This is an open-book exam. You have 75 minutes to complete the exam. There are 5 questions on this examination. The point values for the questions are shown in the table below. Your answers should fit in the space provided on the exam. Read each question carefully before responding.

## SAMPLE SOLUTIONS

Anonymous ID #: __________________________________________

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>Writing your ID above</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

Please complete the Honor Code statement on the last page of the exam booklet.
Question 1

Phone keypads usually display small groups of letters under the numbers that label the keys. For example, in the image of the iPhone shown on the right, the letters A, B, and C appear under 2 and J, K, and L appear under 5. Some companies take advantage of these letters by choosing phone numbers that correspond to words or phrases that connect to their business such as 1 (800) FLOWERS (a.k.a. 1 (800) 3569377) or 1 (800) THRIFTY (which is a good number if you need to rent a car).

Most of us find such numbers annoying because we cannot dial them as fast as a sequence of digits. So, it might be helpful to have a little program that can translate phone numbers including at least some letters into purely numeric form.

An interface for a simple Java program that serves this purpose is shown below.

![Image of phone keypad with letters and numbers]

The program's interface includes a text area in which the user can enter a phone number including letters. When the user has completed entering a number and presses return, the program displays the letter-free equivalent in a JLabel.

An almost complete version of the code for this program is shown on the next page. The only problem is that the body of the `textEntered` method (which does all the work!) is missing. Three possible implementations of `textEntered` labeled (a), (b) and (c) follow the incomplete code. One of the three is correct. The other two have small flaws (i.e., they can be fixed by changing one or two lines or by moving a curly brace back or forth a few lines). For each of these examples, you should either state that it is correct or a) briefly explain how it will fail (i.e., loop forever, produce an incorrect answer, encounter an error such as a string index that is out of range) and b) describe a small change that will correct the code.

All of the versions of `textEntered` given below depend on a private method named `letterToDigit` which expects a single capital letter as a parameter and returns the single digit corresponding to the key on which the letter appears on the phone keypad.

You should not need to understand how `letterToDigit` performs its function, but the idea is fairly simple. The method depends on a `String` named `alpha` which includes all of the letters of the alphabet interspersed with spaces in such a way that if the string is broken up into consecutive substrings of length 4, each substring contains the letters that appear together on one key of the phone keypad (usually together with one or more spaces). As a result, if the position at which a letter is found in `alpha` is divided by 4 and the result rounded down to the nearest integer (which Java does by default), the result will be 1 less than the number on the key displaying the letter.
import squint.*;
import javax.swing.*;

public class PhoneWords extends GUIManager {

    // Change these values to adjust the size of the program's window
    private final int WINDOW_WIDTH = 350, WINDOW_HEIGHT = 100;

    // The letters of the alphabet arranged in groups as they appear on the keys
    private static final String alpha = "           ABC DEF GHI JKL MNO PQRSTUVWXYZ";

    // Field used to enter phone "number"
    private JTextField entry = new JTextField( 12 );

    // Label used to display the result
    private JLabel display = new JLabel( "" );

    public PhoneWords() {
        // Create window and add all the components
        this.createWindow( WINDOW_WIDTH, WINDOW_HEIGHT );
        contentPane.add( new JLabel( "Enter a phone number:" ) );
        contentPane.add( entry );
        contentPane.add( display );
    }

    // Convert a single letter to the number of the key on which it appears
    private String letterToDigit( String letter ) {
        return alpha.indexOf( letter )/4 + 1 + "";
    }

    // When the user presses return, convert the phone number entered into
    // all numbers.
    public void textEntered( ) {
        ...
    }
}
For each of the examples below, either state that it is correct or a) briefly explain how it will fail (i.e., loop forever, produce an incorrect answer, encounter an error such as a string index that is out of range) and b) describe a small change that will correct the code.

(a) // When the user presses return, convert the phone number entered into // all numbers.
   public void textEntered( ) {
      String number = entry.getText();
      String result = "";
      int p = 0;
      while ( p < number.length() ) {
         String letter = number.substring( p, p+1 );
         if ( ! letter.equals(" ") && alpha.contains( letter ) ) {
            result = result + letterToDigit( letter );
         }
         p = p + 1;
      }
      display.setText( result );
   }

The version above only updates result if the variable letter refers to a letter. As a consequence, at the end of the looping, result will contain digits for all of the letters in the phone number entered, but nothing for any of the other symbols entered. For example, 1 (413) 59PIZZA would produce the result 74992.

A good way to fix this is to include an else branch like the one seen in option (b).

(b) // When the user presses return, convert the phone number entered into // all numbers.
   public void textEntered( ) {
      String number = entry.getText();
      String result = "";
      int p = 0;
      while ( p < number.length() ) {
         String letter = number.substring( p, p+1 );
         if ( ! letter.equals(" ") && alpha.contains( letter ) ) {
            result = result + letterToDigit( letter );
         } else {
            result = result + letter;
         }
         p = p + 1;
      }
      display.setText( result );
   }

This version will function correctly.
(c) // When the user presses return, convert the phone number entered into
// all numbers.
public void textEntered( ) {
    String number = entry.getText();
    int p = 0;
    String result = "";
    while ( p < number.length() ) {
        String letter = number.substring( p, p+1 );
        if ( ! letter.equals( " ") && alpha.contains( letter ) ) {
            letter = letterToDigit( letter );
            result = result + letter;
            p = p + 1;
        }
    }
    display.setText( result );
}

This version will get stuck in a loop if any of the symbols in the input provided
are not letters. If letter is a digit or any other non-alphabetic symbol, the as-
signment p = p + 1; will not be executed so the loop will get stuck at that posi-
tion p and just repeatedly execute the if statement without changing p or result.

The easiest way to fix this version is to move the brace that ends the if statement
two lines earlier so that the lines to add letter to result and 1 to p both fall
outside of the if statement. In this way, the loop will progress to the next letter
in “number” each time it executes and always add either the original symbol associat-
ed with letter or the digit corresponding to letter when an capital letter is en-
countered.
Question 2

a) Show how the message ERRATA ARE RARE would be encoded using the Huffman tree shown on the right (which may or may not be a correct Huffman tree for this message) by completing the process of filling in the lower cells in the table below. The first three cells are already filled in to get you off to a good start. A spare copy of the table is provided in case you make a mistake on your first try. If you use both copies, clearly indicate which one we should grade!

```
  E  R  R  A  T  A  _  A  R  E  _  R  A  R  E
1 001 001 000 01 000 001 000 001 1 000 001 000 001 1
```

```
  E  R  R  A  T  A  _  A  R  E  _  R  A  R  E
1 001 001
```

b) In the space provided below, construct a Huffman tree for the message ERRATA ARE RARE. An occurrence count table for this message is provided on the right.

```
<table>
<thead>
<tr>
<th>Letter</th>
<th>Occurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>_</td>
<td>2</td>
</tr>
<tr>
<td>E</td>
<td>3</td>
</tr>
<tr>
<td>A</td>
<td>4</td>
</tr>
<tr>
<td>R</td>
<td>5</td>
</tr>
<tr>
<td>TOTAL</td>
<td>15</td>
</tr>
</tbody>
</table>
```
Below you will find several short phrases (including ERRATA ARE RARE) that use only the four letters A, E, R, and T (and spaces to separate words). In addition, below each phrase you will find a table showing how often each of the five symbols appears in the phrase. For this problem, we would like you to consider the process of sending each message using a separate Huffman code that is customized to minimize the bits required to transmit that message.

<table>
<thead>
<tr>
<th>Letter</th>
<th>Occurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>_</td>
<td>2</td>
</tr>
<tr>
<td>E</td>
<td>3</td>
</tr>
<tr>
<td>A</td>
<td>4</td>
</tr>
<tr>
<td>R</td>
<td>5</td>
</tr>
<tr>
<td>TOTAL</td>
<td>15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Letter</th>
<th>Occurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>2</td>
</tr>
<tr>
<td>R</td>
<td>2</td>
</tr>
<tr>
<td>_</td>
<td>3</td>
</tr>
<tr>
<td>A</td>
<td>4</td>
</tr>
<tr>
<td>T</td>
<td>5</td>
</tr>
<tr>
<td>TOTAL</td>
<td>16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Letter</th>
<th>Occurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>_</td>
<td>2</td>
</tr>
<tr>
<td>R</td>
<td>3</td>
</tr>
<tr>
<td>T</td>
<td>3</td>
</tr>
<tr>
<td>E</td>
<td>6</td>
</tr>
<tr>
<td>TOTAL</td>
<td>15</td>
</tr>
</tbody>
</table>

Consider the four Huffman trees over the 5 symbol alphabet used by our three sample messages.

1) 

2) 

3) 

4) 

6 of 15
c) Because there are often steps in the algorithm for constructing a Huffman tree where one has a choice of which symbols to merge, the process of applying the Huffman algorithm to a message may produce distinct trees for a single message. In fact, at least two of the four trees shown above are valid Huffman trees for one of our three messages. Identify the message and the two trees.

“A TREE TEETERER” could lead to either tree 2 or 4.
Question 3

Suppose you were given the job of writing the phone app for a new smartphone. You would need to write code to produce a phone keypad like the one shown in the window on the right.

Complete Java code to produce such a window is shown on the next page. It takes advantage of a feature of JButtons we have not previously revealed. The text of a JButton can be described using HTML to display text that uses varying font sizes and separates the text into multiple lines. The method makePhoneKey defined on the next page uses this feature to create buttons displaying a digit in large font over a short sequence of letters in a smaller font. You need not understand the details of this method and the HTML it uses to answer the question.

While this code works as desired, the section of code that produces buttons 1 through 9 is very repetitive. It would be better to replace these lines with a short loop. In the space provided below, please provide less than nine lines of code including a loop that will add the nine buttons for 1 through 9 to the program’s window. The string alpha which is declared in the class (but not currently used), should prove helpful when writing this loop. Like the similar string used in the code of Question 1, if the string alpha is divided into subsequences of length 4, each subsequence includes the letters displayed on one of the first 9 buttons (and possibly some extra spaces).

// PLACE THE CODE INCLUDING YOUR LOOP HERE:

```java
int b = 1;
String letters = alpha;
while ( b <= 9 ) {
    contentPane.add( makePhoneKey( b, letters.substring(0,4)));
    letters = letters.substring( 4 );
    b = b + 1;
}
```
or
```
int b = 1;
while ( b <= 9 ) {
    contentPane.add( makePhoneKey( b, alpha.substring( 4*b-4, 4*b )));
    b = b + 1;
}
```
or many other variations.
public class PhoneKeyPad extends GUIManager {

    // Change these values to adjust the size of the program's window
    private final int WINDOW_WIDTH = 240, WINDOW_HEIGHT = 300;

    // The letters of the alphabet arranged in groups as they appear on the keys
    private static final String alpha = "    ABC DEF GHI JKL MNO PQRSTUV WXYZ";

    // Snippets of HTML needed to display text on multiple lines within a JButton
    private final String HTML_PREFIX = "<html><body><center><font size=6>";
    private final String HTML_SUFFIX = "</font></center></body></html>";
    private final String HTML_SEP = "</font><br><font size=3>";

    // Display buttons forming a phone keypad in a window.
    public PhoneKeyPad() {
        // Create window to hold all the components
        this.createWindow( WINDOW_WIDTH, WINDOW_HEIGHT );

        // Add the buttons for digits 1 through 9
        contentPane.add( makePhoneKey( 1, " " ));
        contentPane.add( makePhoneKey( 2, "ABC ");
        contentPane.add( makePhoneKey( 3, "DEF ");
        contentPane.add( makePhoneKey( 4, "GHI ");
        contentPane.add( makePhoneKey( 5, "JKL ");
        contentPane.add( makePhoneKey( 6, "MNO ");
        contentPane.add( makePhoneKey( 7, "PQRS"));
        contentPane.add( makePhoneKey( 8, "TUV ");
        contentPane.add( makePhoneKey( 9, "WXYZ" ));

        // Add the buttons for 0 and special symbols.
        contentPane.add( new JButton( HTML_PREFIX + "*" + HTML_SUFFIX ));
        contentPane.add( makePhoneKey( 0, "+" ));
        contentPane.add( new JButton( HTML_PREFIX + "#" + HTML_SUFFIX ));
    }

    // Create a JButton displaying a digit in a large font over a short
    // string displayed in a smaller font.
    private JButton makePhoneKey( int upper, String lower ) {
        return new JButton( HTML_PREFIX + upper + HTML_SEP + lower + HTML_SUFFIX );
    }
}
Question 4

The program on the next page draws five buttons in a window. These buttons -- button1, button2, button3, button4, and button5 -- are labeled “WITCH”, “VAMPIRE”, “GHOST”, “GHOUL”, and “BAT”, respectively, as shown below. Pressing the buttons changes the labels of various buttons.

![Diagram of buttons labeled WITCH, VAMPIRE, GHOST, GHOUL, and BAT.]

Indicate the labels on each of the buttons after the buttonClicked method is invoked in response to the sequences of button clicks described below. Note that you should assume you start over from scratch each time.

i) Start from scratch. Then click button1 and button5 in sequence.

![Sequence of buttons labeled WITCH, AND, THE, BLACK, and BAT.]

ii) Start from scratch. Then click button1, button4, and button3 in sequence.

![Sequence of buttons labeled BEWARE, OF, THE, SCAREDY, and CAT.]

iii) Start from scratch. Then click button1, button2, button3, button4, and button5 in sequence.

![Sequence of buttons labeled BEWARE, OF, THE, BLACK, and BAT.]

import statements removed to save space

public class ExamButtons extends GUIManager {
    private JButton button1, button2, button3, button4, button5;

    public ExamButtons() {
        this.createWindow( 600, 150 );
        button1 = new JButton( "WITCH" );
        button2 = new JButton( "VAMPIRE" );
        button3 = new JButton( "GHOST" );
        button4 = new JButton( "GHOUL" );
        button5 = new JButton( "BAT" );
        contentPane.add( button1 );
        contentPane.add( button2 );
        contentPane.add( button3 );
        contentPane.add( button4 );
        contentPane.add( button5 );
    }

    public void buttonClicked( JButton which ) {
        button3.setText( "THE" );
        if ( which == button1 ) {
            button2.setText( "AND" );
            button3.setText( "HAT" );
        }
        if ( which == button2 ) {
            button1.setText( "BEWARE" );
            button4.setText( "CHOCOLATE" );
            button5.setText( "CAT" );
        }
        if ( which == button3 ) {
            if ( button5.getText().equals( "BAT" ) ) {
                button5.setText( "CAT" );
            } else {
                button5.setText( "BAT" );
                button1.setText( "NEVER" );
            }
            button2.setText( "OF" );
        }
        if ( which == button4 ) {
            button1.setText( "BEWARE" );
            button3.setText( "ANOTHER" );
        }
        if ( which == button5 ) {
            button4.setText( "BLACK" );
        } else if ( which == button3 ) {
            button4.setText( "SCAREDY" );
        }
    }
}
Question 5

We can describe a transmission scheme by showing the patterns of energy flow used to represent 0 and 1. For example, we described Manchester encoding by showing the diagrams:

![Manchester encoding diagrams](image)

Given such diagrams, we can easily determine how to encode pairs or longer sequences of bits. For example, using Manchester encoding the four possible pairs of 0s and 1s would be encoded as:

![Manchester encoding pairs](image)

An alternative approach is to think about the patterns used to encode pairs of bits directly rather than encoding individual bits. For example, we could use the encodings:

![Pairwise encoding diagrams](image)

Here we are dividing the time used to encode a pair of bits into thirds and using the pattern of transitions at the two boundaries between the thirds to encode the pair of bits. For the purposes of this problem we will refer to this approach as “pairwise encoding”. For example, while the Manchester encoding for 01000111 would be

![Manchester encoding example](image)

the pairwise encoding for the same binary sequence would be:

![Pairwise encoding example](image)

Of course, this approach can only be used if the number of bits to be encoded is even. Given that a byte contains an even number of bits, this will usually be true in practice.

Use the preceding description of pairwise encoding in conjunction with your knowledge of on-off keying and Manchester encoding to answer the following questions.
a) Consider the signal shown below. Assuming this signal represents binary data encoded using on-off keying with each 8-bit byte of data preceded by its own start bit, what data would this signal encode? For this subpart of the problem only, you may assume that the dotted vertical lines represent bit time boundaries.

![Signal Diagram](image)

START BIT 0 0 1 0 0 1 1 1
START BIT 0 0 1 1 1 0 0 0

b) Consider the same signal again. Could this signal be interpreted as a sequence of bits encoded using the pairwise encoding (described in the introduction to this problem) preceded by a start pair (i.e., a 01 encoded using pairwise encoding)? If not, briefly explain why. Otherwise, indicate the shortest sequence of bits the message might encode. Do not assume that there will be exactly 8 bits. Do not assume the vertical lines indicate bit times.

![Signal Diagram](image)

START PAIR 0 1 1 1 1 0 0 1 1 1 1 1 0
c) The main disadvantage of Manchester encoding is that, in the worst case, the number of on/off changes that occur in the signal may be twice the number of bits transmitted (ignoring start sequences). In the worst case, how many transitions per bit may occur when pairwise encoding is used. Describe a sequence of bits that would require this rate of transitions when encoded.

*If the sequence:*

Which represents 0001 were sent repeatedly it would require 3 transition for each pair of bits sent or 1.5 transitions per bit. This is the highest transition rate possible with pairwise encoding and is less than the 2 transitions per bit rate of Manchester encoding.
HONOR CODE STATEMENT

I have neither given nor received aid on this examination.

Signature: ____________________________