Joel Edwards
Course: Java Programming 1
Homework 8
April 28, 2011

8.A)

I followed the instructions for the banner calculator, then added some additional features without removing any of the requested functionality.

- Banner displays the calculation result
- Banner's color changes depending on operation/error
- Speed of the banner's scrolling is a function of the calculation's result (0.25 – 0.75 second delay)
- Method for setting the text of the ScrollingLabel is synchronized in order to prevent potential collisions between the scroll thread and the main thread from which CalculatorLogic sets the text, color, and interval of the ScrollingLabel based upon the operation and result.

Source:

CalculatorGUI.java:

```
import java.applet.Applet;
import java.awt.Button;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Label;
import java.awt.TextField;

public class CalculatorGUI
    extends Applet
{
    public static final long serialVersionUID = 1L;

    protected ScrollingLabel scroller = new ScrollingLabel("Roses are
red; violets are blue! ", 250);
    protected Thread scrollThread;

    protected TextField fieldA  = new TextField(20);
    protected TextField fieldB  = new TextField(20);

    protected Label resultLabel = new Label("Results will be
displayed here.");

    protected Button addButton  = new Button("+");
    protected Button subButton  = new Button("-");
    protected Button mulButton  = new Button("X");
    protected Button divButton  = new Button("/");

    protected CalculatorLogic logic = new CalculatorLogic(fieldA,
```

```
fieldB, resultLabel, scroller);

    public void init() {
        // Add widgets to Applet
        add(scroller);
        add(fieldA);
        add(fieldB);
        add(resultLabel);
        add(addButton);
        add(subButton);
        add(mulButton);
        add(divButton);

        // Attached ActionListeners to our buttons
        addButton.addActionListener(logic);
        subButton.addActionListener(logic);
        mulButton.addActionListener(logic);
        divButton.addActionListener(logic);

        // Configure scroller
        scroller.setBackground(Color.ORANGE);
        System.out.println("Attempting to get font...");
        Font font = scroller.getFont();
        String fontName = "Helvatica";
        if (font != null) {
            fontName = font.getFontName();
        }
        Dimension dimensions = new Dimension(390, 30);
        if (scroller.isPreferredSizeSet()) {
            dimensions = new Dimension(390,
scroller.getPreferredSize().height);
        }
        scroller.setPreferredSize(dimensions);
        scroller.setFont(new Font(fontName, Font.BOLD, 20));
        scrollThread = new Thread(scroller);
        scroller.setOwner(scrollThread);
        scroller.pause();
        scrollThread.start();
    }

    public void start() {
        scroller.play();
    }

    public void close() {
        scroller.pause();
    }
}
```

CalculatorLogic.java:

```
import java.awt.Button;
import java.awt.Color;
import java.awt.Label;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.TextComponent;

public class CalculatorLogic
    implements ActionListener
{
    private TextComponent  inputA   = null;
    private TextComponent  inputB   = null;
    private Label          output   = null;
    private ScrollingLabel scroller = null;

    public CalculatorLogic(TextComponent inputA, TextComponent
inputB,
                           Label output, ScrollingLabel scroller) {
        this.inputA = inputA;
        this.inputB = inputB;
        this.output = output;
        this.scroller = scroller;
    }

    public void actionPerformed(ActionEvent evt) {
        double valueA = 0.0;
        double valueB = 0.0;
        double result = 0.0;
        try {
            valueA = Double.parseDouble(inputA.getText());
            valueB = Double.parseDouble(inputB.getText());
            Button source = (Button)evt.getSource();
            String op = source.getLabel();
            if ("+".compareTo(op) == 0) {
                result = valueA + valueB;
                output.setText("" + result);
                updateScroller(result);
                scroller.setScrollingText(String.format("%f + %f = %f
", valueA, valueB, result));
                scroller.setBackground(Color.BLUE);
            } else if ("-".compareTo(op) == 0) {
                result = valueA - valueB;
                output.setText("" + result);
                updateScroller(result);
                scroller.setScrollingText(String.format("%f - %f = %f
", valueA, valueB, result));
                scroller.setBackground(Color.YELLOW);
```

```
            } else if ("X".compareTo(op) == 0) {
                result = valueA * valueB;
                output.setText("" + result);
                updateScroller(result);
                scroller.setScrollingText(String.format("%f X %f = %f
", valueA, valueB, result));
                scroller.setBackground(Color.ORANGE);
            } else if ("/".compareTo(op) == 0) {
                if (valueB == 0.0) {
                    output.setText("Divide by zero!");
                    scroller.setScrollingText("Divide by zero!  ");
                    scroller.setBackground(Color.LIGHT_GRAY);
                } else {
                    result = valueA / valueB;
                    output.setText("" + result);
                    updateScroller(result);
                    scroller.setScrollingText(String.format("%f / %f
= %f  ", valueA, valueB, result));
                    scroller.setBackground(Color.GREEN);
                }
            } else {
                output.setText("Unsupported Operation!");
                scroller.setScrollingText("Unsupported Operation!
");
                scroller.setBackground(Color.LIGHT_GRAY);
            }
        } catch (NumberFormatException e) {
            output.setText("Invalid Input!");
            scroller.setScrollingText("Invalid Input!  ");
            scroller.setBackground(Color.LIGHT_GRAY);
        }
    }

    private void updateScroller(double value) {
        scroller.setDelay(250 + ((int)value % 500));
    }
}
```

ScrollingLabel.java:

```
import java.awt.Label;

public class ScrollingLabel
extends Label
implements Runnable
{
    public static final long serialVersionUID = 1L;

    private int delay = 1000;
    private boolean running = false;
```

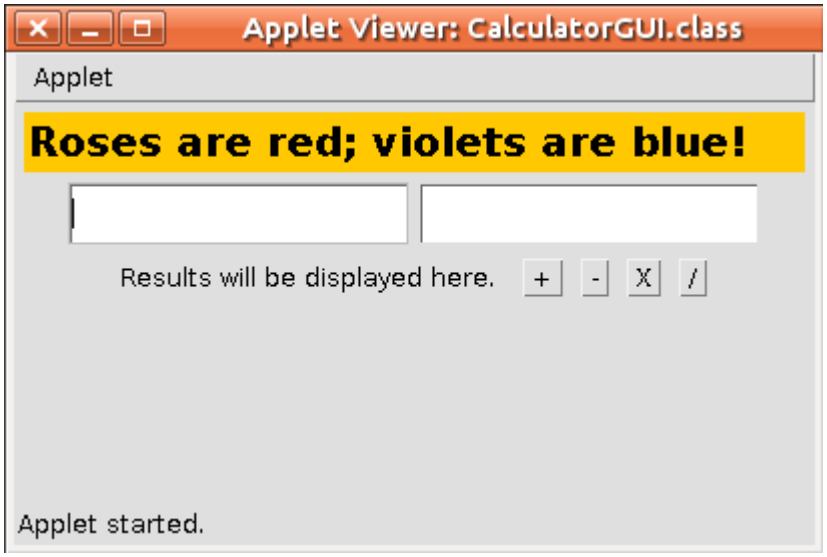
```
private boolean playing = false;
private Thread owner = null;

public ScrollingLabel(String text, int delay) {
    super(text);
    this.delay = delay;
}

public void setOwner(Thread owner) {
    this.owner = owner;
}

public void setDelay(int delay) {
    this.delay = delay;
}

public Thread getOwner() {
    return owner;
}

public int getDelay() {
    return delay;
}

public void run() {
    running = true;
    while (running) {
        try {
            Thread.sleep(delay);
        } catch (InterruptedException e) {;}
        if (playing) {
            setScrollingText(null);
        }
    }
}

public synchronized void setScrollingText(String text) {
    if (text == null) {
        String previous = getText();
        setText(previous.substring(1) + previous.substring(0,1));
    } else {
        setText(text);
    }
}

public void play() {
    playing = true;
}
```

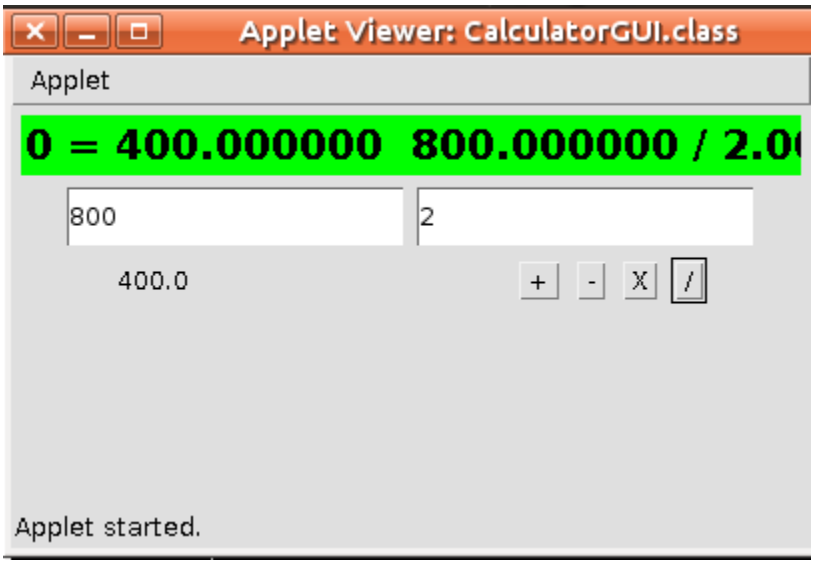
```
    public void pause() {
        playing = false;
    }

    public void halt() {
        running = false;
        if (owner != null) {
            owner.interrupt();
        }
    }
}
```

index.html:

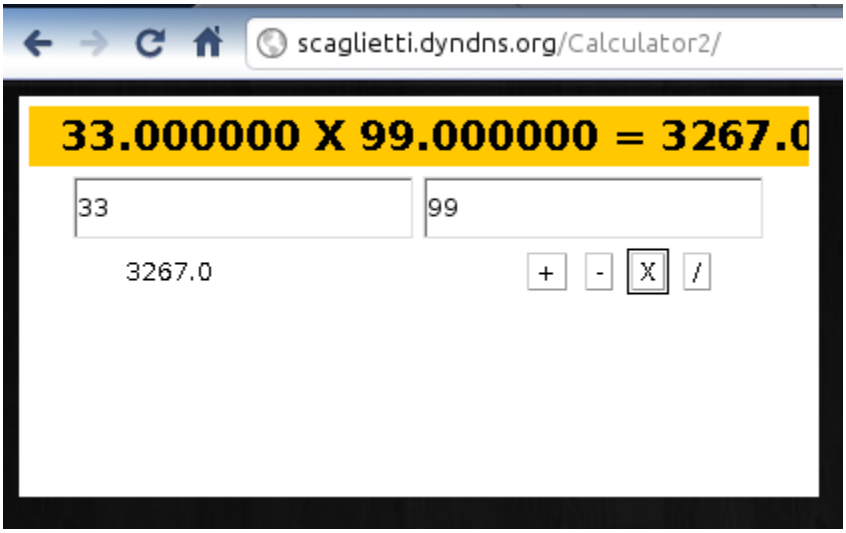
```
<html>
    <head>
        <title>Banner Calculator</title>
        <meta http-equiv="pragma" content="no-cache" />
        <link rel="icon" href="favicon.ico" type="image/x-icon" />
        <link rel="shortcut icon" href="favicon.ico" type="image/x-
icon" />
    </head>
    <body>
        <applet code="CalculatorGUI.class" archive="calculator.jar"
                width="400" height="200">
        </applet>
    </body>
</html>
```

Test Output:





Webpage:
http://scaglietti.dyndns.org/Calculator2/



8.B)

Deposit.java:

```
class Deposit {
    static int balance = 1000;  // simulate the balance kept remotely

    public static void main(String[] args) {
        Account account = new Account();
        DepositThread first, second;

        first  = new DepositThread(account, 1000, "#1");
        second = new DepositThread(account, 1000, "\t\t\t\t#2");

        // start the transactions

        first.start();
        second.start();

        // wait for both transactions to finish

        try {
            first.join();
            second.join();
        } catch (InterruptedException e) {}

        // print out the final balance

        System.out.println("*** Final balance is " + balance);
    }
}
```

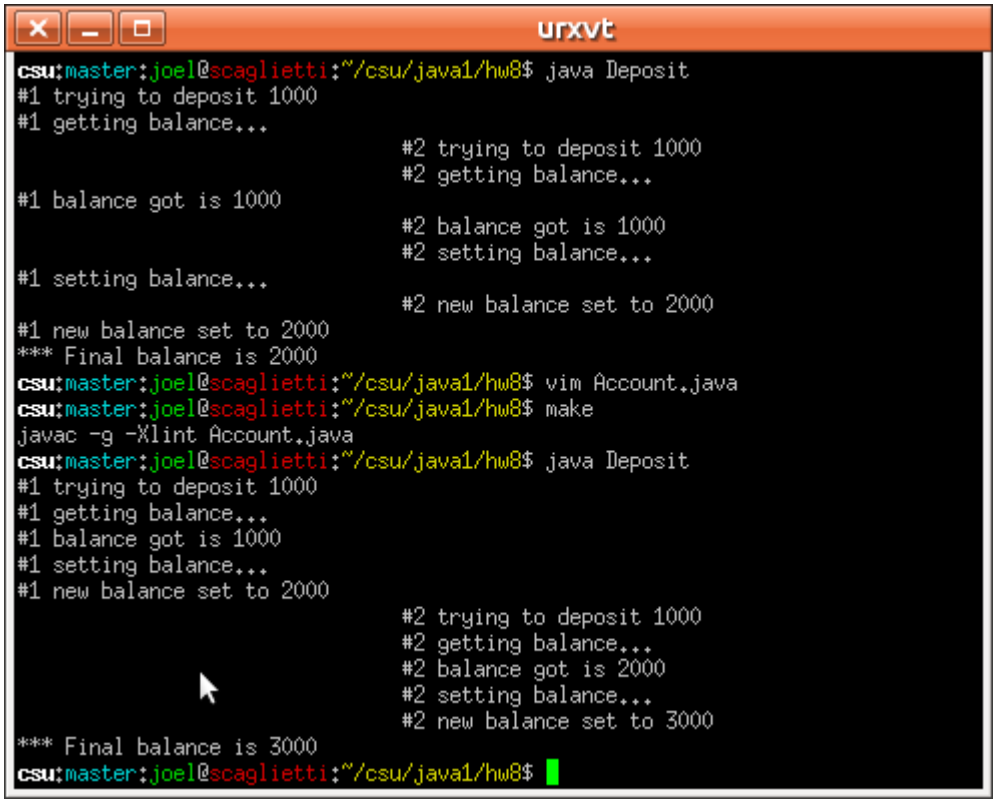
DepositThread.java:

```
class DepositThread extends Thread {

    Account account;
    int     deposit_amount;
    String  message;

    DepositThread(Account account, int amount, String message) {
        this.message  = message;
        this.account  = account;
        this.deposit_amount = amount;
    }

    public void run() {
        account.deposit(deposit_amount, message);
    }
}
```

DepositThread.java:

```
class Account {
    synchronized void deposit(int amount, String name) {
        int balance;

        System.out.println(name + " trying to deposit " + amount);

        System.out.println(name + " getting balance...");
        balance = getBalance();
        System.out.println(name + " balance got is " + balance);

        balance += amount;

        System.out.println(name + " setting balance...");
        setBalance(balance);
        System.out.println(name + " new balance set to " +
                Deposit.balance);
    }

    int getBalance() {
        try {
            // simulate the delay in getting balance remotely
            Thread.sleep(5000);
        } catch (InterruptedException e) {}

        return Deposit.balance;
    }

    void setBalance(int balance) {
        try {
            // simulate the delay in setting new balance remotely
            Thread.sleep(5000);
        } catch (InterruptedException e) {}

        Deposit.balance = balance;
    }
}
```

Test Output:
Below is a screenshot of the account before and after the modification to Account's deposit method.



The following is just a re-work of the account theme from problem B. I successfully built an 'account' package which describes an account that also supports transfers. The program allows the user to specify the number of threads, the delay between sub-operations, and whether the accounts should operate in synchronized or unsynchronized mode.

When the accounts operate in unsynchronized mode, then all operations are interleaved regardless of account. When the accounts operate in synchronized mode, then the only operations which are interleaved are those being performed on different accounts. This is the expected behavior.

There are two accounts, and up to 26 users for these two accounts. They will all attempt to perform the same list of operations on the account simultaneously. If you run with only a single user, you can see the order of operations used by each thread. Make note of the account number when running with multiple threads to confirm that the operation is only being performed on one thread at a time.

The account numbers will increment with every run. They will be reset when you restart the Applet.

I did not include code because this was not assigned, and the code was significantly more complex than either the A or B assignment. The code is available in a ZIP file on the web page.

Output is of the form *C [ACTION] N> Operation_Description* where:
C = *Thread Identifier (an upper-case alpha character)*
ACTION = *The action being performed by this thread*
N = *Account Number (an integer value unique to the account)*
Operation_Description = *Description of the sub-operation of ACTION which is being performed by thread C on account N*

Webpage:
http://scaglietti.dyndns.org/Banking/