Criterion C: Development

1. Representing congruence relations in modular arithmetic-

I first began creating separate classes for congruence relations that are to be displayed in mathematical notation and congruence relations that are to be displayed as a visualization. I found it much more effective, however, to have each of these classes extend an abstract class because they can then share methods and data declarations. This also allows the Slide class to have an array list of the abstract class ModNumber which holds objects of the type ModNumberMathNoation or type ModNumberBoxes. The implementations of ModNumber are then encapsulated and hidden from Slide, which is guaranteed through the abstract class that all objects will have the needed data and methods.

2. Visualizing congruence relations in modular arithmetic- ModNumberBoxes class

To visualize the numbers graphically, the display method of the ModNumberBoxes class first displays a box per whole number dividend in the modulus, and then displays the residual. Two loops were used to do this. I used the java Swing library to create the graphics for the application. This involved extending the JPanel class and overriding the paintComponent function.

I then added a separate type of visualization that displays the number in the modulus as a diophantine equation. Coloring different parts of the solution helps the viewer tie together the steps to solving the problem nicely.

3. The Slide class and class design

I found that overriding constructors in order to create optional arguments was a good way to offer an adaptable structure for creating slides. The Slide class, for example, has optional arguments message and notes. Message becomes a title to the slide, while notes is an arraylist of strings to be displayed in the notes section of the slide. My strategy was to minimize mutation of objects by passing values through, reducing bugs that arise from mutation. Recalculating values allows for dynamic re-scaling of the entire application easily. The math behind re-calculating these types of values is also very cheap, so it has no performance problems. In fact, all of the classes are essentially immutable because of the private variables, except in the case of very few setattribute functions.

4. Writing the Slides

In writing the slides for the lecture, I noticed improvements that could be made and implemented them. One of these improvements was to add a function that converts numbers in mathematical notation to objects to be visualized. I used an enhanced for loop to do this, and a copy constructor to safely duplicate slides without mutating the original slide. The copy constructor receives a slide and returns the new slide.

5. Other notes

Working with the Font class in java was a new experience. I positioned the font along with the graphical entities using the ascent of the font from font metrics. I learned to reuse code and define smaller, helper functions in order to achieve larger tasks. The drawRect function is

used a visualization is needed, which in turn uses the smaller drawCenteredString function. In addition, the interface for using the graphics class relies on mutation, so I learned to protectively undo any changes I made to the Graphics2D object, such as the current color.