

SSE 554 Project 2

Basic Bank Application

Database.java, AccountFrame.java, Account.java

Kei'Shawn Tention

3/20/15

TABLE OF CONTENTS

Introduction	3
Database	3
Conclusion	10
APPENDIX	11
<i>Database.java Class File</i>	11
<i>Account.java Class File</i>	17
<i>AccountFrame.java Class File</i>	18
<i>NON DIRECT ACTIVITY LOG</i>	21

Introduction

The purpose of Project 2 for SSE 554 is to demonstrate one's individual knowledge of an advanced topic as well as become more accustomed to using DVCS. The team consisted of Josh Deremer, Daryl Ebanks, David May, and myself. Each of us had a particular topic we wanted to explore, with some being the same. My particular topic was Database design. Unfortunately, the team agreed that an actual DBMS maybe out of scope for such a simple project. I had the responsibility of completing the Database.java class and ensuring that the accounts would store properly. I also accepted the task of writing those accounts to an XML. Other topics that the group agreed to explore were XML files, Document Object Model (DOM), and Encryption.

Database

The database for the Basic-Bank Application is simple user-defined class that is designed to store the different bank accounts created. The main object that drives the database is an Array List of Account objects. The source code for the Account.java class can be located in the Appendix. The source code for Database.java is located in the Appendix as well. The constructor of the Database.java class has the responsibility of also instantiating the object for the XMLIO.java class and the object for the Document class. The Document object is explained more in the section entitled Using a Document Object Model, which can be found in the complete report.

The Database.java class consists of the following methods:

1. addAccount()
2. removeAccount()
3. withdraw()
4. deposit()
5. getBalance()
6. verifyAcct()
7. writeToFile()

There are more methods that have minor roles for the functionality of the Database.java class. The first 5 methods in the list are self-explanatory and intuitive. However, verifyAcct() and writeToFile() need explaining. Account verification is important for users, whether it is a bank teller or a customer, because it allows the right person to authorize transactions to the proper account they logged into. The following page has the block of code that allows for a secure access to an account.

```
public boolean verifyAcct(Account acct){  
    boolean temp= false;  
    for(Account a:db){  
        if((a.holder.compareTo(acct.holder)==0) &&  
        (a.password.compareTo(acct.password)==0))  
            temp = true;  
    }  
    return temp;  
}
```

The writeToFile() method is another important aspect of the Database.java class. It allows all the accounts created to be stored into an XML file using the Document

Object Model. This concept was one our team wanted to focus on as a major topic for this project. The block of code is provided below.

```
public void writeToFile()
{
    try
    {
        DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();

        DocumentBuilder builder;

        builder = factory.newDocumentBuilder();

        Document doc = builder.newDocument();

        XMLIO io = new XMLIO();

        //root element
        Element rootElement = doc.createElement("Bank");
        doc.appendChild(rootElement);
        for(Account a: db){
            //add account elements
            Element acct_Tag = doc.createElement("Account");
            Element holder = doc.createElement("Holder");
            acct_Tag.appendChild(holder);
            holder.appendChild(doc.createTextNode(a.holder));

            Element psswrđ = doc.createElement("Password");
            acct_Tag.appendChild(psswrđ);
            psswrđ.appendChild(doc.createTextNode(a.password));
```

```

        Element balance = doc.createElement("Balance");
        acct_Tag.appendChild(balance);

balance.appendChild(doc.createTextNode(Double.toString(a.balance)));

        Element accType = doc.createElement("AccountType");
        acct_Tag.appendChild(accType);
        accType.appendChild(doc.createTextNode(a.accType));
        rootElement.appendChild(acct_Tag);

        io.WriteXMLFile(doc, new File("XML/Database.xml"));

        BufferedReader reader = new BufferedReader(new
        FileReader("XML/Database.xml"));
        String line = null, teststr2="";

        while ((line = reader.readLine()) != null) {
            teststr2 += line;
        }

        reader.close();
    }

    // Files.delete(FileSystems.getDefault().getPath("",
    "XML/Database.xml"));

} catch (Exception e) {

```

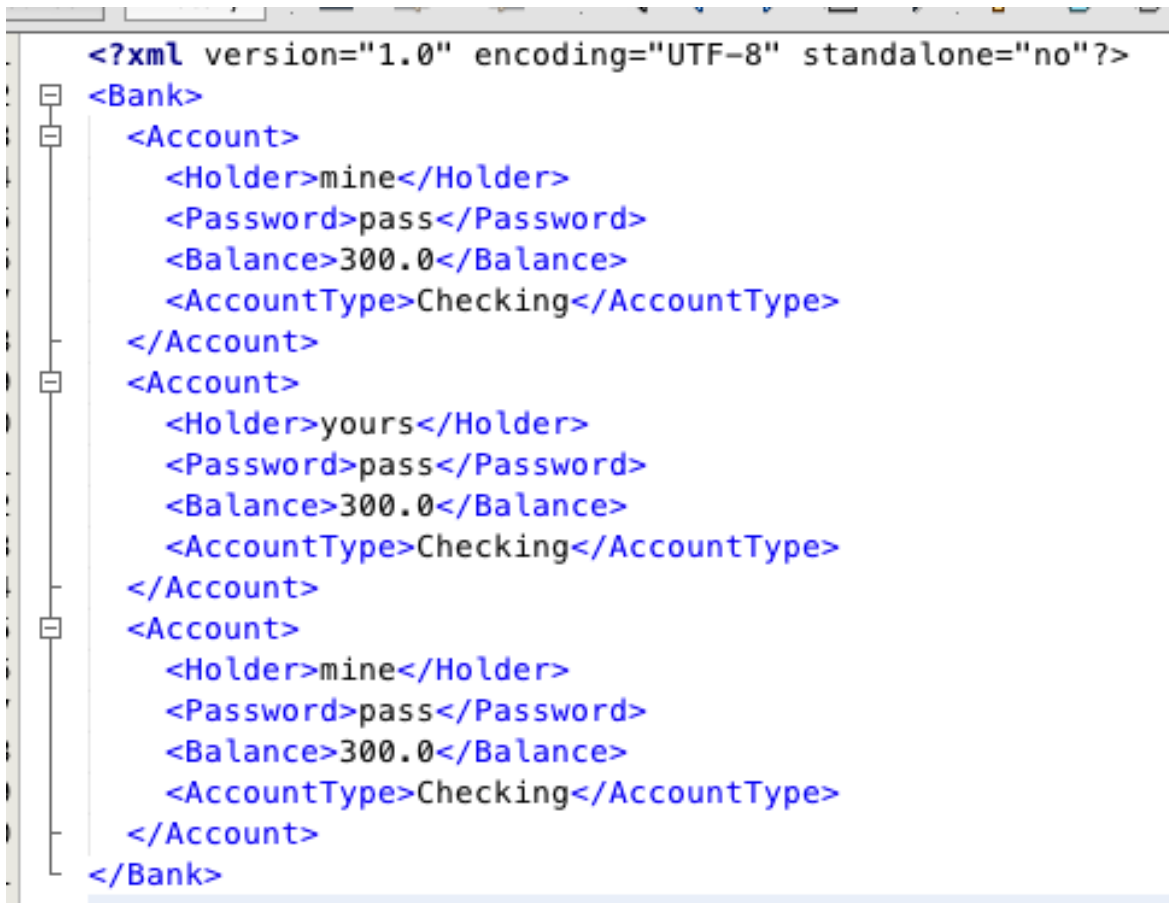
```

        e.printStackTrace();
    }}

}

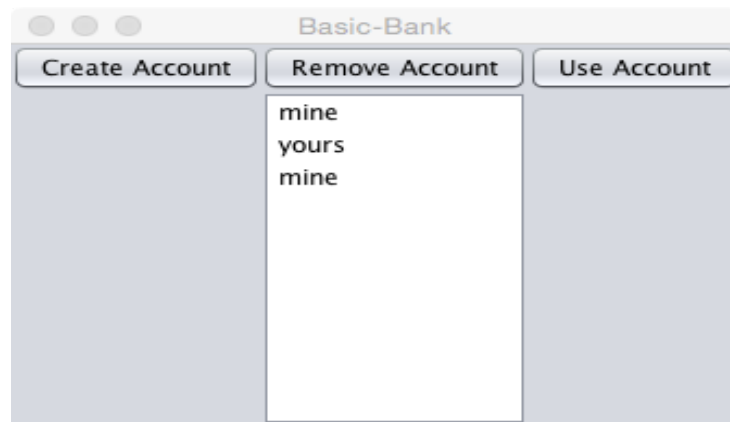
```

A portion of the Document Object Model was used within the Database.java class. As mentioned before, it was used to store the banks accounts into an XML file. The following figures illustrate the creation of banks accounts, the logging in to a specific account, and the transactions of the accounts. Upon exiting the Basic Bank application, the writeToFile() method is called and the accounts are written to an XML file.

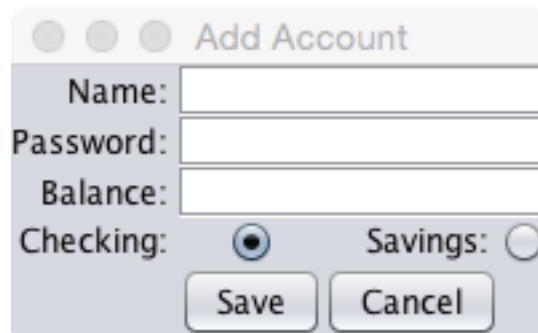


Default accounts of the Basic Bank application.

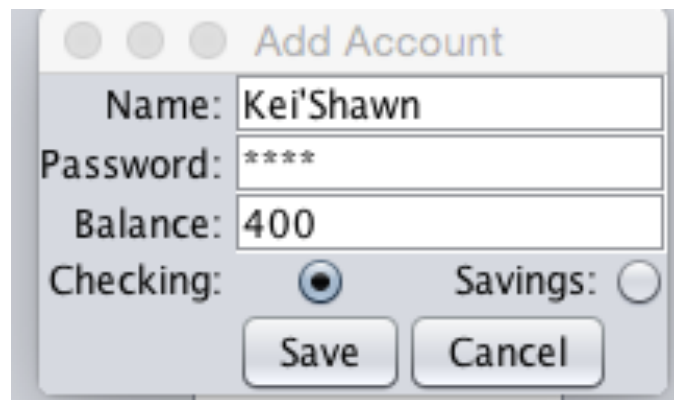
Main BankFrame. The default accounts can be seen.



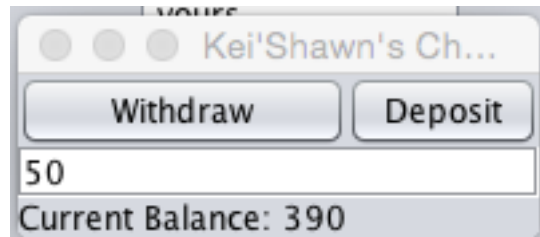
Create a New Account.



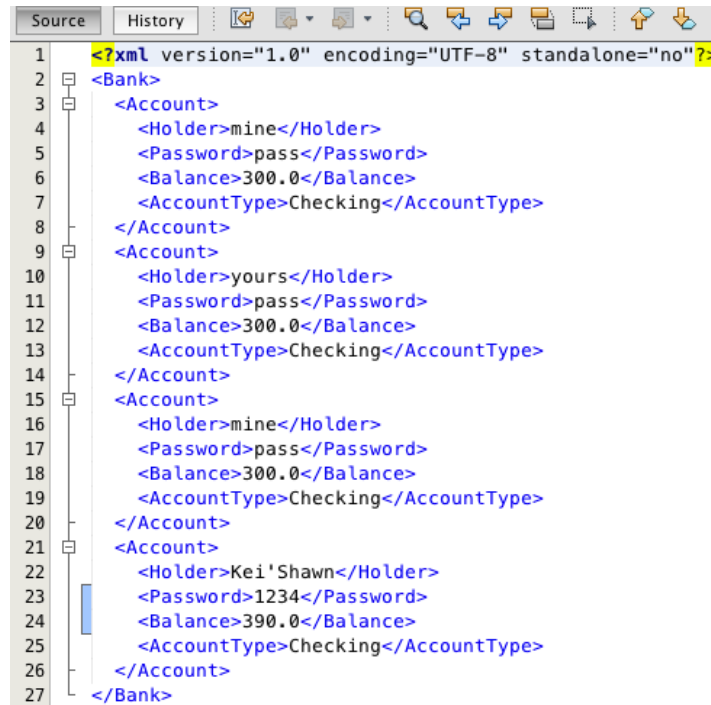
Click save. Clicking save does not add the account details to the XML file.



Now lets make a few transactions. Deposit \$40.00 and withdraw \$50.00.



At this point we have only made changes to the account within the “database” and Array list. In order to write the account details to the XML file. We must back out to the main BankFrame and click on the Exit button. At this point, the `writeToFile()` method of the `Database.java` class is called and the following file, if it doesn’t not already exist, is created. It is entitled “Database.xml”. See the figure below.



```
1 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2 <Bank>
3   <Account>
4     <Holder>mine</Holder>
5     <Password>pass</Password>
6     <Balance>300.0</Balance>
7     <AccountType>Checking</AccountType>
8   </Account>
9   <Account>
10    <Holder>yours</Holder>
11    <Password>pass</Password>
12    <Balance>300.0</Balance>
13    <AccountType>Checking</AccountType>
14  </Account>
15  <Account>
16    <Holder>mine</Holder>
17    <Password>pass</Password>
18    <Balance>300.0</Balance>
19    <AccountType>Checking</AccountType>
20  </Account>
21  <Account>
22    <Holder>Kei' Shawn</Holder>
23    <Password>1234</Password>
24    <Balance>390.0</Balance>
25    <AccountType>Checking</AccountType>
26  </Account>
27 </Bank>
```

Conclusion

Each of the team members had their own topic to explore. One problem we faced in the beginning were actually figuring out what type of project we could develop that would encompass each interested topic—XML file, Document Object Model, Database Design, and Encryption. Selecting Daryl Ebanks previous code, Bank Account from SSE Project 1, was the perfect project and it introduced us to methods of refactoring. Another problem we faced as a group was working together. David May and I were able to physically meet twice, while the others were not. Even though the purpose of utilizing a DVCS is to help programmers work together, but sometimes it is not always effective. We have decided to take Dr. MacNeil's advice and split the group up. David May and I will continue this semester as a team.

Although we ran into complications, I believe that each of us were able to demonstrate our knowledge of the advanced topic we selected. With the help of Josh Deremer, I was able to successfully store the accounts into a array list using the simple in-app Database. With the help of Daryl Ebanks and after gaining a thorough understanding of his XMLIO.java file and DOM, I was able to write the data, account information, to an XML file. Daryl Ebanks and I discussed the possibility of reading in an XML file using the DOM and using the data read in within the Basic Bank app, but time did not permit such an idea.

APPENDIX

Database.java Class File

```
/*  
    * To change this license header, choose License Headers  
    in Project Properties.  
    * To change this template file, choose Tools | Templates  
    * and open the template in the editor.  
    */  
  
package basicbank;  
  
import java.io.File;  
import java.util.ArrayList;  
import javax.xml.parsers.DocumentBuilder;  
import javax.xml.parsers.DocumentBuilderFactory;  
import org.w3c.dom.Document;  
import xmlio.XMLIO;  
import java.io.BufferedReader;  
import java.io.BufferedWriter;  
import java.io.File;  
import java.io.FileReader;  
import java.io.FileWriter;  
import java.nio.file.FileSystems;  
import java.nio.file.Files;
```

```

import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.w3c.dom.Text;
import java.lang.Exception;
import javax.xml.parsers.ParserConfigurationException;

/**
 *
 * @author Josh
 * @author Kei'Shawn
 */
public class Database
{

    private ArrayList<Account> db;

    File file;

    public Database(File file)
    {
        db = new ArrayList<Account>();
        this.file=file;
        //parse document for data
    }

    public Database(ArrayList<Account> db, File file)
    {
        this.db = db;
        this.file=file;
        XMLIO io = new XMLIO();

        Document doc = io.ReadXMLFile(file);
    }

    public void addAccount(String holder, String password,
int balance, boolean checking)
    {
        //Kei'Shawn
        if(checking == true){
            CheckingAccount chck = new
CheckingAccount(balance, holder, password, "Checking");

```

```

        db.add(chck);
    }
    else{
        SavingsAccount svngs = new
SavingsAccount(balance, holder, password, "Savings");
        db.add(svngs);
    }
}

public void removeAccount(String holder)
{
    // Kei'Shawn
    int index = getIndex(holder);

    if( index < 0)
        return;

    db.remove(index);
}

public int getIndex(String holder)
{
    for(Account acct : db)
    {
        if(acct.holder.compareTo(holder) == 0)
            return db.indexOf(acct);
    }
    return -1;
}

public int getBalance()
{
    // Get balance
    return 0;
}

public int withdrawls(String holder, String password)
{
    // Get number of withdrawls
    return 0;
}

```

```

public void withdraw(int amount, Account acct)
{
    //@author Kei'Shawn
    if(verifyAcct(acct))
        acct.withdraw(amount);
}

public void deposit(int amount, Account acct)
{
    //@author Kei'Shawn
    if(verifyAcct(acct))
        acct.deposit(amount);
}

public void writeToFile()
{
    //@author Kei'Shawn
    try
    {
        DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
        DocumentBuilder builder;

        builder = factory.newDocumentBuilder();

        Document doc = builder.newDocument();
        XMLIO io = new XMLIO();

        //root element
        Element rootElement =
doc.createElement("Bank");
        doc.appendChild(rootElement);

        for(Account a: db){

            //add account elements
            Element acct_Tag =
doc.createElement("Account");
            Element holder =
doc.createElement("Holder");
            acct_Tag.appendChild(holder);

            holder.appendChild(doc.createTextNode(a.holder));

            Element psswr =

```

```

doc.createElement("Password");
    acct_Tag.appendChild(psswrld);

psswrld.appendChild(doc.createTextNode(a.password));

    Element balance =
doc.createElement("Balance");
    acct_Tag.appendChild(balance);

balance.appendChild(doc.createTextNode(Double.toString(a.b
alance)));

    Element accType =
doc.createElement("AccountType");
    acct_Tag.appendChild(accType);

accType.appendChild(doc.createTextNode(a.accType));
    rootElement.appendChild(acct_Tag);

    io.WriteXMLFile(doc, new
File("XML/Database.xml"));

    BufferedReader reader = new BufferedReader(new
FileReader("XML/Database.xml"));
    String line = null, teststr2="";

    while ((line = reader.readLine()) != null) {
        teststr2 += line;
    }

    reader.close();
}

//
Files.delete(FileSystems.getDefault().getPath("",
"XML/Database.xml"));

    } catch (Exception e) {
        e.printStackTrace();
    }

}

public int getSize()

```

```

{
    return db.size();
}

public ArrayList<String> getHolders()
{
    ArrayList<String> holders = new
ArrayList<String>();

    for(Account a : db)
    {
        holders.add(a.holder);
    }

    return holders;
}

public ArrayList<Account> getAccounts(){
    return db;
}

public Account setAccount(String holder, String
password){

    //@author Kei'Shawn
    int temp=0;
    for(int i = 0; i<db.size(); i++){
        if(holder.compareTo(db.get(i).holder)==0
            &&
(password.compareTo(db.get(i).password))==0) {
            temp = i;
        }
    }
    return db.get(temp);
}

public boolean verifyAcct(Account acct){

    //@author Kei'Shawn
    boolean temp= false;
    for(Account a:db){
        if((a.holder.compareTo(acct.holder)==0) &&
(a.password.compareTo(acct.password)==0))
            temp = true;
    }
}

```



```

    }
    return temp;
}
}

```

Account.java va Class File

```

package
basicbank
;

/**
 * Created by Daryl Ebanks for SSE 554 Project 1,
 * Spring 2015
 */

public abstract class Account
{
    protected double balance;
    protected String holder;
    protected String password;
    protected double rate = .2;
    protected String accType = "";

    public enum CompoundResult
    {
        NONE,
        RATE,
        PENALTY
    };

    public Account(double balance, String holder,
String password, String accType)
    {
        //Kei'Shawn made adjustments
        this.balance = balance;
        this.holder = holder;
        this.password = password;
    }
}

```

```

        this.accType = accType;
    }

    protected void deposit(int amount)
    {
        if(amount > 0)
            balance+= amount;
    }

    protected abstract void withdraw(int amount);

    protected Double getBalance(String password)
    {
        if(authenticate(password))
            return balance;

        return null;
    }

    protected int getBalance(){
        return (int)balance;
    }

    protected Boolean isOverdrawn(String password)
    {
        if(authenticate(password))
            return balance < 0;

        return null;
    }

    protected abstract CompoundResult
    compoundInterest();

    protected Boolean authenticate(String password)
    {
        if(this.password.compareTo(password) == 0)
            return true;

        return false;
    }
}

```

AccountFrame.java Class File

```

/*
 * To change this license header, choose License Headers in
 * Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

package basicbank;

/**
 *
 * @author Josh
 */

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;

public class AccountFrame extends JFrame
{
    private GridBagPane gridbag;
    private JButton withdraw_button;
    private JButton deposit_button;
    private JTextArea value;
    private JLabel current_balance;

    @SuppressWarnings("unchecked")
    public AccountFrame(Database data, Account acct)
    {
        File file = new File("XML/account.xml");
        gridbag = new GridBagPane(file);
        add(gridbag);

        withdraw_button = (JButton)
gridbag.get("withdraw_button");
        deposit_button = (JButton)
gridbag.get("deposit_button");
        value = (JTextArea) gridbag.get("value");
        current_balance = (JLabel)
gridbag.get("current_balance");

        current_balance.setText("Current Balance: " +
Integer.toString(acct.getBalance()));
    }
}

```

```

        ActionListener withdraw_click = new
        ActionListener(){
            public void actionPerformed(ActionEvent event){
                //@author Kei'Shawn
                //withdraw

                data.withdraw(Integer.parseInt(value.getText()), acct);
                current_balance.setText("Current Balance: "
+ Integer.toString(acct.getBalance()));
            }};

        ActionListener deposit_click = new
        ActionListener(){
            public void actionPerformed(ActionEvent event){
                //@author Kei'Shawn
                //deposit

                data.deposit(Integer.parseInt(value.getText()), acct);
                current_balance.setText("Current Balance: "
+ Integer.toString(acct.getBalance()));
            }};

        withdraw_button.addActionListener(withdraw_click);
        deposit_button.addActionListener(deposit_click);

        pack();
        Dimension dim =
        Toolkit.getDefaultToolkit().getScreenSize();
        this.setLocation((dim.width/2-
this.getSize().width/2), (dim.height/2-
this.getSize().height/2));
        this.setResizable(false);
    }
}

```

NON DIRECT ACTIVITY LOG

Time/Task Log	Activity Type: Non Direct	
Date	Duration (in minutes)	Specific Task/Activity
Feb-3	60	Read through Chapter 8 Event handling pages 264-273 (Java Core Vol 1)
Feb-4	60	Read Java Core Vol 1 pages 276 - 280 of Chapter 8 (Actions)
Feb-5	60	Read pages 68-80 of Chapter 2 in the Java Core Vol 2
Feb-6	120	Read Chapter 4 Database Programming in the Java Core Vol 2. Decided that keeping the database of project 2 simple would suffice
Feb-7	45	Briefly read Chapter 6 (Advance Swing) of Java Core Vol 2 and a little bit of Chapter 7 of Java Core Vol 1
Feb-8	30	Thought about the different advanced topics I could explore individually. Wanted to make sure I had an idea that would fit into project scope.
Feb-9	30	Read w3Schools Introduction to XML
Feb-10	30	Practiced more TDD for my own skill development. Not for project 2
Feb-12	10	Update Logs
Feb-20	120	Traced through the Account.java, CheckingAcct.java SavingsAcct.java class files to get a full understanding of how the BasicBank app would operate
Feb-21	60	Read 12.8(Handling Errors) in Java Core Vol2
Feb-22	120	Ran into some errors and spent time debugging and refactoring code.
Feb-28	60	Watched a youtube video on XML - https://www.youtube.com/watch?v=yUw-aTOwAw8
Mar-1	30	Traced through Josh's skeleton of Database.java
Mar-2	60	Coded part of the Database.java class
Mar-3	45	Made changes to BankFrame.java and AccountFrame.java so that the methods in Database.java would correspond
Mar-4	60	Read an article on XML, Read through Daryl Ebanks XMLIO.java class and test class. Coded the writeToFile() method of Database.java class
Mar-5	60	Read articles on the Document Object Model and

		began constructing the code to make DOM possible. Referred to Daryl Ebanks XMLIO.java class
Mar-6	60	Finished the coding the writeToFile() method required to write data to XML file
Mar-18	90	Documented the rest of my portion of the project
Mar-19	60	Created screen prints of the fully functional Basic Bank App. Separated my portion of the project from other team members
Mar-20	30	Update Logs
TOTAL	1300	
Required	4500	
Remaining for Semester	1295	