To put things simply, Java 2D is an API to render two-dimensional graphics on surfaces such as computer screens, printers, and devices. This powerful API allows you to do things such as drawing geometric shapes, image processing, alpha compositing (combining images), text font rendering, antialiasing, clipping, creating transformations, stroking, filling, and printing.

Breaking news! When giving news I’m sure you’ve heard people say, “I’ve got good news and bad news”. Well, in the case of Java 2D, it is good news and more good news. First, the Java 2D API has virtually been unchanged since its major release (Java 2), which is a testament to good design. Now for more good news, new to Java 7 is that the 2D API is getting a new graphics pipeline called XRender. XRender will have access to hardware accelerated features on systems with modern graphics processing units (GPUs). This is great news to many existing Java applications that use Java 2D already because they will gain excellent rendering performance without changing any code. Things can only get better as the major players get onboard with the Open JDK initiative.

Helper Class for This Chapter

This chapter will familiarize you with recipes pertaining to Java’s 2D APIs. Regarding the 2D API, you will notice that most recipes will rely on a utility helper class to launch an application window to display the examples. Shown below is the utility helper class SimpleAppLauncher.java which will utilize a javax.swing.JComponent object as a drawing surface to be displayed in an application window (javax.swing.JFrame).

---

For more on the latest Java 7 features including XRender, see OpenJDK http://openjdk.java.net/projects/jdk7/features (Oracle Corporation, 2011).

---

```java
package org.java7recipes.chapter12;

import java.awt.BorderLayout;
import java.awt.Component;
import java.awt.Dimension;
import java.awt.Toolkit;
import java.awt.event.ComponentAdapter;
import java.awt.event.WindowAdapter;
```
import java.awt.event.ComponentEvent;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.SwingUtilities;

/**
 * SimpleAppLauncher will create a window and display a component and
 * abide by the event dispatch thread rules.
 *
 * @author cdea
 */
public class SimpleAppLauncher {
    /**
     * @param title the Chapter and recipe.
     * @param canvas the drawing surface.
     */
    protected static void displayGUI(final String title, final JComponent component) {
        // create window with title
        final JFrame frame = new JFrame(title);
        if (component instanceof AppSetup) {
            AppSetup ms = (AppSetup) component;
            ms.apply(frame);
        }
        // set window's close button to exit application
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        component.addComponentListener(new ComponentAdapter() {
            // This method is called after the component's size changes
            public void componentResized(ComponentEvent evt) {
                Component c = (Component)evt.getSource();
                // Get new size
                Dimension newSize = c.getSize();
                System.out.println("component size w,h = "+newSize.getWidth()+", "+
                        newSize.getHeight());
            }
        });

        // place component in the center using BorderLayout
        frame.getContentPane().add(component, BorderLayout.CENTER);

        // size window based on layout
        frame.pack();

        // center window
        Dimension scrnSize = Toolkit.getDefaultToolkit().getScreenSize();
        int scrnWidth = frame.getSize().width;
        int scrnHeight = frame.getSize().height;
        int x = (scrnSize.width - scrnWidth) / 2;
        int y = (scrnSize.height - scrnHeight) / 2;
    }
}