**Lab: 12: Graphics in 2D**

**Muhammad Rizwan Khalid**

**BSCS-6A**

**180459**

**Task: 01: -**

package truck;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.\*;

import java.awt.event.\*;

import java.awt.image.\*;

import javax.imageio.\*;

import javax.swing.\*;

import javax.swing.event.\*;

public class Truck

{

public static void main(String[] args)

{

DrawingPanel panel = new DrawingPanel(500, 500);

panel.setBackground(Color.WHITE);

Graphics g = panel.getGraphics();

drawTruck(g, 10, 30, 25);

drawTruck(g, 100, 45, 40);

drawTruck(g, 10, 300, 2);

drawTruck(g, 100, 300, 61);

drawTruck(g, 25, 300, 3);

drawTruck(g, 45, 300, 4);

drawTruck(g, 10, 320, 5);

}

public static void drawTruck(Graphics g, int x, int y, int s){

int w = s \* 4;

int h = s \* 2;

g.setColor(Color.BLACK);

g.fillRect(x, y, w, h);

int x1 = (int)(x+.1\*w);

int x2 = (int)(x+0.7\*w);

int y1 = (int)(y + .8\*h);

int w1 = (int)(.2\*w);

int h1 = (int)(.2\*w);

g.setColor(Color.RED);

g.fillOval(x1,y1,w1,h1);

g.fillOval(x2,y1,w1,h1);

int y2 = (int)(y+0.20\*h);

int w2 = (int)(w\*0.30);

int h2 = (int)(h\*0.40);

g.setColor(Color.CYAN);

g.fillRect(x2,y2,w2,h2);

}

}

class DrawingPanel implements ActionListener {

public static final int DELAY = 250; // delay between repaints in millis

private static final String DUMP\_IMAGE\_PROPERTY\_NAME = "drawingpanel.save";

private static String TARGET\_IMAGE\_FILE\_NAME = null;

private static final boolean PRETTY = true; // true to anti-alias

private static boolean DUMP\_IMAGE = true; // true to write DrawingPanel to file

private int width, height; // dimensions of window frame

private JFrame frame; // overall window frame

private JPanel panel; // overall drawing surface

private BufferedImage image; // remembers drawing commands

private Graphics2D g2; // graphics context for painting

private JLabel statusBar; // status bar showing mouse position

private long createTime;

static {

TARGET\_IMAGE\_FILE\_NAME = System.getProperty(DUMP\_IMAGE\_PROPERTY\_NAME);

DUMP\_IMAGE = (TARGET\_IMAGE\_FILE\_NAME != null);

}

// construct a drawing panel of given width and height enclosed in a window

public DrawingPanel(int width, int height) {

this.width = width;

this.height = height;

this.image = new BufferedImage(width, height, BufferedImage.TYPE\_INT\_ARGB);

this.statusBar = new JLabel(" ");

this.statusBar.setBorder(BorderFactory.createLineBorder(Color.BLACK));

this.panel = new JPanel(new FlowLayout(FlowLayout.CENTER, 0, 0));

this.panel.setBackground(Color.WHITE);

this.panel.setPreferredSize(new Dimension(width, height));

this.panel.add(new JLabel(new ImageIcon(image)));

// listen to mouse movement

MouseInputAdapter listener = new MouseInputAdapter() {

public void mouseMoved(MouseEvent e) {

DrawingPanel.this.statusBar.setText("(" + e.getX() + ", " + e.getY() + ")");

}

public void mouseExited(MouseEvent e) {

DrawingPanel.this.statusBar.setText(" ");

}

};

this.panel.addMouseListener(listener);

this.panel.addMouseMotionListener(listener);

this.g2 = (Graphics2D)image.getGraphics();

this.g2.setColor(Color.BLACK);

if (PRETTY) {

this.g2.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

this.g2.setStroke(new BasicStroke(1.1f));

}

this.frame = new JFrame("CS307 Drawing Panel");

this.frame.setResizable(false);

this.frame.addWindowListener(new WindowAdapter() {

public void windowClosing(WindowEvent e) {

if (DUMP\_IMAGE) {

DrawingPanel.this.save(TARGET\_IMAGE\_FILE\_NAME);

}

System.exit(0);

}

});

this.frame.getContentPane().add(panel);

this.frame.getContentPane().add(statusBar, "South");

this.frame.pack();

this.frame.setVisible(true);

if (DUMP\_IMAGE) {

createTime = System.currentTimeMillis();

this.frame.toBack();

} else {

this.toFront();

}

// repaint timer so that the screen will update

new Timer(DELAY, this).start();

}

// used for an internal timer that keeps repainting

public void actionPerformed(ActionEvent e) {

this.panel.repaint();

if (DUMP\_IMAGE && System.currentTimeMillis() > createTime + 4 \* DELAY) {

this.frame.setVisible(false);

this.frame.dispose();

this.save(TARGET\_IMAGE\_FILE\_NAME);

System.exit(0);

}

}

// obtain the Graphics object to draw on the panel

public Graphics2D getGraphics() {

return this.g2;

}

// set the background color of the drawing panel

public void setBackground(Color c) {

this.panel.setBackground(c);

}

// show or hide the drawing panel on the screen

public void setVisible(boolean visible) {

this.frame.setVisible(visible);

}

// makes the program pause for the given amount of time,

// allowing for animation

public void sleep(int millis) {

try {

Thread.sleep(millis);

} catch (InterruptedException e) {}

}

// take the current contents of the panel and write them to a file

public void save(String filename) {

String extension = filename.substring(filename.lastIndexOf(".") + 1);

// create second image so we get the background color

BufferedImage image2 = new BufferedImage(this.width, this.height, BufferedImage.TYPE\_INT\_RGB);

Graphics g = image2.getGraphics();

g.setColor(panel.getBackground());

g.fillRect(0, 0, this.width, this.height);

g.drawImage(this.image, 0, 0, panel);

// write file

try {

ImageIO.write(image2, extension, new java.io.File(filename));

} catch (java.io.IOException e) {

System.err.println("Unable to save image:\n" + e);

}

}

// makes drawing panel become the frontmost window on the screen

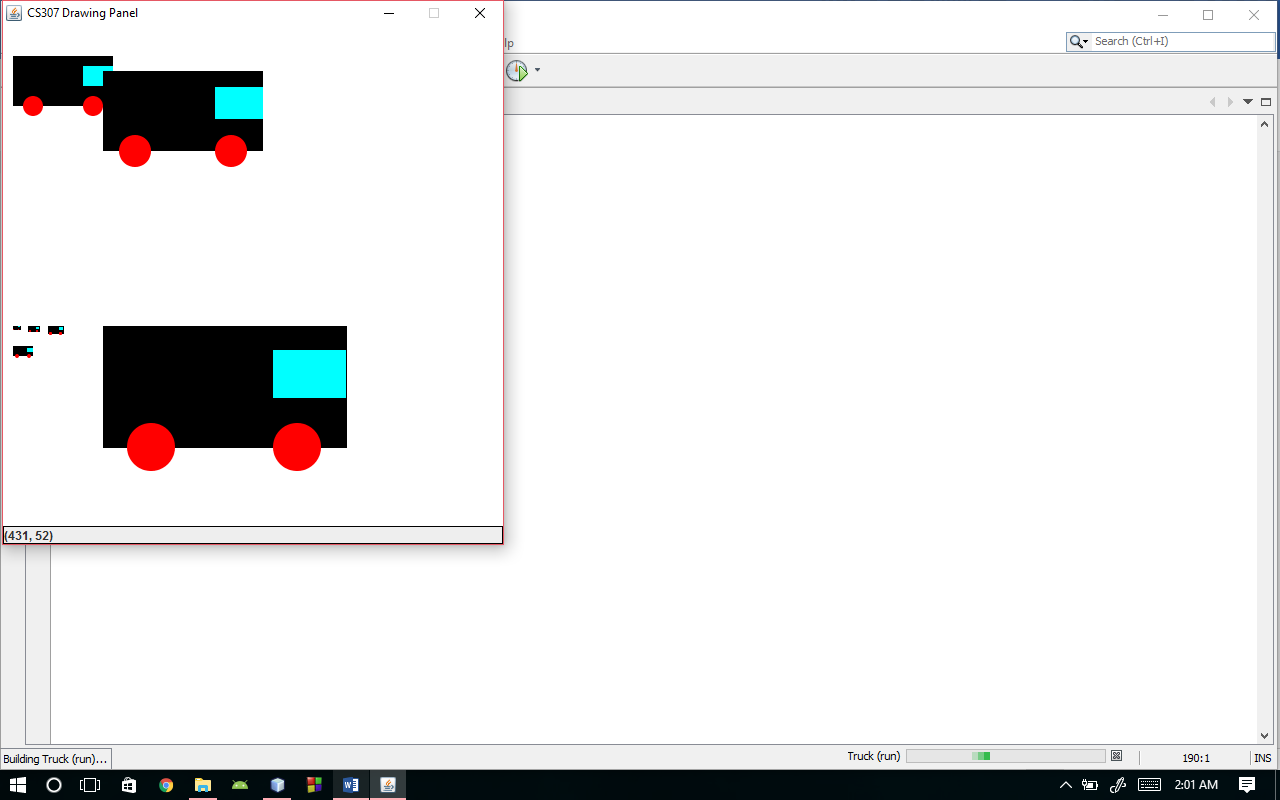
public void toFront() {

this.frame.toFront();

}

}

**Output: -**



**Task: 02: -**

package test;

/\*\*

\* Paints an analog clock synchronized with the system clock.

\*/

import java.awt.BorderLayout;

import java.awt.Color;

import java.awt.Dimension;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.Point;

import java.awt.RenderingHints;

import java.util.Calendar;

import javax.swing.JFrame;

import javax.swing.JPanel;

class Test extends JFrame {

private static final long serialVersionUID = 1L;

private static final Color BACKGROUND\_COLOR = new Color(24, 116, 205);

public Test() {

ClockPanel container = new ClockPanel();

add(container, BorderLayout.CENTER);

setBackground(BACKGROUND\_COLOR);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

setResizable(false);

pack();

setVisible(true);

}

public static void main(String[] args) {

Test test = new Test();

}

}

class ClockPanel extends JPanel implements Runnable {

private static final long serialVersionUID = 1L;

Thread t = new Thread(this);

/\*\* The coordinates used to paint the clock hands. \*/

int xHandSec, yHandSec, xHandMin, yHandMin, xHandHour, yHandHour;

/\*\* The size of the clock. \*/

private final int HORIZONTAL\_SIZE = 500;

private final int VERTICAL\_SIZE = 500;

/\*\* The length of the clock hands relative to the clock size. \*/

private final int secondHandLength = HORIZONTAL\_SIZE / 2 - 50;

private final int minuteHandLength = HORIZONTAL\_SIZE / 2 - 70;

private final int hourHandLength = HORIZONTAL\_SIZE / 2 - 100;

/\*\* The distance of the dots from the origin (center of the clock). \*/

private final int DISTANCE\_DOT\_FROM\_ORIGIN = HORIZONTAL\_SIZE / 2 - 40;

private final int DIAMETER\_BIG\_DOT = 8;

private final int DIAMETER\_SMALL\_DOT = 4;

private final static Color GREY\_COLOR = new Color(160,160,160);

public ClockPanel() {

setMinimumSize(new Dimension(HORIZONTAL\_SIZE, VERTICAL\_SIZE));

setMaximumSize(new Dimension(HORIZONTAL\_SIZE, VERTICAL\_SIZE));

setPreferredSize(new Dimension(HORIZONTAL\_SIZE, VERTICAL\_SIZE));

setLayout(null);

t.start();

}

/\*\*

\* At each iteration we recalculate the coordinates of the clock hands,

\* and repaint everything.

\*/

public void run(){

while(true){

try{

/\* TODO: Get Seconds from the Calender Class; Replace zero with your code \*/

int currentSecond = Calendar.getInstance().get(Calendar.SECOND) ;

/\* TODO: Get Minutes from the Calender Class; Replace zero with your code \*/

int currentMinute = Calendar.getInstance().get(Calendar.MINUTE);

/\* TODO: Get Hours from the Calender Class; Replace zero with your code \*/

int currentHour = Calendar.getInstance().get(Calendar.HOUR);

/\*TODO: Calculate the X Coordinate of Seconds Handle using minToLocation method \*/

xHandSec = minToLocation(currentSecond,secondHandLength).x;

/\*TODO: Calculate the Y Coordinate of Seconds Handle using minToLocation method \*/

yHandSec = minToLocation(currentSecond,secondHandLength).y;

/\*TODO: Calculate the X Coordinate of Minutes Handle using minToLocation method \*/

xHandMin = minToLocation(currentMinute,minuteHandLength).x;

/\*TODO: Calculate the Y Coordinate of Minutes Handle using minToLocation method \*/

yHandMin = minToLocation(currentMinute,minuteHandLength).y;

/\*TODO: Calculate the X Coordinate of Hours Handle using minToLocation method \*/

xHandHour = minToLocation(currentHour,hourHandLength).x;

/\*TODO: Calculate the Y Coordinate of Hours Handle using minToLocation method \*/

yHandHour = minToLocation(currentHour,hourHandLength).y;

repaint();

Thread.sleep(500);

} catch(InterruptedException ie){

ie.printStackTrace();

}

}

}

/\*\*

\* Returns how much the hour hand should be ahead

\* according to the minutes value.

\* 04:00, return 0.

\* 04:12, return 1, so that we move the hour handle ahead of one dot.

\* @param min The current minute.

\* @return The relative offset to add to the hour hand.

\*/

private int getRelativeHour(int min) {

return min / 12;

}

protected void paintComponent(Graphics g){

Graphics2D g2 = (Graphics2D)g;

g2.setRenderingHint(RenderingHints.KEY\_ANTIALIASING,

RenderingHints.VALUE\_ANTIALIAS\_ON);

g2.clearRect(0, 0, getWidth(), getHeight());

// Draw the dots

g2.setColor(GREY\_COLOR);

for (int i = 0; i < 60; i++) {

Point dotCoordinates = minToLocation(i, DISTANCE\_DOT\_FROM\_ORIGIN);

g2.setColor((i <= Calendar.getInstance().get(Calendar.SECOND)) ? Color.BLACK : GREY\_COLOR);

if (i % 5 == 0) {

// big dot

/\*TODO: Draw the big dots using the fillOval method \*/

g2.fillOval(dotCoordinates.x, dotCoordinates.y, DIAMETER\_BIG\_DOT, DIAMETER\_BIG\_DOT);

} else {

// small dot

/\*TODO: Draw the small dots using the fillOval method \*/

g2.fillOval(dotCoordinates.x, dotCoordinates.y, DIAMETER\_SMALL\_DOT, DIAMETER\_SMALL\_DOT);

}

}

// Draw the clock hands

g2.setColor(Color.BLACK);

g2.drawLine(HORIZONTAL\_SIZE / 2, VERTICAL\_SIZE / 2, xHandSec, yHandSec);

/\*TODO: Draw the minute hand here, just like the seconds hand \*/

g2.drawLine(HORIZONTAL\_SIZE / 2, VERTICAL\_SIZE / 2, xHandMin, yHandMin);

/\*TODO: Draw the hour hand here, just like the seconds hand \*/

g2.drawLine(HORIZONTAL\_SIZE / 2, VERTICAL\_SIZE / 2, xHandHour, yHandHour);

}

/\*\*

\* Converts current second/minute/hour to x and y coordinates.

\* @param min The current minute

\* @param radius The radius length

\* @return the coordinates point

\*/

private Point minToLocation(int timeStep, int radius) {

double t = 2 \* Math.PI \* (timeStep-15) / 60;

int x = (int)(HORIZONTAL\_SIZE / 2 + radius \* Math.cos(t));

int y = (int)(VERTICAL\_SIZE / 2 + radius \* Math.sin(t));

return new Point(x, y);

}

}

**Output: -**

