**Classification of Dataset based on Hierarchy**

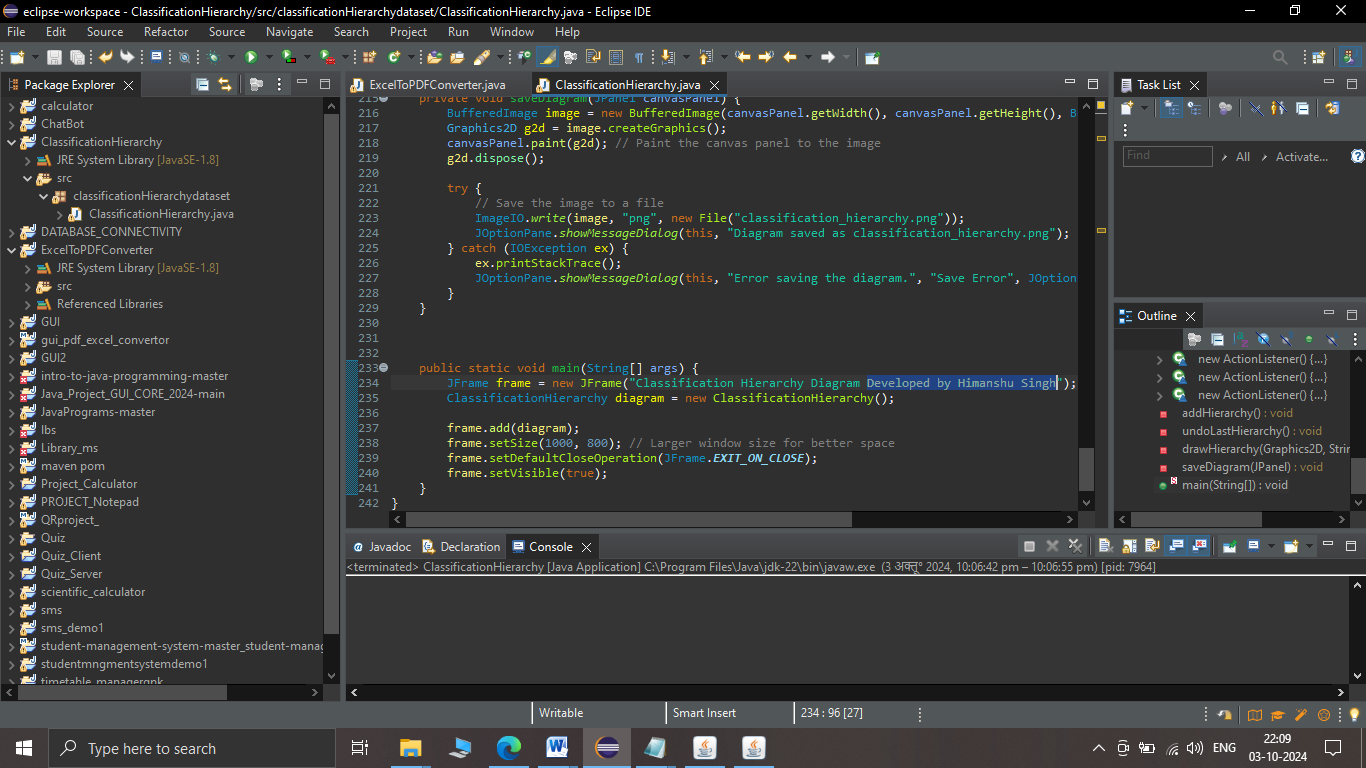
**image generator using Java**

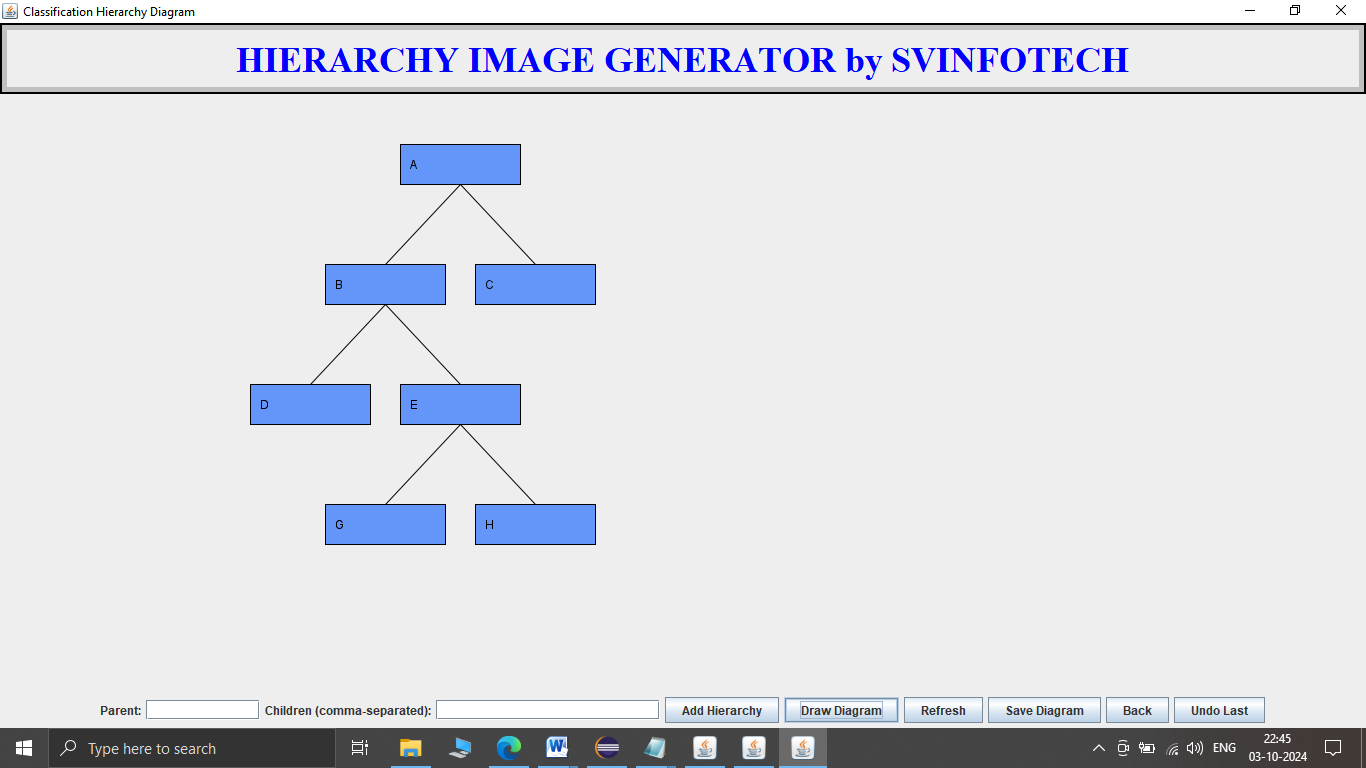
**github link-https://github.com/himanshuSinghworkPort/hierarchy\_chart\_generator-using\_java**

**Project name- ClasificationHierarchy**

**Src>> package\_name-clasificationhierarchydataset**

**Package\_name >>class\_name-clasificationhierarchy.java**





**Source code:**

**package classificationHierarchydataset;**

**import javax.swing.\*;**

**import javax.swing.border.\*;**

**import java.awt.\*;**

**import java.awt.event.ActionEvent;**

**import java.awt.event.ActionListener;**

**import java.awt.image.BufferedImage;**

**import java.io.File;**

**import java.io.IOException;**

**import java.util.HashMap;**

**import java.util.Map;**

**import java.util.Stack;**

**import javax.imageio.ImageIO;**

**public class ClassificationHierarchy extends JPanel {**

**private Map<String, String[]> hierarchy = new HashMap<>();**

**private Stack<String> undoStack = new Stack<>(); // Stack to track hierarchy additions**

**private JTextField parentField;**

**private JTextField childrenField;**

**private JButton addButton;**

**private JButton drawButton;**

**private JButton refreshButton;**

**private JButton saveButton;**

**private JButton backButton;**

**private JButton undoButton;**

**public ClassificationHierarchy() {**

**setLayout(new BorderLayout());**

**JPanel titlePanel = new JPanel();**

**JLabel titleLabel = new JLabel("HIERARCHY IMAGE GENERATOR by SVINFOTECH", SwingConstants.CENTER);**

**titleLabel.setFont(new Font("Serif", Font.BOLD, 36));**

**// Set custom font and size**

**titleLabel.setForeground(Color.BLUE);**

**// Set title color**

**// Create double-shaded border**

**Border outerBorder = new LineBorder(Color.BLACK, 2);**

**// Outer black border**

**Border innerBorder = new LineBorder(Color.LIGHT\_GRAY, 5);**

**// Inner lighter border**

**titlePanel.setBorder(BorderFactory.createCompoundBorder(outerBorder, innerBorder));**

**titlePanel.add(titleLabel); // Add label to the panel**

**add(titlePanel, BorderLayout.NORTH);**

**// Add title panel at the top**

**// Input panel for parent and children fields**

**JPanel inputPanel = new JPanel();**

**inputPanel.setLayout(new FlowLayout());**

**parentField = new JTextField(10);**

**childrenField = new JTextField(20);**

**addButton = new JButton("Add Hierarchy");**

**drawButton = new JButton("Draw Diagram");**

**refreshButton = new JButton("Refresh");**

**saveButton = new JButton("Save Diagram");**

**backButton = new JButton("Back");**

**undoButton = new JButton("Undo Last");**

**inputPanel.add(new JLabel("Parent:"));**

**inputPanel.add(parentField);**

**inputPanel.add(new JLabel("Children (comma-separated):"));**

**inputPanel.add(childrenField);**

**inputPanel.add(addButton);**

**inputPanel.add(drawButton);**

**inputPanel.add(refreshButton);**

**inputPanel.add(saveButton);**

**inputPanel.add(backButton);**

**inputPanel.add(undoButton);**

**add(inputPanel, BorderLayout.SOUTH);**

**// Canvas panel for drawing the hierarchy diagram**

**JPanel canvasPanel = new JPanel(true) {**

**@Override**

**protected void paintComponent(Graphics g) {**

**super.paintComponent(g);**

**Graphics2D g2d = (Graphics2D) g;**

**// Enable anti-aliasing for smoother lines and shapes**

**g2d.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);**

**// Draw the hierarchy starting from the first root**

**if (hierarchy.size() > 0) {**

**for (String root : hierarchy.keySet()) {**

**drawHierarchy(g2d, root, 400, 50, 120, 40); // Centering root at 400**

**break; // Draw only one root for now**

**}**

**}**

**}**

**};**

**add(canvasPanel, BorderLayout.CENTER);**

**// Button to add hierarchy data**

**addButton.addActionListener(new ActionListener() {**

**@Override**

**public void actionPerformed(ActionEvent e) {**

**addHierarchy();**

**}**

**});**

**// Button to redraw the diagram**

**drawButton.addActionListener(new ActionListener() {**

**@Override**

**public void actionPerformed(ActionEvent e) {**

**canvasPanel.repaint();**

**}**

**});**

**// Refresh button to clear the diagram**

**refreshButton.addActionListener(new ActionListener() {**

**@Override**

**public void actionPerformed(ActionEvent e) {**

**hierarchy.clear(); // Clear the hierarchy map**

**canvasPanel.repaint(); // Redraw the canvas**

**parentField.setText(""); // Clear the input fields**

**childrenField.setText("");**

**undoStack.clear(); // Clear the undo stack**

**}**

**});**

**// Save button to save the diagram as an image**

**saveButton.addActionListener(new ActionListener() {**

**@Override**

**public void actionPerformed(ActionEvent e) {**

**saveDiagram(canvasPanel); // Call method to save the diagram**

**}**

**});**

**// Back button to clear input fields**

**backButton.addActionListener(new ActionListener() {**

**@Override**

**public void actionPerformed(ActionEvent e) {**

**parentField.setText(""); // Clear parent field**

**childrenField.setText(""); // Clear children field**

**JOptionPane.showMessageDialog(ClassificationHierarchy.this, "Input fields cleared. You can enter new data."); // Notify the user**

**}**

**});**

**// Undo button to undo the last added hierarchy**

**undoButton.addActionListener(new ActionListener() {**

**@Override**

**public void actionPerformed(ActionEvent e) {**

**undoLastHierarchy();**

**canvasPanel.repaint(); // Redraw the canvas**

**}**

**});**

**}**

**// Method to add hierarchy based on user input**

**private void addHierarchy() {**

**String parent = parentField.getText().trim();**

**String children = childrenField.getText().trim();**

**if (parent.isEmpty() || children.isEmpty()) {**

**JOptionPane.showMessageDialog(this, "Both parent and children fields must be filled!",**

**"Input Error", JOptionPane.ERROR\_MESSAGE);**

**return;**

**}**

**String[] childrenArray = children.split(",");**

**for (int i = 0; i < childrenArray.length; i++) {**

**childrenArray[i] = childrenArray[i].trim();**

**}**

**hierarchy.put(parent, childrenArray);**

**undoStack.push(parent); // Store the parent in undo stack**

**parentField.setText("");**

**childrenField.setText("");**

**}**

**// Method to undo the last added hierarchy**

**private void undoLastHierarchy() {**

**if (!undoStack.isEmpty()) {**

**String lastAddedParent = undoStack.pop();**

**hierarchy.remove(lastAddedParent); // Remove the last added hierarchy**

**JOptionPane.showMessageDialog(this, "Last hierarchy removed: " + lastAddedParent);**

**} else {**

**JOptionPane.showMessageDialog(this, "No hierarchy to undo!", "Undo Error", JOptionPane.WARNING\_MESSAGE);**

**}**

**}**

**// Recursive method to draw the classification hierarchy**

**private void drawHierarchy(Graphics2D g2d, String parent, int x, int y, int width, int height) {**

**// Set color for parent node**

**g2d.setColor(new Color(100, 150, 250)); // Blue for parent**

**g2d.fillRect(x, y, width, height); // Fill the rectangle for parent**

**g2d.setColor(Color.BLACK); // Set border color**

**g2d.drawRect(x, y, width, height); // Draw border for parent**

**g2d.drawString(parent, x + 10, y + 25); // Draw parent label**

**String[] children = hierarchy.get(parent);**

**if (children != null) {**

**int numChildren = children.length;**

**int childXStart = x - (numChildren - 1) \* 150 / 2; // Center children under the parent**

**int childY = y + height + 80; // Increased space below parent for children**

**for (int i = 0; i < numChildren; i++) {**

**int childX = childXStart + i \* 150; // Space children evenly**

**g2d.drawLine(x + width / 2, y + height, childX + width / 2, childY); // Draw line to child**

**// Set color for child node**

**g2d.setColor(new Color(200, 250, 100)); // Green for children**

**g2d.fillRect(childX, childY, width, height); // Fill the rectangle for child**

**g2d.setColor(Color.BLACK); // Set border color**

**g2d.drawRect(childX, childY, width, height); // Draw border for child**

**g2d.drawString(children[i], childX + 10, childY + 25); // Draw child label**

**drawHierarchy(g2d, children[i], childX, childY, width, height); // Recursively draw children**

**}**

**}**

**}**

**// Method to save the diagram as an image**

**private void saveDiagram(JPanel canvasPanel) {**

**BufferedImage image = new BufferedImage(canvasPanel.getWidth(), canvasPanel.getHeight(), BufferedImage.TYPE\_INT\_RGB);**

**Graphics2D g2d = image.createGraphics();**

**canvasPanel.paint(g2d); // Paint the canvas into the BufferedImage**

**g2d.dispose(); // Dispose graphics context**

**try {**

**// Specify the file path and format**

**File outputfile = new File("classification\_hierarchy\_diagram.png");**

**ImageIO.write(image, "png", outputfile); // Save the image as PNG**

**JOptionPane.showMessageDialog(this, "Diagram saved as " + outputfile.getAbsolutePath());**

**} catch (IOException e) {**

**e.printStackTrace();**

**JOptionPane.showMessageDialog(this, "Error saving diagram: " + e.getMessage(),**

**"Save Error", JOptionPane.ERROR\_MESSAGE);**

**}**

**}**

**public static void main(String[] args) {**

**JFrame frame = new JFrame("Classification Hierarchy Diagram");**

**ClassificationHierarchy diagram = new ClassificationHierarchy();**

**frame.add(diagram);**

**frame.setSize(1200, 800); // Larger canvas size**

**frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);**

**frame.setVisible(true);**

**}**

**}**

**TESTING CODE:**

**package classificationHierarchydataset;**

**import javax.swing.\*;**

**import javax.swing.border.\*;**

**import java.awt.\*;**

**import java.awt.event.ActionEvent;**

**import java.awt.event.ActionListener;**

**import java.awt.image.BufferedImage;**

**import java.io.File;**

**import java.io.IOException;**

**import java.util.HashMap;**

**import java.util.Map;**

**import java.util.Stack;**

**import javax.imageio.ImageIO;**

**public class ClassificationHierarchy extends JPanel {**

**private Map<String, String[]> hierarchy = new HashMap<>();**

**private Stack<String> undoStack = new Stack<>();**

**// Stack to track hierarchy additions**

**private JTextField parentField;**

**private JTextField childrenField;**

**private JButton addButton;**

**private JButton drawButton;**

**private JButton refreshButton;**

**private JButton saveButton;**

**private JButton backButton;**

**private JButton undoButton;**

**public ClassificationHierarchy() {**

**setLayout(new BorderLayout());**

**// Create the title panel with double-shaded border and custom //font**

**JPanel titlePanel = new JPanel();**

**JLabel titleLabel = new JLabel("HIERARCHY IMAGE GENERATOR by SVINFOTECH", SwingConstants.CENTER);**

**titleLabel.setFont(new Font("Serif", Font.BOLD, 36));**

**// Set custom font and size**

**titleLabel.setForeground(Color.BLUE);**

**// Set title color**

**// Create double-shaded border**

**Border outerBorder = new LineBorder(Color.BLACK, 2);**

**// Outer black border**

**Border innerBorder = new LineBorder(Color.LIGHT\_GRAY, 5);**

**// Inner lighter border**

**titlePanel.setBorder(BorderFactory.createCompoundBorder(outerBorder, innerBorder));**

**titlePanel.add(titleLabel); // Add label to the panel**

**add(titlePanel, BorderLayout.NORTH);**

**// Add title panel at the top**

**// Input panel for parent and children fields and buttons**

**JPanel inputPanel = new JPanel();**

**inputPanel.setLayout(new FlowLayout());**

**// You can use GridLayout or BoxLayout for a different arrangement**

**parentField = new JTextField(10);**

**childrenField = new JTextField(20);**

**addButton = new JButton("Add Hierarchy");**

**drawButton = new JButton("Draw Diagram");**

**refreshButton = new JButton("Refresh");**

**saveButton = new JButton("Save Diagram");**

**backButton = new JButton("Back");**

**undoButton = new JButton("Undo Last");**

**inputPanel.add(new JLabel("Parent:"));**

**inputPanel.add(parentField);**

**inputPanel.add(new JLabel("Children (comma-separated):"));**

**inputPanel.add(childrenField);**

**inputPanel.add(addButton);**

**inputPanel.add(drawButton);**

**inputPanel.add(refreshButton);**

**inputPanel.add(saveButton);**

**inputPanel.add(backButton);**

**inputPanel.add(undoButton);**

**// Add the input panel directly below the title**

**add(inputPanel, BorderLayout.CENTER);**

**// Center position, right under the title panel**

**// Canvas panel for drawing the hierarchy diagram**

**JPanel canvasPanel = new JPanel(true) {**

**@Override**

**protected void paintComponent(Graphics g) {**

**super.paintComponent(g);**

**Graphics2D g2d = (Graphics2D) g;**

**// Enable anti-aliasing for smoother lines and shapes**

**g2d.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);**

**// Draw the hierarchy starting from the first root**

**if (hierarchy.size() > 0) {**

**for (String root : hierarchy.keySet()) {**

**drawHierarchy(g2d, root, 400, 50, 120, 40);**

**// Centering root at 400**

**break;**

**// Draw only one root for now**

**}**

**}**

**}**

**};**

**add(canvasPanel, BorderLayout.SOUTH);**

**// Below the input panel**

**// Button to add hierarchy data**

**addButton.addActionListener(new ActionListener() {**

**@Override**

**public void actionPerformed(ActionEvent e) {**

**addHierarchy();**

**}**

**});**

**// Button to redraw the diagram**

**drawButton.addActionListener(new ActionListener() {**

**@Override**

**public void actionPerformed(ActionEvent e) {**

**canvasPanel.repaint();**

**}**

**});**

**// Refresh button to clear the diagram**

**refreshButton.addActionListener(new ActionListener() {**

**@Override**

**public void actionPerformed(ActionEvent e) {**

**hierarchy.clear();**

**// Clear the hierarchy map**

**canvasPanel.repaint();**

**// Redraw the canvas**

**parentField.setText("");**

**// Clear the input fields**

**childrenField.setText("");**

**undoStack.clear();**

**// Clear the undo stack**

**}**

**});**

**// Save button to save the diagram as an image**

**saveButton.addActionListener(new ActionListener() {**

**@Override**

**public void actionPerformed(ActionEvent e) {**

**saveDiagram(canvasPanel);**

**// Call method to save the diagram**

**}**

**});**

**// Back button to clear input fields**

**backButton.addActionListener(new ActionListener() {**

**@Override**

**public void actionPerformed(ActionEvent e) {**

**parentField.setText("");**

**// Clear parent field**

**childrenField.setText("");**

**// Clear children field**

**JOptionPane.showMessageDialog(ClassificationHierarchy.this, "Input fields cleared. You can enter new data."); // Notify the user**

**}**

**});**

**// Undo button to undo the last added hierarchy**

**undoButton.addActionListener(new ActionListener() {**

**@Override**

**public void actionPerformed(ActionEvent e) {**

**undoLastHierarchy();**

**canvasPanel.repaint(); // Redraw the canvas**

**}**

**});**

**}**

**// Method to add hierarchy based on user input**

**private void addHierarchy() {**

**String parent = parentField.getText().trim();**

**String children = childrenField.getText().trim();**

**if (parent.isEmpty() || children.isEmpty()) {**

**JOptionPane.showMessageDialog(this, "Both parent and children fields must be filled!",**

**"Input Error", JOptionPane.ERROR\_MESSAGE);**

**return;**

**}**

**String[] childrenArray = children.split(",");**

**for (int i = 0; i < childrenArray.length; i++) {**

**childrenArray[i] = childrenArray[i].trim();**

**}**

**hierarchy.put(parent, childrenArray);**

**undoStack.push(parent); // Store the parent in undo stack**

**parentField.setText("");**

**childrenField.setText("");**

**}**

**// Method to undo the last added hierarchy**

**private void undoLastHierarchy() {**

**if (!undoStack.isEmpty()) {**

**String lastAddedParent = undoStack.pop();**

**hierarchy.remove(lastAddedParent);**

**// Remove the last added hierarchy**

**JOptionPane.showMessageDialog(this, "Last hierarchy removed: " + lastAddedParent);**

**} else {**

**JOptionPane.showMessageDialog(this, "No hierarchy to undo!", "Undo Error", JOptionPane.WARNING\_MESSAGE);**

**}**

**}**

**// Recursive method to draw the classification hierarchy**

**private void drawHierarchy(Graphics2D g2d, String parent, int x, int y, int width, int height) {**

**// Set color for parent node**

**g2d.setColor(new Color(100, 150, 250));**

**// Blue for parent**

**g2d.fillRect(x, y, width, height);**

**// Fill the rectangle for parent**

**g2d.setColor(Color.BLACK);**

**// Set border color**

**g2d.drawRect(x, y, width, height);**

**// Draw border for parent**

**g2d.drawString(parent, x + 10, y + 25);**

**// Draw parent label**

**String[] children = hierarchy.get(parent);**

**if (children != null) {**

**int numChildren = children.length;**

**int childXStart = x - (numChildren - 1) \* 150 / 2;**

**// Center children under the parent**

**int childY = y + height + 80;**

**// Increased space below parent for children**

**for (int i = 0; i < numChildren; i++) {**

**int childX = childXStart + i \* 150;**

**// Space children evenly**

**g2d.drawLine(x + width / 2, y + height, childX + width / 2, childY);**

**// Draw line to child**

**// Set color for child node**

**g2d.setColor(new Color(200, 250, 100));**

**// Green for children**

**g2d.fillRect(childX, childY, width, height);**

**// Fill the rectangle for child**

**g2d.setColor(Color.BLACK);**

**// Set border color**

**g2d.drawRect(childX, childY, width, height);**

**// Draw border for child**

**g2d.drawString(children[i], childX + 10, childY + 25);**

**// Draw child label**

**drawHierarchy(g2d, children[i], childX, childY, width, height); // Recursively draw children**

**}**

**}**

**}**

**// Method to save the diagram as an image**

**private void saveDiagram(JPanel canvasPanel) {**

**BufferedImage image = new BufferedImage(canvasPanel.getWidth(), canvasPanel.getHeight(), BufferedImage.TYPE\_INT\_RGB);**

**Graphics2D g2d = image.createGraphics();**

**canvasPanel.paint(g2d);**

**// Paint the canvas panel to the image**

**g2d.dispose();**

**try {**

**// Save the image to a file**

**ImageIO.write(image, "png", new File("classification\_hierarchy.png"));**

**JOptionPane.showMessageDialog(this, "Diagram saved as classification\_hierarchy.png");**

**} catch (IOException ex) {**

**ex.printStackTrace();**

**JOptionPane.showMessageDialog(this, "Error saving the diagram.", "Save Error", JOptionPane.ERROR\_MESSAGE);**

**}**

**}**

**public static void main(String[] args) {**

**JFrame frame = new JFrame("Classification Hierarchy Diagram Developed by Himanshu Singh");**

**ClassificationHierarchy diagram = new ClassificationHierarchy();**

**frame.add(diagram);**

**frame.setSize(1000, 800);**

**// Larger window size for better space**

**frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);**

**frame.setVisible(true);**

**}**

**}**

**Create this project in eclipse ide:**

**Run this project with alt+shift+x**

# Project Report: Classification Hierarchy Diagram

## 1. Introduction

The Classification Hierarchy Diagram project aims to create an interactive Java Swing application that allows users to visualize hierarchical relationships among various classifications. The program provides a user-friendly interface for entering parent-child relationships, drawing a diagram to represent these relationships, and managing the hierarchy through various features.

## 2. Objectives

* To develop a graphical user interface (GUI) that allows users to input hierarchical data.
* To visualize the entered data as a hierarchical diagram with arrows indicating relationships.
* To implement functionality for adding, undoing, refreshing, saving, and displaying the hierarchy.

## 3. Tools and Technologies

* **Programming Language**: Java
* **Framework**: Java Swing
* **Development Environment**: Eclipse IDE
* **Graphics**: AWT and Swing for rendering graphics
* **Image Handling**: Java ImageIO for saving diagrams

## 4. Features

## ……………………………………………………………………………..

## ……………………………………………………………………………..

## ……………………………………………………………………………..

### 4.1 User Input

* **Parent Input Field**: A text field for entering the parent classification.
* **Children Input Field**: A text field for entering children classifications in a comma-separated format.
* **Add Button**: To add the hierarchy based on user inputs.
* **Undo Button**: To remove the last added hierarchy entry.
* **Refresh Button**: To clear the current diagram and input fields.
* **Save Button**: To save the diagram as a PNG image.
* **Back Button**: To clear the input fields for new entries.

### 4.2 Visualization

* The program visually represents the hierarchy using rectangles for nodes (parent and child) with arrows connecting them.
* Different colors are used for parent and child nodes to enhance clarity.
* A title "SVINFOTECH" at the top with customized color styling.

### 4.3 Diagram Management

* **Drawing Logic**: The program uses recursive methods to render hierarchical structures, accommodating multi-level hierarchies.
* **Undo Functionality**: Users can undo the last entry, enhancing flexibility and ease of use.

## 5. Implementation

The project was implemented as follows:

### 5.1 Project Structure

* **Main Class**: ClassificationHierarchy
* **Components**:
  + Input fields for parent and children.
  + Buttons for adding, undoing, refreshing, saving, and back functionality.
  + A drawing area for rendering the hierarchy.

### 5.2 Code Overview

java

// Main class

public class ClassificationHierarchy extends JPanel {

// Define components

private Map<String, String[]> hierarchy = new HashMap<>();

private Stack<String> undoStack = new Stack<>();

private JTextField parentField;

private JTextField childrenField;

private JButton addButton, drawButton, refreshButton, saveButton, backButton, undoButton;

public ClassificationHierarchy() {

// GUI setup and layout

...

}

// Add hierarchy method

private void addHierarchy() {

...

}

// Undo last hierarchy

private void undoLastHierarchy() {

...

}

// Drawing the hierarchy

private void drawHierarchy(Graphics2D g2d, String parent, int x, int y, int width, int height) {

...

}

// Save diagram as image

private void saveDiagram(JPanel canvasPanel) {

...

}

public static void main(String[] args) {

...

}

}

## 6. Results

The application successfully allows users to input and visualize hierarchical relationships. Users can interact with the application by adding classifications, undoing entries, refreshing the canvas, and saving diagrams. The GUI is intuitive and responsive, enhancing user experience.

## 7. Conclusion

The Classification Hierarchy Diagram project meets its objectives of providing an interactive tool for visualizing hierarchical data. The use of Java Swing allowed for the creation of a responsive and user-friendly interface, while the graphics handling enabled clear visualization of relationships.

## 8. Future Work

Potential improvements include:

* Adding features for editing existing entries.
* Implementing functionality for exporting diagrams in various formats (e.g., JPEG, SVG).
* Enhancing the GUI design with more visual customization options.

## 9. References

* [Java Swing Documentation](https://docs.oracle.com/javase/tutorial/uiswing/)
* [Java ImageIO Documentation](https://docs.oracle.com/javase/7/docs/api/javax/imageio/ImageIO.html)
* [Java AWT Graphics Documentation](https://docs.oracle.com/javase/7/docs/api/java/awt/Graphics.html)

Feel free to modify or expand on any sections according to your specific project needs or to include additional details such as screenshots, flowcharts, or user feedback.

Bottom of Form