|  |
| --- |
| import javax.swing.event.\*;  import javax.swing.\*;  import java.awt.\*;  import java.awt.event.\*;  import javax.swing.border.\*;  import java.util.\*;  import java.io.\*;  public class GUI implements ActionListener, MouseListener, MouseMotionListener  {  JFrame f;    //Store the current Coordinates of the Shape drawn  private int x1, y1, x2, y2;    //Store the coordinates of the shape that has to be stored in the Shape Class  private int x1Coord, y1Coord, x2Coord, y2Coord;    //File Opening and Saving Components  private JFileChooser fileChooser;  private File fileSave, fileOpen;  private int userSave, userOpen;    //Saving shapes class  SaveFile saveClass;    //Opening existing shapes class  OpenFile openClass;    //Whether new shape drawing has started or not  boolean dragged;    //Type of drawing - rectangle or line  String drawing;    JPanel mainPanel, paintPanel, btnPanel;    //Create JButtons for the user at the bottom of the mainPanel  JButton btn[];  String[] btnString = {"Line", "Rectangle", "Clear", "Save", "Load"};    //Draw all the shapes  PaintClass paint;    //Class that stores the detailed information of the shape  Shape shape;    //ArrayList to store all the shapes drawn \*\*Main ArrayList\*\*  ArrayList<Shape> shapeArray;    public GUI()  {  f = new JFrame("PAINT");    fileChooser = new JFileChooser();    saveClass = new SaveFile();  openClass = new OpenFile();    dragged = false;  drawing = "line";    paint = new PaintClass();    btnPanel = new JPanel(new FlowLayout());    //Set buttons on the screen  btn = new JButton[5];  for(int b = 0; b < btn.length; b++)  {  btn[b] = new JButton(btnString[b]);    //Add actionListener to all the buttons  btn[b].addActionListener(this);    //Add the buttons to the btnPanel  btnPanel.add(btn[b]);    }//end for    shapeArray = new ArrayList<Shape>();    mainPanel = new JPanel(new BorderLayout());  mainPanel.setPreferredSize(new Dimension(550, 400));  mainPanel.setBorder(new BevelBorder(BevelBorder.RAISED));    paint.setBorder(new SoftBevelBorder(SoftBevelBorder.LOWERED));    paint.addMouseListener(this);  paint.addMouseMotionListener(this);    mainPanel.add(paint, BorderLayout.CENTER);  mainPanel.add(btnPanel, BorderLayout.SOUTH);    f.add(mainPanel);    f.pack();  f.setLocationRelativeTo(null);  f.setVisible(true);    }    public void actionPerformed(ActionEvent e)  {  //If the user wants to draw line on the screen  if(e.getSource() == btn[0])  {  drawing = "Line";  }    //If the user wants to draw rectangles on the screen  else if(e.getSource() == btn[1])  {  drawing = "Rectangle";  }    //If the user wants to clear the screen  else if(e.getSource() == btn[2])  {  shapeArray.clear();    x1 = 0; x2 = 0; y1 = 0; y2 = 0;  paint.repaint();  }    //If the user wants to save the shapes drawn on the screen  else if(e.getSource() == btn[3])  {  fileChooser.setDialogTitle("Save File");  userSave = fileChooser.showSaveDialog(f);    if(userSave == JFileChooser.APPROVE\_OPTION)  {  fileSave = fileChooser.getSelectedFile();  file\_Save("" + fileSave);    }//end if    }//end if    //If the user wants to open the file from the directory  // to drawn the previous shapes  else  {    //Open the file  fileChooser.setDialogTitle("Open File");  fileChooser.setCurrentDirectory(new File(System.getProperty("user.home")));  userOpen = fileChooser.showOpenDialog(f);    //If the user has to open the file  if(userOpen == JFileChooser.APPROVE\_OPTION)  {  //Get the name of the selected file  fileOpen = fileChooser.getSelectedFile();    //Clear the screen  shapeArray.clear();    x1 = 0; x2 = 0; y1 = 0; y2 = 0;  paint.repaint();    //Open the selected file and draw the shapes saved in it  file\_Open("" + fileOpen);    }//end if  }//end if    }//end actionPerformed(ActionEvent)    /\*\*  \* Method Name : file\_Save  \* Purpose : Save the shapes on the screen to the file  \* @param String filePath - path of the file selected  \* @return void  \* \*/  public void file\_Save(String filePath)  {  saveClass.saveFile(filePath, shapeArray);    }//end file\_Save(String)    /\*\*  \* Method Name : file\_Open  \* Purpose : Open the selected file and store the shapes into the  \* @param String filePath - path of the file selected  \* @return void  \* \*/  public void file\_Open(String filePath)  {  //open the selected file  openClass.openFile(filePath);    //Add all the shapes from the file to the shapeArray  shapeArray.addAll(openClass.getArray());    //repaint the screen  paint.repaint();    }//end file\_Open(String)    /\*\*  \* MouseListener and MouseMotionListeners that are not used  \*/  public void mouseClicked(MouseEvent e){}  public void mouseEntered(MouseEvent e){}  public void mouseExited(MouseEvent e){}  public void mouseMoved(MouseEvent e){}    /\*\*  \* Purpose : create a new Shape and store it in the shapeArray  \*/  public void mouseReleased(MouseEvent e)  {  //Create a new shape if the mouse if dragged  if(dragged)  {  //Create a new Shape Object  shape = new Shape(x1Coord, x2Coord, y1Coord, y2Coord, drawing);    //Add the Shape object to the arrayList  shapeArray.add(shape);    //Set the current coordinates of the shape to 0  x1 = 0; x2 = 0; y1 = 0; y2 = 0;    paint.repaint();  }  }//end mouseReleased(MouseEvent)    /\*\*  \* Purpose : Get the starting and final x and y coordinates of  \* the mouse when the mouse button is pressed  \*/  public void mousePressed(MouseEvent e)  {  //Set the initial x1 and y1 coordinates of the mouse  x1 = e.getX();  y1 = e.getY();    dragged = false;    }//end mousePressed(MouseEvent)    /\*\*  \* Purpose : Get the final mouse coordinate with the mouse is  \* dragged while pressing  \*/  public void mouseDragged(MouseEvent e)  {  //Store the x2 and y2 coordinates of the mouse  x2 = e.getX();  y2 = e.getY();    dragged = true;    paint.repaint();    }//end mouseDragged(MouseEvent)    //Main  public static void main(String[] args)  {  new GUI();  }    //PAINT CLASS  class PaintClass extends JPanel  {  int width, height, x, y;    public void paintComponent(Graphics g)  {  super.paintComponent(g);    //Set the background to white color  g.setColor(Color.WHITE);  g.fillRect(0, 0, getWidth(), getHeight());    //Draw all the previous shapes  for(Shape x: shapeArray)  {    //If the Shape is line, draw the line  if(x.get\_shape().equalsIgnoreCase("line"))  {  line(g, x.get\_x1(), x.get\_y1(), x.get\_x2(), x.get\_y2());  }    //If the Shape is Rectangle, draw rectangle  else  {  rectangle(g, x.get\_x1(), x.get\_y1(), x.get\_x2(), x.get\_y2());  }    }//end for    //Draw the current Line  if(drawing.equalsIgnoreCase("Line"))  {  line(g, x1, y1, x2, y2);  }    //Draw the current Rectangle  else if(drawing.equalsIgnoreCase("Rectangle"))  {  rectangle(g, x1, y1, x2, y2);  }//end if    }//end paintComponent(Graphics)    //Draw Lines  public void line(Graphics g, int x1, int y1, int x2, int y2)  {  x1Coord = x1; x2Coord = x2; y1Coord = y1; y2Coord = y2;    g.setColor(Color.RED);  g.drawLine(x1, y1, x2, y2);    }//end line(Graphics, int, int, int, int)    //Draw rectangles  public void rectangle(Graphics g, int x1, int y1, int x2, int y2)  {  x1Coord = x1; x2Coord = x2; y1Coord = y1; y2Coord = y2;    width = Math.abs(x2 - x1);  height = Math.abs(y2 - y1);    x = Math.min(x2, x1);  y = Math.min(y2, y1);    g.setColor(Color.BLUE);  g.fillRect(x, y, width, height);    }//end rectangle(Graphics, int, int, int, int)    }//end PaintClass    }//end GUI  class Shape implements Serializable  {    private int x1, y1, x2, y2;  private String shape;    /\*\*  \* Constructor Name : Shape  \* @param int x1 - initial x coordinate of the shape  \* @param int x2 - final x coordinate of the shape  \* @param int y1 - initial x coordinate of the shape  \* @param int y2 - final y coordinate of the shape  \* @param String shape - type of the shape  \* \*/  public Shape(int x1, int x2, int y1, int y2, String shape)  {  this.x1 = x1;  this.x2 = x2;  this.y1 = y1;  this.y2 = y2;  this.shape = shape;    }//end Shape(int, int, int, int, String) Constructor    /\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Get Methods  \* \*\*\*\*\*\*\*\*\*\*\*/    /\*\*  \* Method Name : get\_x1  \* Purpose : Get the x1 coordinate of the shape  \* @param none  \* \*/  public int get\_x1()  {  return this.x1;  }//end get\_x1()    /\*\*  \* Method Name : get\_x2  \* Purpose : Get the x2 coordinate of the shape  \* @param none  \* \*/  public int get\_x2()  {  return this.x2;  }//end get\_x2()    /\*\*  \* Method Name : get\_y1  \* Purpose : Get the y1 coordinate of the shape  \* @param none  \* \*/  public int get\_y1()  {  return this.y1;  }//end get\_y1()    /\*\*  \* Method Name : get\_y2  \* Purpose : Get the y2 coordinate of the shape  \* @param none  \* \*/  public int get\_y2()  {  return this.y2;  }//end get\_y2()    /\*\*  \* Method Name : get\_shape  \* Purpose : Get the type of the shape  \* @param none  \* \*/  public String get\_shape()  {  return this.shape;  }//end get\_shape()    @Override  /\*\*  \* Method Name : toString  \* Purpose : Outputs the coordinates of the shape and its type  \* \*/  public String toString()  {  String xString = "x2", yString = "y2";    //If the shape is a rectangle  if(!this.shape.equalsIgnoreCase("line"))  {  xString = "width";  yString = "height";  }    return "Dimensions of " + this.shape + " are : x1 = " + this.x1 + " , y1 = " + this.y1  + " , "+ xString + " = " + this.x2 + ", "+ yString + " = " + this.y2;  }  }//end Shape() |
| //Required java packages to open the file  import java.io.\*;  import java.util.ArrayList;  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* READING OBJECTS FROM FILES \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  public class OpenFile  {    private ObjectInputStream input;  private ArrayList<Shape> openList;    public OpenFile(){}    public void openFile(String filePath)  {    //Attempt to open the shape file  try  {  //Create a filePath file with an ObjectOutputStream  input = new ObjectInputStream(new FileInputStream(filePath));    //Read the ArrayList of Shape Objects from the filePath file  // and add them to the openList ArrayList  openList = (ArrayList<Shape>)input.readObject();    //Close the filePath file  input.close();    }    //Exception warning from the reading of the filePath file  catch(IOException e)  {  System.out.println(e);  }    //Exception warning from the genetic method readObject()  catch(ClassNotFoundException c)  {  System.out.println(c);  }    }//end openFile(String)    /\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Get Method  \* \*\*\*\*\*\*\*\*\*\*\*/    /\*\*  \* Method Name : getArray  \* Purpose : returns the ArrayList of Shapes from the filepath file  \* \*/  public ArrayList<Shape> getArray()  {  return this.openList;  }//end getArray()    }//end OpenFile() |
| //Required java packages to save the Shapes in a file  import java.io.\*;  import java.util.ArrayList;  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* WRITING OBJECTS TO FILES \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  public class SaveFile  {    private FileOutputStream output;  private ObjectOutputStream fileOutput;    public SaveFile(){}    public void saveFile(String filePath, ArrayList<Shape> shapeArray)  {    //Attempt to write the file that stores the Shapes drawn on the screen  try{    //Create a filepath file with an ObjectOutputStream  output = new FileOutputStream(filePath);  fileOutput = new ObjectOutputStream(output);    //Write the ArrayList of Shapes to the file  fileOutput.writeObject(shapeArray);    //flush the object output stream  fileOutput.flush();    //close the filePath  fileOutput.close();    }    //Exception warning if the file is not saved or created  catch(IOException e)  {  System.out.println(e);  }    }//end saveFile(String, ArrayList<Shape>)    }//end SaveFile |
| Screenshots: |