**Project1- Fall CS596-019**                                                Name:  Manisha Amulakonda

                                                                                                       Student id : 1503012030

JavaBean makes it easy to reuse software components. Developers can use software components written by others without having to understand their inner workings.

In this Java Beans project, given the jar file Clock.jar which contains following:

Source files**Clock.java,ClockBeanInfo.java**,

and the respective compiled  class files **Clock.class** and **ClockBeaninfo.class**,

a manifest file - **clock.mf**,

**ClockIcon16,ClockIcon32** gif files.

If you are creating new components or beans for use within the visual editor (for example, adding components to the palette), you can control their behavior by supplying a BeanInfo class.

A BeanInfo class implements java.beans.BeanInfo and is used by the visual editor to determine the behavior of the properties view for the bean, and whether a customizer is available. The visual editor includes BeanInfo classes for common AWT and Swing controls.

Manifest file is a text file containing information about bean. In this Clock.mf file we have Clock.class as the java bean path is declared ;Hence the attribute **Java-Bean : True**

**Makefile**

JCC = javac

JFLAGS = -g

default: Clock.class ClockBeanInfo.class

Clock.class: Clock.java

       $(JCC) $(JFLAGS) Clock.java

ClockBeanInfo.class: ClockBeanInfo.java

       $(JCC) $(JFLAGS) ClockBeanInfo.java

clean:

       $(RM) \*.class

**Source Code**

// Clock Bean Class

// Clock.java

package HTPJB.Chap13.Clock;

// Imports

import java.awt.\*;

import java.awt.event.\*;

import java.beans.\*;

import java.io.Serializable;

import java.util.Calendar;

public class Clock extends Canvas implements Serializable, Runnable {

 private static final int    SPACING = 4;

 private transient Dimension offSize;

 private transient Image     offImage;

 private transient Graphics  offGrfx;

 private transient Thread    clockThread;

 private boolean             raised;

 private boolean             digital;

 // Constructors

 public Clock() {

   this(false, false);

 }

 public Clock(boolean r, boolean d) {

   // Allow the superclass constructor to do its thing

   super();

   // Set properties

   raised = r;

   digital = d;

   setBackground(Color.lightGray);

   // Set a default size

   setSize(100, 100);

   // Create the clock thread

   clockThread = new Thread(this, "Clock");

   clockThread.start();

 }

 // Accessor methods

 public boolean isRaised() {

   return raised;

 }

 public void setRaised(boolean r) {

   raised = r;

   repaint();

 }

 public boolean isDigital() {

   return digital;

 }

 public void setDigital(boolean d) {

   digital = d;

   repaint();

 }

 // Other public methods

 public void run() {

   while (clockThread != null) {

     repaint();

     try {

       // Sleep for a second

       clockThread.sleep(1000);

     } catch (InterruptedException e) {

       System.out.println(e);

     }

   }

 }

 public void update(Graphics g) {

   paint(g);

 }

 public synchronized void paint(Graphics g) {

   Dimension d = getSize();

   // Create the offscreen graphics context

   if (offGrfx == null || (offSize.width != d.width) ||

     (offSize.height != d.height)) {

     offSize = d;

     offImage = createImage(d.width, d.height);

     offGrfx = offImage.getGraphics();

   }

   // Paint the background with 3D effects

   offGrfx.setColor(getBackground());

   offGrfx.fillRect(1, 1, d.width - 2, d.height - 2);

   offGrfx.draw3DRect(0, 0, d.width - 1, d.height - 1, raised);

   offGrfx.setColor(getForeground());

   // Paint the clock

   if (digital)

     drawDigitalClock(offGrfx);

   else

     drawAnalogClock(offGrfx);

   // Paint the image onto the screen

   g.drawImage(offImage, 0, 0, null);

 }

 // Private support methods

 private void drawAnalogClock(Graphics g) {

   int   radius = (Math.min(getSize().height, getSize().width) -

     (SPACING \* 2)) / 2;

   Point center = new Point(getSize().width / 2, getSize().height / 2);

   // Draw the clock shape

   g.setColor(Color.black);

   g.drawOval(center.x - radius, center.y - radius, radius \* 2, radius \* 2);

   g.setColor(Color.white);

   g.fillOval(center.x - radius + 1, center.y - radius + 1, (radius \* 2) - 2,

     (radius \* 2) - 2);

   // Draw the hour marks and numbers

   g.setFont(getFont());

   FontMetrics fm = g.getFontMetrics();

   for (int h = 1; h < 13; h++) {

     // Draw an hour mark

     double hourAngle = h \* Math.PI / 6.0;

     g.setColor(Color.black);

     g.drawLine(center.x + (int)(Math.round(0.9 \* radius \*

       Math.sin(hourAngle))), center.y - (int)(Math.round(0.9 \* radius \*

       Math.cos(hourAngle))), center.x + (int)(Math.round(radius \*

       Math.sin(hourAngle))), center.y - (int)(Math.round(radius \*

       Math.cos(hourAngle))));

     // Draw an hour number, as long as it will fit

     if (fm.getAscent() - fm.getDescent() < (radius / 4)) {

       String num = String.valueOf(h);

       g.setColor(getForeground());

       g.drawString(num, center.x + (int)(Math.round(0.7 \* radius \*

         Math.sin(hourAngle))) - (fm.stringWidth(num) / 2), center.y -

         (int)(Math.round(0.7 \* radius \* Math.cos(hourAngle))) +

         ((fm.getAscent() - fm.getDescent()) / 2));

     }

   }

   // Draw the hour hand

   Calendar now = Calendar.getInstance();

   g.setColor(Color.black);

   g.drawLine(center.x, center.y, center.x + (int)(Math.round(radius \* 0.4 \*

     Math.sin((now.get(Calendar.HOUR) % 12) \* Math.PI / 6.0 + (Math.PI \*

     now.get(Calendar.MINUTE) / 360.0)))), center.y -

     (int)(Math.round(radius \* 0.4 \* Math.cos((now.get(Calendar.HOUR) % 12) \*

     Math.PI / 6.0 + (Math.PI \* now.get(Calendar.MINUTE) / 360.0)))));

   // Draw the minute hand

   g.drawLine(center.x, center.y, center.x + (int)(Math.round(0.8 \*

     radius \* Math.sin(now.get(Calendar.MINUTE) \* Math.PI / 30.0))),

     center.y - (int)(Math.round(0.8 \* radius \*

     Math.cos(now.get(Calendar.MINUTE) \* Math.PI / 30.0))));

   // Draw the second hand and center

   g.setColor(Color.red);

   g.drawLine(center.x, center.y, center.x + (int)(Math.round(0.6 \*

     radius \* Math.sin(now.get(Calendar.SECOND) \* Math.PI / 30.0))),

     center.y - (int)(Math.round(0.6 \* radius \*

     Math.cos(now.get(Calendar.SECOND) \* Math.PI / 30.0))));

   g.setColor(Color.black);

   g.fillOval(center.x - 2, center.y - 2, 4, 4);

 }

 private void drawDigitalClock(Graphics g) {

   Dimension d = getSize();

   // Get the time as a string

   Calendar      now = Calendar.getInstance();

   int           h = now.get(Calendar.HOUR) % 12;

   StringBuffer  time = new StringBuffer("");

   if (h == 0)

     time.append("12:");

   else

     time.append(String.valueOf(h) + ":");

   String  s = String.valueOf(now.get(Calendar.MINUTE));

   if (s.length() < 2)

     time.append("0" + s + ":");

   else

     time.append(s + ":");

   s = String.valueOf(now.get(Calendar.SECOND));

   if (s.length() < 2)

     time.append("0" + s);

   else

     time.append(s);

   // Draw the time

   g.setColor(getForeground());

   g.setFont(getFont());

   FontMetrics fm = g.getFontMetrics();

   g.drawString(time.toString(), (d.width - (SPACING \* 2) -

     fm.stringWidth(time.toString())) / 2, (d.height - (SPACING \* 2) +

     fm.getAscent() - fm.getDescent()) / 2);

 }

}

**CLOCKBEANINFO CLASS**

package HTPJB.Chap13.Clock;

// Imports

import java.beans.\*;

public class ClockBeanInfo extends SimpleBeanInfo {

 // Get the appropriate icon

 public java.awt.Image getIcon(int iconKind) {

   if (iconKind == BeanInfo.ICON\_COLOR\_16x16) {

     java.awt.Image img = loadImage("ClockIcon16.gif");

     return img;

   }

   if (iconKind == BeanInfo.ICON\_COLOR\_32x32) {

     java.awt.Image img = loadImage("ClockIcon32.gif");

     return img;

   }

   return null;

 }

}

**Manifest  File:**

Manifest-Version: 1.0

Name: HTPJB/Chap13/Clock/Clock.class

Java-Bean: True



