Practice Final Exam

This "open-book" examination will last approximately 2 hours. You may use any notes, texts, or other references at your disposal; however, we advise you not to waste too much time leafing through your materials.

This exam contains 100 points; in order to properly budget your time, plan on spending a little more than one minute per point. If you succeed in budgeting your time like this, you may have 20 minutes or more left over to check your work. Note, however, some of you might find this to be a long exam, so don't worry if you don't actually have time to check your work or if you can't completely finish solving all the problems.

Do not bother to "dummy-proof" your code (e.g., you need not check for valid input values) unless specifically asked to do so.

Show all of your work right here on the exam. If any question appears ambiguous or especially peculiar, ask for help! It is silly to misinterpret a question that you are capable of answering.

As always, *Best of luck!*

Your Name ______________________ Your TF's Name ______________________

[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	TOTAL

Dr. H. H. Leitner

Part A — 2 Java Execution Questions *15 points total*

For each of the following two questions, indicate precisely what will be printed (or drawn) when the program is executed. Hint: Each of these programs does compile! For possible partial credit, show all of your work in the limited space provided.

[1] 6 points

```
public static void prob1 (ArrayList<Integer> list)
{
    for (int i = list.size() - 1; i >= 0; i--)
    {
        if (i % 2 == 0) list.add (list.get(i));
        else list.add(0, list.get (i));

    }
    System.out.println (list);
}
```

What output is produced if the **prob1** method is passed an *ArrayList* containing the values **[10, 20, 30]** ?

[2] 9 points

```
import java.awt.*;
import javax.swing.*;


public class Prob2

{

   public static void main(String[] args)

   {
      JPanel center = new JPanel(new GridLayout(2, 2));
      center.add(new JButton ("Button4"));

      center.add(new JButton ("Button6"));
      center.add(new JButton ("Button5"));
      center.add(new JButton ("Button7"));
      JPanel north = new JPanel(new GridLayout(1, 3));
      north.add(new JButton("Button1"));
      north.add(new JButton("Button2"));
      north.add(new JButton("Button3"));

      JPanel south = new JPanel();
      south.add(new JLabel("Type stuff:"));
      south.add(new JTextField(10));

      JFrame frame = new JFrame ("Good thing I studied!");
      frame.setSize(285, 400);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.add(north, BorderLayout.NORTH);
      frame.add(center, BorderLayout.CENTER);
      frame.add(south, BorderLayout.SOUTH);
      frame.setVisible(true);
   }

}
```

// Draw the output of this program as precisely as you can!

Part B — Short Problems *45 points total*

[3] 9 points

Assume that the following four classes have been defined:

```
public class Truman extends FDR  // first class
{
    public void democrat() {
            System.out.println ("Truman - D");
    }
}


public abstract class Prez  // second class
{
    public void republican()
    {
        System.out.println ("Republican Prez");
    }
    public void democrat()
    {
        republican();
        System.out.println ("Democratic Prez ");
    }

    public String toString() {
            return "I am the President!";
    }
}

public class Lincoln extends Prez  // third class
{
    public void republican ()
    {
        super.republican ();
        System.out.println ("Lincoln-R");
    }
}


public class FDR extends Prez    // fourth class
{
    public String toString()
    {
        return "FDR";
    }

    public void democrat()
    {
        System.out.println ("FDR - D");
        super.democrat();
    }
}
```

Given the classes above, precisely what output is produced by the following code?

```
Prez [] p = { new FDR(), new Lincoln(), new Truman() };

for (int i = 0; i < p.length; i++)
{
    System.out.println (p[i]);
    p[i].democrat();
    p[i].republican();
    System.out.println();
}
```

[4] **16 points** **(Stacks** and **Queues)**

Write a method called **isSorted** that takes a stack of integers and returns **true** if the stack is sorted and **false** otherwise. A stack is considered sorted when its integers are in non-decreasing order (i.e. increasing order with duplicates allowed) when read from bottom to top.

So, a *sorted stack* has its smallest integer on the bottom and its largest integer on the top. A stack that contains fewer than two integers is sorted by definition. For example, suppose that a variable named **s** stores the following sequence of values:

```
bottom [-12, 0, 1, 8, 8, 8] top
```

then a call on **`isSorted(s)`** should return *true*.

If **s** had instead contained the following values:

```
bottom [-9, 10, 43, 24, 97] top
```

then a call on isSorted(s) should return false, because 24 is less than 43.

You may use one *queue* as auxiliary storage to solve this problem. You may not use any other auxiliary data structures to solve this problem, although you can have as many simple variables as you like. You may not use recursion to solve this problem. Assume the *Queue* and *Stack* classes discussed in lecture (along with their associated methods) are available for your use.

You have access to the following two methods and may call them as needed to help you solve the problem:

```
public static void s2q (Stack<Integer> s, Queue<Integer> q)
{
    while (!s.isEmpty()) q.add(s.pop());
       // Transfers the entire contents
       // of stack s to queue q
}

public static void q2s (Queue<Integer> q, Stack<Integer> s)
{
    while (!q.isEmpty()) s.push(q.remove());
      // Transfers the entire contents
      // of queue q to stack s
}
```

*In answering question [5] assume the **Bitset** class has been defined as discussed in lecture and in a course handout. We have reproduced the first part of this code in an Appendix at the end of this document, for your convenience. You may assume the **byteArray** instance variable is **public** (not private) in answering these questions.*

[5] 7 points

Suppose a **`Bitset`** named **`setA`** has been declared and properly initialized to some value.

What would be the effect of the statement

```
setA.byteArray[0] ^= 17;
```

Express your answer in terms of setA's resulting "set value." For example, your answer might state something like "the value 17 is removed from setA if that value was already present. Otherwise, this statement has no effect." (You may assume that the instance variable **byteArray** has been made *public* for the purpose of solving this problem.)

[6] 13 points

Recall the **`ListNode`** class as defined in lecture, except now the data fields store integer values instead of *String* object:

```
public class ListNode
{
    private int data;
    private ListNode link;

    public ListNode ()
    {
        link = null;
        data = null;
    }


    public ListNode (int newData, ListNode linkValue)
    {
        data = newData;
        link = linkValue;
    }
```

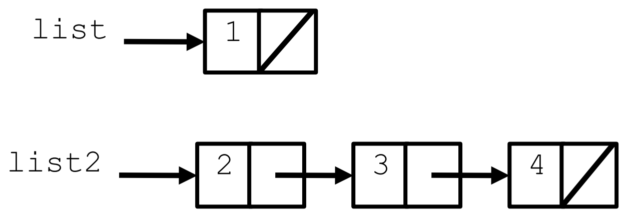
```
        public void setData (int newData)
        {
            data = newData;
        }


        public int getData ()
        {
            return data;
        }


        public void setLink (ListNode newLink)
        {
            link = newLink;
        }

        public ListNode getLink()
        {
            return link;
        }
    }
```

Suppose the following linked lists have been created:



Draw a precise picture to show the value of **list** and **list2** after the following code gets executed:

```
list2.getLink().getLink().setLink (list);
list = list2.getLink ();
list.getLink().getLink().setLink (null);
list.getLink().setData (17);
```

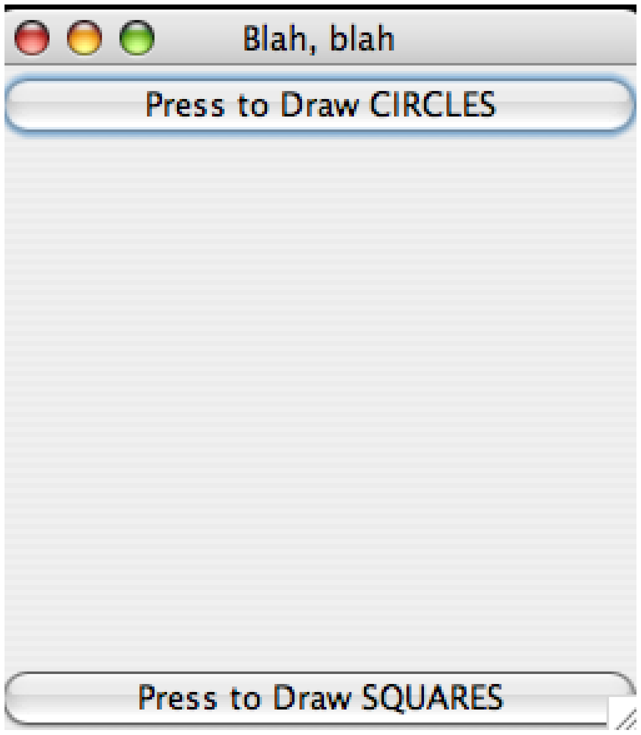
Part C — Programming Problems *40 points total*

[7] 20 points

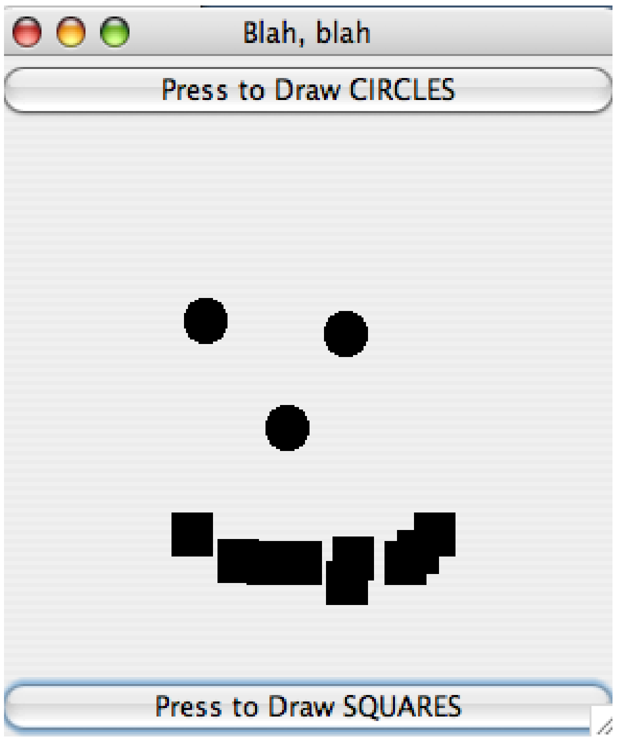
A *Java* program has been constructed to present a very simple user interface. The main program, `Prob7.java`, appears below:

```
public class Prob7
{
  public static void main(String args[])
  {
     Prob7Frame f = new Prob7Frame ("blah, blah", 500, 500);
  }
}
```

When this program is executed, the following appears: a **`JFrame`** containing a **`JButton`** at the top and a **`JButton`** at the bottom.



The center space has been reserved for the user to draw small circles and squares simply by clicking the mouse inside a **`JPanel`**. The user can toggle between drawing circles versus squares by first clicking on one or the other **`JButton`**. Here's what the screen is supposed to look like after doing a bit of clicking:



Each circle or square is 20 pixels wide and 20 pixels high. The upper left-hand corner of each circle or square appears where I clicked the mouse. A somewhat incomplete version of this program is shown below. Your task is to fill in the missing code for the constructor `public MyPanel ()` and the method `actionPerformed(ActionEvent e).` As little as ONE instruction can go in each place!

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

class Prob7Frame extends JFrame
{
    JButton circleButton = new JButton("Press to Draw CIRCLES");
    JButton squareButton = new JButton("Press to Draw SQUARES");
    MyPanel mp = new MyPanel();
    boolean circleOrSquare = false;
    int xValue, yValue;

    public Prob7Frame(String name, int h, int w)
    {
       setTitle (name);
       setSize (h, w);
       Container c = getContentPane();
       ButtonListener bl = new ButtonListener();
       circleButton.addActionListener ( bl );
       squareButton.addActionListener ( bl );
```

```
          c.add (mp, BorderLayout.CENTER);
          c.add (squareButton, BorderLayout.SOUTH);
          c.add (circleButton, BorderLayout.NORTH);
          setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE);
          setVisible( true);
        }

        //inner classes
        class MyPanel extends JPanel implements MouseListener
        {
           MyPanel()
           {
               // missing code goes here
           }

           public void mouseClicked( MouseEvent e )
           {
               Graphics g = getGraphics();
               xValue = e.getX();
               yValue = e.getY();

               if (circleOrSquare)
                       g.fillRect(xValue, yValue, 20,20);
               else  g.fillOval(xValue, yValue, 20, 20);
           }

           public void mousePressed( MouseEvent e ) { }
           public void mouseReleased( MouseEvent e ) {}
           public void mouseEntered( MouseEvent e )  {}
           public void mouseExited( MouseEvent e )  {}
        }


        class ButtonListener implements ActionListener
        {
            public void actionPerformed(ActionEvent e)
            {
                  // missing code goes here }
            }

   }
```

[8] MIPS Assembly Language Programming: 20 points

Write a MIPS *subroutine* named **decode**. This subroutine expects as its only argument a pointer to an ASCIIZ string that contains a "secret message" in *lower-case alphabetic letters* along with "punctuation."

This "secret code" has every letter in the original message being stored as a character that is three ahead. In other words, the letter 'a' is stored as a 'd', the letter 'b' is stored as an 'e', ..., and the letters 'x', 'y' and 'z' have been converted into 'a', 'b', and 'c', respectively. Your subroutine should convert the secret message back into ordinary English. Thus, for example, if the encoded message is:

```
mdyda!!!
```

then your **decode** subroutine will return with this message now reading

```
javax!!!
```

Note that the encoded string (and the decoded result) might contain non-alphabetic characters. Upper-case alphabetic characters can be treated as though they are "punctuation."

This subroutine is written using the standard MIPS conventions for argument passing and installing a stack frame, and so on. Most of the code is shown below. Fill in the missing pieces!

```
# E50b Practice Final Exam
# Decodes a message
        .text
main:            # SPIM starts by jumping to main.
                 # First read the string from keyboard
        la      $a0, string_space
        li      $a1, 1024
        li      $v0, 8 # load read_string code into $v0
        syscall

        la      $a0, string_space
        jal     decode
        la      $a0, string_space
        li      $v0, 4
        syscall
        li      $v0, 10      # load "exit" into $v0
        syscall

decode: # Here is the subroutine!
        subu    $sp, $sp, 8  # first install stack frame
        sw      $ra, 4($sp)
        sw      $fp, 0($sp)
        addu    $fp, $sp, 8

loop:   lb      $t3, ($a0)   # load one char into $t3
        beqz    $t3, exit    # if $t3==null char, branch
        blt     $t3, 'a', incr
        bgt     $t3, 'z', incr
```

// Fill in missing statements right here!

```
incr:   sb      $t3, ($a0)
        addu    $a0, $a0, 1
        b       loop
```

exit: **// Fill in more missing statements right here!**

```
## Here's where data for this program is stored:
                .data
string_space:   .space  1024
```

Appendix A: Bitset.java code from Course Handout

Note that only the first part of this file is shown.

```
import java.util.*;                  // this is file Bitset.java

class Bitset
{
    private byte [] byteArray;     // the array of bytes (8-bit integers)
    private int maxSize;           // max # of set elements allowed

    // First 3 constructor methods
    public Bitset()                // make an empty set of capacity zero.
    {
        maxSize = 0;       byteArray = null;
    }

    // make a set able to hold 'size' elements each in range 0 .. size-1
    public Bitset (int size)
    {
        maxSize   = size;
        int nbyte = (size + 7) / 8;
        byteArray = new byte [nbyte];    // new array, all zeroes
    }

    // make a copy of 'setA'
    public Bitset (Bitset setA)
    {
        maxSize   = setA.maxSize;
        int nbyte = setA.byteArray.length;
        byteArray = new byte [nbyte];        // new array, all zeroes
        System.arraycopy (setA.byteArray, 0,
                          byteArray, 0, setA.byteArray.length);
    }

    // Sets the bit at given offset from address byteArray to 1
    // i.e., adds element 'offset' to the set
    private void setBit (int n)
    {
        int whichByte = n / 8;
        int whichBit = n % 8;
        byteArray[whichByte] |= (1 << whichBit);
    }

    // Returns true if the bit at given offset from address
    // .. byteArray is set; else false.
    // i.e., determines whether element 'offset' is in the set
    private boolean getBit (int n)
    {
        int whichByte = n / 8;
        int whichBit = n % 8;
        return ( (byteArray[whichByte] & (1 << whichBit)) != 0 );
    }
```