Swing and MVC

To simplify these problems, I have made a class in my swing-mvc-exercises project called JListBase. By using JListBase, all you have to do is to create a JList, and the code will automatically create and pop up a window containing your JList. Copy JListBase and WindowUtilities to your new project as a starting point. Make a subclass of JListBase and implement these methods:

- **makeJList.** Have this return the JList you want. This is the *only* method you need to override for problems 1, 2, 4, and 5. Note that the base code takes this return value and puts it into a protected instance variable call jList.
- **addStufftoListPanel.** Have this call "add" on the JPanel argument, if you want to add something extra to the top window. For problem 3, you will use this method to make a JButton, attach a listener, and then call listPanel.add(yourNewJButton). But if all you want is a JList and nothing else, ignore this method totally.

Once you have done this, just make a "main" method that instantiates your class. For example, here is a simple program that pops up a window that contains a list showing some names.

```
public class JListTest extends JListBase {
  @Override
  protected JList makeJList() {
     // In your code, uou have to write this part to create a JList
    String[] names = { ”Joe”, ”Jane”, ”John”, ”Juan”, ”Jean” };
    JList nameList = new JList(names);
    return(nameList);
  }

  public static void main(String[] args) {
    new JListTest();
  }
}
```

1. Make an Employee class that stores a first name, last name, and salary. Make an array of Employee objects. Make a JList that shows the names: Specifically, make an array of Strings by loopingdown the list of Employees and looking up the name, then display those Strings in a JList.

2. Repeat the previous problem, but this time don't make an array of Strings first. (Hint: give your class a toString method and note that you can pass an Object[] (e.g., the Employee[]) to the JList constructor.)

3. Add a push button that, when pressed, pops up a dialog box showing the salary of the currently selected name. Hint1: call listBox.getSelectedValue() and cast the result to Employee. Hint2: assuming "this" refers to the subclass of JListBase, pop up a dialog box with JOptionPane.showMessageDialog(this, someMessage).

4. Make a List<Employee>. Implement the ListModel interface in order to put the employees into a JList. (Hint: as in the class example, you can have empty bodies for addListDataListener and removeListDataListener).