```
package OurNeighborsChild;

import java.awt.Color;

public class DataCheckDialog extends JDialog implements ActionListener,
ListSelectionListener
{
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    private static final int NUM_ROWS_TO_DISPLAY = 10;
    private static final Integer MAXIMUM_ONC_NUMBER = 9999;

    public JTable dupTable;
    private DefaultTableModel dupTableModel;
    private JCheckBox[] cbArray;
    private JLabel lblCount;
    private JButton btnPrint;
    private ArrayList<DupItem> dupAL;
    private ArrayList<ONCFamily> fAL;
    private ChildDB cDB;

    private boolean bChangingTable;

    DataCheckDialog(JFrame pf, ImageIcon oncIcon, Families fdb, ChildDB
cdb)
    {
        super(pf, false);
        fAL = fdb.getFamilyDB();
        cDB = cdb;
        this.setTitle("Our Neighbor's Child - Child Database Checks");

        //Initialize Dup Table data structure
        dupAL = new ArrayList<DupItem>();

        JPanel contentPane = (JPanel) this.getContentPane();
        contentPane.setLayout(new BoxLayout(contentPane,
BoxLayout.Y_AXIS));

        JPanel topPanel = new JPanel();
        topPanel.setLayout(new FlowLayout(FlowLayout.LEFT));
```

```
        topPanel.setBorder(BorderFactory.createTitledBorder("Child Data
Check Criteria"));

        JLabel lblONCIcon = new JLabel(oncIcon);

        JPanel checkCriteriaPanel = new JPanel(new
FlowLayout(FlowLayout.LEFT));
        topPanel.setBorder(BorderFactory.createTitledBorder("Child Data
Check Criteria"));

        cbArray = new JCheckBox[6];
        cbArray[0] = new JCheckBox("Different Families?");
        cbArray[0].setSelected(true);
        cbArray[1] = new JCheckBox("Date of Birth");
        cbArray[1].setSelected(true);
        cbArray[2] = new JCheckBox("Gender");
        cbArray[2].setSelected(true);
        cbArray[3] = new JCheckBox("First Name");
        cbArray[4] = new JCheckBox("Exact Last Name");
        cbArray[4].setSelected(true);
        cbArray[5] = new JCheckBox("Partial Last Name");

        for(int i=0; i<cbArray.length; i++)
        {
            cbArray[i].addActionListener(this);
            checkCriteriaPanel.add(cbArray[i]);
        }

        topPanel.add(lblONCIcon);
        topPanel.add(checkCriteriaPanel);

        //Create the table to display duplicate check results
        dupTable = new JTable()
        {
            private static final long serialVersionUID = 1L;

            //Implement table header tool tips.
            protected String[] columnToolTips = {"Family #1 ONC Number",
"Child #1 First Name", "Child #1 Last Name",
                                                 "Child #1 Gender", "Child
#1 DOB", "Result",
                                                 "Family #2 ONC Number",
```

```
"Child #2 First Name", "Child #2 Last Name",
                                                    "Child #2 Gender", "Child
#2 DOB"};

            public boolean getScrollableTracksViewportWidth()
            {
                return getPreferredSize().width < getParent().getWidth();
            }

            protected JTableHeader createDefaultTableHeader()
            {
                return new JTableHeader(columnModel)
                {
                    private static final long serialVersionUID = 1L;

                    public String getToolTipText(MouseEvent e)
                    {
                        java.awt.Point p = e.getPoint();
                        int index = columnModel.getColumnIndexAtX(p.x);
                        int realIndex =
columnModel.getColumn(index).getModelIndex();
                        return columnToolTips[realIndex];
                    }
                };
            }

            public Component prepareRenderer(TableCellRenderer
renderer,int Index_row, int Index_col)
            {
              Component comp = super.prepareRenderer(renderer, Index_row,
Index_col);

              if(isRowSelected(Index_row))
                  comp.setBackground(comp.getBackground());
              else if (Index_row % 2 == 1)
                  comp.setBackground(new Color(240,248,255));
              else
                  comp.setBackground(Color.white);

              return comp;
            }
        };
```

```
        String[] columns = {"ONC#", "First Name","Last Name", "Gen.",
"DOB", "Result",
                            "ONC #", "First Name", "Last Name", "Gen.",
"DOB"};
        dupTableModel = new DefaultTableModel(columns, 0)
        {
            private static final long serialVersionUID = 1L;
            @Override
            //All cells are locked from being changed by user
            public boolean isCellEditable(int row, int column) {return
false;}
        };


        dupTable.setModel(dupTableModel);

dupTable.setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECTION);

        dupTable.setAutoResizeMode( JTable.AUTO_RESIZE_OFF);

      //Set table column widths
            int tablewidth = 0;
            int[] colWidths = {40, 88, 96, 48, 72, 72, 40, 88, 96, 48,
72};
            for(int i=0; i < colWidths.length; i++)
            {

dupTable.getColumnModel().getColumn(i).setPreferredWidth(colWidths[i]);
                tablewidth += colWidths[i];
            }
            tablewidth += 24;   //Account for vertical scroll bar


        JTableHeader anHeader = dupTable.getTableHeader();
        anHeader.setForeground( Color.black);
        anHeader.setBackground( new Color(161,202,241));

        //mouse listener for header click
        anHeader.addMouseListener(new MouseAdapter()
        {
            @Override
```

```
            public void mouseClicked(MouseEvent e)
            {
                if(dupTable.columnAtPoint(e.getPoint()) == 0)   //Sort on
Family 1 ONC Family
                    Collections.sort(dupAL, new
DupItemONCNumComparator());
                else if(dupTable.columnAtPoint(e.getPoint()) == 2)  //
Sort on Batch Number
                    Collections.sort(dupAL, new
DupItemChild1LastNameComparator());
                else if(dupTable.columnAtPoint(e.getPoint()) == 4)  //
Sort on ONC Family Number
                    Collections.sort(dupAL, new
DupItemChild1AgeComparator());
                else
                    return;

                displayDupTable();
            }
        });

        //Center cell entries for Batch # and Region
        DefaultTableCellRenderer dtcr = new DefaultTableCellRenderer();
        dtcr.setHorizontalAlignment(SwingConstants.CENTER);
        dupTable.getColumnModel().getColumn(3).setCellRenderer(dtcr);
        dupTable.getColumnModel().getColumn(5).setCellRenderer(dtcr);
        dupTable.getColumnModel().getColumn(9).setCellRenderer(dtcr);


dupTable.setBorder(UIManager.getBorder("Table.scrollPaneBorder"));
        dupTable.setFillsViewportHeight(true);

        dupTable.getSelectionModel().addListSelectionListener(this);

        //Create the scroll pane and add the table to it.
        JScrollPane dupScrollPane = new JScrollPane(dupTable,
JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,

JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);

        dupScrollPane.setPreferredSize(new Dimension(tablewidth,
dupTable.getRowHeight()*NUM_ROWS_TO_DISPLAY));
```

```
        //Create the bottom panel
        JPanel bottomPanel = new JPanel();
        bottomPanel.setLayout(new BoxLayout(bottomPanel,
BoxLayout.X_AXIS ));

        //Create the count panel
        JPanel countPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
        lblCount = new JLabel("Child Count: 0");
        bottomPanel.add(lblCount);

        //Create the control panel
        JPanel cntlPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));
        btnPrint = new JButton("Print");
        btnPrint.setEnabled(false);
        btnPrint.addActionListener(this);
        cntlPanel.add(btnPrint);

        bottomPanel.add(countPanel);
        bottomPanel.add(cntlPanel);

        contentPane.add(topPanel);
        contentPane.add(dupScrollPane);
        contentPane.add(bottomPanel);

        pack();
//      setSize(780, 220);
        setResizable(true);
        Point pt = pf.getLocation();
        setLocation(pt.x + 20, pt.y + 20);

    }

    void buildDupTable()
    {
        dupAL.clear();
        btnPrint.setEnabled(false);
        dupTable.setRowSelectionAllowed(false);

        DuplicateDataCheck datachecker = new DuplicateDataCheck();
        boolean[] criteria = {cbArray[0].isSelected(),
```

```
cbArray[1].isSelected(), cbArray[2].isSelected(),
                                cbArray[3].isSelected(),
cbArray[4].isSelected(), cbArray[5].isSelected()};

        //If child comparison returns a match, sort by child 1 last name,
allow table row selections and user print
        if(datachecker.duplicateChildCheck(fAL, criteria, cDB, dupAL))
        {
            Collections.sort(dupAL, new
DupItemChild1LastNameComparator());
            dupTable.setRowSelectionAllowed(true);
            btnPrint.setEnabled(true);
        }

        displayDupTable();
    }

    void displayDupTable()
    {
        bChangingTable = true;  //don't process table messages while
being changed

        while (dupTableModel.getRowCount() > 0) //Clear the current table
            dupTableModel.removeRow(0);

        for(DupItem di:dupAL)   //Build the new table
            dupTableModel.addRow(di.getDupChildTableRow());

        lblCount.setText("Child Count: " +
Integer.toString(dupTableModel.getRowCount()));

        bChangingTable = false;
    }

    void onPrintDataCheck()
    {
        try
        {
             MessageFormat headerFormat = new MessageFormat("Child Data
Check");
             MessageFormat footerFormat = new MessageFormat("- {0} -");
             dupTable.print(JTable.PrintMode.FIT_WIDTH, headerFormat,
```

```
footerFormat);
        }
        catch (PrinterException e)
        {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    ListSelectionModel getDupTableLSM() { return
dupTable.getSelectionModel(); }

    int getSelectedDupItemONCID() { return
dupAL.get(dupTable.getSelectedRow()).getDupItem1ONCID(); }

    boolean isDupTableChanging() { return bChangingTable; }

    @Override
    public void valueChanged(ListSelectionEvent arg0) {
        // TODO Auto-generated method stub

    }

    @Override
    public void actionPerformed(ActionEvent ae)
    {
        if(ae.getSource() == btnPrint) { onPrintDataCheck(); }
        else
        {
            if(ae.getSource() == cbArray[0] && cbArray[0].isSelected())
                cbArray[3].setSelected(false);
            else if(ae.getSource() == cbArray[0] && !
cbArray[0].isSelected())
                cbArray[3].setSelected(true);
            else if(ae.getSource() == cbArray[4] &&
cbArray[4].isSelected())
                cbArray[5].setSelected(false);
            else if(ae.getSource() == cbArray[5] &&
cbArray[5].isSelected())
                cbArray[4].setSelected(false);
        }
```

```
        buildDupTable();
    }

    public static boolean isNumeric(String str)
    {
      return str.matches("-?\\d+(\\.\\d+)?");  //match a number with
optional '-' and decimal.
    }

    private class DupItemONCNumComparator implements Comparator<DupItem>
    {
        @Override
        public int compare(DupItem d1, DupItem d2)
        {
            Integer onc1, onc2;

            if(!d1.getDupItemFamily1ONCNum().isEmpty() &&
isNumeric(d1.getDupItemFamily1ONCNum()))
                onc1 = Integer.parseInt(d1.getDupItemFamily1ONCNum());
            else
                onc1 = MAXIMUM_ONC_NUMBER;

            if(!d2.getDupItemFamily1ONCNum().isEmpty() &&
isNumeric(d2.getDupItemFamily1ONCNum()))
                onc2 = Integer.parseInt(d2.getDupItemFamily1ONCNum());
            else
                onc2 = MAXIMUM_ONC_NUMBER;

            return onc1.compareTo(onc2);
        }
    }

    private class DupItemChild1AgeComparator implements
Comparator<DupItem>
    {
        @Override
        public int compare(DupItem d1, DupItem d2)
        {
            return
d1.getDupItemChild1DOB().compareTo(d2.getDupItemChild1DOB());
        }
    }
```

```
    private class DupItemChild1LastNameComparator implements
Comparator<DupItem>
    {
        @Override
        public int compare(DupItem d1, DupItem d2)
        {
            return
d1.getDupItemChild1LastName().compareTo(d2.getDupItemChild1LastName());
        }
    }
}
```