Alisha Adam and Rohit Talreja
CS 106A – Summer 2016

Practice Final #2

Final Exam Time: Friday, August 12th, 12:15pm - 3:15pm

Final Exam Location: Split by last name

Last name starting with A – P	**NVIDIA Auditorium**
Last name starting with Q – Z	**Skilling Auditorium**

Based on a past version of this handout by Marty Stepp and Keith Schwarz

This handout is intended to give you practice solving problems that are comparable in format and difficulty to those which will appear on the final exam.

Final Exam is open book, closed notes, closed computer

The examination is open-book (specifically the course textbook *The Art and Science of Java*). You may **not** use any handouts, course notes/slides, printouts of your programs or other notes you've taken in the class. You may **not** use a computer or electronic device of any kind.

Coverage

The final exam covers the material presented throughout the class (with the exception of the Karel material), which means that you are responsible for Chapters 1 through 13 of the class textbook *The Art and Science of Java*.

General instructions

Answer each of the questions included in the exam. Write all of your answers directly on the examination paper, including any work that you wish to be considered for partial credit.

Each question is marked with the number of points assigned to that problem. The total number of points is 180. We intend for the number of points to be roughly comparable to the number of minutes you should spend on that problem.

In all questions, you may include methods or definitions that have been developed in the course, either by writing the `import` line for the appropriate package or by giving the name of the method and the textbook chapter number in which that definition appears.

Unless otherwise indicated as part of the instructions for a specific problem, comments will not be required on the exam. Uncommented code that gets the job done will be sufficient for full credit on the problem. On the other hand, comments may help you to get partial credit if they help us determine what you were trying to do.

Blank pages for solutions omitted in practice exam (but will be available on real exam)

In an effort to save trees, the blank pages that would be provided in a regular exam for writing your solutions have been omitted from this practice exam.

Problem 1: Value/Reference Semantics Trace (15 points)

The following program produces 4 lines of output. Write the output below, as it would appear on the console.

```
public class ReferenceMystery1 extends ConsoleProgram {
    public void run() {
        int y = 1;
        int x = 3;
        int[] a = new int[4];

        mystery(a, y, x);
        println(x + " " + y + " " + Arrays.toString(a));

        x = y - 1;
        mystery(a, y, x);
        println(x + " " + y + " " + Arrays.toString(a));
    }

    public void mystery(int[] a, int x, int y) {
        if (x < y) {
            x++;
            a[x] = 17;
        } else {
            a[y] = 17;
        }
        println(x + " " + y + " " + Arrays.toString(a));
    }
}
```

Problem 2: Multi-dimensional Arrays Trace (15 points)

Consider the following method. For each multi-dimensional array listed below, write the **final array state** that would result if the given array were passed as a parameter to the method.

```
public void array2dMystery3(int[][] a) {
    for (int r = 0; r < a.length - 1; r++) {
        for (int c = 0; c < a[0].length - 1; c++) {
            if (a[r][c + 1] > a[r][c]) {
                a[r][c] = a[r][c + 1];
            } else if (a[r + 1][c] > a[r][c]) {
                a[r][c] = a[r + 1][c];
            }
        }
    }
}
```

<u>Method Call</u> <u>Final Array State</u>

a)

```
int[][] a1 = {
    {3, 4, 5, 6},
    {4, 2, 6, 1},
    {1, 6, 7, 2}
};
array2dMystery3(a1);
```

____

b)

```
int[][] a2 = {
    {1, 2, 3, 0, 5},
    {2, 4, 6, 8, 10},
    {9, 5, 1, 2, 4}
};
array2dMystery3(a2);
```

____

Problem 3: Collections Trace (15 points)

Write the output produced when the following method is passed each of the following maps. It does not matter what order the key/value pairs appear in your answer, so long as you have the right overall set of key/value pairs.

```
public void collectionMystery1(HashMap<String, String> map) {
    HashMap<String, String> result = new HashMap<String, String>();
    for (String k : map.keySet()) {
        String v = map.get(k);
        if (k.charAt(0) <= v.charAt(0)) {
            result.put(k, v);
        } else {
            result.put(v, k);
        }
    }
    println(result);
}
```

Map	Output
a) `{two=deux, five=cinq, one=un, three=trois, four=quatre}`	________________
b) `{skate=board, drive=car, program=computer, computer=awesome}`	________________
c) `{siskel=ebert, girl=boy, heads=tails, ready=begin, first=last}`	________________

Problem 4: ConsoleProgram (30 points)

Suppose you want to hold a never-ending birthday party, where every day of the year someone at the party has a birthday. How many people do you need to get together to have such a party?

Your task in this program is to write a method **`neverEndingBirthdayParty`** that simulates building a group of people one person at a time. Each person is presumed to have a birthday that is randomly chosen from all possible birthdays. Once it becomes the case that each day of the year, someone in your group has a birthday, your program should return the number of people in the group and exit.

In writing your solution, you should assume the following:

- There are 366 possible birthdays (this includes February 29).
- All birthdays are equally likely, including February 29.

  You might find it useful to represent birthdays as integers between 0 and 365, inclusive.

Problem 5: Arrays (20 points)

Write a method named **`longestSortedSequence`** that accepts an array of integers as a parameter and that returns the length of the longest sorted (non-decreasing) sequence of integers in the array. For example, if a variable named array stores the following values:

```
int[] array1 = {3, 8, 10, 1, 9, 14, -3, 0, 14, 207, 56, 98, 12};
```

Then the call of `longestSortedSequence(array1)` should return 4 because the longest sorted sequence in the array has four values in it (the sequence -3, 0, 14, 207). Notice that sorted means non-decreasing, which means that the sequence could contain duplicates. For example, if the array stores the following values:

```
int[] array2 = {17, 42, 3, 5, 5, 5, 8, 2, 4, 6, 1, 19};
```

Then the method would return 5 for the length of the longest sequence (the sequence 3, 5, 5, 5, 8). Your method should return 0 if passed an empty array. Your method should return 1 if passed an array that is entirely in decreasing order or contains only one element.

Constraints: You may not use any auxiliary data structures (arrays, lists, strings, etc.) to solve this problem. Your method should not modify the array that is passed in.

Problem 6: Classes and Objects (25 points)

Suppose that you are provided with a pre-written class `BankAccount` as described at right. (The headings are shown, but not the method bodies, to save space.) Assume that the fields, constructor, and methods shown are already implemented. You may refer to them or use them in solving this problem if necessary.

Write a method named **`transactionFee`** that will be placed inside the `BankAccount` class to become a part of each `BankAccount` object's behavior. The `transactionFee` method accepts a fee amount (a real number) as a parameter, and applies that fee to the user's past transactions. The fee is applied once for the first transaction, twice for the second transaction, three times for the third, and so on. These fees are subtracted out from the user's overall balance. If the user's balance is large enough to afford all of the fees with greater than $0.00 remaining, the method returns true. If the balance cannot afford all of the fees, the balance is left as 0.0 and the method returns false.

For example, given the following `BankAccount` object:

```
BankAccount savings =
    new BankAccount("Jimmy");
savings.deposit(10.00);
savings.deposit(50.00);
savings.deposit(10.00);
savings.deposit(70.00);
```

The account at that point has a balance of $140.00. If the following call were made:

```
savings.transactionFee(5.00)
```

Then the account would be deducted $5 + $10 + $15 + $20 for the four transactions, leaving a final balance of $90.00. The method would return true. If a second call were made,

```
savings.transactionFee(10.00)
```

Then the account would be deducted $10 + $20 + $30 + $40 for the four transactions, leaving a final balance of $0.00. The method would return `false`.

```
// A BankAccount keeps track of a
// user's money balance and ID,
// and counts how many transactions
// (deposits/withdrawals) are made.

public class BankAccount {
    private String id;
    private double balance;
    private int transactions;

    // Constructs a BankAccount
    // object with the given id, and
    // 0 balance and transactions.
    public BankAccount(String id)

    // returns the field values
    public double getBalance()
    public String getID()

    // Adds the amount to the balance
    // if it is between 0-500.
    // Also counts as 1 transaction.
    public void deposit(double amount)

    // Subtracts the amount from
    // the balance if the user has
    // enough money.
    // Also counts as 1 transaction.
    public void withdraw(double amount)


    // your method would go here

}
```

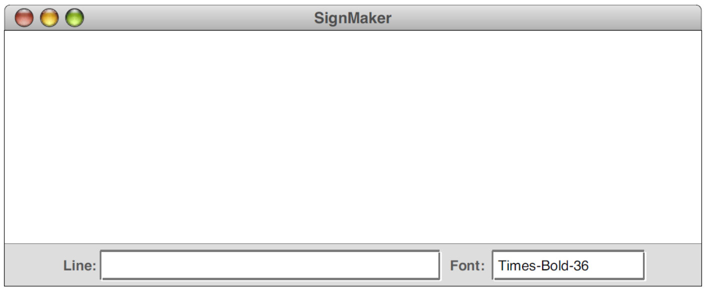
Problem 7: Collections (30 points)

Write a method named **`isSubMap`** that accepts two hash maps from strings to strings as its parameters and returns true if every key in the first map is also contained in the second map and maps to the same value in the second map. For example, `{Smith=949-0504, Marty=206-9024}` is a sub-map of `{Marty=206-9024, Hawking=123-4567, Smith=949-0504, Newton=123-4567}`. The empty map is considered to be a sub-map of every map.

Constraints: You may not declare any auxiliary data structures in solving this problem.

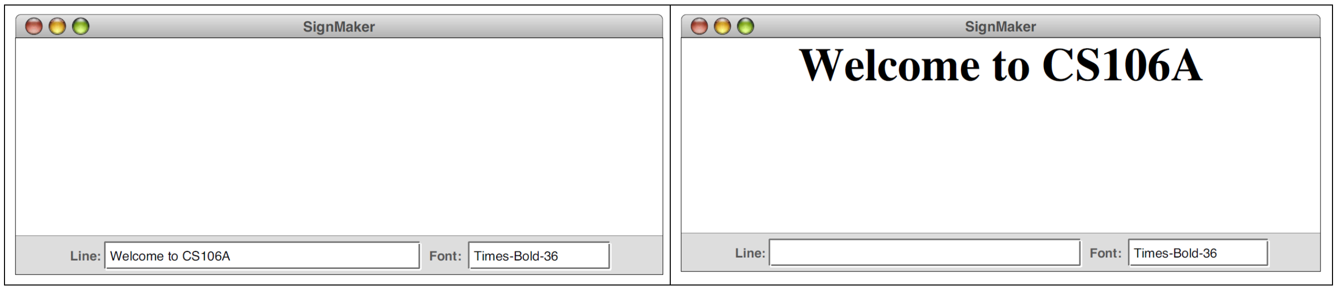
Problem 8: Graphical User Interfaces (30 points)

Write a complete program named **`SignMaker`** that implements a graphical user interface for creating simple signs, each of which consists of lines of centered text displayed in different fonts. When you start the program, the user interface looks like the screenshot below. In its bottom region it contains a 30-character-wide text field labeled "Line:" and a 15-character-wide text field labeled "Font:". The initial text of the "Line" field is blank, and the initial text of the "Font" Field is "Times-Bold-36".



Times-Bold-36

You can then add a line to the display by entering text into the Line field and pressing Enter, as shown in the screenshots below. That label should be centered in the window and set in the font specified in the 15-character-wide Font text field. The first label you add should be positioned so that its text is at the very top of the window. The first line shown is set in Times-Bold-36 and appears very close to the top of the window. Pressing ENTER also clears the text field in the control strip making it easier for the user to enter the next line of the sign.



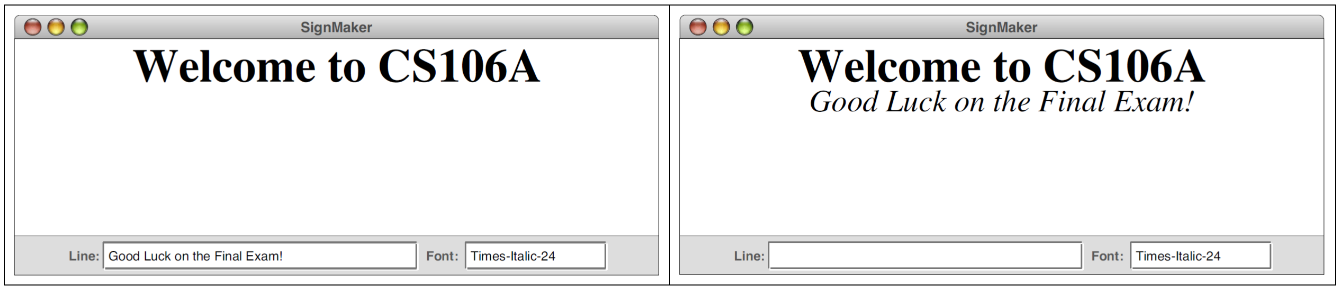
Welcome to CS106A

Times-Bold-36

Welcome to CS106A

Times-Bold-36

The user can change the font of each subsequently added line by typing a new font name in the Font field. For example, if the user wanted to add a second line to the message is a smaller, italic font, that user could do so by changing the contents of the Font field to Times-Italic-24 and then typing in a new message. Typing Enter at this point would add a new centered label below the first one. The distance to the next baseline from the previous one should be the height of the new label you're adding. The screenshots below show the window state before and after adding a second label.



Welcome to CS106A

Good Luck on the Final Exam!

Times-Italic-24

Welcome to CS106A

Good Luck on the Final Exam!

Times-Italic-24

You may assume that the user types valid font strings into the Font field.