Final Exam Solutions

Problem 1: Short Answer (15 points)

Answer for 1a:

```
X = 520, Y = 520
```

Answer for 1b:

```
X = 65, Y = 66
```

Problem 2: Using the graphics libraries (40 points)

```
import acm.program.*;
import acm.graphics.*;
import java.awt.event.*;
import javax.swing.*;

public class LineDrawing extends GraphicsProgram {

   // Set up Undo button and listeners.  (Could also have be done in init method.)
   public void run() {
      add(new JButton("Undo"), SOUTH);
      addActionListeners();
      addMouseListeners();
   }

   public void mouseClicked(MouseEvent e) {
      // If this is not the first click, then draw line to previous click position.
      if (lastX != -1) {
         lastLine = new GLine(lastX, lastY, e.getX(), e.getY());
         add(lastLine);       // Add line to the canvas
         numLines++;          // Increment number of lines on the screen
      }

      // Store the last click position
      lastX = e.getX();
      lastY = e.getY();
   }

   // Determine if Undo button was pressed and take appropriate action
   public void actionPerformed(ActionEvent e) {
      String cmd = e.getActionCommand();
      if (cmd.equals("Undo")) {
         if (lastLine != null) {    // If there is a last line on screen, remove it
            remove(lastLine);       // Remove line from canvas
            numLines--;             // Decrement number of lines on screen

            // Set next point to draw from as start point of line we just removed
            lastX = lastLine.getX();
            lastY = lastLine.getY();

            lastLine = null;        // Indicate there is no last line to remove
         }

         // If we just removed the only line on screen, indicate that we once again
         // need a "first click" to draw first line.  Note: this will also include
         // the case where the user clicked once and then pressed the Undo button.
         if (numLines == 0) {
            lastX = -1;
         }
      }
   }

   /* Private instance variables */
   int numLines = 0;                // keep track of the number of lines drawn
   // lastX, lastY indicate position of last click on canvas by the user
   private double lastX = -1;       // Use -1 to indicate "no first click"
   private double lastY;
   private GLine lastLine = null;   // keep track of last line drawn on screen
}
```

Problem 3: Strings and characters (20 points)

```
public String fixCapitalization(String str) {

   /* Start with an empty result string that we will build up.             */
   String result = "";

   /* nextCap keeps track of whether next letter in input string should    *
    * be capitalized.  Note that the first letter must be capitalized.     */
   boolean nextCap = true;

   for(int i = 0; i < str.length(); i++) {
      char ch = str.charAt(i);

      /* Only consider capitalization if character is a letter.            */
      if (Character.isLetter(ch)) {
         if (nextCap) {
            ch = Character.toUpperCase(ch);
            nextCap = false;
         } else {
            ch = Character.toLowerCase(ch);
         }

      /* If we didn't have letter, see if it's end-of-sentence punctuation *
       * mark (in which case we should capitalize the next letter.         */
      } else if (ch == '.' || ch == '!' || ch == '?') {
         nextCap = true;
      }
      result += ch;
   }

   return result;
}
```

Problem 4: Arrays (20 points)

```
public void gridInsert(char[][] grid, int row, int col, char ch) {

   // Shift the row where new character will be inserted
   char last = shiftArray(grid[row], col);

   // Insert new character ch
   grid[row][col] = ch;

   // Shift all remaining rows in the grid
   for(int i = row + 1; i < grid.length; i++) {
      char nextLast = shiftArray(grid[i], 0);
      grid[i][0] = last;
      last = nextLast;
   }
}


// Shift elements in array over one position starting from index pos.
// Return the last element previously in the array.
private char shiftArray(char[] arr, int pos) {
   char last = arr[arr.length - 1];
   for(int i = arr.length - 1; i > pos; i--) {
      arr[i] = arr[i-1];
   }
   return last;
}
```

Problem 5: Data structure design (40 points)

```
import java.util.*;

public class BigNum {

   public BigNum(ArrayList<Integer> digits) {
      number = new ArrayList<Integer>();
      for(int i = 0; i < digits.size(); i++) {
         number.add(digits.get(i));
      }
   }

   // Return integer at given index in list, if index exist.  Otherwise return 0.
   private int safeGetDigit(ArrayList<Integer> list, int index) {
      if (index >= list.size()) {
         return 0;
      } else {
         return list.get(index);
      }
   }

   // Add NigNum num passed in to this BigNum
   public void add(BigNum num) {
      int carry = 0;       // Keep track of "carry" when two digits added

      // Determine which BigNum has more digits
      int maxSize = number.size();
      if (num.number.size() > maxSize) {
         maxSize = num.number.size();
      }

      // Loop through digits/columns of the BigNums
      for(int i = 0; i < maxSize; i++) {

         // Add digits in current column of BigNum (including carry)
         int sum = safeGetDigit(number, i) + safeGetDigit(num.number, i) + carry;

         // Determine if there is a carry to the next column
         if (sum >= 10) {
            carry = 1;
            sum = sum - 10;
         } else {
            carry = 0;
         }

         // Either set the result in this BigNum (if we have existing space)
         // or add a new digit if needed.
         if (i < number.size()) {
            number.set(i, sum);
         } else {
            number.add(i, sum);
         }
      }

      // Check for a carry on the last digit/column we added
      if (carry == 1) {
         number.add(carry);
      }
   }
```

```
   // Create a string of the BigNum
   public String toString() {
      String result = "";
      for(int i = 0; i < number.size(); i++) {
         // Add digits to string in reverse
         result = number.get(i) + result;
      }
      return result;
   }

   private ArrayList<Integer> number;  // Store digits of BigNum
}
```

Problem 6: Java programming (25 points)

```
import acm.program.*;
import acm.util.*;
import java.io.*;

public class TextProcessor extends ConsoleProgram {

   /* Constants */
   private static final String DATAFILE = "data.txt";
   private static final char APPEND = '+';
   private static final char BACKSPACE = '-';
   private static final char INSERT = '#';

   public void run() {
      String str = "";                          // string we are building up

      // read file of edit operations line-by-line
      try {
         BufferedReader rd = new BufferedReader(new FileReader(DATAFILE));

         while (true) {
            String line = rd.readLine();
            if (line == null) break;
            str = updateString(str, line);      // update str based on edit "line"
         }
         rd.close();
      } catch (IOException ex) {
         throw new ErrorException(ex);
      }

      println(str);                             // print built-up string
   }

   // start with string str, and return an updated version based
   // on the edit operation in the string line
   private String updateString(String str, String line) {
      String result = str;
      char op = line.charAt(0);     // first character determines edit operation

      if (op == APPEND) {
         result = str + line.substring(1);
      } else if (op == BACKSPACE) {
         result = str.substring(0, str.length() - 1);
      } else if (op == INSERT) {
         char ch = line.charAt(1);
         int pos = Integer.parseInt(line.substring(3));  // parse insert position
         result = insertChar(str, ch, pos);
      }

      return result;
   }

   // return string that is str with character ch inserted at position pos
   private String insertChar(String str, char ch, int pos) {
      String prefix = str.substring(0, pos);
      String suffix = str.substring(pos);
      return (prefix + ch + suffix);
   }
}
```

Problem 7: Using data structures (20 points)

```
public HashMap<String, String> invert(HashMap<String, String> map) {

   // create a new map for the resulting inverted map
   HashMap<String,String> result = new HashMap<String,String>();

   // Iterate over the keySet of the original map
   // (Note: could also do with with a "for each" loop)
   Iterator<String> it = map.keySet().iterator();
   while (it.hasNext()) {
      String key = it.next();
      String value = map.get(key);

      // if a value in the original map already exists as a key
      // in the result map, then the original map is not invertible
      // since the result map would not have unique keys.
      if (result.containsKey(value)) return null;

      // add the inverted value/key pair to the result map
      result.put(value, key);
   }

   return result;
}
```