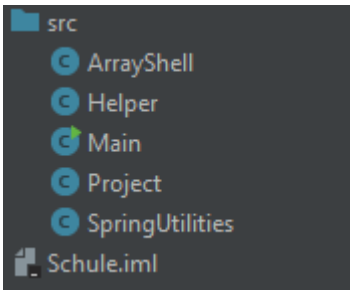
Criterion C: Development

All Classes:



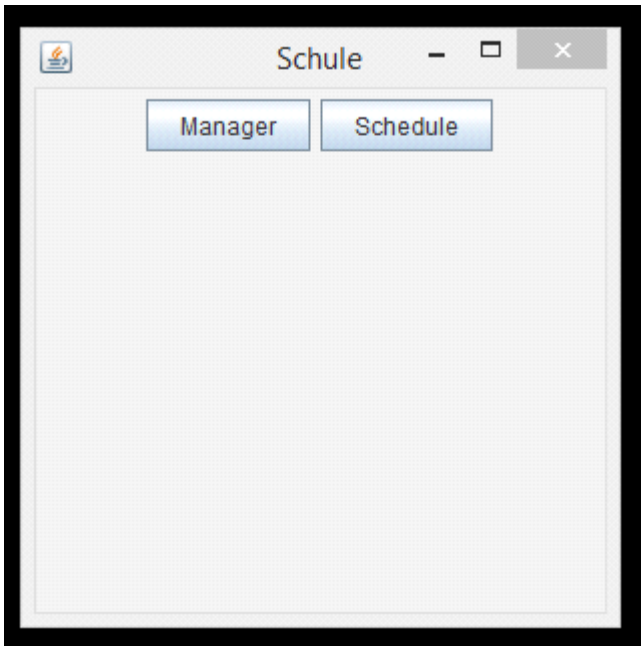
List of techniques:

- Project is the main "object" that represents assignments, and is used in the Main class whenever input is received
- Helper class contains methods to quickly create Swing components
- ArrayShell contains helper methods to sort arraylists, as well as a custom to-array method and a print array method for testing
- Includes JLists, ArrayLists, and ListModels
- Static methods and variables

Program File:

HELP

Main Window:



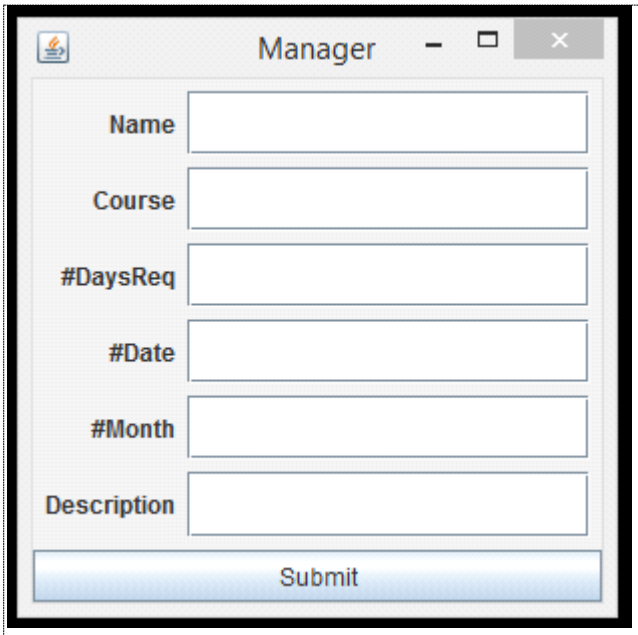
Using pre-written methods to quickly create Swing components, such as the main JFrame, with custom settings.

```
JFrame schule = Helper.createFrame("Schule", 300, 300);

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);
    }
});

JPanel schulePanel = new JPanel();
schulePanel.add(schuleManager);
schulePanel.add(schuleSchedule);
schule.add(schulePanel);
schule.setVisible(true);
```

The exit button terminates the program.

Upon clicking the "Manager" button, initial Schule JFrame dissappears. JFrame Manager will pop up, and allow for user input using a GUI constructed by the SpringUtilities class, distributed by Oracle and available for use.

Manager Window:



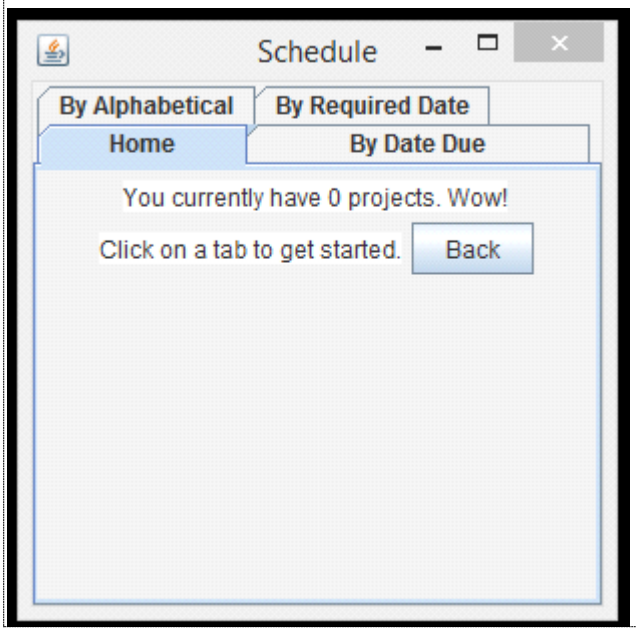

```
JPanel managerPanel = new JPanel();
managerPanel.setLayout(new SpringLayout());
JPanel managerPanel2 = new JPanel();
managerPanel2.setLayout(new BorderLayout());
JTextField textField;
JTextField[] actualTextFields = new JTextField[6];
for (int i = 0; i < managerLabels.length; i++) {
    JLabel managerLabel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

layout.SpringUtilities.makeCompactGrid(managerPanel, 6, 2, 6, 6, 6, 6);
manager.add(managerPanel);
```

The JButton Submit's actionListener will launch the various sort methods, in the event that the user will then want to see them sorted in the "Schedule" window. This button will bring the user back to the main JFrame Schule, and make this frame invisible again. Clicking Submit will also create a new Project object based on user entries.

The exit button terminates the program.

```
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);
```

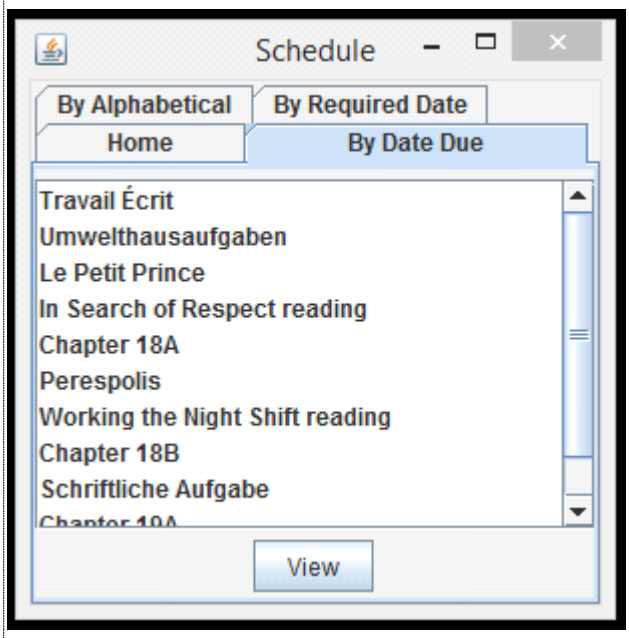
Schedule Window:



Upon clicking the "Schedule" button, initial Schule JFrame dissappears. JFrame Schedule will pop up, and allow users to see their current Project count. This label updates with the Project count. The "Back" button will hide the Schedule JFrame and and make the main Schule JFrame visible again. Each tabbedPane contains a changeListener, so that when a user clicks on a tab, the following code will be run:

```
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());
        }

    }
};
```

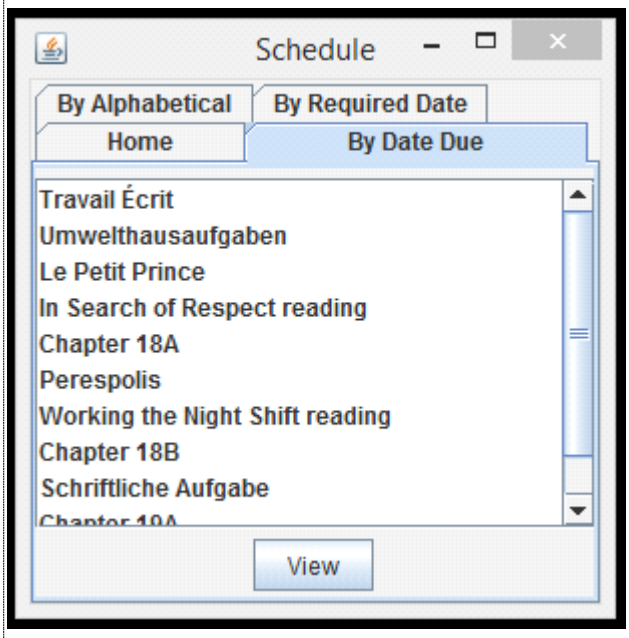
The tabbedPanes By Alphabetical, By Required Date, and By Date Due are identically set up, but they will display the Projects in different orders (relative to the sort used). They look like this when populated, and feature a functional vertical scrollbar.

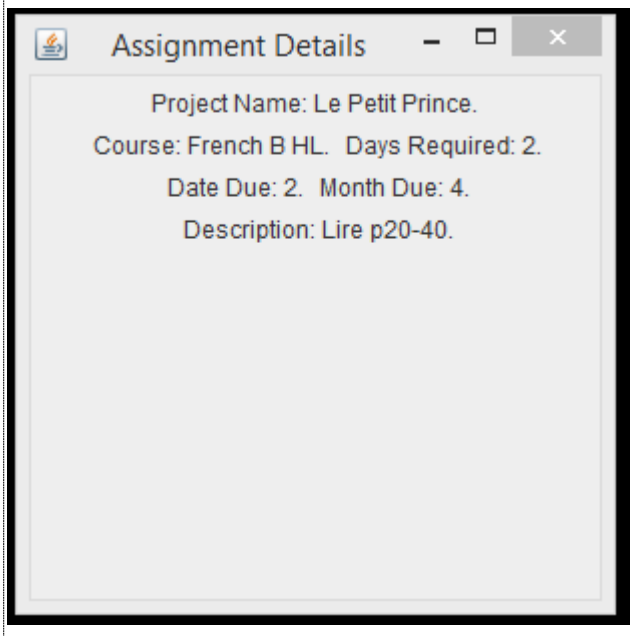



```
dateSort.setLayout( new ScrollPaneLayout());
dateSort.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
dateSortedList.setSelectionMode(SINGLE_SELECTION);
dateSort.getViewport().setView(dateSortedList);
dateSortedList.setFont(font);
buttons1.add(dateSort);
dateSort.setPreferredSize(new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

The exit button terminates the program.

The lists show only the name of the Project. Upon selecting an element, and clicking the "View" button, a window will then pop up to provide more information on the assignment. The exit button on this JFrame, Assignment Details, does not terminate the program. Instead, it closes the JFrame Assignment Details. Also, the parent JFrame Schedule is not hidden while Assignment Details is visible.





```
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).getDaysReq() + ".");
        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);
    }
});
```

Methods Used from Helper Class:

The following method from class Helper will create a JFrame when called, using the passed values title, width, and height, to quickly create a JFrame with these parameters.

```
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;
}
```

The following method from class Helper will create a JLabel when called, using the passed value text, to quickly create a JLabel with a custom Font.

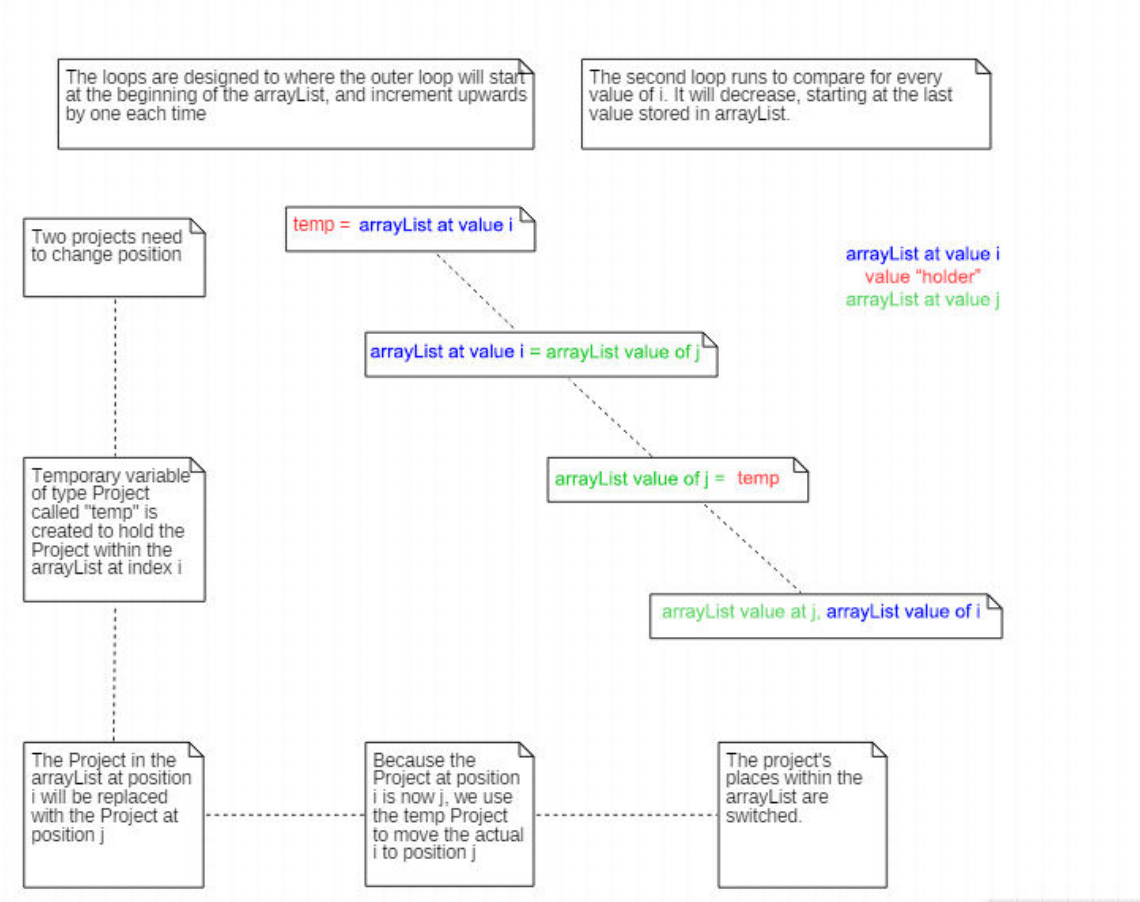
```
public static JLabel createLabel(String text){
    Font font = new Font("Dialog", PLAIN, 12);
    JLabel newLabel = new JLabel(text);
    newLabel.setFont(font);
    return newLabel;
}
```

The following method from class Helper will create a JButton when called, using the passed value title, to quickly create a JButton with a custom Font.

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

}
```

Methods in ArrayShell (Including Sorts):

Sorting/replacement explanation diagram:



The following method from class ArrayShell is used to pass to the JLists seen in the tabbedPanes the title of the projects.

```
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;
}
```

The following method from class ArrayShell was used for testing to ensure that ArrayLists that have been sorted were actually sorted and useable.

```
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() + " ");
    }
}
```

The following method from class ArrayShell is used to sort assignments by alphabetical order. By converting the initial ArrayList into a char[], I am able to use the Arrays.sort function in order to sort the ArrayList alphabetically. After sorting using Arrays.sort, I convert the now sorted characters back into their full ArrayList. While this functions properly, it can lead to issues when two elements in the ArrayList have the same initial character for their respective ProjectName field.

```
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;
}
```

The following method from class ArrayShell is used to sort assignments according to when they should be started. To clarify, each Project has a due date, stored as numDate, and the amount of time finishing the project is expected to take, daysReq. This method will sort Projects by the time they should be started, in order, to be completed. It also considers each project's respective month value as well.

```
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;
}
```

The following method from class ArrayShell is used to sort assignments according to their due date. This method will sort Projects by the times they are due, according to month. If two Projects have the same month, they will then be sorted by their numDate.

```
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;
}
```

Word Count: 658 (1,490 including code)