```java
import javafx.application.Application;
import java.util.*;

import javafx.application.Platform;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.geometry.Point2D;
import javafx.scene.control.Label;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.event.EventType;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.Node;
import javafx.scene.control.Button;
import javafx.scene.control.TextField;
import javafx.scene.input.MouseEvent;
import javafx.event.EventHandler;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class ExpressionEditor extends Application {
  public static void main (String[] args) {
    launch(args);
  }

  /**
   * Mouse event handler for the entire pane that constitutes the ExpressionEditor
   */
  private static class MouseEventHandler implements EventHandler<MouseEvent> {
    private int focusLevel;
    private Pane _pane;
    private CompoundExpression _rootExpression;
    private Expression selectedExp;
    private Node deepCopy;
    private CompoundExpression selectedExpParent;
    private HBox helpMeP;
    private Node selectedExpNode;

    private List<Expression> expressionList;

    private double curIndex;

    private boolean wasSwapped;



    MouseEventHandler(Pane pane_, CompoundExpression rootExpression_) {
      _pane = pane_;
      _rootExpression = rootExpression_;
    }


    public void handle(MouseEvent event) {
      if (event.getEventType() == MouseEvent.MOUSE_PRESSED) {
        highlight(event);
      } else if (event.getEventType() == MouseEvent.MOUSE_DRAGGED) {
        if (focusLevel > 0) {
          selectedExpNode.setOpacity(0.6);
          if (!_pane.getChildren().contains(deepCopy)) {
            _pane.getChildren().add(deepCopy);
          }
          moveNode(deepCopy, event.getX(), event.getY());
          rearrange2(event);
        }
      } else if (event.getEventType() == MouseEvent.MOUSE_RELEASED) {
        if (focusLevel > 0 && wasSwapped) {
          doTheSwap();
          _pane.getChildren().remove(deepCopy);
          selectedExpNode.setOpacity(1); //redundant
          focusLevel = 0;
          resetBordersAndOpacity(_pane.getChildren().get(0));
          System.out.println(_rootExpression.convertToString(0));
          wasSwapped = false;
        }
      }
    }

    /**
     * deals with the selecting of the expression
     * checks first if it is in bounds of the whole thing, then
     * goes through all the children and trees and tries to find an expression
     * that is valid based on the mouse's click location and the current focus level
     *
     * @param event the mouse (click) event
     */
    private void highlight(MouseEvent event) {
      //first child of _pane is one hBox, aka the ROOT NODE
      //so we want to start with the children of that one node
      Pane temp = _pane;
      HBox curHBox = (HBox) temp.getChildren().get(0);
      resetBordersAndOpacity(curHBox);
      Point2D mouse = curHBox.sceneToLocal(event.getSceneX(), event.getSceneY());
      if (!curHBox.contains(mouse)) {
        focusLevel = 0;
        return;
      } //if out of bounds
      int currentFocus = 0;
      CompoundExpression curExp = _rootExpression; //this is the current expression we are on.
      //this, and expCounter, are used to know which expression we are highlighting (keep track of it)
      while (true) {
        int expCounter = 0;
        for (Node n : curHBox.getChildren()) {
          //sets mouse coordinates equal to n's local coordinates so we can use contains method
          mouse = n.sceneToLocal(event.getSceneX(), event.getSceneY());
          if (n.contains(mouse)) {
            //if we are on the right focus level and the node is an expression...
            if (currentFocus == focusLevel && isValidExpression(n)) {
              focusLevel++; //increase focus level (for future selections)
```

```java

       final Expression theExpression = curExp.getChildList().get(expCounter); //get the expression
       System.out.println(theExpression.getParent().getChildList().size());
       if (selectedExp != null) {
        selectedExp.getNode().setOpacity(1);
        _pane.getChildren().remove(deepCopy);
       }
       selectedExp = theExpression; //based on the tracking that has been going on
       deepCopy = theExpression.deepCopy().getNode();
       selectedExpParent = curExp;
       helpMeP = curHBox; //del
       selectedExpNode = n;
       curIndex = expCounter; // goal is to record index of thing we are dragging
       //needed below:
       CompoundExpression tempCopy = (CompoundExpression) selectedExp.getParent().deepCopy();
       expressionList = tempCopy.getChildList();
       System.out.println(expressionList.size()); //this produces 2. should be 4.
       try {
        Pane pane = (Pane) n;
        pane.setBorder(Expression.RED_BORDER);
       } catch (Exception e) {
        Label label = (Label) n;
        label.setBorder(Expression.RED_BORDER);
       }
       return; //exit loop.
      }
      //ELSE... (aka we need a bigger focus...)
      currentFocus++; //increase this var
      if (n.getClass().equals(new HBox().getClass())) { //if n is an hBox, that means
       //that there are more expressions to check. if it is not hBox, it is the most focused.
       curHBox = (HBox) n;
       curExp = (CompoundExpression) curExp.getChildList().get(expCounter); //go to that child
       expCounter = 0;
      } else {
       //if n is NOT hbox, that means that there is no node to find, so we reset focus
       focusLevel = 0;
       return; //and exit out of the loop
      }

     } else if (isValidExpression(n)) {
      //checks if the node is a valid expression.
      expCounter++;
     }
    }
   }
  }

  /**
   * removes the border of node n and all children
   *
```

```java
     * @param n the node to have its border modified
     */
    private void resetBordersAndOpacity(Node n) {
      if (n.getClass().equals(new HBox().getClass())) {
        Pane hBox = (Pane) n;
                hBox.setBorder(Expression.NO_BORDER);
                hBox.setOpacity(1);
        for (Node x : hBox.getChildren()) {
          resetBordersAndOpacity(x);
        }
      } else {
        final Label label = (Label) n;
        label.setBorder(Expression.NO_BORDER);
        label.setOpacity(1);
      }

    }

    /**
     * checks if the node is a valid expression to highlight
     * a single node of +, ·, (, or ) is invalid; all else is valid
     * so only need to check if node n is a label and has one of the four above values.
     *
     * @param n the node to be checked
     * @return true if the node follows above conditions, false otherwise
     */
    private boolean isValidExpression(Node n) {
      if (n.getClass().equals(new Label().getClass())) {
        final Label label = (Label) n;
        if (label.getText().equals("+") || label.getText().equals("·")
            || label.getText().equals("(") || label.getText().equals(")")) {
          return false;
        }
      }
      return true;
    }

    private void moveNode(Node n, double deltaX, double deltaY) {
      n.setLayoutY(deltaY);
      n.setLayoutX(deltaX);
    }

    private void doTheSwap() {
      //doesn't work!
      CompoundExpression parent = selectedExpParent;
      parent.getChildList().removeAll(parent.getChildList());

      for (Expression e : expressionList) {
        parent.addSubexpression(e);
      }
      /*Pane p = (Pane) deepCopy.getParent();
      p.getChildren().remove(p);*/

    }

    //for loop of size of children
    //create a rearrangement and actually like DO IT (by removing all children and re-adding them)
    //check if that distance is closer than the current "closest one"
    //if it is, write down a way to get it back (NOT COPY)
    //then for the one that is closest, set it as it

    //doesn't work with parenthese (f off)
    private void rearrange2(MouseEvent event) {
      //helpMeP is HBox
      double startingX = helpMeP.localToScene(helpMeP.getBoundsInLocal()).getMinX();
      double widthOfOperator = helpMeP.getChildren().get(1).getLayoutBounds().getWidth();
      double expWidth = selectedExpNode.getLayoutBounds().getWidth(); //can prob delete

      //double targetPoint = selectedExpNode.localToScene(selectedExpNode.getBoundsInLocal()).getMinX() + selectedExpNode.getLayoutBounds().getWidth()/2;
      //mid point above
      List<Double> listOfValues = new ArrayList<Double>();
      List<Node> listOfNodes = new ArrayList<Node>();
      int theIndex = 0;
      int pastIndex = 0;
      for (int i = 0; i < helpMeP.getChildren().size(); i += 2) {
        //double xCord = helpMeP.getChildren().get(i).localToScene(helpMeP.getBoundsInLocal()).getMinX();
        double xCord = helpMeP.getChildren().get(i).getLayoutBounds().getWidth();
        //so this has 0 + or * labels
        if (!helpMeP.getChildren().get(i).equals(selectedExpNode)) {
          listOfValues.add(xCord);
          listOfNodes.add(helpMeP.getChildren().get(i));
        } else {
          theIndex = i;
          pastIndex = i / 2;
        }
      }

      double closestValue = -1;
      int index = -1;
      for (int i = 0; i < listOfValues.size() + 1; i++) {
        final double testValue = calculateX(i, listOfValues, widthOfOperator, startingX);
        if (i == 1) {
          /*System.out.println(closestValue);
          System.out.println(testValue);
          System.out.println(deepCopy.getLayoutBounds().getWidth()/2 + deepCopy.getLayoutX());*/
        }
        if (isClosestValue(closestValue, testValue)) {

          closestValue = testValue;
          index = i;
        }
      }
      //System.out.println(index);
      if (curIndex != index && index != -1) {
        instantiateNewOrdering(index, listOfValues.size() + 1, pastIndex, theIndex);
        curIndex = index;
      }

      //PROBLEM: THIS IS BASED ON MOUSEX AND NOOOOT ON THE DRAGGED EXPRESSION
      //WE IGNORE THIS FOR NOW

      //both of them will swap by one.

      //then we calculate the value of the x.
    }

    private void instantiateNewOrdering(int index, int size, int pastIndex,
int theIndex) {
      wasSwapped = true;
      List<Node> newChildList = new ArrayList<Node>();
      final Label op = (Label) helpMeP.getChildren().get(1);
      //final Label opCopy = new Label(op.getText());
      for (int i = 0; i < index - 1; i++) {
        //add all NON-DRAGGED NON-OPERATION NODES
        newChildList.add(helpMeP.getChildren().get(i * 2));
        newChildList.add(op);
      }
      newChildList.add(selectedExpNode);
      for (int i = index + 1; i < size; i++) {
        newChildList.add(op);
        newChildList.add(helpMeP.getChildren().get(i * 2));
      }
      Collection<Node> children = helpMeP.getChildren();

      System.out.println(children.equals(newChildList));
      //ADD DRAGGED

      if (pastIndex > index) {
        //move dragged to the left

        Node temp = helpMeP.getChildren().get(theIndex - 2);
        helpMeP.getChildren().set(theIndex, new Label());
        helpMeP.getChildren().set(theIndex - 2, new Label());
        helpMeP.getChildren().set(theIndex, temp);
        helpMeP.getChildren().set(theIndex - 2, selectedExpNode);
        Expression tempE = expressionList.get(pastIndex);
        expressionList.set(pastIndex, expressionList.get(pastIndex - 1));
        expressionList.set(pastIndex - 1, tempE);

      } else {
        //move dragged to the right
        Node temp = helpMeP.getChildren().get(theIndex + 2);
        helpMeP.getChildren().set(theIndex, new Label());
        helpMeP.getChildren().set(theIndex + 2, new Label());
        helpMeP.getChildren().set(theIndex, temp);
        helpMeP.getChildren().set(theIndex + 2, selectedExpNode);
        Expression tempE = expressionList.get(pastIndex);
        expressionList.set(pastIndex, expressionList.get(pastIndex + 1));
        expressionList.set(pastIndex + 1, tempE);
      }
    }


    private double calculateX(int index, List<Double> listOfValues, double
opWidth, double startingPoint) {
      double sum = startingPoint +
selectedExpNode.getLayoutBounds().getWidth() / 2;
```

```java
      for (int i = 0; i < index; i++) {
        sum += listOfValues.get(i);
        sum += opWidth;
      }
      return sum;
    }

    private boolean isClosestValue(double oldV, double newV) {
      if (oldV < 0) {
        return true;
      }
      double oldTwo = Math.abs(deepCopy.getLayoutX() + deepCopy.getLayoutBounds().getWidth() / 2 - oldV);
      double newTwo = Math.abs(deepCopy.getLayoutX() + deepCopy.getLayoutBounds().getWidth() / 2 - newV);
      return newTwo < oldTwo;
      //like if we want to do this based on expression dragged location, we would have to compare minX and maxX
      //and see, out of EVERYTHING, which is the smallest. hardest part with that is calculating
      //min and max x (JK THIS IS SUPER EASY)

    }

  }

  /**
   * Size of the GUI
   */
  private static final int WINDOW_WIDTH = 500, WINDOW_HEIGHT = 250;

  /**
   * Initial expression shown in the textbox
   */
  private static final String EXAMPLE_EXPRESSION = "2*x+3*y+4*z+(7+6*z)";

  /**
   * Parser used for parsing expressions.
   */
  private final ExpressionParser expressionParser = new SimpleExpressionParser();

  @Override
  public void start (Stage primaryStage) {
    primaryStage.setTitle("Expression Editor");

    // Add the textbox and Parser button
    final Pane queryPane = new HBox();
    final TextField textField = new TextField(EXAMPLE_EXPRESSION);
    final Button button = new Button("Parse");
    queryPane.getChildren().add(textField);

    final Pane expressionPane = new Pane();

    // Add the callback to handle when the Parse button is pressed
    button.setOnMouseClicked(new EventHandler<MouseEvent>() {
      public void handle (MouseEvent e) {
        // Try to parse the expression
        try {
          // Success! Add the expression's Node to the expressionPane
          final Expression expression = expressionParser.parse(textField.getText(), true);
          System.out.println(expression.convertToString(0));
          expressionPane.getChildren().clear();
                    final Node node = expression.getNode();
          expressionPane.getChildren().add(node);
          node.setLayoutX(WINDOW_WIDTH/4);
          node.setLayoutY(WINDOW_HEIGHT/2);
          // If the parsed expression is a CompoundExpression, then register some callbacks
          if (expression instanceof CompoundExpression) {
            ((Pane) expression.getNode()).setBorder(Expression.NO_BORDER);
            final MouseEventHandler eventHandler = new MouseEventHandler(expressionPane, (CompoundExpression) expression);
            expressionPane.setOnMousePressed(eventHandler);
            expressionPane.setOnMouseDragged(eventHandler);
            expressionPane.setOnMouseReleased(eventHandler);

          }
        } catch (ExpressionParseException epe) {
          // If we can't parse the expression, then mark it in red
          textField.setStyle("-fx-text-fill: red");
        }
      }
    });
    queryPane.getChildren().add(button);

    // Reset the color to black whenever the user presses a key
    textField.setOnKeyPressed(e -> textField.setStyle("-fx-text-fill: black"));

    final BorderPane root = new BorderPane();
    root.setTop(queryPane);
    root.setCenter(expressionPane);

    primaryStage.setScene(new Scene(root, WINDOW_WIDTH, WINDOW_HEIGHT));
    primaryStage.show();
  }

}
```