import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
// Define a Node class representing each step in the puzzle tree  
class Node {  
 int step; // The value of the step at this node  
 Node left, right; // Pointers to the left and right children  
  
 // Constructor to initialize a node with a step value  
 public Node(int step) {  
 this.step = step;  
 this.left = null; // No left child initially  
 this.right = null; // No right child initially  
 }  
}  
  
public class PuzzleGameGUI extends JFrame {  
 private Node root; // Root of the puzzle tree  
 private Node currentNode; // Track the current node during navigation  
  
 private JLabel stepLabel; // Label to show the current step  
 private JButton leftButton, rightButton, quitButton; // Navigation buttons  
 private JButton createTreeButton; // Button to start tree creation  
  
 // Constructor to set up the GUI  
 public PuzzleGameGUI() {  
 // Setting up the frame  
 setTitle("Puzzle Game");  
 setSize(400, 300);  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 setLayout(new BorderLayout());  
  
 // Create and add the step label to the frame  
 stepLabel = new JLabel("Click 'Create Tree' to start", SwingConstants.*CENTER*);  
 add(stepLabel, BorderLayout.*CENTER*);  
  
 // Panel for navigation buttons  
 JPanel buttonPanel = new JPanel();  
 buttonPanel.setLayout(new FlowLayout());  
  
 // Create Tree button to initialize the tree  
 createTreeButton = new JButton("Create Tree");  
 createTreeButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 createTree(); // Start the tree creation process  
 }  
 });  
 buttonPanel.add(createTreeButton);  
  
 // Left navigation button  
 leftButton = new JButton("Left");  
 leftButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 navigate("l");  
 }  
 });  
 buttonPanel.add(leftButton);  
  
 // Right navigation button  
 rightButton = new JButton("Right");  
 rightButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 navigate("r");  
 }  
 });  
 buttonPanel.add(rightButton);  
  
 // Quit button  
 quitButton = new JButton("Quit");  
 quitButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 System.*exit*(0); // Quit the game  
 }  
 });  
 buttonPanel.add(quitButton);  
  
 // Add the button panel to the frame  
 add(buttonPanel, BorderLayout.*SOUTH*);  
  
 // Set navigation buttons to be disabled initially  
 setNavigationButtons(false);  
 }  
  
 // Method to create the binary tree with user input  
 public void createTree() {  
 int rootStep = Integer.*parseInt*(JOptionPane.*showInputDialog*(this, "Enter the value for the root node:"));  
 root = new Node(rootStep);  
 createNode(root);  
 currentNode = root; // Start from the root for navigation  
 stepLabel.setText("Tree created! Start navigation.");  
 setNavigationButtons(true); // Enable navigation buttons  
 }  
  
 // Recursive helper method to create nodes for the tree  
 private void createNode(Node parent) {  
 // Prompt for left child  
 int leftValue = Integer.*parseInt*(JOptionPane.*showInputDialog*(this, "Enter left child of " + parent.step + " (-1 for no child):"));  
 if (leftValue != -1) {  
 parent.left = new Node(leftValue);  
 createNode(parent.left);  
 }  
  
 // Prompt for right child  
 int rightValue = Integer.*parseInt*(JOptionPane.*showInputDialog*(this, "Enter right child of " + parent.step + " (-1 for no child):"));  
 if (rightValue != -1) {  
 parent.right = new Node(rightValue);  
 createNode(parent.right);  
 }  
 }  
  
 // Method to handle navigation through the tree  
 private void navigate(String direction) {  
 if (currentNode == null) return;  
  
 // Move left or right based on the direction  
 if (direction.equals("l")) {  
 if (currentNode.left != null) {  
 currentNode = currentNode.left;  
 } else {  
 JOptionPane.*showMessageDialog*(this, "No left child. You’ve reached a dead end!");  
 return;  
 }  
 } else if (direction.equals("r")) {  
 if (currentNode.right != null) {  
 currentNode = currentNode.right;  
 } else {  
 JOptionPane.*showMessageDialog*(this, "No right child. You’ve reached a dead end!");  
 return;  
 }  
 }  
  
 // Update the label to show the current step  
 stepLabel.setText("You are at puzzle step: " + currentNode.step);  
  
 // Check if we've reached a leaf node  
 if (currentNode.left == null && currentNode.right == null) {  
 JOptionPane.*showMessageDialog*(this, "Congratulations! You have reached the end of this puzzle path at step: " + currentNode.step);  
 setNavigationButtons(false); // Disable navigation as we reached the end  
 }  
 }  
  
 // Enable or disable navigation buttons  
 private void setNavigationButtons(boolean enable) {  
 leftButton.setEnabled(enable);  
 rightButton.setEnabled(enable);  
 }  
  
 // Main method to start the game  
 public static void main(String[] args) {  
 // Create and show the GUI  
 SwingUtilities.*invokeLater*(new Runnable() {  
 public void run() {  
 new PuzzleGameGUI().setVisible(true);  
 }  
 });  
 }  
}