1 Objectives

There are several objectives for this project. First, you will gain experience designing a medium-scale object-oriented system. You will have significant flexibility in how you choose to realize your implementation. Second, you will gain experience using some of the most important components within the Java API, including classes within the Swing library. Third, you will gain experience applying the Singleton pattern, the Strategy pattern, and the Observer pattern. Fourth, you will gain experience using generic types. Finally, you will gain experience working with JAR-based class libraries.

It is worth emphasizing that a—perhaps the—key objective for this project is to give you a chance to refine your basic design skills. Please take advantage of this. It is one of the few opportunities where someone will review your source code in detail and offer friendly advice.

2 Overview

For this assignment, you will develop a basic email client using the Java programming language, the Swing API, and the JavaMail API. When you have completed the assignment, you will have a fully functional system that you can use to email your friends and impress your potential employers. You should take the latter point seriously: Several students have used their implementations to secure summer internships.

Your email client will provide three sets of features. The most basic set of features will allow a user to configure the email client as appropriate to their system and network. The second set of features will allow a user to manage an email contact database to store information about those contacts that are emailed most frequently. The last set of features will allow a user to send email to contacts in their database.

3 Requirements

An important goal of this assignment is for you to gain some design experience. Hence, you will not receive a complete description of the system design. You have significant flexibility in how you complete the project. Your system must, however, satisfy several requirements. The subsections that follow describe the key system features, user interface elements, and system classes that your design must include.
3.1 The Configuration Class

The Configuration class will be a data class that maintains information about the user’s system configuration. At a minimum, your class must include support for storing the primary user’s email address and the IP address of their outgoing SMTP server. The class will provide getter and setter methods for accessing each configuration attribute. To simplify data storage and retrieval, your Configuration class will implement Serializable.

3.2 The Contact Class

The Contact class will be a data class that maintains information about a single email contact. At a minimum, your class must include support for storing a contact name, a postal address, a phone number, and an email address. The class will provide getter and setter methods for accessing each contact attribute. To simplify data storage and retrieval, your Contact class will implement Serializable.

3.3 The DataStore Class

The DataStore class will serve two purposes. First, the class will encapsulate all of the logic required to store and retrieve configuration and contact data from disk. The class will provide methods for loading configuration and contact data, and methods for storing configuration and contact data. The first set of methods will be used at system startup, and the second set of methods will be used at system shutdown. The storage and retrieval methods will throw (but not catch) appropriate exceptions if input or output problems are encountered.

The second purpose of the DataStore class is to provide access to one Configuration object and one collection of Contact objects. (You may choose any collection realization that you wish (e.g., Vector, HashSet).) The class will provide appropriate getter methods to access these elements.

To ensure that exactly one DataStore object is constructed, and to make this object globally accessible, you will apply the Singleton pattern.

3.4 The MainFrame Class

The MainFrame class will subclass JFrame to implement the main window of your application. The window must include the following user interface elements and support the following features.

- The window will use an appropriately selected icon — the cornerstone of a well-designed user interface! (Note that this icon may not show on all operating systems.)
- The window will include a menu-bar that mirrors the standard Windows menu-bar. The menu-bar will include a File menu, a Configuration menu, and a Help menu. The File menu will include an Exit menu-item that terminates the application. The Configuration menu will include a Configure menu-item that displays the configuration window (see below). The Help menu will include an About menu-item that displays a standard system information window (see below).
- The main content pane of your window will display a detail view of the contacts in your contact database. To implement this view, you will use the JTable class provided by the
Swing API. Your table will include appropriate column headers and will allow the user to scroll horizontally and vertically. The data displayed within this table must remain consistent with the information stored by the DataStore object. For a good tutorial on how to use the JTable class, have a look at the following website:
http://java.sun.com/docs/books/tutorial/uiswing/components/table.html

- When a user double-clicks on one of the rows within the detail table, the main window will display the email transmission window (see below). The destination address displayed in the transmission window will be based on the selected row in the detail table.
- Your content pane will include a row of three buttons arranged horizontally just beneath your table. These buttons will be labeled Add, Edit, and Delete. When the Add or Edit buttons are clicked, the contact editing window will be displayed. In the case of Edit, the text fields within the editing window will be populated based on the row selected within the detail table. In the case of Add, the fields will be empty. When the user clicks the Delete button, the user will be prompted to determine whether they really want to delete the selected row. If they agree, the row will be deleted from the DataStore (and the detail table). These buttons should only be enabled when they are applicable. The Delete button, for example, should be disabled when there are no rows selected within the table. You will apply the Mediator pattern to localize the interaction logic within the MainFrame class.

3.5 The ConfigurationDlg Class

The ConfigurationDlg class will subclass JDialog to implement the configuration window of your application. The purpose of the window is to allow the user to update the system configuration parameters. The window must include the following user interface elements and support the following features.

- The window will behave as a modal dialog. It will not allow focus to reach any other window until it is closed.
- The window will include appropriately labeled text fields corresponding to each of the configuration attributes in the system Configuration object. The fields will initially be populated based on the data stored by this object. The user will be allowed to modify the text fields.
- The window will include two buttons at the bottom of the form. These buttons will be labeled Save and Cancel. If the user clicks the Save button, the text fields will be validated (to ensure syntactic correctness), and the DataStore will be appropriately updated. (Validation will check, for example, that a reasonable SMTP address has been entered, and generate an error message if one has not.) If the user clicks Cancel, the DataStore will not be updated.

Note that this window will not trigger reads or writes to the disk. The contents of the DataStore object will be retrieved and stored from disk only at system startup and shutdown, respectively.

3.6 The SystemInformationDlg Class

The SystemInformationDlg class will subclass JDialog to implement the system information window of your application. The purpose of the window is to display basic information about the application.
and its designers. Please have a look at the Help→About Eclipse SDK menu-item in Eclipse as an example. The window will behave as a modal dialog. It will not allow focus to reach any other window until it is closed.

### 3.7 The ContactEditingDlg Class

The ContactEditingDlg class will subclass JDialog to implement the contact editing window of your application. The purpose of the window is to allow the user to update an existing contact in the DataStore, or to add a new contact. The window must include the following user interface elements and support the following features.

- The window will behave as a modal dialog. It will not allow focus to reach any other window until it is closed.
- The window will include appropriately labeled text fields corresponding to each of the contact attributes. As mentioned previously, the initialization of the text fields depends on whether a new contact is being added or an existing contact is being modified. In either case, the user will be allowed to modify the text fields.
- The window will include two buttons at the bottom of the form. These buttons will be labeled Save and Cancel. If the user clicks the Save button, the text fields will be validated (to ensure syntactic correctness), and the DataStore will be appropriately updated. (Validation will check, for example, that a reasonable email address has been entered and generate an error message if one has not.) If the user clicks Cancel, the DataStore will not be updated.

Note that this window will not trigger reads or writes to the disk. The contents of the DataStore object will be retrieved and stored from disk only at system startup and shutdown, respectively.

### 3.8 The EmailTransmissionDlg Class

The EmailTransmissionDlg class will subclass JDialog to implement the email transmission window of your application. The purpose of the window is to allow the user to draft and transmit an email to one of the contacts in their database. The window must include the following user interface elements and support the following features.

- The window will behave as a modal dialog. It will not allow focus to reach any other window until it is closed.
- The window will include appropriately labeled text fields for drafting a basic email. This includes support for (i) a source address, (ii) a list of destination addresses, (iii) a subject line, and (iv) an email body. With the exception of the source address, all of the fields will be modifiable by the user. The source address will be populated from the system Configuration object. The destination address will be populated based on the contact selected in the detail view of the main window. The remaining fields will be initially empty.
- The window will include two buttons at the bottom of the form. These buttons will be labeled Send and Cancel. If the user clicks the Send button, the text fields will be validated, and the email will be transmitted. (Validation will check, for example, that reasonable email
addresses have been provided, and generate an error message if not.) If the user clicks Cancel, the window will close, and the email will not be sent.

Note that I will provide sample source code on the course website to show you how to transmit email using the JavaMail API.

3.9 Generics

Use generic types wherever appropriate to reduce type-casting in your implementation and to reduce the likelihood of data typing errors.

4 Submission Instructions

This project is due by midnight on April 29th. Absolutely no late assignments will be accepted; there will be no time for extensions. Please start early. This is a time-consuming project.

When your project is complete, archive your source materials and use the handin command to submit your work. The command for this project is: handin.215.1 4 <filename>. You may use any standard archive format you like.

You must also slide a hard copy of your assignment under your instructor’s office door.

5 Grading

Your project will be graded based on your adherence to the specified requirements and the programming style guidelines discussed in class.

This is an intermediate course in software development. Your source materials should be properly documented. Your source must compile. Your application must not crash. A violation of any of these requirements will result in an automatic zero. Test your application thoroughly.

Note that due to the scale of this project, it will be weighted more than your previous two projects.

6 Collaboration

You may work independently or with one partner. As always, you must not discuss the problem or the solution with classmates outside of your group. Keep this in mind before you choose to work independently. Collaborating outside your group will be considered academic misconduct.