# Criterion C: Development

1. For loops
2. Overloading
3. Multidimensional Arrays
4. Use of GUI Builder

1. The for loops allowed the total number of lines of code to be greatly condensed, primarily on the functions that would be output on matrixC, as each of the nine parts of that would be determined based on the previous ones. I was unable to use for loops to receive input from the text boxes, so I had to individually set the values of the matrices to the text in each box that the user inputted. The greatest amount of lines condensed was in the multiplication function, where there were three total for loops inside of each other, as shown below.

```
for(int i = 0; i < 3; i++) {

    for(int j = 0; j < 3; j++) {

        int modifier = ((((i + j + 1) %2) * 2) - 1);

        int value = 0;

        for(int k = 0; k < 3; k++) {
            //we're real deep now
            value += matrixA[i][k] * matrixB[k][j];
        }

        matrixC[i][j] = modifier * value;

    }
}
```

2. I used overloading in the determinant method, which is called by the user performing the determinant operator, or by taking the inverse of matrix A. This was helpful because it could be used to define the determinant somewhat recursively  I originally had the determinant just as the hard calculation to get the number at the end, though it was a

single very long convoluted line, and it only worked for matrices of size 3x3. Using overloading allowed determinants to be solved for any square matrix under and including size 3x3. I could have further expanded this to solve for square matrices of size 4, 5, and so on, potentially to the point of recursively solving for any matrix of size n. As I only had 3x3 matrices as options, such recursion would not have been necessary, and overloading elegantly solved my problem (except for the 9 needed floats for the 3x3.)

```
public float determinant(float p11, float p12, float p13, float p21, float p22, float p23, float p31, float p32, float p33) {
    float answerp = p11 * determinant(p22, p23, p32, p33) - p12 * determinant(p21, p23, p31, p33) + p13 * determinant(p21, p22, p31, p32);
    return answerp;

}

//overloading is such fun wohoho
public float determinant(float q11, float q12, float q21, float q22) {
    float answerq = (q11 * determinant(q22)) - (q12 * determinant(q21));
    return answerq;
}

//now im just being extravagant
public float determinant(float r11) {
    return r11;
}
```

3. I chose to use multidimensional arrays (2 dimensions specifically) for the matrices because it was a better solution than just declaring a variable for each of the 9 cells for both matrices, to just 2. The only part that I had to do for each individual cell of the matrix was receive the text that the user input via the text boxes, which was not possible using some form of loops. Having each variable in part of an array was especially convenient when doing calculations with matrices because the way that the variables in two dimensional arrays are stored in programming is highly similar to the layout of matrices, such as [1][0] in programming being the second row and first column, the specification of placements of values in the matrix is the same way, that is, row then column, the only difference being that "1" denotes the first row or column, where in programming "0" does this, making everything have a difference of 1 when converting between the two types of data.

4. As the GUI for my project required 28 text boxes and 5 buttons, as well as multiple labels, I had significant difficulty in trying to program those in manually. Attempts to do so ended with displaced text boxes and numerous errors. Due to the significant difficulty resulting from the sheer number of objects present in the GUI, I used the NetBeans Java GUI builder, which made creating the GUI much easier, so when creating this project I could focus more on having the calculations work instead of having working buttons and text appear when said button is pressed, and so on.