COMP1406 - Assignment #10

(Due: **Monday, March 31st @ 11:30pm**)

In this assignment you will practice using recursion with data structures.

(1) Consider the following **BinaryTree** class:

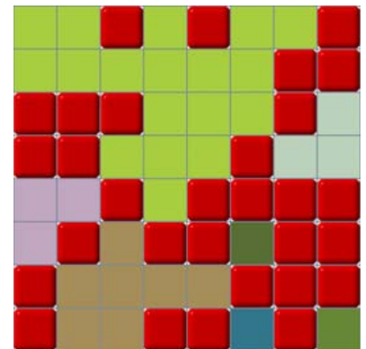
```
public class BinaryTree {
      private String          data;
      private BinaryTree      leftChild;
      private BinaryTree      rightChild;

      public BinaryTree(String d) {
            data = d;
            leftChild = null;
            rightChild = null;
      }

      public BinaryTree(String d, BinaryTree left, BinaryTree right) {
            data = d;
            leftChild = left;
            rightChild = right;
      }
}
```

Write a **recursive** method in this class called **hasSameStructureAs(BinaryTree tree)** that returns whether or not a tree has the same structure as another tree. Two trees have the same left/right children locations all the way through from the root to the leaves. The data at each node need not be the same, however. You will not receive any marks if your code is non-recursive or if it contains any loops. Use the following to test your code:

```
public class BinaryTreeTest {
   public static void main(String[] args) {
      BinaryTree[] trees = new BinaryTree[8];

      trees[0] = new BinaryTree("A");
      trees[1] = new BinaryTree("A", new BinaryTree("B"), new BinaryTree("C"));
      trees[2] = new BinaryTree("A",
            new BinaryTree("B", new BinaryTree("C"), null), new BinaryTree("D"));
      trees[3] = new BinaryTree("A", null, new BinaryTree("C", new BinaryTree("D"),
                     new BinaryTree("E",
                        new BinaryTree("F", new BinaryTree("G"), null),
                        new BinaryTree("H"))));
      trees[4] = new BinaryTree("A",
               new BinaryTree("B",
                  new BinaryTree("C",
                     new BinaryTree("D"),
                     new BinaryTree("E",
                        new BinaryTree("F",
                           new BinaryTree("G"),
                           new BinaryTree("I")),
                        new BinaryTree("H"))),
                  new BinaryTree("J",
                     new BinaryTree("K",
                        null,
                        new BinaryTree("L",
                           null,
```

```
                            new BinaryTree("M"))),
                        new BinaryTree("N",
                           null,
                           new BinaryTree("O")))),
                new BinaryTree("P",
                   new BinaryTree("Q"),
                   new BinaryTree("R",
                      new BinaryTree("S",
                         new BinaryTree("T"),
                         null),
                      new BinaryTree("U"))));
        trees[5] = new BinaryTree("A", null, new BinaryTree("P", null,
                   new BinaryTree("R", null, new BinaryTree("U"))));
        trees[6] = new BinaryTree("A", null, new BinaryTree("P", null,
                   new BinaryTree("R", new BinaryTree("W"), new BinaryTree("U"))));
        trees[7] = new BinaryTree("A",
                new BinaryTree("B",
                   new BinaryTree("C",
                      null,
                      new BinaryTree("E",
                         new BinaryTree("F",
                            new BinaryTree("G"),
                            new BinaryTree("I")),
                         new BinaryTree("H"))),
                   new BinaryTree("J",
                      new BinaryTree("K",
                         null,
                         new BinaryTree("L",
                            null,
                            new BinaryTree("M"))),
                      new BinaryTree("N",
                         null,
                         new BinaryTree("O")))),
                null);

        for (int i=0; i<8; i++)
           for (int j=0; j<8; j++)
              System.out.println("Tree[" + i + "] == trees[" + j + "]: " +
                                    trees[i].hasSameStructureAs(trees[j]));
     }
  }
```

In this test case, the results should show that only a tree compared against itself will result in **true** and all the rest should be **false**.



(2) Below is a **RoomMaze** class. This class represents a grid in which there are walls at various locations in an 8 x 8 grid. The walls separate *rooms*. Your task is to complete the **identifyRooms()** method (it MUST be recursive) and the **traceRoomFrom()** method so that it calculates and returns an integer indicating the number of rooms in the maze. The code must assign a unique color index to each room, starting with the number 1 (i.e., don't start at 0 because 0 represents a wall location). So, each location in the maze should be assigned an integer indicating the room that that grid location belongs to. These indices will be used to color the room in the test program. Note that if two wall locations are diagonally adjacent to one another (i.e., their corners touch) then this represents a separation between rooms. The image here shows walls (as red) and 7 rooms that have been uniquely identified:

```java
public class RoomMaze {
   public static byte   ROWS = 8;
   private byte[][]     wallTable;


   public RoomMaze() {
      wallTable = new byte[ROWS][ROWS];
      resetWalls();
   }

   public void resetWalls() {
      for (int r=0; r<ROWS; r++)
         for (int c=0; c<ROWS; c++) {
            wallTable[r][c] = (byte)(Math.random()*2);
         }

   }

   public byte getWall(int r, int c) { return wallTable[r][c]; }

   public int identifyRooms() {
      // Go through each location in the maze. If there is not a
      // wall there, then start tracing out the room recursively.
      // Keep count of how many rooms are traced out and assign
      // a colorIndex (i.e., store it in the wallTable) to each
      // room being traced.   Wall locations MUST have a value
      // of zero in the wallTable.  You may add attributes to this
      // class if you need to.
      return 1; // Replace this line
   }

   public void traceRoomFrom(int r, int c, byte colorIndex) {
      // Code missing.  This method MUST be directly recursive
   }
}
```

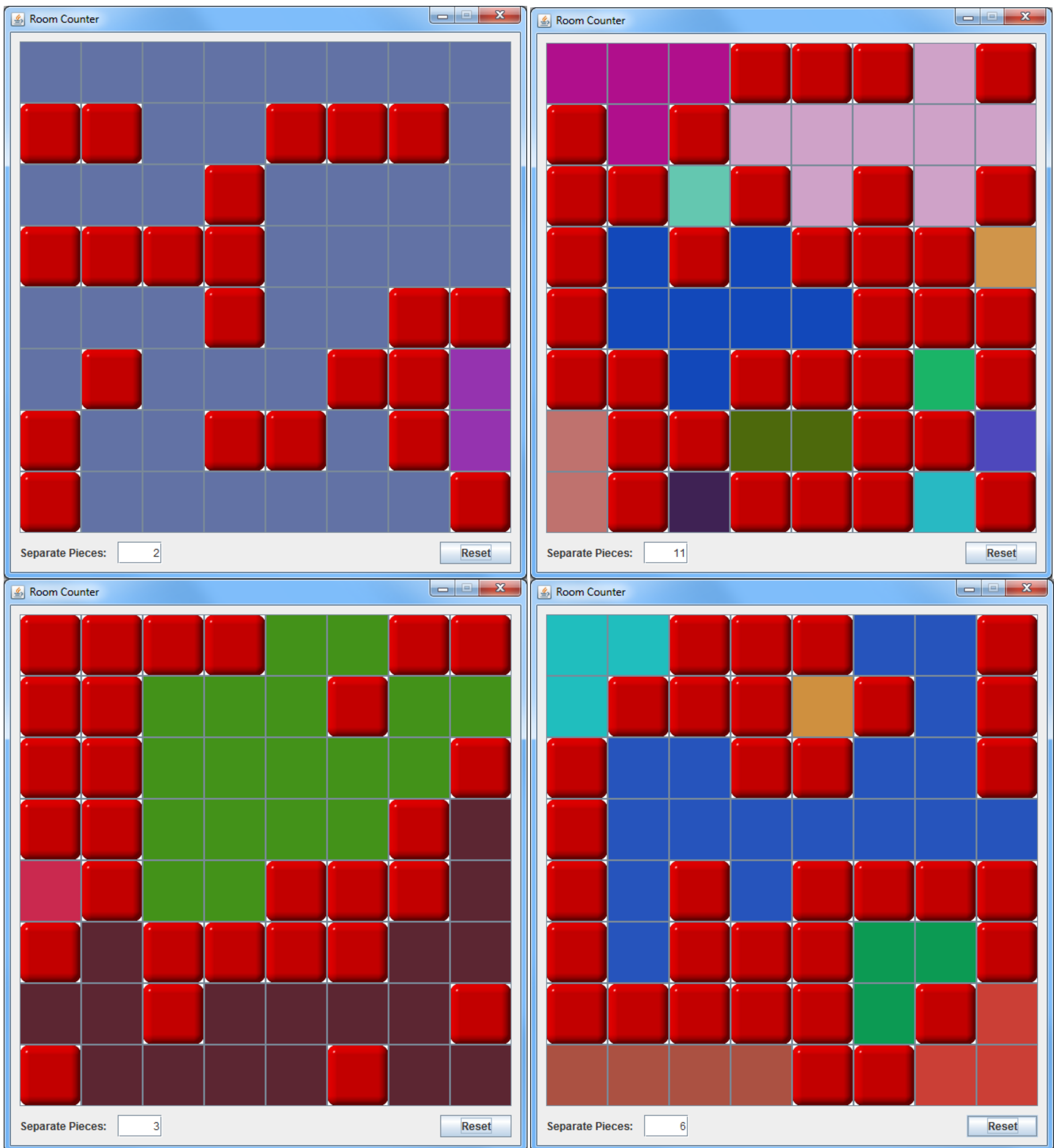
To test your code, use the following class:

```java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

public class RoomCounterApp extends JFrame {
   public static ImageIcon    wall = new ImageIcon("squareButton.png");

   // These are the model-specific variables
   private RoomMaze     maze;

   // These are window components
   private JButton[][]        buttons;
   private JTextField         countField;

   // This constructor builds the window
   public RoomCounterApp(String title) {
      super(title);
      initializeBoard();

      setupComponents();

      setDefaultCloseOperation(EXIT_ON_CLOSE);
      setSize(578,634);
```

```java
    setResizable(false);

    update();
  }


  // Initialize the board.  Do this whenever a new game is started.
  private void initializeBoard() {
      maze = new RoomMaze();
  }

  // Here we add all the components to the window accordingly
  private void setupComponents() {
    getContentPane().setLayout(null);

    // Setup the panel with the buttons
    buttons = new JButton[RoomMaze.ROWS][ RoomMaze.ROWS];
    JPanel tiles = new JPanel();
    tiles.setLayout(null);

    // Add the buttons to the tile panel
    for (int r=0; r<RoomMaze.ROWS; r++) {
       for (int c=0; c<RoomMaze.ROWS; c++) {
          buttons[r][c] = new JButton(wall);
          buttons[r][c].setBackground(Color.WHITE);
          buttons[r][c].setSize(69,69);
          buttons[r][c].setLocation(c*69, r*69);
          tiles.add(buttons[r][c]);
       }
    }
    tiles.setLocation(10,10);
    tiles.setSize(552,552);
    getContentPane().add(tiles);

    // Add the score and buttons
    JLabel aLabel = new JLabel("Separate Pieces:");
    aLabel.setSize(150,25);
    aLabel.setLocation(10, 572);
    getContentPane().add(aLabel);

    countField = new JTextField();
    countField.setSize(50,25);
    countField.setLocation(120, 572);
    countField.setHorizontalAlignment(JTextField.RIGHT);
    getContentPane().add(countField);

    JButton resetButton = new JButton("Reset");
    resetButton.setSize(80,25);
    resetButton.setLocation(482, 572);
    getContentPane().add(resetButton);
    resetButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
             maze.resetWalls();
             update();
         }
     });
 }

 // Update the board
 private void update() {
    int count = maze.identifyRooms();
    Color[] colors = new Color[count];
```

```
      for (int i=0; i<count; i++) {
         colors[i] = new Color((int)(10+Math.random()*200),
                        (int)(10+Math.random()*200), (int)(10+Math.random()*200));
      }
      countField.setText("" + count);

      // Update the look of the buttons
      for (int r=0; r<RoomMaze.ROWS; r++) {
         for (int c=0; c<RoomMaze.ROWS; c++) {
            if (maze.getWall(r,c) == 0) {
               buttons[r][c].setIcon(wall);
            }
            else {
               buttons[r][c].setIcon(null);
               buttons[r][c].setBackground(colors[maze.getWall(r,c)-1]);
            }
         }
      }
   }

   public static void main(String[] args) {
      JFrame frame = new RoomCounterApp("Room Counter");
      frame.setVisible(true);
   }
}
```

On the next page there are some screen snapshots showing what your results should look like, depending on the random arrangement of the walls and the randomly chosen colors:



NOTE: Submit all **.java** files needed to run as well as any image icon files. You **MUST NOT use packages** in your code, **nor projects**. Submit ALL of your files in one folder such that they can be opened and compiled individually in JCreator. Some IDEs may create packages and/or projects automatically. You **MUST** export the .java files and remove the package code at the top if it is there. Do NOT submit JCreator projects either. **JUST SUBMIT the JAVA FILES and image icon files**. Note that if your internet connection at home is down or does not work, we will not accept this as a reason for handing in an assignment late ... so make sure to submit the assignment WELL BEFORE it is due !

Please NOTE that you WILL lose marks on this assignment if any of your files are missing. You will also lose marks if your code is not written neatly with proper indentation. See examples in the notes for proper style.