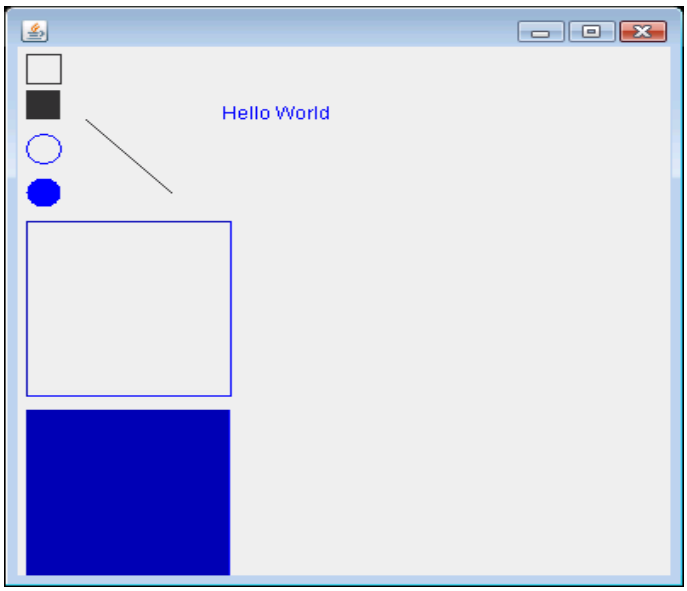
MyPanel.java

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

public class MyPanel extends JPanel {// extending class from JPanel

// overriding paintComponent method
public void paintComponent(Graphics g){

    // Down casting Graphics object to Graphics2D
    Graphics2D g2 = (Graphics2D) g;

    // drawing rectanle
    g2.drawRect(5,5,20,20);
    g2.fillRect(5,30,20,20);

    // drawing line
    g2.drawLine(40,50,90,100);

    // changing the color to blue
    g2.setColor(Color.blue);

    // drawing filled oval with color i.e. blue
    g2.drawOval(5,60,20,20);
    g2.fillOval(5,90,20,20);

    g2.draw3DRect(5, 120, 120, 120, false);
    g2.fill3DRect(5, 250, 120, 120, false);

    // drawing string
    g2.drawString("Hello World", 120, 50);

}// end paintComponent
} // end MyPanel class
```

Step 2: Create JFrame (top level container) and add JPanel (General purpose container) on JFrame

```
// importing required packages
import javax.swing.*; import java.awt.*;
public class Test {
JFrame f;

// declaring Reference of MyPanel class
MyPanel p;

// parameter less constructor
public Test(){

    f = new JFrame();
    Container c = f.getContentPane();
    c.setLayout(new BorderLayout());

    // instantiating reference
    p = new MyPanel();

    // adding MyPanel into container
    c.add(p);
    f.setSize(400,400);
    f.setVisible(true);
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
} // end constructor

// main method
public static  void main(String args[ ]){
    Test t = new Test();
}
} // end of class
```

Output:



How to Animate?

If we want to animate something like ball, moving from one place to another, we constantly need to call *paintComponent( )* method and to draw the shape (ball etc.) at new place means at new coordinates.
Painting is managed by system, so calling *paintComponent()* directly is not recommended at all. Similarly calling *paint( )* method is also not recommended. Why? Because such code may be invoked at times when it is not appropriate to paint -- for instance, before the component is visible or has access to a valid Graphics object.
Java gives us a solution in the from of *repaint( )* method. Whenever we need to repaint, we call this method that in fact makes a call to *paint( )* method at appropriate time.

Example:
The ball is continuously moving freely inside the corner of the frames. The sample outputs are shown below:

First we examine the MyPanel.java class that is drawing a filled oval.

<u>Example # 02</u> (Basic Example of create drawing and insert some interactivity in components)

Step 1: Create a general purpose component

MyPanel.java

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

// extending class from JPanel
public class MyPanel extends JPanel {

int mX = 200;
int mY = 0;
// overriding paintComponent method
public void paintComponent(Graphics g){
// erasing behaviour this will clear all the previous painting
super.paintComponent(g);

// Down casting Graphics object to Graphics2D
Graphics2D g2 = (Graphics2D)g;

// changing the color to blue
g2.setColor(Color.blue);
// drawing filled oval with blue color

// using instance variables
g2.fillOval(mX,mY,20,20);

}// end paintComponent
} //send of MyPanel class
```

Step 2: Create JFrame (top level container) and add JPanel (General purpose container) on JFrame

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

public class Test implements ActionListener {
JFrame f;
MyPanel p;

// used to control the direction of ball
int x, y;

public Test(){
    f = new JFrame();
    Container c = f.getContentPane();
    c.setLayout(new BorderLayout());
    x = 5;
    y = 3;
    p = new MyPanel();
    c.add(p);

    f.setSize(400,400);
    f.setVisible(true);
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

/*creating a Timer class object, used for firing one or more
action events after a specified delay
Timer class constructor requires time in milliseconds
and object of class that handles action events*/
    Timer t = new Timer (5, this);
// starts the timer,causing it to start sending action events to listeners
    t.start();
} // end constructor

// event handler method
public void actionPerformed(ActionEvent ae){

// if ball reached to maximum width of frame minus 40
//since diameter of ball is 40 then change the X-direction of ball
if (f.getWidth()-40 == p.mX)
x = -5;

// if ball reached to maximum height of frame minus 40
//then change the Y-direction of ball
if (f.getHeight()-40 == p.mY)
y = -3;

// if ball reached to min. of width of frame,change the X-direction of bal
if (p.mX == 0 )
x = 5;

// if ball reached to min. of height of frame, change the Y-direction of b
if (p.mY == 0 )
y = 3;

// Assign x,y direction to MyPanel's mX & mY
p.mX += x;
p.mY += y;

// call to repaint() method so that ball is drawn on// new locations
p.repaint();

} // end actionPerformed() method
// main method
public static void main(String args[ ]){
Test at = new Test();
}
} // end of AnimTest class
```

References:

 Painting in AWT & Swing
http://java.sun.com/products/jfc/tsc/articles/painting/index.html
 Performing Custom Painting
http://java.sun.com/docs/books/tutorial/uiswing/14painting/index.html