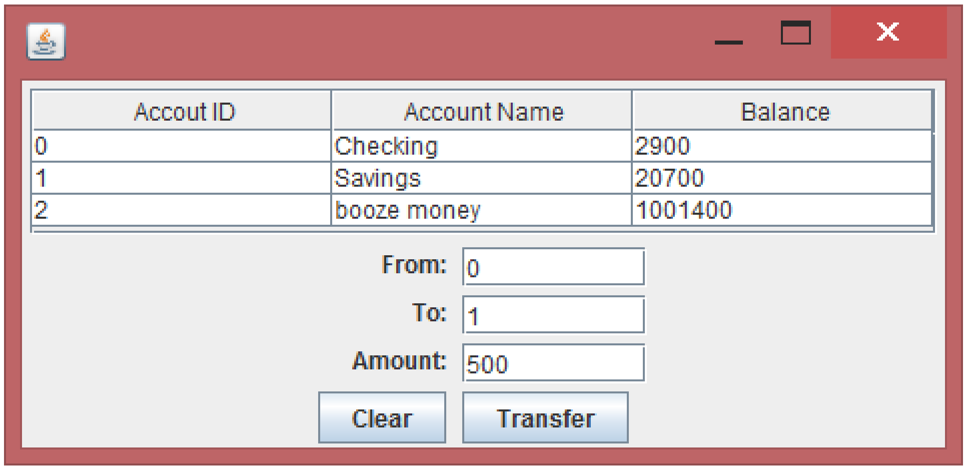
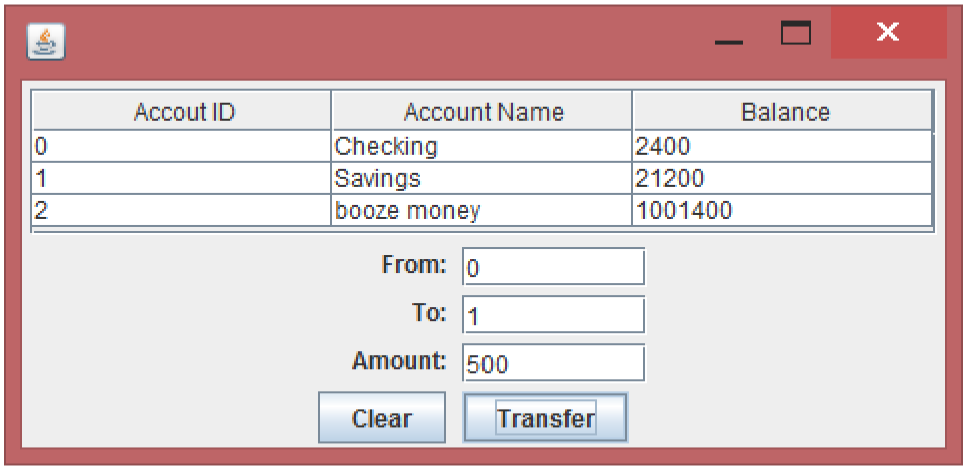
Eric Wilson
Java Lab #7



Accout ID	Account Name	Balance
0	Checking	2900
1	Savings	20700
2	booze money	1001400

From: 0
To: 1
Amount: 500



Accout ID	Account Name	Balance
0	Checking	2400
1	Savings	21200
2	booze money	1001400

From: 0
To: 1
Amount: 500

This one might be slower than most but that's because it operates directly on the DB to reduce the amount of erroneous data caused my multiple users updating the table at the same time. It's not a beautiful program but it gets the job done.

MySqlExample.java

```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.*;
import java.io.*;
import java.util.Vector;

public class MySqlExample {

    private Connection con;
    private Statement stmt;

    // Load class and create a connection to the database
    public MySqlExample() throws SQLException {
```

```java
        // Connection string
        String connectionString = "jdbc:mysql://kc-sce-appdb01.kc.umkc.edu/eawwwd";
        String userID = "eawwwd";
        String password = "ZDkWx5Dydgwe";

        try {
            Class.forName("com.mysql.jdbc.Driver");
        } catch(java.lang.ClassNotFoundException e) {
            System.out.println(e);
            System.exit(0);
        }

        con = DriverManager.getConnection(connectionString, userID, password);
        stmt = con.createStatement();

        ResultSet rs = stmt.executeQuery("SELECT * FROM accounts;");
        displayResultSet(rs, System.out);
        Vector<Vector<Object>> dataModel = createObjectArray(rs);

        AccountTransactionLayout atl = new AccountTransactionLayout(dataModel);

        atl.transferAddActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent actionEvent) {
                int amount = atl.getAmount();
                int from = atl.getfromNum();
                int to = atl.getToNum();
                int fromActAmt = 0;
                ResultSet temp;
                ResultSet query;
                try {
                    temp = stmt.executeQuery("SELECT balance FROM accounts WHERE acct
= " +

                    from + ";");
                    temp.first();
                    fromActAmt = temp.getInt(1);
                } catch (SQLException e) {
                    e.printStackTrace();
                }
                if (fromActAmt >= amount){
                    try {
                        stmt.executeUpdate("UPDATE accounts SET balance = balance +  "
+

                        amount + " WHERE acct = " + to + ";");
                        stmt.executeUpdate("UPDATE accounts SET balance = balance - "
+

                        amount + " WHERE acct = " + from + ";");
                        /*TEST*/ query = stmt.executeQuery("SELECT * from accounts");
                        displayResultSet(query, System.out);
                        Vector<Vector<Object>> newTable = createObjectArray(query);
                        atl.updateWholeTable(newTable);
                    } catch (SQLException e) {
                        e.printStackTrace();
                    }

                    }
                else {
                    JOptionPane.showMessageDialog(atl, "Invalid Amount! Do you want
overdraft fees because "+
                    "this is how you get overdraft fees");
                }
            }
        });

        atl.clearAddActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent actionEvent) {
                atl.clearInputFields();
            }
        });
    }

    private Vector<Vector<Object>> createObjectArray(ResultSet results) throws
SQLException {
        Vector<Vector<Object>> vectorvector = new Vector();
        ResultSetMetaData md = results.getMetaData();
        int numColumns = md.getColumnCount();
        results.first();
        do {
            Vector<Object> currentRowData = new Vector();
            for(int i = 0; i < numColumns; i++){
                currentRowData.add(i,results.getString(i + 1) );
            }
            vectorvector.addElement(currentRowData);
        } while (results.next());
        return vectorvector;
    }


    public void createTables() throws SQLException {
        //String sqlcmd = "USE eawwwd";

        String createTable = "CREATE TABLE IF NOT EXISTS Accounts" + "(acct INT,"
                + "acct_name VARCHAR(64),"
                + "balance FLOAT);";
        //stmt.execute(sqlcmd);
        stmt.execute(createTable);
    }

    public void updateTable() throws SQLException {

    }

    public void queryTable(int acct) throws SQLException {
        String sqlCmd = "SELECT acct_name,balance FROM accounts " +
                "WHERE acct = " + Integer.toString(acct) + ";";
        ResultSet rs = stmt.executeQuery(sqlCmd);
        displayResultSet(rs,System.out);
    }

    public void displayResultSet (ResultSet rs, PrintStream out) {
        int i;

        out.println();
        out.println();
        out.println("Result Set:");
        out.println("------------------------");
        out.println();

        try {
            ResultSetMetaData rsmd = rs.getMetaData ();

            int numCols = rsmd.getColumnCount ();
            for (i=1; i<=numCols; i++) {
                out.print(rsmd.getColumnLabel(i));
                out.print(" ");
            }
            out.println();

            boolean more = rs.next ();
            while (more) {
                for (i=1; i<=numCols; i++) {
                    // Every field value is fetched as a string
                    // JDBC will convert the values from their
                    // stored type to string type
                    out.print(rs.getString(i));
                    out.print(" ");
                }
                out.println();

                more = rs.next ();
            }
            out.println();
        }
        catch (SQLException ex) {
            out.println ("*** SQLException");

            while (ex != null) {
                out.println ("SQLState: " + ex.getSQLState ());
                out.println ("Message: " + ex.getMessage ());
                out.println ("Vendor: " + ex.getErrorCode ());
                ex = ex.getNextException ();
                out.println ("");
            }
        }
    }

    public static void main(String args[]) {
        try {
            MySqlExample ex = new MySqlExample();
            System.out.println("If no exceptions, you successfully connected to your
DB.");

          ex.createTables();
            ex.queryTable(1);
        }
        catch (SQLException e) {
            System.err.println(e);
        }
    }
}
```

AccountTransactionLayout.java

```
// This code gives the layout for the UI and
//   demonstrates two ways of updating the data
//   in a JTable.
// Another option to consider when using JTable is
//   creating your own data model by overriding
//   AbstractTableModel. You might use this option
//   if data for table was coming from say a DB.
//   One example: http://www.java2s.com/Code/Java/Swing-
JFC/CreatingsimpleJTableusingAbstractTableModel.htm

import java.awt.*;
```

```java
import java.awt.event.ActionListener;

import java.util.Vector;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;

public class AccountTransactionLayout extends JFrame {

    private JTextField fromField = new JTextField("3",8);
    private JTextField toField = new JTextField("4",8);
    private JTextField amountField = new JTextField("100",8);

    JButton clearButton = new JButton("Clear");
    JButton transferButton = new JButton("Transfer");


    private JTable table;
    private Vector<String> columnNames = new Vector();

    JFrame frame = this;

    public AccountTransactionLayout(Vector<Vector<Object>> results) {
        //DefaultTableModel dtm = tabMod;
        columnNames.add("Accout ID");
        columnNames.add("Account Name");
        columnNames.add("Balance");

        table = new JTable(results, columnNames );
        Container contentPane = getContentPane();
        contentPane.setLayout(new GridBagLayout());


        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


        // The default size of a JTable is something like
        // 450 X 400.
        Dimension smallerSize = new Dimension(450, 50);
        table.setPreferredScrollableViewportSize(smallerSize );

        JScrollPane scrollPaneForTable = new JScrollPane(table);

        GridBagConstraints constraints = new GridBagConstraints();

        constraints.gridx = 0;
        constraints.gridy = 0;
        constraints.gridwidth = 2;
        constraints.gridheight = 1;
        constraints.weightx = 1;
        constraints.weighty = 1;
        constraints.insets = new Insets(4, 4, 4, 4);
        constraints.fill = GridBagConstraints.BOTH;

        contentPane.add(scrollPaneForTable,constraints);

        constraints.gridx = 0;
//    constraints.gridy = 1;
        constraints.weighty = 0;
        constraints.gridy = GridBagConstraints.RELATIVE;
        constraints.insets = new Insets(2, 4, 2, 4);
        constraints.fill = GridBagConstraints.NONE;
        constraints.gridwidth = 1;
        constraints.anchor = GridBagConstraints.NORTHEAST;
        JLabel toLabel = new JLabel("From:");
        contentPane.add(toLabel,constraints);

        constraints.gridx = 1;
        constraints.anchor = GridBagConstraints.NORTHWEST;

        // Workaround, because of:
http://bugs.java.com/bugdatabase/view_bug.do?bug_id=4247013
        fromField.setMinimumSize(fromField.getPreferredSize());
        contentPane.add(fromField,constraints);

        constraints.gridx = 0;
//      constraints.gridy = 2;
        constraints.anchor = GridBagConstraints.NORTHEAST;
        JLabel fromLabel = new JLabel("To:");
        contentPane.add(fromLabel,constraints);

        constraints.gridx = 1;
//      constraints.gridy = 2;
        constraints.anchor = GridBagConstraints.NORTHWEST;

        toField.setMinimumSize(toField.getPreferredSize());
        contentPane.add(toField,constraints);

        constraints.gridx = 0;
//      constraints.gridy = 2;
        constraints.anchor = GridBagConstraints.NORTHEAST;
        JLabel amountLabel = new JLabel("Amount:");
        contentPane.add(amountLabel,constraints);

        constraints.gridx = 1;
//      constraints.gridy = 2;
        constraints.anchor = GridBagConstraints.NORTHWEST;

        amountField.setMinimumSize(amountField.getPreferredSize());
        contentPane.add(amountField,constraints);

        constraints.gridx = 0;
//      constraints.gridy = 2;
        constraints.anchor = GridBagConstraints.NORTHEAST;

        contentPane.add(clearButton,constraints);
        // ATTENTION!!! The action here is just another
        //   example of how to update JTable. It is
        //   certainly not the logic for clearing the
        //   values in the GUI.




        constraints.gridx = 1;
//      constraints.gridy = 2;
        constraints.anchor = GridBagConstraints.NORTHWEST;

        contentPane.add(transferButton,constraints);

        frame.pack();
        frame.setVisible(true);
    }

    public void transferAddActionListener(ActionListener
al){transferButton.addActionListener(al);}
    public void clearAddActionListener(ActionListener
al){clearButton.addActionListener(al);}
    public void clearInputFields(){fromField.setText(""); toField.setText("");
amountField.setText("");}
    public int getfromNum(){
        return Integer.parseInt(fromField.getText());
    }
    public int getToNum(){
        return Integer.parseInt(toField.getText());
    }
    public int getAmount(){
        return Integer.parseInt(amountField.getText());
    }
   public void updateWholeTable(Vector<Vector<Object>> newVector){
       DefaultTableModel newModel = new DefaultTableModel(newVector, columnNames);
       table.setModel(newModel);
       table.updateUI();
    }
}
```