**CIS 163 – Computer Science II**

**Project 3 – Bank Application**

**GROUP project (3 to 4 persons per group)**

**Objectives**

* Use inheritance and polymorphism
* Sort objects on different fields using java.util.Comparator class
* Use advanced Swing components like JList, JTable and Abstract Models
* Save and restore objects using the Serialization API
* Save and restore objects to/from a text file
* Bonus: Save and restore objects to and from an XML file

**Due Date: See schedule.**

**Problem Statement**

For Project 3, you will develop a program that maintains accounts for a bank. The application allows you to add, delete, update, and sort accounts. It also provides functionality to save and load/restore accounts to and from a file using three different formats – binary (serialized), text, and XML**. A demo of the project will be done in class, showing all the functionality required. There are also screen shots provided below to help your group understand functionality. As you work on the project, you are welcome to stop into the office and see the programming running.**

**Project Requirements**

Implement your program using the MVC (Model-View-Controller) design. This project is divided into three tiers. **You must complete one tier before beginning the next**.

**Tier 1 (50 points)**

Implement the basic functionality to manage accounts using a **JLIST**.

1. Add an account.
2. Remove an account.
3. Update an account.

|  |  |  |
| --- | --- | --- |
| **Tier 1: look on R:drive for help**   * GUI is functional and easy to use (JList, menu bar, etc) * Add Account works correctly * Delete Account works correctly * Update Account works correctly | 10  15  10  15 |  |

**Tier 2 (15 points)**

Add functionality to save/load accounts to/from a binary (serialized) file:

1. Save, and load accounts as a serialized (binary) file
2. Save, and load accounts from a Text file.

|  |  |  |
| --- | --- | --- |
| **Tier 2:**   * Save, and load accounts as a serialized (binary) file * Save, and load accounts from a Text file | 5  10 |  |

**Tier 3 (15 points)**

Add functionality to give more options to save and load accounts:

(Saving and loading from text files was demonstrated in class but not on R: drive.)

You will need to research JTable and XML

1. Use a JTable not a JList

|  |  |  |
| --- | --- | --- |
| **Tier 3**   * Use a JTable (Just comment out JList code, don’t delete it) | 10 |  |

**Challenging section: (15 points and a gold Star)**

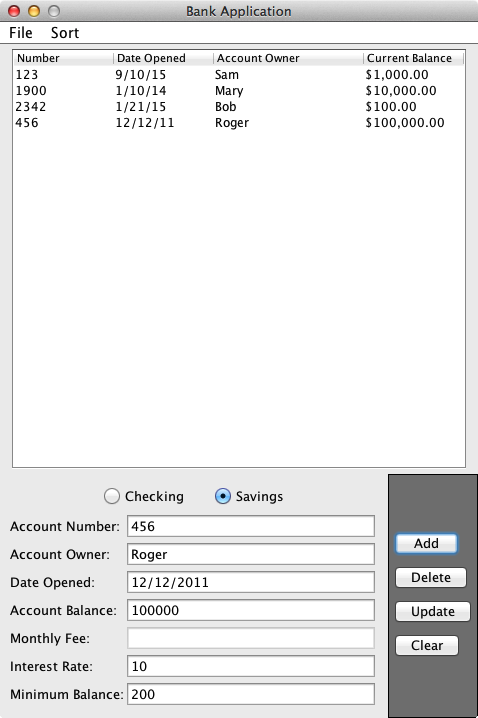
0. A presentation of your program to the class (just 10 mins)

1. Save and Load (recreate) accounts from an XML file.
2. Sort accounts by account number, account owner, and date account opened.
   1. YOU MUST use Collections class (do not write your own sort)

|  |  |  |
| --- | --- | --- |
| **Challenging section**   * A presentation of your program to the class (just 7 mins) * Save and Load (recreate) accounts from an XML file. * Sort accounts by account number, account owner, and date account opened. | 5  5  5 |  |

**Suggested Sample Screenshost; you also can use the GUI that was done in class**

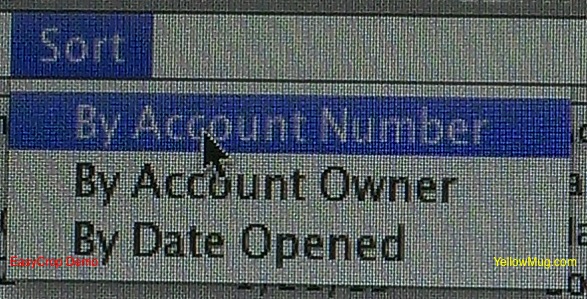
The following screenshots suggest a possible GUI for this project:



Note: Monthly Fee is grayed out.



Interest Rate and Minimum balance are grayed out!



**Suggested Approach**

Include the following classes in your program. You are also free to add additional classes.

1. Account class with the following fields and methods:

import java.io.Serializable;

import java.util. GregorianCalendar;

public abstract class Account implements Serializable {

private static final long serialVersionUID = 1L;

private int number;

private String owner;

private GregorianCalendar dateOpened;

private double balance;

// add constructor

// add getter, setter, equals(), and toString() methods

}

1. CheckingAccount class with the following fields and methods:

public class CheckingAccount extends Account {

private static final long serialVersionUID = 1L;

private double monthlyFee;

// add constructor

// add getter, setter, equals(), and toString() methods

}

1. SavingsAccount class with the following fields and methods:

public class SavingsAccount extends Account {

private static final long serialVersionUID = 1L;

private double minBalance;

private double interestRate;

// add constructor

// add getter, setter, equals(), and toString() methods

}

1. BankGUI class builds the GUI for the application:

public class BankGUI extends JFrame {

// declare GUI components (menu items, buttons, etc.) needed

// constructor method that prepares the GUI

// event handlers and other methods needed to build the GUI

}

1. BankModel class stores a list of accounts (as an ArrayList of Account objects) and contains methods to manage (retrieve, add, delete, update, sort, load, save, etc.) the accounts.

public class BankModel extends AbstractListModel{

private ArrayList<Accounts> acts;

// constructor method that initializes the arraylist

// override these two methods from AbstractListModel class

public Object getElementAt(int arg0) {

}

public int getSize() {

}

// add methods to find, add, delete, and update accounts

// add methods to sort accounts on required fields

// add methods to load/save accounts from/to a binary file

// add methods to load/save accounts from/to a text file

// add methods to load/save accounts from/to an XML file

// add other methods as needed

}

1. To make updates to the accounts in the model immediately visible in the JList on your GUI, it is important that the methods in the BankModel class that modify (add, delete, and update) the accounts notify the JList immediately after any changes. These notifications can be sent from BankModel class using one of these methods: fireIntervalAdded(), fireIntervalRemoved(), and fireContentsChanged(). The BankModel class inherits these methods from the AbstractListModel class.
2. BankController class maps the user’s interactions with the GUI (view) into appropriate method calls in the BankModel class. The BankGUI class makes calls to methods in the BankController class.
3. BankController class maintains an instance of BankModel class and also provides a getter method for the BankModel instance it stores.
4. BankGUI class maintains an instance of BankController. In the BankGUI class, set the BankModel as the model for the JList in the BankGUI class using the setModel() method of the JList class.
5. Your instructor will provide more details of how various classes involved in this project work.
6. The following figure shows how the model, view, and controller classes work together to provide the functionality in the program. When a user interacts with the view, the following actions occur:
7. The view recognizes (via event handlers) that a user action (such as a button click, menu selection, etc.) has occurred.
8. The view then calls appropriate method(s) in the controller.
9. The controller accesses the model and calls appropriate method(s) on the model.
10. As a result of actions on the model, if there are any changes in the model, the model notifies the view of the changes. The view may also pull changes form the model as needed.

View

Controller

Model

**Deliverables and Submission**

* You must demonstrate your program. Programs that are not personally demonstrated may not be graded. Details on demonstration times will be provided to you later.

**Grading Rubric**

* See the next page for grading rubric.
* We will be using this grading sheet during your demonstration of the project.

**Grading Rubric for Project 3, CIS 163, Winter 2012**

|  |  |
| --- | --- |
| Student Name |  |
| Date Submitted |  |
| Days Late / Late Penalty |  |

|  |  |  |
| --- | --- | --- |
| **Graded Item** | **Max Points** | **Points Won** |
| **JSG** |  |  |
| Javadoc Comments and Coding Style/Technique  (<http://www.cis.gvsu.edu/studentsupport/javaguide>)   * Code Indentation (auto format source code in IDE) * Naming Conventions (see Java style guide) * Proper access modifiers for fields and methods * Use of helper (private) methods * Using good variable names * Header/class comments * Every method uses @param and @return (1 sentence after) * Every method uses a /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* separator * Overall layout, readability, No text wrap * Using /\*\* … / for each Instance variable * Has many inner “inner” comments | 10 |  |
| **Tier 1: look on R:drive for help**   * GUI is functional and easy to use (JList, menu bar, etc) * Add Account works correctly * Delete Account works correctly * Update Account works correctly | 10  15  10  15 |  |
| **Tier 2:**   * Save, and load accounts as a serialized (binary) file * Save, and load accounts from a Text file | 5  10 |  |
| **Tier 3**   * Use a JTable (Just comment out JList code, don’t delete it) | 10 |  |
| **Challenging section**   * A presentation of your program to the class (just 7 mins) * Save and Load (recreate) accounts from an XML file. * Sort accounts by account number, account owner, and date account opened. | 5  5  5 |  |
| **Total Possible Points** | **100** |  |

|  |
| --- |
| **Comments** |