1. **A short introduction**
I used NetBeans to develop my Java program, using Java's Swing tools to create an operationalize the GUI. In doing so, the program was able to store, and display a list of recipes. Additionally, the program automatically made certain calculations such as profit margins.

2. **Summary List of All Techniques**
- Parameter Passing
- Return Methods
- Array List
- For Loops
- Nested For Loops
- Encapsulation
  - Privatizing variables
- Inheritance
- Pointing at the same object between two classes
- GUI Tabs
- GUI Table
- GUI buttons
- Saving and Loading Recipes
  - Tokenizing data
- Error Trapping when inputting recipes
- Multiple Constructors
- Try-Catch-Throw Errors
  - Log exceptions
- Error Trapping
  - ".matches" method to ensure only digits or only letters
- JOptionPane Prompts

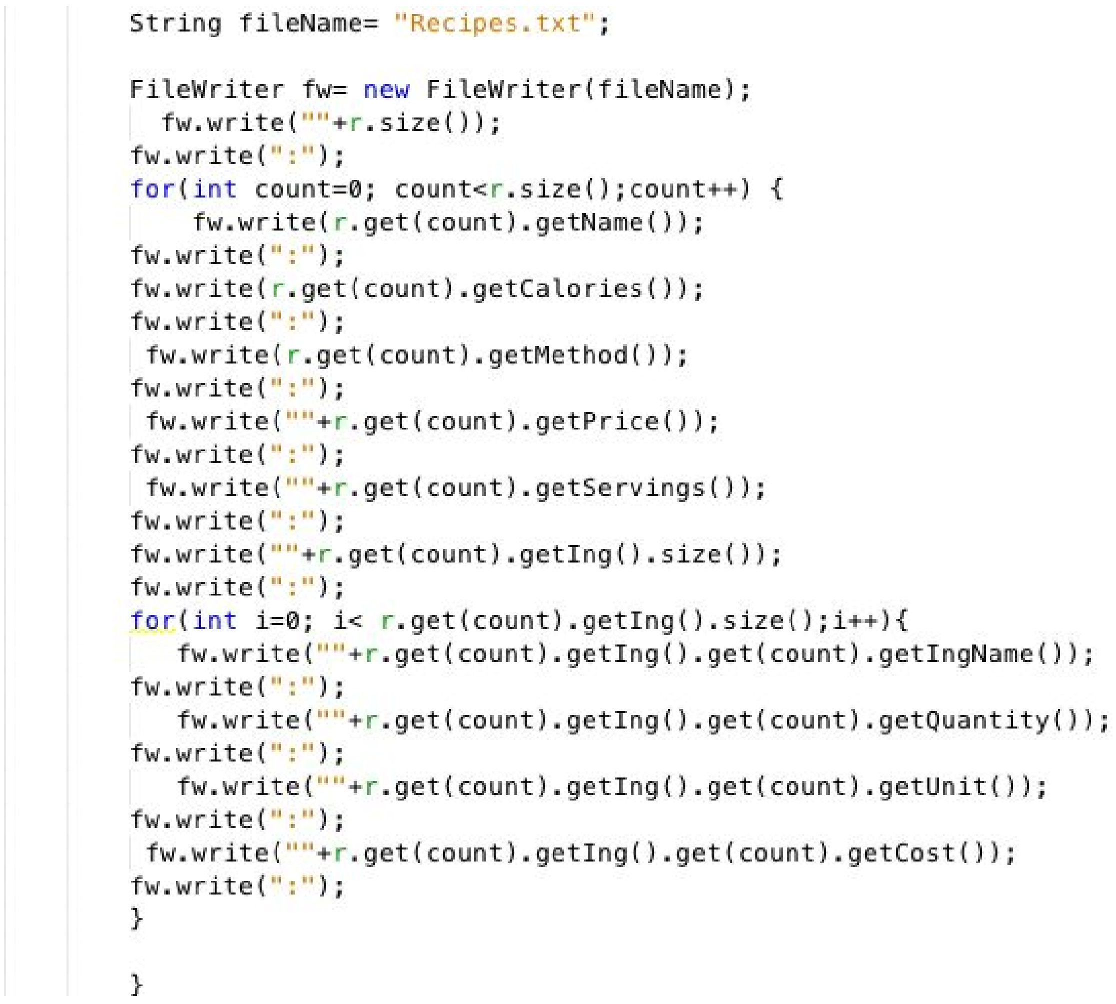
3. **Structure of the Program**
In total, I use three GUI classes, one Saving and loading methods class and two Object Classes. The "main GUI" is where Recipes are centrally inputted and displayed. It contains the instantiation of the variable: recipeList which is pointed at by other classes.

When the user wants to add ingredients to a recipe, a button from the main GUI opens a second Ingredients Builder that saves a collection of ingredients as an Array List into the recipeList housed by the main GUI.

The "details GUI", is opened upon the press of a button in the main GUI. The main GUI passes the attributes of a selected recipe from recipeList to details GUI which then visually displays them and allows the user to edit them. Upon closing the details GUI, the edited attributes are saved to same recipeList as the details GUI points to the same variable housed by the main GUI.

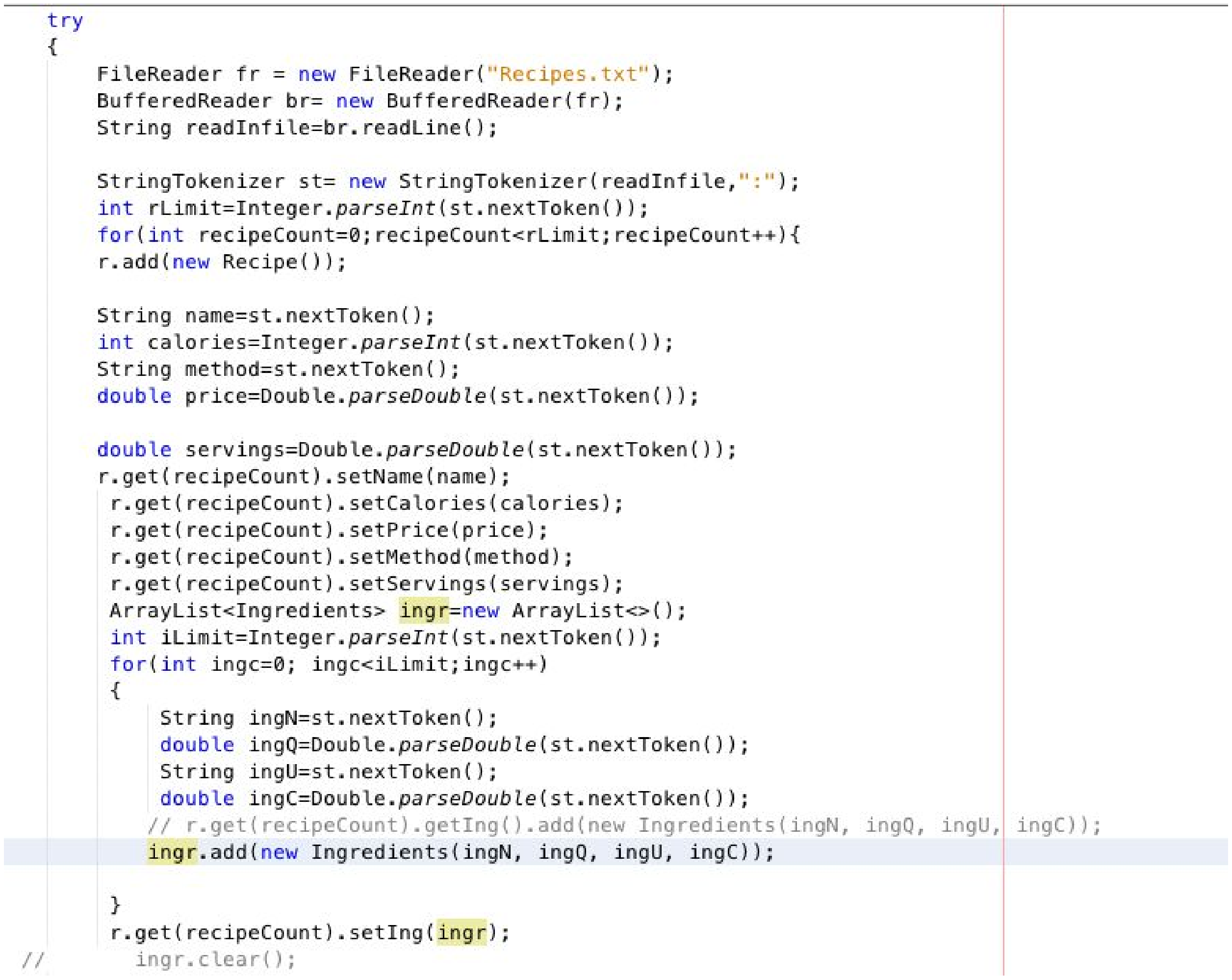
4. **Data Structures Used**
The only non-primitive data structure used were Arraylists. Recipes were stored in an Arraylist of Recipes and within this Arraylist, another Arraylist of Ingredient objects was stored. I used an Arraylist because it is a dynamic data type, and neither recipes or ingredients are consistent throughout the use of the programs. Recipes and ingredients will be added and removed by the user and as such it is most efficient to use a datatype that can adjust in size accordingly.

5. **Main Unique Algorithms**
My program stores Recipes in an arraylist which itself has contains an ArrayList of Ingredients. As such the program is storing essentially an Array of ArrayLists, combined with other parameters. To save this data, each parameter must be saved individually in a particular order. When loading, each parameter must be loaded in the same order as they were saved. The code to save each parameter is shown below.



```
String fileName= "Recipes.txt";

FileWriter fw= new FileWriter(fileName);
  fw.write(""+r.size());
fw.write(":");
for(int count=0; count<r.size();count++) {
    fw.write(r.get(count).getName());
fw.write(":");
fw.write(r.get(count).getCalories());
fw.write(":");
 fw.write(r.get(count).getMethod());
fw.write(":");
 fw.write(""+r.get(count).getPrice());
fw.write(":");
 fw.write(""+r.get(count).getServings());
fw.write(":");
fw.write(""+r.get(count).getIng().size());
fw.write(":");
for(int i=0; i< r.get(count).getIng().size();i++){
   fw.write(""+r.get(count).getIng().get(count).getIngName());
fw.write(":");
   fw.write(""+r.get(count).getIng().get(count).getQuantity());
fw.write(":");
   fw.write(""+r.get(count).getIng().get(count).getUnit());
fw.write(":");
 fw.write(""+r.get(count).getIng().get(count).getCost());
fw.write(":");
}

}
```

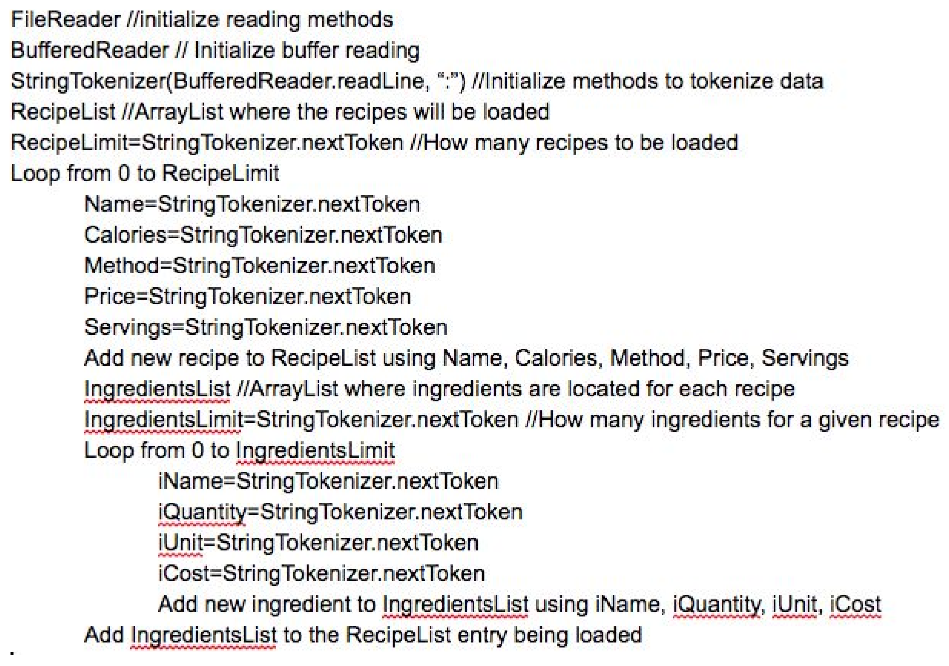
Evidently, I utilise two nested for-loops. The first is simply controlling for whenever a new recipe begins to be saved in the “Recipe.txt” file. The second, controls for when each ingredient (and its encapsulated parameters) are saved in the same file. They must be saved due to the algorithm used to load each recipe:



```
try
{
    FileReader fr = new FileReader("Recipes.txt");
    BufferedReader br= new BufferedReader(fr);
    String readInfile=br.readLine();

    StringTokenizer st= new StringTokenizer(readInfile,":");
    int rLimit=Integer.parseInt(st.nextToken());
    for(int recipeCount=0;recipeCount<rLimit;recipeCount++){
    r.add(new Recipe());

    String name=st.nextToken();
    int calories=Integer.parseInt(st.nextToken());
    String method=st.nextToken();
    double price=Double.parseDouble(st.nextToken());

    double servings=Double.parseDouble(st.nextToken());
    r.get(recipeCount).setName(name);
     r.get(recipeCount).setCalories(calories);
     r.get(recipeCount).setPrice(price);
     r.get(recipeCount).setMethod(method);
     r.get(recipeCount).setServings(servings);
     ArrayList<Ingredients> ingr=new ArrayList<>();
     int iLimit=Integer.parseInt(st.nextToken());
     for(int ingc=0; ingc<iLimit;ingc++)
     {
         String ingN=st.nextToken();
         double ingQ=Double.parseDouble(st.nextToken());
         String ingU=st.nextToken();
         double ingC=Double.parseDouble(st.nextToken());
        // r.get(recipeCount).getIng().add(new Ingredients(ingN, ingQ, ingU, ingC));
        ingr.add(new Ingredients(ingN, ingQ, ingU, ingC));

     }
     r.get(recipeCount).setIng(ingr);
//       ingr.clear();
```

In the algorithm above, the data that was previously saved is tokenized. This means that each parameter is separated and treated as an individual token to be loaded. The separation occurs on the instance of a “:”. Referring back to the saving algorithm, a colon separated each parameter entry.

Below is the pseudo-code version of the algorithm:

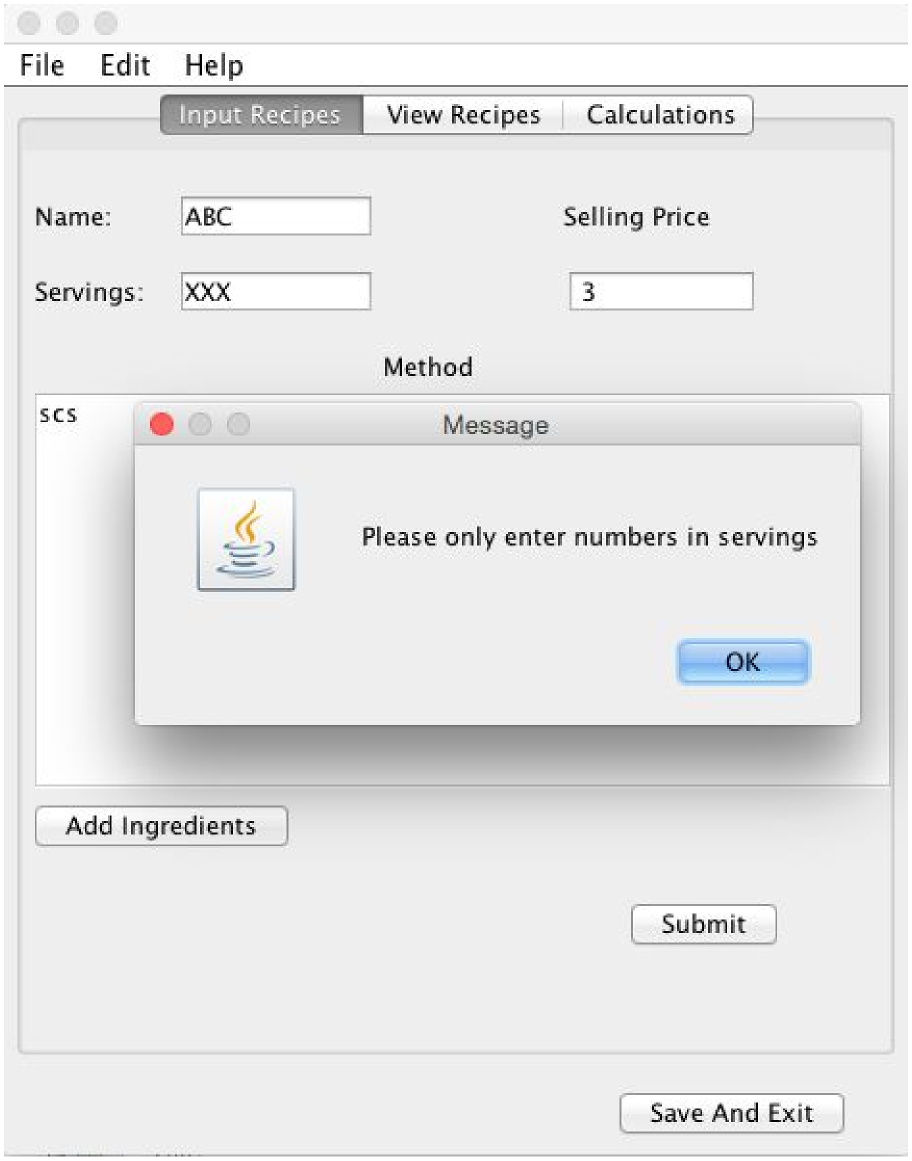


```
FileReader //initialize reading methods
BufferedReader // Initialize buffer reading
StringTokenizer(BufferedReader.readLine, “:”) //Initialize methods to tokenize data
RecipeList //ArrayList where the recipes will be loaded
RecipeLimit=StringTokenizer.nextToken //How many recipes to be loaded
Loop from 0 to RecipeLimit
    Name=StringTokenizer.nextToken
    Calories=StringTokenizer.nextToken
    Method=StringTokenizer.nextToken
    Price=StringTokenizer.nextToken
    Servings=StringTokenizer.nextToken
    Add new recipe to RecipeList using Name, Calories, Method, Price, Servings
    IngredientsList //ArrayList where ingredients are located for each recipe
    IngredientsLimit=StringTokenizer.nextToken //How many ingredients for a given recipe
    Loop from 0 to IngredientsLimit
        iName=StringTokenizer.nextToken
        iQuantity=StringTokenizer.nextToken
        iUnit=StringTokenizer.nextToken
        iCost=StringTokenizer.nextToken
        Add new ingredient to IngredientsList using iName, iQuantity, iUnit, iCost
    Add IngredientsList to the RecipeList entry being loaded
.
```

My GUI mostly uses editable JtextFields so that the user can input data that is presumably short in length. The only parameters for which I don't use JTextFields are the Method and Ingredients. Method extends past a single line of text into paragraphs and numbered steps. As such a JTextArea was used. Meanwhile, Ingredients is a list of specified objects, as such as JList was used so that the user could select from a pre-generated list. These ingredients were displayed in a JTextArea.

6. User Interface/GUI Work

My GUI uses various swing components to error-trap and facilitate use by the client. An example is given below for when the user enters letters instead of numbers for servings.



Name: ABC

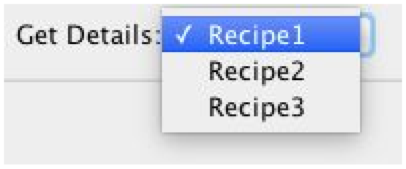
Servings: XXX

Selling Price: 3

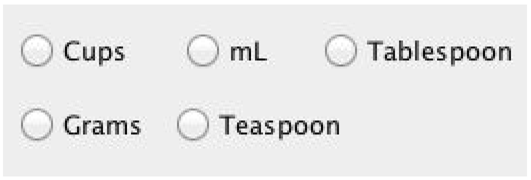
scs

Please only enter numbers in servings

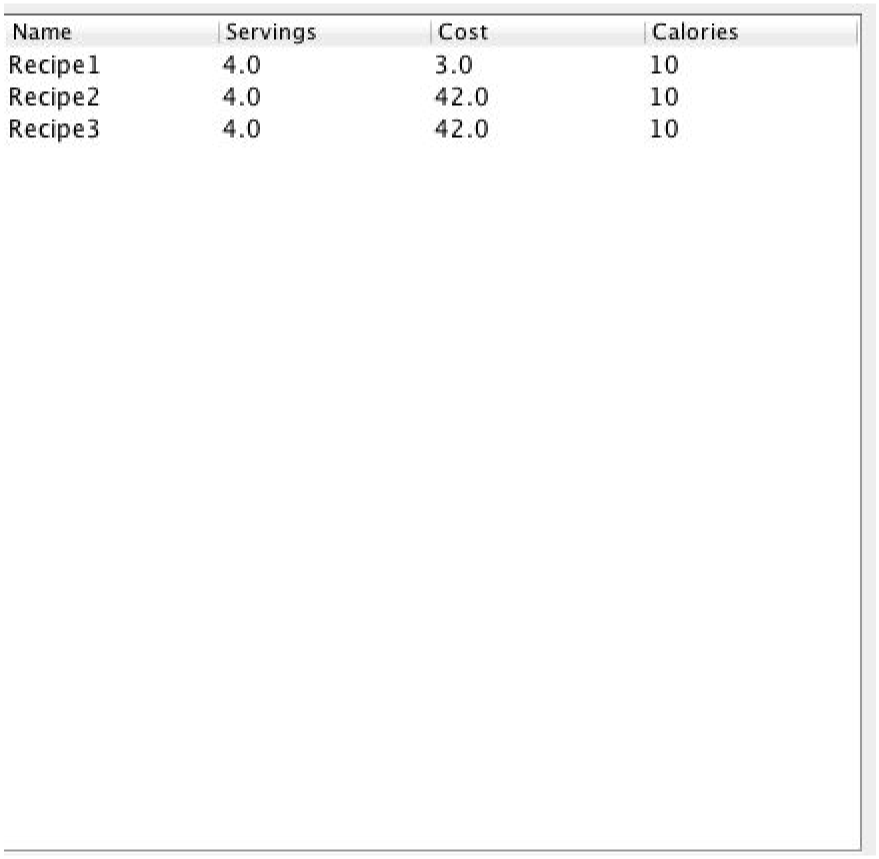
Combo boxes are used to in displaying a list of recipes for the client so that the client does not have to type in the full name of a recipe to search.



Radio Buttons are used so that the client can not only view all options for Ingredient units, but also select and reselect at their behest.

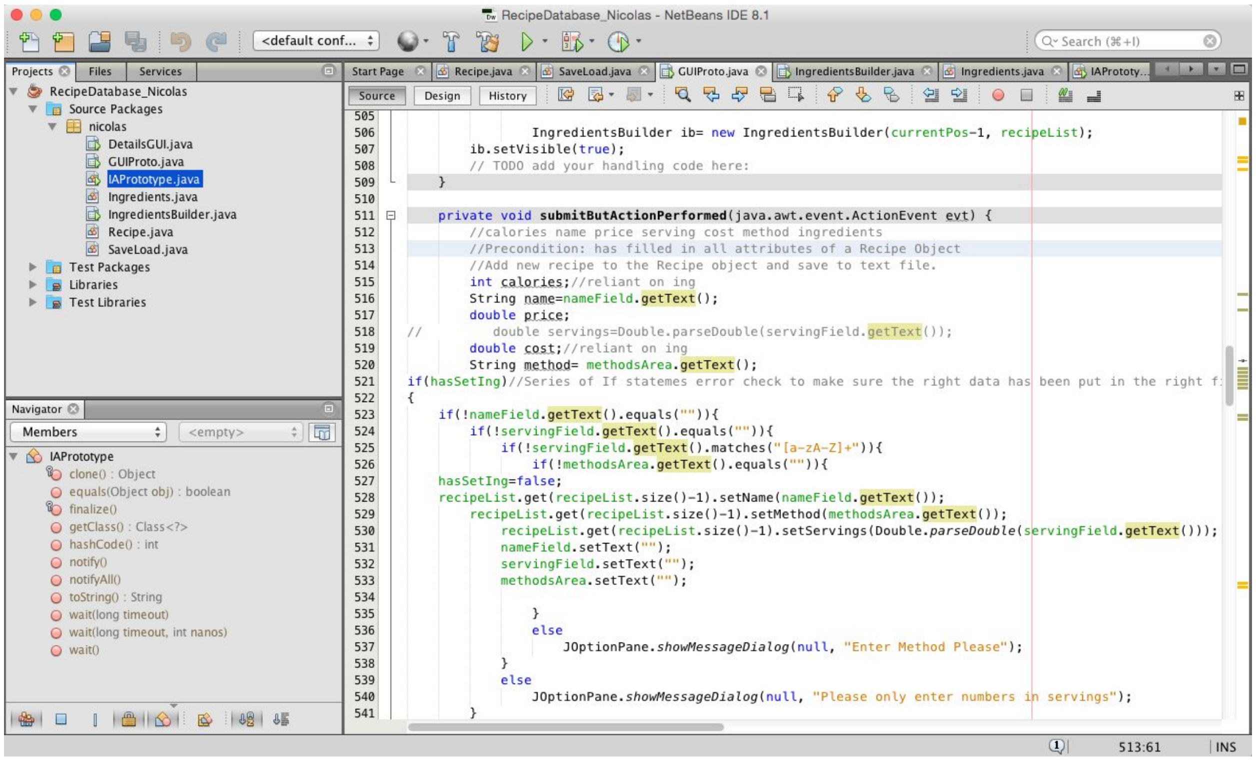


One biggest swing components used was a JTable. This functioned as the main display for the recipes where the client could view upwards of 20 recipes in a single screen. Furthermore the table allows the client to also view certain crucial elements of teh recipe such as calories, servings and cost.



Name	Servings	Cost	Calories
Recipe1	4.0	3.0	10
Recipe2	4.0	42.0	10
Recipe3	4.0	42.0	10

7. Software Tools Used



```
                IngredientsBuilder ib= new IngredientsBuilder(currentPos-1, recipeList);
        ib.setVisible(true);
        // TODO add your handling code here:
    }

    private void submitButActionPerformed(java.awt.event.ActionEvent evt) {
        //calories name price serving cost method ingredients
        //Precondition: has filled in all attributes of a Recipe Object
        //Add new recipe to the Recipe object and save to text file.
        int calories;//reliant on ing
        String name=nameField.getText();
        double price;
//         double servings=Double.parseDouble(servingField.getText());
        double cost;//reliant on ing
        String method= methodsArea.getText();
if(hasSetIng)//Series of If statemes error check to make sure the right data has been put in the right f
{
    if(!nameField.getText().equals("")){
        if(!servingField.getText().equals("")){
            if(!servingField.getText().matches("[a-zA-Z]+")){
                if(!methodsArea.getText().equals("")){
    hasSetIng=false;
    recipeList.get(recipeList.size()-1).setName(nameField.getText());
        recipeList.get(recipeList.size()-1).setMethod(methodsArea.getText());
            recipeList.get(recipeList.size()-1).setServings(Double.parseDouble(servingField.getText()));
            nameField.setText("");
            servingField.setText("");
            methodsArea.setText("");

                }
                else
                    JOptionPane.showMessageDialog(null, "Enter Method Please");
            }
            else
                JOptionPane.showMessageDialog(null, "Please only enter numbers in servings");
        }
```

The NetBeans Integrated Development Environment was used to develop this program. It allowed me to program in Java from my computer. Furthermore, using netbeans allows for easy and repeated compilation so as to test the program on different computers and ensure compatibility. Lastly, the refractor tools where highly useful when variables needed to be renamed throughout a large class.