# Answers to Additional Practice Final Problems

## Problem – Interactors

```
public class InteractorsSample extends GraphicsProgram{
	private GLine fwdslash;
	private GLine backslash;

	private static final int LINE_WIDTH = 10;
	private static final int LINE_HEIGHT = 10;

	public void run() {
		//add buttons
		add(new JButton("NORTH"), SOUTH);
		add(new JButton("SOUTH"), SOUTH);
		add(new JButton("EAST"), SOUTH);
		add(new JButton("WEST"), SOUTH);

		//calculate coordinates for cross
		double xLeft = getWidth()/2-LINE_WIDTH/2;
		double xRight = getWidth()/2+LINE_WIDTH/2;
		double yTop = getHeight()/2-LINE_HEIGHT/2;
		double yBot = getHeight()/2+LINE_WIDTH/2;

		//create two GLines for diagonals
		fwdslash = new GLine(xLeft, yBot, xRight, yTop);
		backslash = new GLine(xLeft, yTop, xRight, yBot);
		add(fwdslash);
		add(backslash);

		validate();
		addActionListeners();
	}

	public void actionPerformed(ActionEvent e) {
		if(e.getActionCommand().equals("NORTH")) {
			moveCross(0, -20);
		}else if(e.getActionCommand().equals("SOUTH")) {
			moveCross(0, 20);
		}else if(e.getActionCommand().equals("EAST")) {
			moveCross(20, 0);
		}else if(e.getActionCommand().equals("WEST")) {
			moveCross(-20, 0);
		}
	}

	public void moveCross(int dx, int dy) {
		double oldX = backslash.getX()+LINE_WIDTH/2;
		double oldY = backslash.getY()+LINE_HEIGHT/2;

		fwdslash.move(dx, dy);
		backslash.move(dx, dy);

		double newX = backslash.getX()+LINE_WIDTH/2;
		double newY = backslash.getY()+LINE_WIDTH/2;
```

```
            //create new line based on calculations
            GLine line = new GLine(oldX, oldY, newX, newY);
            line.setColor(Color.RED);
            add(line);
      }
```

**Problem —Strings (15 points)**

```
/* Method: findFirstMatchingPosition(shortDNA, longDNA) */
/**
 * Returns the first index position at which the short DNA
 * strand would bind to the longDNA strand, or -1 if no such
 * position exists.
 */
   private int findFirstMatchingPosition(String shortDNA,
                                         String longDNA) {
      return longDNA.indexOf(matchingStrand(shortDNA));
   }

/* Method: matchingStrand(strand) */
/**
 * Returns a string in which each base has been replaced
 * by the counterpart base (C <-> G and A <-> T).  Any
 * other characters are replaced by an X.
 */
   private String matchingStrand(String strand) {
      String match = "";
      for (int i = 0; i < strand.length(); i++) {
         match += matchingBase(strand.charAt(i));
      }
      return match;
   }

/* Method: matchingBase(base) */
/**
 * Returns a character which is the DNA match for the
 * specified base (C <-> G and A <-> T).  Any other
 * character returns an X.
 */
   private char matchingBase(char base) {
      switch (base) {
        case 'C': return 'G';
        case 'G': return 'C';
        case 'A': return 'T';
        case 'T': return 'A';
        default:  return 'X';
      }
   }
```

You can also write the method with no loops. Here's one possibility:

```
   private int findFirstMatchingPosition(String shortDNA,
                                         String longDNA) {
      String pattern = shortDNA;
      pattern = pattern.replace('C', 'g');
      pattern = pattern.replace('G', 'c');
      pattern = pattern.replace('A', 't');
      pattern = pattern.replace('T', 'a');
      return longDNA.indexOf(pattern.toUpperCase());
   }
```

**Problem —Arrays (10 points)**

```
/* Method: checkUpperLeftCorner */
/**
 * This method checks the upper left corner of a Sudoku array
 * to see if it correctly contains one copy of each digit
 * between 1 and 9.  If so, the method returns true.  If it
 * contains values that are duplicated or out of range, the
 * method returns false.
 */
   private boolean checkUpperLeftCorner(int[][] matrix) {
      boolean[] alreadyUsed = new boolean[10];
      for (int i = 0; i < 3; i++) {
         for (int j = 0; j < 3; j++) {
            int digit = matrix[i][j];
            if (digit < 1 || digit > 9) return false;
            if (alreadyUsed[digit]) return false;
            alreadyUsed[digit] = true;
         }
      }
      return true;
   }
```

**Problem —Java programming (15 points)**

```
/*
 * File: TelephoneMnemonic.java
 * ----------------------------
 * This program displays all seven-letter words that match
 * a phone number.
 */

import acm.program.*;
import acm.util.*;

import java.io.*;

public class TelephoneMnemonic extends ConsoleProgram {


/** Runs the program */
   public void run() {
      String phoneNumber = readLine("Enter phone number: ");
      generateMnemonics(phoneNumber);
   }

/* Method: generateMnemonics(phoneNumber) */
/**
 * Generates the mnemonics for a telephone number using the
 * standard keypad codes:
 *
 *   1 (none)    2 (ABC)     3 (DEF)
 *   4 (GHI)     5 (JKL)     6 (MNO)
 *   7 (PQRS)    8 (TUV)     9 (WXYZ)
 *               0 (none)
 */
   private void generateMnemonics(String phoneNumber) {
      try {
         BufferedReader rd = new BufferedReader(
                                    new FileReader(DATA_FILE));
         while (true) {
            String word = rd.readLine();
            if (word == null) break;
            if (toPhoneDigits(word).equals(phoneNumber)) {
               println(word);
            }
         }
         rd.close();
      } catch (Exception ex) {
         throw new ErrorException("Data file missing");
      }
   }
```

**Problem —Java programming (continued)**

```
/* Method: toPhoneDigits(word) */
/**
 * Translates a word into a string with the corresponding
 * keypad numbers.
 */
   private String toPhoneDigits(String word) {
      String phoneNumber = "";
      for (int i = 0; i < word.length(); i++) {
         char ch = word.charAt(i);
         switch (ch) {
           case 'A': case 'B': case 'C': ch = '2'; break;
           case 'D': case 'E': case 'F': ch = '3'; break;
           case 'G': case 'H': case 'I': ch = '4'; break;
           case 'J': case 'K': case 'L': ch = '5'; break;
           case 'M': case 'N': case 'O': ch = '6'; break;
           case 'P': case 'Q': case 'R': case 'S': ch = '7'; break;
           case 'T': case 'U': case 'V': ch = '8'; break;
           case 'W': case 'X': case 'Y': case 'Z': ch = '9'; break;
         }
         phoneNumber += ch;
      }
      return phoneNumber;
   }

/* Private constants and instance variables */

   private static final String DATA_FILE = "words7.txt";

}
```