Following patterns are found in TreeGUIBasic.java :

1. STATE DESIGN PATTERN

```
input1 = new JPanel(new FlowLayout()); // The First Panel

tree_kind_label = new JLabel("Tree Kind:");
input1.add(tree_kind_label);
tree_kind = new Choice();
tree_kind.addItem("Normal Tree");
tree_kind.addItem("Dup Tree");
input1.add(tree_kind);

elem_kind_label = new JLabel("   Element Kind:");
input1.add(elem_kind_label);
element_kind = new Choice();
element_kind.addItem("Integer");
element_kind.addItem("String");
input1.add(element_kind);

input_elem_text = new JTextField("enter value here");
input_elem_text.requestFocus(true);
input_elem_text.selectAll();

insertButton = new JButton("Insert");
insertButton.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent e) {
            String s = input_elem_text.getText();
            int index = element_kind.getSelectedIndex();
            {
                  if (is_new_tree) {
                        if (index == 0) {
                              if (tree_kind.getSelectedIndex() == 0)
                                    tree = new Tree(
                                                new IntElem(Integer.parseInt(s)));
                              else
                                    tree = new Duptree(new IntElem(Integer
                                                .parseInt(s)));
                        } else {
                              if (tree_kind.getSelectedIndex() == 0)
                                    tree = new Tree(new StringElem(s));
                               else
                                    tree = new Duptree(new StringElem(s));
                        }
                        is_new_tree = false;
                  }
                  else {
```

```
                    if (index == 0)
                        tree.insert(new IntElem(Integer.parseInt(s)));
                    else
                        tree.insert(new StringElem(s));
                }
            }
            input_elem_text.selectAll();
        }
    });

    input_elem_text.setEditable(true);
    input1.add(input_elem_text);
    input1.add(insertButton);
```

2. COMPOSITE DESIGN PATTERN

```
input2 = new JPanel();                  // The Second Panel

    minButton = new JButton("Minimum");
    minButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            min = tree.min().get_value();
            min_text.setText(min);
        }
    });
    input2.add(minButton);
    min_text = new JTextField(10);
    min_text.setEditable(false);
    input2.add(min_text);

    maxButton = new JButton("Maximum");
    maxButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            max = tree.max().get_value();
            max_text.setText(max);
        }
    });
    input2.add(maxButton);
    max_text = new JTextField(10);
    max_text.setEditable(false);
    input2.add(max_text);
```

```
        clearButton = new JButton("Clear");
        clearButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                is_new_tree = true;
                input_elem_text.setText("");
                min_text.setText("");
                max_text.setText("");
            }
        });
        input2.add(clearButton);
```

3. OBSERVATION DESIGN PATTERN

```
output = new JPanel();          // The Third Panel


        printButton = new JButton("Print Values");
        output.add(printButton);
        output_text  = new JTextArea("",1,30);
        output.add(output_text);

        printButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String str;
                str = tree.inorder("");
                output_text.setText(str);
            }
        });


        inputPanel.add("North", input1);
        inputPanel.add("South", input2);

        add("North", inputPanel);
        add("Center", output);

        setSize(600, 400); // for the frame
        setVisible(true);
    }
```

4. TEMPLATE METHOD PATTERN

```
abstract class Abs_tree {


protected abstract void count_duplicates();
protected abstract Abs_tree add_node(Element n);
public abstract String get_value();
}

class Tree extends Abs_tree {
        protected Abs_tree add_node(Element n) {
                return new Tree(n);
        }

        protected void count_duplicates() {
        }

        public String get_value() {
                return (node.get_value());
        }

}

class Duptree extends Abs_tree {
        protected Abs_tree add_node(Element n) {
                return new Duptree(n);
        }

        protected void count_duplicates() {
                count++;
        }

        public String get_value() {
                return (node.get_value() + '/' + count);
        }
```

5. FACTORY METHOD PATTERN

```
abstract class Abs_tree {

protected abstract void count_duplicates();
protected abstract Abs_tree add_node(Element n);
```

```
public abstract String get_value();
}

class Tree extends Abs_tree {
protected Abs_tree add_node(Element n) {
                return new Tree(n);
        }

        protected void count_duplicates() {
        }

        public String get_value() {
                return (node.get_value());
        }

}

class Duptree extends Abs_tree {
        protected Abs_tree add_node(Element n) {
                return new Duptree(n);
        }

        protected void count_duplicates() {
                count++;
        }

        public String get_value() {
                return (node.get_value() + '/' + count);
        }
}
```

6. STRATEGY PATTERN

```
interface Element {
        public boolean equal(Element n);

        public boolean less_than(Element n);

        public String get_value();
}



class IntElem implements Element {

        public boolean equal(Element n) {
```

```
    return (this.value == ((IntElem) n).getValue());
  }

  public boolean less_than(Element n) {
    return (this.value < ((IntElem) n).getValue());
  }

}


class StringElem implements Element {

  public boolean equal(Element n) {
    return (this.value.equals(((StringElem) n).get_value()));
  }

  public boolean less_than(Element n) {
    return (value.compareTo(((StringElem) n).get_value()) < 0);
  }

}
```