Classes

For this assignment you are required to modify and extend the **Account** class created in Assignment 2 as well as create a new **Customer** class.

The **Account** class should include the following instance variables:

fAcctNo: a string representing the account number.
fSSN: a string representing the social security number.
fBalance: a double that stores the current account balance.

In addition, it should support the following methods:

constructor: the constructor should be passed initial values for the account number, social security number, and balance.
getAcctNo(): an accessor method that returns the fAcctNo instance variable.
getSSN(): an accessor method that returns the fSSN instance variable.
getBalance(): an accessor method that returns the fBalance instance variable.
deposit(double amt): an action method that increases the fBalance instance variable by the amount passed to the method.
doMonthEnd(): an abstract method implemented by the subclasses of this class.
withdrawal(double amt): an abstract method implemented by the subclasses of this class.

You should extend the **Account** class into two subclasses—a **Savings** class and a **Checking** class.

The **Savings** class adds the following instance variable:

fIntRate: a double that stores the *annual* interest rate that applies to the account.

The **Savings** class includes the following methods:

constructor: executes the superclass's constructor.
getIntRate(): an accessor method that returns the fIntRate instance variable.
setIntRate(double rate): an accessor method that sets the fIntRate instance variable.
doMonthEnd(): causes the fBalance instance variable to be increased by applying the value stored in fIntRate. Remember that fIntRate is the *annual* rate and this method is just applying *one month's* worth of interest.
withdrawal(double amt): decreases the fBalance instance variable by the amount passed to the method.

The **Checking** class extends the **Account** class by adding the following instance variable:

fTransCharge: a double that stores the transaction charge associated with withdrawing funds (writing checks).

The **Checking** class includes the following methods:

constructor: executes the superclass's constructor.
getTransCharge(): an accessor method that returns the fTransCharge instance variable.
setTransCharge(double rate): an accessor method that sets the fTransCharge instance variable.
doMonthEnd(): the method does not do anything in this class but must be included because the method is abstract in its superclass.
withdrawal(double amt): decreases the fBalance instance variable by the amount passed to the method. The value of fTransCharge is also subtracted from fBalance for withdrawal.

All three classes should be stored in the same package.

The **Customer** class, which should be stored in a separate package, contains information about a customer and the accounts owned by that customer. The specific instance variables for the **Customer** class are:

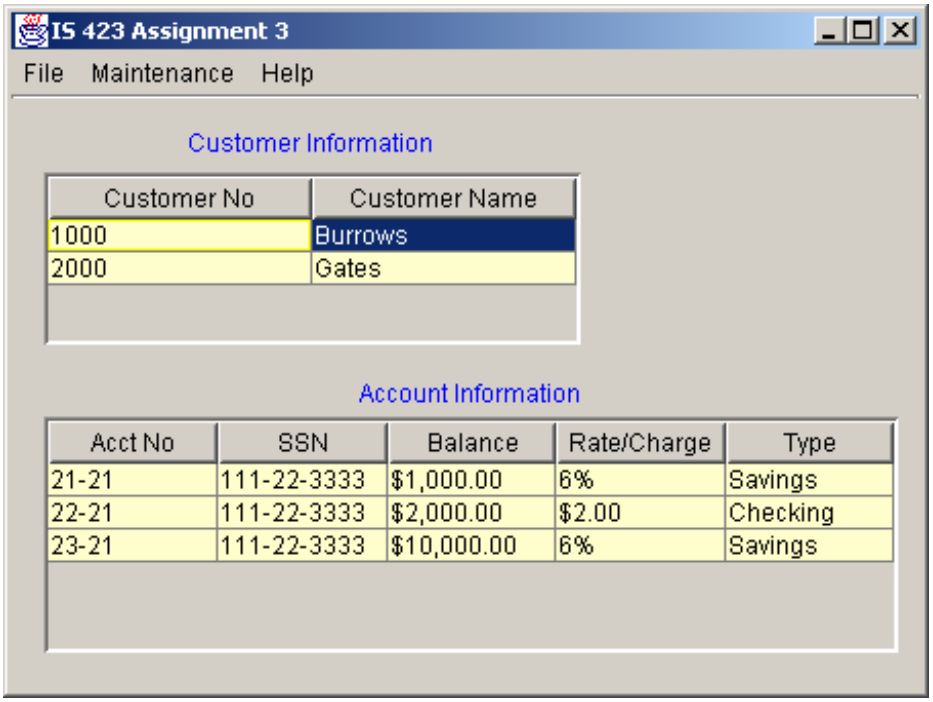
fCustNo: a string storing the customer number.
fName: a string storing the customer name.
fAcctList: a Vector that stores the accounts (savings and checking) that are owned by the customer. The **Vector** class can store any type of object and includes methods to add objects, get objects, and determine the current number of objects stored in the Vector object. Information on the **Vector** class will be provided in class.

The methods of the **Customer** class include:

constructor: uses the values passed to the constructor to define fCustNo and fName instance variables. The constructor must also instantiate the fAcctList instance variable.
getCustNo(): an accessor method that returns the fCustNo instance variable.
getName(): an accessor method that returns the fName instance variable.
setName(String name): an accessor method that sets the fName instance variable.
addAcct(Object acct): causes the specific Savings or Checking object stored in the "acct" parameter to be added to the fAcctList instance variable.
getAllAccts(): returns all the accounts stored in the fAcctList instance variable by returning a clone of fAcctList.

Test Application

The user interface is shown below:

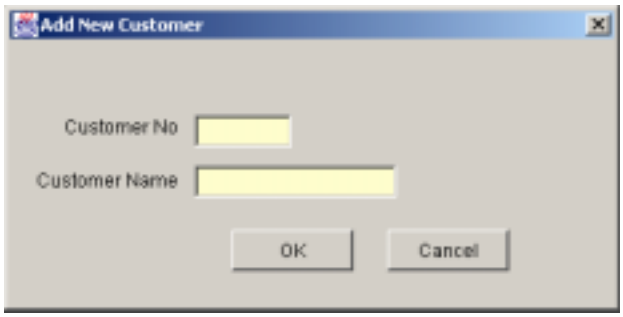


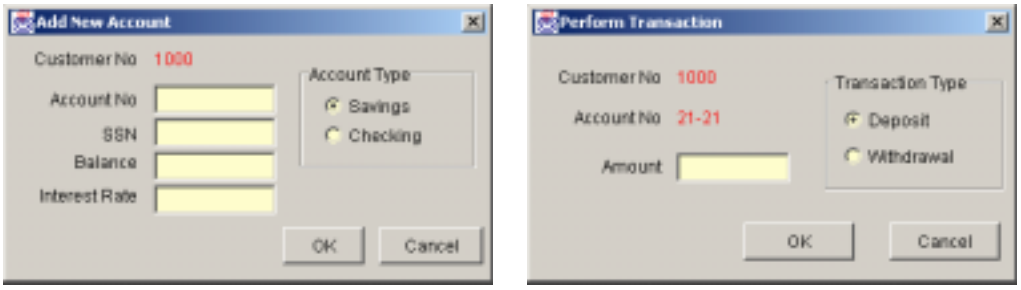
The Customer Information and Account Information JTables are updated automatically whenever a new customer or account is added or a balance changes. You can determine an object's type by using the getClass() and getName() methods that are available for any object as follows:

```
objFoo.getClass().getName()
```

The getName() method returns a fully qualified String containing the object's class name.

You should use JDialog class objects to get customer, account, and transaction information from the user. The dialog boxes below show examples.



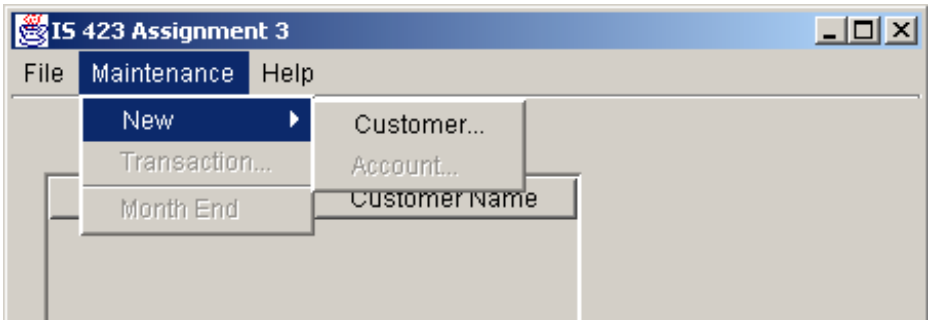


Check to be sure that all fields are completed when the user clicks on the OK button. If there are blank fields when the user clicks on OK, use a JOptionPane to display an alert to the user as shown below.

Also be sure that if the user clicks on the Cancel button, the application functions as expected. Review class handouts for information on using dialog boxes in your application. Be aware that you will have to create instance variables within the JDialog class and also program some accessor methods to get the values from these instance variables. You also might want an instance variable (and a "get" method) to determine if the user clicked on the OK or Cancel button.

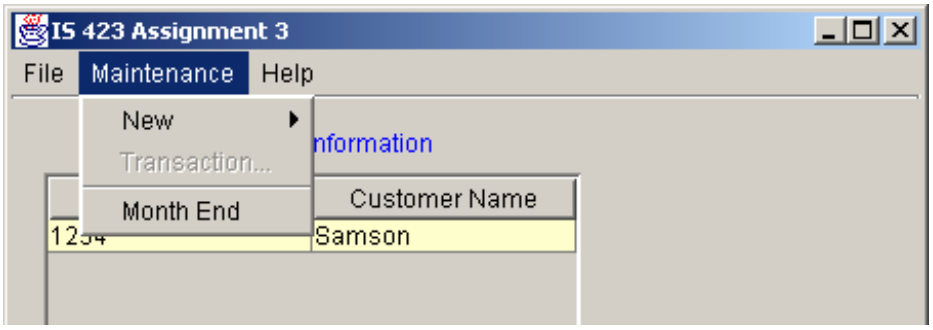
Sequence of Events

The Maintenance menu should drive the application. This menu is shown below.

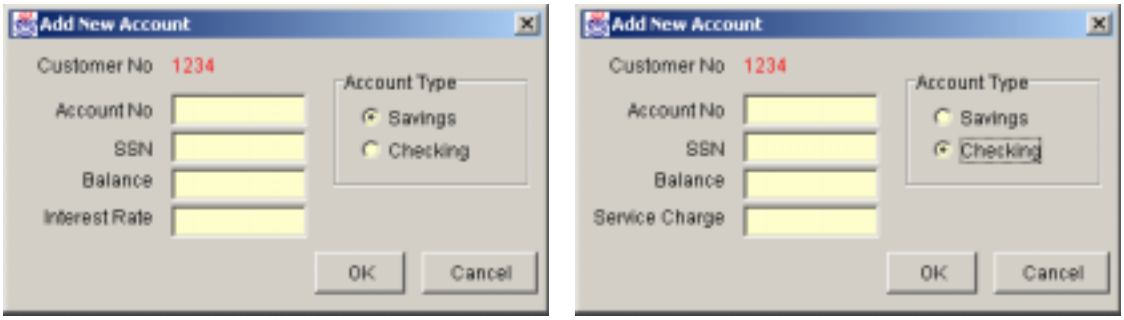


The "New" submenu options allow you create new customers and new accounts for a specific customer. The "Transaction…" menu is used to process either a withdrawal or deposit transaction on a specific account. The "Month End" menu item is used to cause the "month-end" behavior to be applied to ***all*** customers and ***all*** accounts. When the program first starts, the only menu item that is enabled should be the New ▶ Customer…. Choosing this option should bring up the Add New Customer dialog box as shown on the previous page.

After a customer is added, the “Month End” function should be enabled and remain enabled for the remainder of the application.

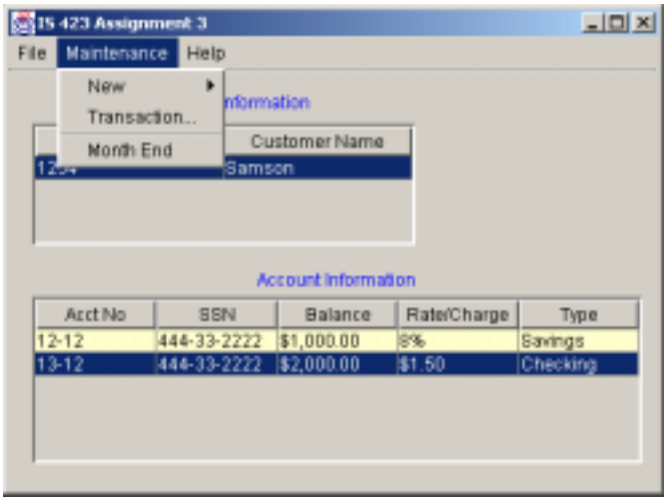


When the user selects a customer in the Customer Information table, the New ▶ Account… menu item should be enabled. It is assumed that the new account will be created for the customer that is selected. In addition, when the user selects a customer in the Customer Information table, the current set of accounts for that customer should be displayed in the Account Information table. Selecting New ▶ Account… causes the Add New Account dialog box to be displayed.

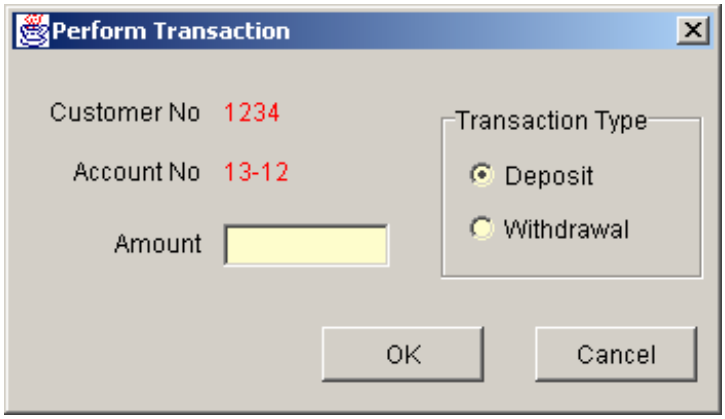


Notice that the customer number has been transferred to the dialog box. Also note that this dialog box is used for both Savings and Checking accounts. When the user selects an account type, the label identifying the bottom text field changes (either “Interest Rate” or “Service Charge”). You will need a number of instance variables for the JDialog class (including one to record the type of account) as well as a number of accessor methods that set/get the instance variable values.

As soon as some accounts are shown in the Account Information table, the user should be able to click and select one of the accounts. When this happens, the “Transaction…” button should be enabled.



Clicking on this button brings up the Perform Transaction dialog box.



Like the Add New Account dialog box, this dialog box is used for processing both Deposit and Withdrawal transactions.

The Customer and Account Information tables should both automatically be updated as changes take place. For example, when a new account is created, it should be added to the Account Information table. Similarly, when a transaction has been completed (including the month-end transaction), the Account Information table should be updated to show the new balance. Also, when a new customer or new account is added, there should be no selected rows in the corresponding table after it is updated. Finally, the “New Account” and “Transaction” menu items should be enabled/disabled as rows are selected and deselected.

Final Comments

You'll need a strategy to keep track of which customer and account the user selected from the Customer and Account Information tables. My solution used two private JFrame class variables to hold this information:

```
private Customer activeCust = null;
private Account activeAccount = null;
```

These variables (as well as others) were declared before the first constructor method for the class in my solution.

This is a complex project so please plan accordingly. It will likely take you many hours to complete and test the application so start early. One good strategy to use is to create your classes first. Then create a simple test application to test them. That is, try creating a customer object and then add an account to it. When this works, try processing a transaction against the account. Do this with a very simple GUI. In fact, you might even want to "hard code" the tests into a button's action_performed event. Once you have the classes tested, make a copy of them and create the project that will support the GUI described here. In this project, you might want to start with the "add new customer function" since you really cannot do anything until you have a customer to work with.

The statements below are all the class-wide variables I declared for my application. I provide them for you so that they might help you think about the structure of the solution.

```
private Vector custList = new Vector();
private Customer activeCust = null;
private Account activeAccount = null;

DefaultTableModel custInfoModel = new DefaultTableModel();
DefaultTableModel acctInfoModel = new DefaultTableModel();
```

The `custList` Vector stores all the customers.

To Turn In

Use the same procedure as you did for Assignments #1 and #2. That is, please store your project on the computer's C: drive in a folder named "is423". Within that folder, the project and its files should be stored in a folder named A3. After you have finished your project, use a ZIP utility to zip the project folder (A3 in this case) and all its subfolders and email it as an attachment to is423wb@u.washington.edu. I will grade the project and return an evaluation form and grade.