Linked List Visualization Project

# Project Overview

This project demonstrates a graphical user interface (GUI) based linked list visualization built using Java Swing. It allows users to perform various operations on a linked list, including inserting nodes at the head, tail, or a specific position, deleting nodes, and reversing the list. The linked list is displayed dynamically, reflecting changes in real-time with a visual representation of node connections.

# Main Class

The entry point of the application is the Main class, which is responsible for launching the GUI.

/\*\*  
 \* The entry point of the LinkedList visualization application.  
 \* This class is responsible for launching the GUI of the linked list.  
 \*/  
public class Main {  
  
 /\*\*  
 \* Main method that serves as the entry point of the program.  
 \* It creates and displays the LinkedList GUI.  
 \*  
 \* @param args Command-line arguments (not used in this program).  
 \*/  
 public static void main(String[] args) {  
 // Create a new instance of the LinkedListGui and make it visible  
 new LinkedListGui().setVisible(true);  
 }  
}

# LinkedListGui Class

This class manages the overall layout and functionality of the linked list visualization GUI. It uses Swing components like JPanel, JLabel, JTextField, and JButton for user interaction.

/\*\*  
 \* LinkedListGui manages the GUI for displaying and interacting with the linked list.  
 \* It includes buttons and input fields for various linked list operations.  
 \*/  
public class LinkedListGui extends JFrame {  
  
 private final LinkedList linkedList;  
 private JTextField inputField;  
 private JTextField positionField;  
 private JLabel sizeLabel;  
 private LinkedListPanel linkedListPanel;  
  
 public LinkedListGui() {  
 linkedList = new LinkedList();  
 setupUI();  
 }  
   
 // GUI setup for linked list visualization and controls  
 private void setupUI() {  
 setTitle("Linked List Visualization");  
 setSize(800, 600);  
 setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  
 setLocationRelativeTo(null);  
   
 // Setup linked list visualization panel  
 linkedListPanel = new LinkedListPanel(linkedList);  
 linkedListPanel.setBackground(Color.BLACK);  
 linkedListPanel.setPreferredSize(new Dimension(800, 400));  
   
 // Setup input fields and buttons for user interactions  
 inputField = new JTextField(10);  
 positionField = new JTextField(10);  
 sizeLabel = new JLabel("Size: 0");  
 sizeLabel.setForeground(Color.WHITE);  
   
 JButton insertAtHeadButton = new JButton("Insert at Head");  
 JButton insertAtTailButton = new JButton("Insert at Tail");  
 JButton insertAtPositionButton = new JButton("Insert at Position");  
 JButton deleteAtPositionButton = new JButton("Delete at Position");  
 JButton deleteAtHeadButton = new JButton("Delete at Head");  
 JButton deleteAtTailButton = new JButton("Delete at Tail");  
 JButton reverseButton = new JButton("Reverse List");  
   
 // Add buttons and inputs to the control panel  
 JPanel controlPanel = new JPanel(new GridBagLayout());  
 controlPanel.setBackground(Color.BLACK);  
   
 // Add panels to the frame  
 add(linkedListPanel, BorderLayout.CENTER);  
 add(controlPanel, BorderLayout.SOUTH);  
 }  
  
 // Update the size label to reflect the size of the linked list  
 private void updateSize() {  
 sizeLabel.setText("Size: " + linkedList.getSize());  
 }  
  
 // Main method to launch the GUI application  
 public static void main(String[] args) {  
 SwingUtilities.invokeLater(() -> new LinkedListGui().setVisible(true));  
 }  
}

# LinkedListPanel Class

This class extends JPanel and is responsible for visually drawing the linked list. It iterates over the linked list nodes and represents each node with a box, showing the data and next node address, along with arrows indicating the node connections.

/\*\*  
 \* LinkedListPanel extends JPanel to handle custom rendering of the linked list.  
 \* It draws nodes, arrows, and their connections based on the linked list data.  
 \*/  
public class LinkedListPanel extends JPanel {  
  
 private LinkedList linkedList;  
  
 public LinkedListPanel(LinkedList linkedList) {  
 this.linkedList = linkedList;  
 setPreferredSize(new Dimension(800, 400));  
 }  
  
 @Override  
 protected void paintComponent(Graphics g) {  
 super.paintComponent(g);  
 drawLinkedList(g);  
 }  
   
 // Draw each node of the linked list on the panel  
 private void drawLinkedList(Graphics g) {  
 Node current = linkedList.getHead();  
 int x = 20;  
 int y = 50;  
 int boxWidth = 150;  
 int boxHeight = 40;  
 int gap = 50;  
  
 while (current != null) {  
 // Draw node and its address, and handle visual layout  
 // (See full code for detailed comments)  
 current = current.next;  
 }  
 }  
}

# Linked List Operations

The linked list supports the following operations, each triggered by buttons in the GUI:

- Insert at Head: Adds a new node at the beginning of the list.

- Insert at Tail: Adds a new node at the end of the list.

- Insert at Position: Adds a new node at a specified position.

- Delete at Head: Removes the first node in the list.

- Delete at Tail: Removes the last node in the list.

- Delete at Position: Removes the node at a specified position.

- Reverse List: Reverses the entire list.