1a (4 points)
What is contained in the array list at the end of this code?

```java
int[] list = { 5, 4, 3, 2, 1 };
for ( int index = 0; index < list.length-1; index += 2 )
{
    list[index] += list[index + 1];
}
{ 9, 4, 5, 2, 1 }
```

1b (4 points)
What is the output of the following program?

```java
public class Midterm {
    public static void main(String[] args) {
        int lhs = 9;
        int rhs = 3;
        int[] list = { 5, 4, 3, 2, 1 };
        scram(list, lhs, rhs);
        System.out.println("Main: lhs = "+ lhs + ", rhs = "+ rhs);
        System.out.println("Main: list[3] = "+ list[3]);
    }
    public static void scram(int[] list, int lhs, int rhs) {
        list[rhs] = lhs;
        lhs = list[1];
        rhs = list[list.length – 1];
        System.out.println("scram: lhs = " + lhs + ", rhs = " + rhs);
        System.out.println("scram: list[3] = " + list[3]);
    }

    scram: lhs = 4, rhs = 1
    scram: list[3] = 9
    Main: lhs = 9, rhs = 3
    Main: list[3] = 9
```
1c (4 points)

What is the output of the following code?

```java
int x = 3;
int y = 8;
int z = 5;

if (x > 3)
    if (y > 8)
        System.out.println("First line");
    else if (z >= 5)
        System.out.println("Second line");
    else
        System.out.println("Third line");
System.out.println("Fourth line");
```

Fourth line

1d (4 points)

What is the output of the following code?

```java
int number = 11;
while (number > 0) {
    if (number % 2 == 0 || number % 3 == 0)
        System.out.print(number);
    else if (number % 5 == 0)
        System.out.println();
    number -= 3;
}
System.out.println("!");
```

```
8
2!
```

1e (4 points)

How many times does the following loop execute?

```java
int index = 2;
do {
    if (index > 3)
        continue;
    index *= 2;
} while (index < 8);
```

Infinite loop
2 (16 points)

Write three overloaded versions of the method `printVertical`:

- One version accepts a `String` and prints it out vertically, one character at a time.
- Second version accepts an integer and prints its binary representation vertically, one character at a time (use the `Integer.toBinaryString` method to get the binary representation of the integer).
- Third version accepts a real and prints its hexadecimal representation vertically, one character at a time (use the `Double.toHexString` method to get the hexadecimal representation of the real).

Hints:

- Can you use one of the methods to help implement the other?
- The `String` methods `charAt(int index)` and/or `toCharArray()` might help.

```java
void printVertical(String item)
{
    for( int index = 0; index < item.length; ++index )
        System.out.println(item.charAt(index));
}

void printVertical(int item)
{
    printVertical(Integer.toBinaryString(item));
}

void printVertical(double item)
{
    printVertical(Double.toHexString(item));
}
```
3 (16 points)

Write a method to estimate the value of \( \pi \) using the following summation:

\[
\pi = 4 \times (1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} \ldots)
\]

Terminate the calculation when your estimated \( \pi \) is within \( 10^{-6} \) (0.000001) of Math.PI. Output the estimated value and the number of terms used:

After 1000001 terms, computed value of PI = 3.1415936535887745

Hints:

- The estimated value of \( \pi \) can be either greater or less than the actual value. The first few results of the series are approximately: 4.000, 2.667, 3.467, 2.895, 3.340, 2.976. Can you use the method Math.abs to help you?
- How do you get the sign to alternate between positive and negative?
- Pay attention in the change in denominator rather than change in the fraction.
- Divide the problem in 4 sub-problems: initialization, end condition, next term and output.

```java
public static void computePI()
{
    int terms = 1;
    double nominator = 4;
    double denominator = 1;
    double computedPI = nominator / denominator;
    while ( Math.abs(computedPI - Math.PI) > 0.000001 )
    {
        nominator *= -1;
        denominator += 2;
        ++terms;
        computedPI += nominator / denominator;
    }
    System.out.println("After " + terms + " terms, computed value of PI = " + computedPI);
}
```
4 (16 points)

Write a method that given two integers will return a matrix (2 dimensional array) with the multiplication table of the numbers in the range given by the arguments:

<table>
<thead>
<tr>
<th>multiplicationTable(2, 6) returns</th>
<th>multiplicationTable(3, 5) returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>{{4, 6, 8, 10, 12},</td>
<td>{{9, 12, 15},</td>
</tr>
<tr>
<td>{6, 9, 12, 15, 18},</td>
<td>{12, 16, 20},</td>
</tr>
<tr>
<td>{8, 12, 16, 20, 24},</td>
<td>{15, 20, 25} }</td>
</tr>
<tr>
<td>{10, 15, 20, 25, 30},</td>
<td></td>
</tr>
<tr>
<td>{12, 18, 24, 30, 36}}</td>
<td></td>
</tr>
</tbody>
</table>

```java
public static int[][] multiTable(int low, int high)
{
    if ( low > high )
    {
        int temp = low;
        low = high;
        high = temp;
    }
    int[][] result = new int[high - low + 1][high - low + 1];
    for ( int row = low; row <= high; ++row )
        for ( int col = low; col <= high; ++col )
            result[row - low][col - low] = row * col;
    return result;
}

Alternative solution:

```java
public static int[][] multiTable(int low, int high)
{
    if ( low > high )
        return multiTable(high, low);
    int[][] result = new int[high - low + 1][high - low + 1];
    for ( int row = low; row <= high; ++row )
        for ( int col = low; col <= high; ++col )
            result[row - low][col - low] = row * col;
    return result;
}
Some prisoners communicate by organizing the letters of the alphabet in a matrix and then tapping on the cell wall the number of the row and column of the matrix containing the next letter.

If the letters of the alphabet are organized in the following matrix:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
</tr>
<tr>
<td>3</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>4</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
</tr>
<tr>
<td>5</td>
<td>U</td>
<td>V</td>
<td>X</td>
<td>Y</td>
<td>W</td>
</tr>
<tr>
<td>6</td>
<td>Z</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

the word “class” would be tapped out with the sequence (1,3) – (3,2) – (1,1) – (4,4) – (4,4)

The above matrix is 6x5, but different set of prisoners may come up with different matrix organization. If the alphabet was organized in a 4x7 matrix:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
</tr>
<tr>
<td>2</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
</tr>
<tr>
<td>3</td>
<td>O</td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
</tr>
<tr>
<td>4</td>
<td>V</td>
<td>X</td>
<td>Y</td>
<td>W</td>
<td>Z</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

the word “class” would be tapped out with the sequence (1,3) – (2,5) – (1,1) – (3,5) – (3,5)
5 (16 points)

Write a method that, given a row and column, will return the corresponding letter in the alphabet matrix as described on page 6. The arguments of the method are the alphabet matrix, stored in a two dimensional array of arbitrary dimensions, a row number and a column number.

The method **must check** that the row and column values are valid. If either is invalid, the method returns a blank space: ' '.

With the 6x5 matrix on page 6, calling the method with row 4 and column 3 would return the letter 'R'. With the 7x4 matrix on page 6, calling the method with row 2 and column 7 would return the letter 'N'.

**Notes:**
- You don't know the array dimensions in advance.
- Humans think in terms of row and column that start at 1, Java does not.

```java
public static char findLetter(char[][] table, int row, int col) {
    char retValue = ' '; // Initialize returned value
    --row; // Adjust row index
    --col; // Adjust column index
    if ( (row >= 0) && (row < table.length) 
        && (col >= 0) && (col < table[row].length) )
    {
        retValue = table[row][col]; // Get the letter
    }
    return retValue; // Return the letter
}
```
6 (16 points)

Write a method that, given a letter, will return an array with the row and column of the letter in the alphabet matrix as described on page 6. The arguments of the method are the alphabet matrix, stored in a two dimensional array of arbitrary dimensions, and the letter.

With the 6x5 matrix on page 6, calling the method with letter 'R' would return the array {4, 3}. With the 4x7 matrix on page 6, calling the method with the letter 'N' would return the array {2, 7}.

Notes:
• You don't know the array dimensions in advance.
• Humans think in terms of row and column that start at 1, Java does not.

```java
public static int[] findTuple(char[][] table, char letter) {
    int[] retValue = {0, 0};
    for (int row = 0; row < table.length; ++row)
        for (int col = 0; col < table[row].length; ++col)
            if (table[row][col] == letter) {
                retValue[0] = row + 1;
                retValue[1] = col + 1;
                return retValue;
            }
    return retValue;
}
```