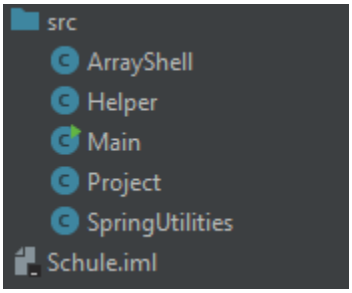
Appendix

All Classes:



Project Class:

```
public class Project

{

    private String projectName;
    private String course;
    private String daysReq;
    private String numDate;
    private String month;
    private String description;

    public Project(String projectName, String course, String daysReq, String numDate,
String month, String description) {
        this.setProjectName(projectName);
        this.setCourse(course);
        this.setDaysReq(daysReq);
        this.setNumDate(numDate);
        this.setMonth(month);
        this.setDescription(description);

    }

    public static Project newProject(String projectName, String course, String day, String
numDate, String month, String description) {
        Project newProject = new Project(projectName, course, day, numDate, month,
description);
        return newProject;
    }

    ;

    public Project() {
    }

    ;

    public String getProjectName() {
        return projectName;
    }

    public void setProjectName(String projectName) {
        this.projectName = projectName;
    }

    public String getCourse() {
        return course;
    }

    public void setCourse(String course) {
        this.course = course;
    }

    public String getDaysReq() {
        return daysReq;
    }

    public void setDaysReq(String daysReq) {
        this.daysReq = daysReq;
    }

    public String getNumDate() {
        return numDate;
    }
```

```
    public void setNumDate(String numDate) {
        this.numDate = numDate;
    }

    public String getMonth() {
        return month;
    }

    public void setMonth(String month) {
        this.month = month;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }


}
```

Helper Class:

```
public class Helper {


    Font font = new Font("Dialog", PLAIN, 12);

    public static JFrame createFrame(String title, int width, int height){
        JFrame newFrame = new JFrame(title);
        newFrame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        newFrame.setSize(width, height);
        newFrame.setLocationRelativeTo(null);
        newFrame.setVisible(false);
        newFrame.setLayout(new BorderLayout());
        return newFrame;
    }

    public static JLabel createLabel(String text){
        Font font = new Font("Dialog", PLAIN, 12);
        JLabel newLabel = new JLabel(text);
        newLabel.setFont(font);
        return newLabel;
    }

    public static JButton createButton(String title){
        Font font = new Font("Dialog", PLAIN, 12);
        JButton newButton = new JButton(title);
        newButton.setFont(font);
        return newButton;
    }
```

ArrayShell Class:

```
import java.util.*;

@SuppressWarnings("unchecked")
public class ArrayShell {



    public static String[] actualToArray(ArrayList<Project> sortedList) {
        String[] newString = new String[sortedList.size()];
        for(int i = 0; i < sortedList.size(); i++) {
            newString[i] = sortedList.get(i).getProjectName();
        }

        return newString;
    }

    public static ArrayList sortByName(ArrayList<Project> projectList) {
        ArrayList<Project> newArrayList = new ArrayList<>();
        for (int i = 0; i < projectList.size(); i++) {
            newArrayList.add(i, projectList.get(i));
        }
        if (newArrayList.size() > 1) {
            char[] letters = new char[projectList.size()];
            for (int i = 0; i < newArrayList.size(); i++) {
                letters[i] = (newArrayList.get(i).getProjectName().charAt(0));
            }
            Arrays.sort(letters);
            for (int i = 0; i < newArrayList.size(); i++) {
                for (int j = newArrayList.size() - 1; j > 1; j--) {
                    if (letters[i] == newArrayList.get(j).getProjectName().charAt(0)) {
                    Project temp = newArrayList.get(i);
                    newArrayList.set(i, newArrayList.get(j));
                    newArrayList.set(j, temp);
                    }
                }
            }
        }
        return newArrayList;
    }

    public static ArrayList sortArrayByDateReq(ArrayList<Project> projectList) {
        ArrayList<Project> newArrayList = new ArrayList<>();
        for (int i = 0; i < projectList.size(); i++) {
            newArrayList.add(i, projectList.get(i));
        }
        if(newArrayList.size() > 1) {
            sortArrayByDateDue(newArrayList);
            int tempDate = 0;
            int tempDate2 = 0;
            for(int i = 0; i < newArrayList.size() - 1; i++) {
                tempDate = Integer.parseInt(newArrayList.get(i).getNumDate()) -
(Integer.parseInt(newArrayList.get(i).getDaysReq()));
                tempDate2 = Integer.parseInt(newArrayList.get(i + 1).getNumDate()) -
(Integer.parseInt(newArrayList.get(i + 1).getDaysReq()));
                    if (tempDate > 0) {
```

```
                        if (tempDate > tempDate2) {
                            Project temp = newArrayList.get(i);
                            newArrayList.set(i, newArrayList.get(i + 1));
                            newArrayList.set(i + 1, temp);
                        }
                    }
                    if (tempDate <= 0) {
                        if(tempDate > tempDate2){
                            Project temp = newArrayList.get(i);
                            newArrayList.set(i, newArrayList.get(i + 1));
                            newArrayList.set(i + 1, temp);
                        }
                    }
                }
            }


        return newArrayList;
    }

    public static ArrayList sortArrayByDateDue(ArrayList<Project> projectList) {
        ArrayList<Project> newArrayList = new ArrayList<>();
        for (int i = 0; i < projectList.size(); i++) {
            newArrayList.add(i, projectList.get(i));
        }
        if (newArrayList.size() > 1) {
            for (int i = 0; i < newArrayList.size(); i++) {
                for (int j = newArrayList.size() - 1; j > i; j--) {
                    if (Integer.parseInt(newArrayList.get(i).getMonth()) >
Integer.parseInt(newArrayList.get(j).getMonth())) {
                        Project temp = newArrayList.get(i);
                        newArrayList.set(i, newArrayList.get(j));
                        newArrayList.set(j, temp);
                    }
                    if (Integer.parseInt(newArrayList.get(i).getMonth()) ==
Integer.parseInt(newArrayList.get(j).getMonth())) {
                        if(Integer.parseInt(newArrayList.get(i).getNumDate()) >
Integer.parseInt(newArrayList.get(j).getNumDate())){
                            Project temp = newArrayList.get(i);
                            newArrayList.set(i, newArrayList.get(j));
                            newArrayList.set(j, temp);
                        }
                    }
                }
            }
        }
        return newArrayList;
    }



    public static void printArray(ArrayList<Project> projectList) {
        for(int i = 0; i < projectList.size(); i++) {
            System.out.println(" this is the printArray method running " +
projectList.get(i).getProjectName() + " " + projectList.get(i).getMonth() + " ");
```

```
            }
        }
}


}
```

SpringUtilities Class (Provided by Oracle):

```java
/*
 * Copyright (c) 1995, 2008, Oracle and/or its affiliates. All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *   - Redistributions of source code must retain the above copyright
 *     notice, this list of conditions and the following disclaimer.
 *
 *   - Redistributions in binary form must reproduce the above copyright
 *     notice, this list of conditions and the following disclaimer in the
 *     documentation and/or other materials provided with the distribution.
 *
 *   - Neither the name of Oracle or the names of its
 *     contributors may be used to endorse or promote products derived
 *     from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
 * IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

package layout;

import javax.swing.*;
import javax.swing.SpringLayout;
import java.awt.*;

/**
 * A 1.4 file that provides utility methods for
 * creating form- or grid-style layouts with SpringLayout.
 * These utilities are used by several programs, such as
 * SpringBox and SpringCompactGrid.
 */
public class SpringUtilities {
    /**
     * A debugging utility that prints to stdout the component's
     * minimum, preferred, and maximum sizes.
     */
    public static void printSizes(Component c) {
        System.out.println("minimumSize = " + c.getMinimumSize());
        System.out.println("preferredSize = " + c.getPreferredSize());
        System.out.println("maximumSize = " + c.getMaximumSize());
    }

    /**
```

```
     * Aligns the first <code>rows</code> * <code>cols</code>
     * components of <code>parent</code> in
     * a grid. Each component is as big as the maximum
     * preferred width and height of the components.
     * The parent is made just big enough to fit them all.
     *
     * @param rows number of rows
     * @param cols number of columns
     * @param initialX x location to start the grid at
     * @param initialY y location to start the grid at
     * @param xPad x padding between cells
     * @param yPad y padding between cells
     */
    public static void makeGrid(Container parent,
                                int rows, int cols,
                                int initialX, int initialY,
                                int xPad, int yPad) {
        SpringLayout layout;
        try {
            layout = (SpringLayout)parent.getLayout();
        } catch (ClassCastException exc) {
            System.err.println("The first argument to makeGrid must use SpringLayout.");
            return;
        }

        Spring xPadSpring = Spring.constant(xPad);
        Spring yPadSpring = Spring.constant(yPad);
        Spring initialXSpring = Spring.constant(initialX);
        Spring initialYSpring = Spring.constant(initialY);
        int max = rows * cols;

        //Calculate Springs that are the max of the width/height so that all
        //cells have the same size.
        Spring maxWidthSpring = layout.getConstraints(parent.getComponent(0)).
                getWidth();
        Spring maxHeightSpring = layout.getConstraints(parent.getComponent(0)).
                getHeight();
        for (int i = 1; i < max; i++) {
            SpringLayout.Constraints cons = layout.getConstraints(
                    parent.getComponent(i));

            maxWidthSpring = Spring.max(maxWidthSpring, cons.getWidth());
            maxHeightSpring = Spring.max(maxHeightSpring, cons.getHeight());
        }

        //Apply the new width/height Spring. This forces all the
        //components to have the same size.
        for (int i = 0; i < max; i++) {
            SpringLayout.Constraints cons = layout.getConstraints(
                    parent.getComponent(i));

            cons.setWidth(maxWidthSpring);
            cons.setHeight(maxHeightSpring);
        }
```

```
        //Then adjust the x/y constraints of all the cells so that they
        //are aligned in a grid.
        SpringLayout.Constraints lastCons = null;
        SpringLayout.Constraints lastRowCons = null;
        for (int i = 0; i < max; i++) {
            SpringLayout.Constraints cons = layout.getConstraints(
                    parent.getComponent(i));
            if (i % cols == 0) { //start of new row
                lastRowCons = lastCons;
                cons.setX(initialXSpring);
            } else { //x position depends on previous component
                cons.setX(Spring.sum(lastCons.getConstraint(SpringLayout.EAST),
                        xPadSpring));
            }

            if (i / cols == 0) { //first row
                cons.setY(initialYSpring);
            } else { //y position depends on previous row
                cons.setY(Spring.sum(lastRowCons.getConstraint(SpringLayout.SOUTH),
                        yPadSpring));
            }
            lastCons = cons;
        }

        //Set the parent's size.
        SpringLayout.Constraints pCons = layout.getConstraints(parent);
        pCons.setConstraint(SpringLayout.SOUTH,
                Spring.sum(
                        Spring.constant(yPad),
                        lastCons.getConstraint(SpringLayout.SOUTH)));
        pCons.setConstraint(SpringLayout.EAST,
                Spring.sum(
                        Spring.constant(xPad),
                        lastCons.getConstraint(SpringLayout.EAST)));
    }

    /* Used by makeCompactGrid. */
    private static SpringLayout.Constraints getConstraintsForCell(
            int row, int col,
            Container parent,
            int cols) {
        SpringLayout layout = (SpringLayout) parent.getLayout();
        Component c = parent.getComponent(row * cols + col);
        return layout.getConstraints(c);
    }

    /**
     * Aligns the first <code>rows</code> * <code>cols</code>
     * components of <code>parent</code> in
     * a grid. Each component in a column is as wide as the maximum
     * preferred width of the components in that column;
     * height is similarly determined for each row.
     * The parent is made just big enough to fit them all.
     *
     * @param rows number of rows
```

```
     * @param cols number of columns
     * @param initialX x location to start the grid at
     * @param initialY y location to start the grid at
     * @param xPad x padding between cells
     * @param yPad y padding between cells
     */
    public static void makeCompactGrid(Container parent,
                                       int rows, int cols,
                                       int initialX, int initialY,
                                       int xPad, int yPad) {
        SpringLayout layout;
        try {
            layout = (SpringLayout)parent.getLayout();
        } catch (ClassCastException exc) {
            System.err.println("The first argument to makeCompactGrid must use
SpringLayout.");
            return;
        }

        //Align all cells in each column and make them the same width.
        Spring x = Spring.constant(initialX);
        for (int c = 0; c < cols; c++) {
            Spring width = Spring.constant(0);
            for (int r = 0; r < rows; r++) {
                width = Spring.max(width,
                        getConstraintsForCell(r, c, parent, cols).
                                getWidth());
            }
            for (int r = 0; r < rows; r++) {
                SpringLayout.Constraints constraints =
                        getConstraintsForCell(r, c, parent, cols);
                constraints.setX(x);
                constraints.setWidth(width);
            }
            x = Spring.sum(x, Spring.sum(width, Spring.constant(xPad)));
        }

        //Align all cells in each row and make them the same height.
        Spring y = Spring.constant(initialY);
        for (int r = 0; r < rows; r++) {
            Spring height = Spring.constant(0);
            for (int c = 0; c < cols; c++) {
                height = Spring.max(height,
                        getConstraintsForCell(r, c, parent, cols).
                                getHeight());
            }
            for (int c = 0; c < cols; c++) {
                SpringLayout.Constraints constraints =
                        getConstraintsForCell(r, c, parent, cols);
                constraints.setY(y);
                constraints.setHeight(height);
            }
            y = Spring.sum(y, Spring.sum(height, Spring.constant(yPad)));
        }
```

```
        //Set the parent's size.
        SpringLayout.Constraints pCons = layout.getConstraints(parent);
        pCons.setConstraint(SpringLayout.SOUTH, y);
        pCons.setConstraint(SpringLayout.EAST, x);


    }
}
```

Main Class:

```
import javax.swing.*;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import java.awt.*;
import java.awt.event.*;
import java.util.ArrayList;

import static java.awt.Font.PLAIN;
import static javax.swing.ListSelectionModel.SINGLE_SELECTION;


public class Main {
    public static JTextArea homeText2;
    public static JList <String> dateSortedList;
    public static JList <String> numSortedList;
    public static JList <String> reqSortedList;
    private static DefaultListModel<String> listModelDate;
    private static DefaultListModel<String> listModelNum;
    private static DefaultListModel<String> listModelReq;
    public static JScrollPane dateSort = new JScrollPane();
    public static JScrollPane numSort = new JScrollPane();
    public static JScrollPane reqSort = new JScrollPane();
    public static ArrayList<Project> dateSorted;
    public static ArrayList<Project> numSorted;
    public static ArrayList<Project> reqSorted;
    public static ArrayList<Project> projectList;

    /*.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e){

            }
       });
       */

@SuppressWarnings("unchecked")
    public static void main(String[] args) {


    Font font = new Font("Dialog", PLAIN, 12);

    Dimension max = new Dimension(30, 30);
    projectList = new ArrayList<>();
    dateSorted = ArrayShell.sortArrayByDateDue(projectList);
    numSorted = ArrayShell.sortByName(projectList);
    reqSorted = ArrayShell.sortArrayByDateReq(projectList);
    homeText2 = new JTextArea("You currently have " + projectList.size() + " projects.
Wow!");
    dateSortedList = new JList(ArrayShell.actualToArray(dateSorted));
    numSortedList = new JList(ArrayShell.actualToArray(numSorted));
    reqSortedList = new JList(ArrayShell.actualToArray(reqSorted));
    dateSortedList.setMaximumSize(max);
    dateSortedList.setMinimumSize(max);
    numSortedList.setMaximumSize(max);
    numSortedList.setMinimumSize(max);
    reqSortedList.setMaximumSize(max);
```

```
    reqSortedList.setMaximumSize(max);
    listModelDate = new DefaultListModel();
    listModelNum = new DefaultListModel();
    listModelReq = new DefaultListModel();



    //the Frame Depository
    JFrame schule = Helper.createFrame("Schule", 300, 300);
    JFrame manager = Helper.createFrame("Manager", 300, 300);
    JFrame schedule = Helper.createFrame("Schedule", 300, 300);
    String[] managerLabels = {"Name", "Course", "#DaysReq", "#Date", "#Month",
"Description"};


    //buttons for schule
    JButton schuleManager = Helper.createButton("Manager");
    schuleManager.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            schule.setVisible(false);
            manager.setVisible(true);
        }
    });
    JButton schuleSchedule = Helper.createButton("Schedule");
    schuleSchedule.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            schule.setVisible(false);
            schedule.setVisible(true);
        }
    });


    //putting the "buttons" in "schuleButtons ;)"
    JPanel schulePanel = new JPanel();
    schulePanel.add(schuleManager);
    schulePanel.add(schuleSchedule);
    schule.add(schulePanel);
    schule.setVisible(true);


    //  manager panel
    JPanel managerPanel = new JPanel();
    managerPanel.setLayout(new SpringLayout());
    JPanel managerPanel2 = new JPanel();
    managerPanel2.setLayout(new BorderLayout());
    JTextField textField;
    JTextField[] actualTextFields = new JTextField[6];
    for (int i = 0; i < managerLabels.length; i++) {
        JLabel managerLabel = new JLabel(managerLabels[i], JLabel.TRAILING);
        managerPanel.add(managerLabel);
        textField = new JTextField();
        actualTextFields[i] = textField;
        managerLabel.setLabelFor(textField);
        managerPanel.add(textField);
    }
```

```
    //spring utilities junk
    layout.SpringUtilities.makeCompactGrid(managerPanel, 6, 2, 6, 6, 6, 6);
    manager.add(managerPanel);
    JButton managerSubmit = Helper.createButton("Submit");
    managerSubmit.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            Project newProject;
            newProject = new Project(actualTextFields[0].getText(),
actualTextFields[1].getText(), actualTextFields[2].getText(),
actualTextFields[3].getText(), actualTextFields[4].getText(),
actualTextFields[5].getText());
            projectList.add(newProject);
            dateSorted = ArrayShell.sortArrayByDateDue(projectList);
            numSorted = ArrayShell.sortByName(projectList);
            reqSorted = ArrayShell.sortArrayByDateReq(projectList);
            homeText2.setText("You currently have " + projectList.size() + " projects.
Wow!");
            manager.setVisible(false);
            schule.setVisible(true);
            listModelDate = new DefaultListModel();
            listModelNum = new DefaultListModel();
            listModelReq = new DefaultListModel();
            for(int i =0; i < projectList.size(); i++){
                listModelDate.addElement(dateSorted.get(i).getProjectName());
                listModelNum.addElement(numSorted.get(i).getProjectName());
                listModelReq.addElement(reqSorted.get(i).getProjectName());
            }


            dateSortedList = new JList<String>(listModelDate);
            numSortedList = new JList<String>(listModelNum);
            reqSortedList = new JList<String>(listModelReq);
            dateSort.setViewportView(dateSortedList);
            numSort.setViewportView(numSortedList);
            reqSort.setViewportView(reqSortedList);
            for (int i = 0; i < actualTextFields.length; i++) {
                actualTextFields[i].setText("");
            }
        }
    });
    manager.add(managerSubmit, BorderLayout.PAGE_END);


    //schedule
    ChangeListener changeListener = new ChangeListener() {
        public void stateChanged(ChangeEvent changeEvent) {
            JTabbedPane sourceTabbedPane = (JTabbedPane) changeEvent.getSource();
            listModelDate = new DefaultListModel();
            listModelNum = new DefaultListModel();
            listModelReq = new DefaultListModel();
            for(int i =0; i < projectList.size(); i++){
                listModelDate.addElement(dateSorted.get(i).getProjectName());
                listModelNum.addElement(numSorted.get(i).getProjectName());
```

```
                listModelReq.addElement(reqSorted.get(i).getProjectName());
            }

        }
    };
    JTabbedPane schedulePane = new JTabbedPane();
    schedule.add(schedulePane);
    schedulePane.addChangeListener(changeListener);


    //home tab
    JPanel home = new JPanel();
    JTextArea homeText = new JTextArea("Click on a tab to get started.");
    homeText.setEditable(false);


    homeText2.setEditable(false);
    home.add(homeText2, BorderLayout.CENTER);

    //little array of surprise text or ascii art to be randomly generated here in free time
    home.add(homeText, BorderLayout.PAGE_START);

    schedulePane.addTab("Home", home);
    JButton homeReturn = Helper.createButton("Back");
    home.add(homeReturn, BorderLayout.PAGE_END);
    homeReturn.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            schule.setVisible(true);
            schedule.setVisible(false);
        }
    });



    //.removeElement(Object obj);


    JButton view1 = Helper.createButton("View");
    view1.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            JFrame dialogue = new JFrame("Assignment Details");
            dialogue.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
            dialogue.setSize(300, 300);
            dialogue.setLocationRelativeTo(null);
            dialogue.setVisible(true);
            dialogue.setLayout(new BorderLayout());


            int dateIndex = dateSortedList.getSelectedIndex();

            JLabel l1 = Helper.createLabel("Project Name: " +
dateSorted.get(dateIndex).getProjectName() + ".");
            JLabel l2 = Helper.createLabel(" Course: " +
dateSorted.get(dateIndex).getCourse() + ".");
            JLabel l3 = Helper.createLabel(" Days Required: " +
```

```
dateSorted.get(dateIndex).getDaysReq() + ".");
            JLabel l4 = Helper.createLabel(" Date Due: " +
dateSorted.get(dateIndex).getNumDate() + ".");
            JLabel l5 = Helper.createLabel(" Month Due: " +
dateSorted.get(dateIndex).getMonth() + ".");
            JLabel l6 = Helper.createLabel(" Description: " +
dateSorted.get(dateIndex).getDescription() + ".");
            JPanel labelPanel = new JPanel();
            labelPanel.setLayout(new FlowLayout());
            labelPanel.add(l1);
            labelPanel.add(l2);
            labelPanel.add(l3);
            labelPanel.add(l4);
            labelPanel.add(l5);
            labelPanel.add(l6);
            dialogue.add(labelPanel);

        }
    });

    JPanel buttons1 = new JPanel();
    buttons1.setLayout(new FlowLayout());



    dateSort.setLayout( new ScrollPaneLayout());
    dateSort.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
    dateSortedList.setSelectionMode(SINGLE_SELECTION);
    dateSort.getViewport().setView(dateSortedList);
    dateSortedList.setFont(font);
    buttons1.add(dateSort);
    dateSort.setPreferredSize(new Dimension(280,175));
    buttons1.add(view1);
    schedulePane.add(buttons1);
    schedulePane.addTab("By Date Due", buttons1);


    JButton view2 = Helper.createButton("View");
    view2.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            JFrame dialogue = new JFrame("Assignment Details");
            dialogue.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
            dialogue.setSize(300, 300);
            dialogue.setLocationRelativeTo(null);
            dialogue.setVisible(true);
            dialogue.setLayout(new BorderLayout());


            int numIndex = dateSortedList.getSelectedIndex();

            JLabel l1 = Helper.createLabel("Project Name: " +
numSorted.get(numIndex).getProjectName() + ".");
            JLabel l2 = Helper.createLabel(" Course: " +
numSorted.get(numIndex).getCourse() + ".");
            JLabel l3 = Helper.createLabel(" Days Required: " +
```

```
numSorted.get(numIndex).getDaysReq() + ".");
            JLabel l4 = Helper.createLabel(" Date Due: " +
numSorted.get(numIndex).getNumDue() + ".");
            JLabel l5 = Helper.createLabel(" Month Due: " +
numSorted.get(numIndex).getMonth() + ".");
            JLabel l6 = Helper.createLabel(" Description: " +
numSorted.get(numIndex).getDescription() + ".");
            JPanel labelPanel = new JPanel();
            labelPanel.setLayout(new FlowLayout());
            labelPanel.add(l1);
            labelPanel.add(l2);
            labelPanel.add(l3);
            labelPanel.add(l4);
            labelPanel.add(l5);
            labelPanel.add(l6);
            dialogue.add(labelPanel);

        }
    });

    JPanel buttons2 = new JPanel();
    buttons2.setLayout(new FlowLayout());



    numSort.setLayout( new ScrollPaneLayout());
    numSort.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
    numSortedList.setSelectionMode(SINGLE_SELECTION);
    numSort.getViewport().setView(numSortedList);
    numSortedList.setFont(font);
    buttons2.add(numSort);
    numSort.setPreferredSize(new Dimension(280,175));
    buttons2.add(view2);
    schedulePane.add(buttons2);
    schedulePane.addTab("By Alphabetical", buttons2);


    JButton view3 = Helper.createButton("View");
    view3.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            JFrame dialogue = new JFrame("Assignment Details");
            dialogue.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
            dialogue.setSize(300, 300);
            dialogue.setLocationRelativeTo(null);
            dialogue.setVisible(true);
            dialogue.setLayout(new BorderLayout());


            int reqIndex = reqSortedList.getSelectedIndex();


            JLabel l1 = Helper.createLabel("Project Name: " +
reqSorted.get(reqIndex).getProjectName() + ".");
            JLabel l2 = Helper.createLabel(" Course: " +
reqSorted.get(reqIndex).getCourse() + ".");
```

```
            JLabel l3 = Helper.createLabel(" Days Required: " +
reqSorted.get(reqIndex).getDaysReq() + ".");
            JLabel l4 = Helper.createLabel(" Date Due: " +
reqSorted.get(reqIndex).getNumDue() + ".");
            JLabel l5 = Helper.createLabel(" Month Due: " +
reqSorted.get(reqIndex).getMonth() + ".");
            JLabel l6 = Helper.createLabel(" Description: " +
reqSorted.get(reqIndex).getDescription() + ".");
            JPanel labelPanel = new JPanel();
            labelPanel.setLayout(new FlowLayout());
            labelPanel.add(l1);
            labelPanel.add(l2);
            labelPanel.add(l3);
            labelPanel.add(l4);
            labelPanel.add(l5);
            labelPanel.add(l6);
            dialogue.add(labelPanel);

        }
    });

    JPanel buttons3 = new JPanel();
    buttons3.setLayout(new FlowLayout());



    reqSort.setLayout( new ScrollPaneLayout());
    reqSort.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
    reqSortedList.setSelectionMode(SINGLE_SELECTION);
    reqSort.getViewport().setView(reqSortedList);
    reqSortedList.setFont(font);
    buttons3.add(reqSort);
    reqSort.setPreferredSize(new Dimension(280,175));
    buttons3.add(view3);
    schedulePane.add(buttons3);
    schedulePane.addTab("By Required Date", buttons3);




    schedulePane.setVisible(true);

    }
}
```