Criterion C:

List of Simple Techniques through the IA to code functionality of the system, though not included due to word count restrictions:
Encapsulation If statements While loops For loops Data hiding Polymorphism

Technique: Linear Regression Model
Link To Success Criterion/Criteria: - The system should be able to accurately predict the grade of a student in a specific subject, and allow the teacher to have some influence - The system should be able to display a trend graph of student grades against time
Source: (Mohit Gupta_OMG , et al. "ML: Linear Regression.") → Mathematics (3Blue1Brown, director. *Gradient Descent, How Neural Networks Learn | Deep Learning, Chapter 2*.) → LinReg Theory Note: The Code was implemented myself by understanding the theory and coding it. This code is tailored to my purposes. This is my own version of a linear regression algorithm.

Justification:

The **Lin-Reg** Model forms a Line of Best Fit between all grades of a student in a **specific class**, displaying the trend of a student's grades over time on a GUI LineChart, achieving a success criteria. The Model predicts the likely next grade of the student allowing the client predict grades.

Explanation:

The **LinReg** Class instantiates a **LinReg** object. Each LinReg Object has its own dataset (i.e. grades) named **data**[][], and **ArrayList** of variables. Each object has a function, **predict**(), that returns the variables, which are the information needed to create the function of the LOBF. The function where the variables fit to the data are the **gradientDescent**(): it computes the mathematics described in **B**.

Example:

```java
public ArrayList<Double> predict() {
    fit(epoch, 0.01);
    ArrayList<Double> variables = new ArrayList<Double>();
    for (double b : weights) {

        variables.add(b);
    }
    variables.add(bias);

    return variables;
}
```

Returns the prediction model variables
In a straight line y=Mx+c, the function returns m and c
In an ArrayList

```java
public void fit(int epochs, double alpha) {
    for (int epoch = 0; epoch < epochs; epoch++) {
        gradientDescent(alpha);

    }

    return;
}

}
```

```java
public void gradientDescent(double alpha)// run until cost is near 0
{
    for (int sid = 0; sid < 10000; sid++) {
        //we have w and b
        double[] preds = new double[inputs];//dpreds to weights in order
        double[] costs = new double[inputs + 1];//dcosts to weights
        double dpred_dbias = 1;
        double dCost_dpred = 0;
        // to minizize  sosts, we have to this equation costs = sum of (prediction-actual)^2/ data points.
        //dcostsdpred * dpreddweight
        for (int u = 0; u < inputs; u++) {
            double sum = 0;
            for (int i = 0; i < args.size(); i++) {//loops through ArrayList to compute costs
                double predictionFromWeights = data[i][u] * weights[u] + bias;
                double y = data[i][1];
                sum += (predictionFromWeights - y) * data[i][u]; //calculating derivating for weight
            }
            sum /= args.size();
            weights[u] -= alpha * sum; //updating the weights
        }
        double sum = 0;
        for (int u = 0; u < inputs; u++) {
            for (int i = 0; i < args.size(); i++) {
                double predictionFromWeights = data[i][u] * weights[u] + bias;
                double y = data[i][1];
                sum += (predictionFromWeights - y);
            }
            sum /= args.size();
        }
        bias -= alpha * sum; //updating the bias
    }
    return;
}
```

Sum over all data points to compute sum as stated in mathematics

$m = m - \alpha \frac{\partial J}{\partial m}$

$c = c - \alpha \frac{\partial J}{\partial c}$

$$\frac{\partial J}{\partial m} = \sum \frac{\partial J}{\partial h(X_i)} \cdot \frac{\partial h(X_i)}{\partial m}$$

$$\frac{\partial J}{\partial h(X_i)} = 2(h(X_i) - Y_i); \quad \frac{\partial h(X_i)}{\partial m} = \frac{\partial}{\partial m}(mX_i + c) = X_i$$

$$\frac{\partial J}{\partial m} = \frac{1}{v} \sum_{i=0}^{v} [2(h(X_i) - Y_i)] X_i]$$

$$\frac{\partial J}{\partial c} = \sum \frac{\partial J}{\partial h(X_i)} \cdot \frac{\partial h(X_i)}{\partial c}$$

$$\frac{\partial J}{\partial h(X_i)} = 2(h(X_i) - Y_i); \quad \frac{\partial h(X_i)}{\partial c} = \frac{\partial}{\partial c}(mX_i + c) = 1$$

$$\frac{\partial J}{\partial c} = \frac{1}{v} \sum_{i=0}^{v} [2(h(X_i) - Y_i)] \cdot 1]$$

Technique: Neural Network (Python Programming)
Link To Success Criterion/Criteria: - The system should be able to accurately predict the grade of a student in a specific subject, and allow the teacher to have some influence
Sources: Java implementation: (Deeplearning4j - Skymind, director. *Lecture 7 | Import a Keras Neural Net Model into Deeplearning4j*.) Deepleaning4j documentation: ("Multilayer Network." *Deeplearning4j*,) Keras documentation :("Keras: The Python Deep Learning Library.")

Justification:

Predicted grades are best calculated based on how past students with similar trends performed. These indicators are a reliable form of predicting grades from past data if a student has a grade trend that isn't clear for a teacher to predict. This helps achieve the success criteria as predicting grades becomes easier for the Teacher.

Explanation:

There are two **neural network algorithms** in **Python** using **Keras**; Both trained on 1000 data points provided by my client. Each model has 5 Dense Layers. The Input Layers are an input of 14 grades. The Hidden Layers increases in neuron count to accurately fit the data: I conducted trial and error to scout the highest accuracy. The first algorithm will output a **regression** output (i.e. single decimal grade from 1.0~7.9), and the second algorithm will output the **probability** that the student with the inputted grades will attain each grade level as its output is a **softmax** layer. Each Model was saved as ".**h5**" files and was accessed in java by using the **deeplearning4j**, and **Nd4j** libraries. The **NNModel.java** class uses aforementioned libraries to load both models: the function **Dense**() loads the regression model, and the **Probabilities**() loads the probability model .

Python Models and Java implementation of Code:

```
import numpy as np
from numpy import loadtxt
from keras.models import Sequential
from keras.models import load_model
from keras.layers import Dense
# load the dataset

dataset = loadtxt('hellobello.csv', delimiter=',')
# split into input (X) and output (y) variables
X = dataset[:,0:14]

y = dataset[:,15:22]

# define the keras model
model = Sequential()
model.add(Dense(32, input_dim=14, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(128, activation='sigmoid'))
model.add(Dense(16, activation='sigmoid'))
model.add(Dense(7, activation='softmax'))
# compile the keras model
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
# fit the keras model on the dataset
model.fit(X, y, epochs=200, batch_size=10)
# evaluate the keras model

model.save('prob')  # save everything in HDF5 format

model_json = model.to_json()  # save just the config. replace with "to_yaml" for YAML serialization
with open("prob_config", "w") as f:
    f.write(model_json)

model.save_weights('prob_weights') # save just the weights.

_, accuracy = model.evaluate(X, y)
print('Accuracy: %.2f' % (accuracy*100))

predict_sample = model.predict(np.array([[5.6,5.8,7.0,7.5,6.8,6.8,6.8,7.8,7.6,5.4,7.3,7.3,6.0,7.5]]))

print(predict_sample)
```

'hellobello.csv' contains 800 data sets for the model to train.

14 inputs for 14 grades

7 neuron layer, including softmax, To output 7 probabilities as intended

Save the model to it can be Used in java

```
import numpy as np
from numpy import loadtxt
from keras.models import Sequential
from keras.models import load_model
from keras.layers import Dense
# load the dataset
dataset = loadtxt('sheet1.csv', delimiter=',', usecols=range(15), skiprows=1)
# split into input (X) and output (y) variables
X = dataset[:,0:14]
y = dataset[:,14:15]

print(X[1],y[1])
# define the keras model
model = Sequential()
model.add(Dense(32, input_dim=14, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(128, activation='sigmoid'))
model.add(Dense(16, activation='sigmoid'))
model.add(Dense(1))
# compile the keras model
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
# fit the keras model on the dataset
model.fit(X, y, epochs=200, batch_size=10)
# evaluate the keras model
_, accuracy = model.evaluate(X, y)
print('Accuracy: %.2f' % (accuracy*100))
predict_sample = model.predict(np.array([[5.6,5.8,7.0,7.5,6.8,6.8,6.8,7.8,7.6,5.4,7.3,7.3,6.0,7.5]]))
print(predict_sample)

model.save('regular')  # save everything in HDF5 format

model_json = model.to_json()  # save just the config. replace with "to_yaml" for YAML serialization
with open("regular_config", "w") as f:
    f.write(model_json)

model.save_weights('regular_weights') # save just the weights.
```

'sheet1.csv contains 800 data sets for the model to train.

14 inputs for 14 grades

Single layer output for single predictive grade

Save the model to it can be Used in java

```java
public ArrayList<Double> Probability(ArrayList<Double> grades) throws Exception {

    //Loading the model from python
    MultiLayerNetwork model = KerasModelImport.importKerasSequentialModelAndWeights("prob");

    //MultiLayerNetwork model take an input of a a INDArray
    //so, I load 14 grades from the arrayList into an array
    double[] arr_1d = new double[14];
    for (int i = 0; i < grades.size(); i++) {
        arr_1d[i] = grades.get(i);
    }

    //then from an double[] array to an INDArray
    INDArray x = Nd4j.zeros(1, arr_1d.length);
    for (int i = 0; i < arr_1d.length; i++) {
        x.putScalar(0, i, arr_1d[i]);
    }

    //out will use the loaded model and the data in INDArray x to predict
    //7 probabilities. It will create a probability distribution
    INDArray output = model.output(x);

    //createing an arrayList to store each of the 7 probabilities
    ArrayList<Double> probs = new ArrayList<Double>();
    for (int i = 0; i < 7; i++) {
        //rounding each probability to two decimals
        double gg = Math.round(output.getDouble(0, i) * 100.0) / 100.0;
        probs.add(gg);
    }
    return probs;
}

//note to self: USE UPDATED DEPENDANCIES.
public double Dense(ArrayList<Double> grades) throws Exception {

    // LOADING THE MODEL from python
    MultiLayerNetwork model = KerasModelImport.importKerasSequentialModelAndWeights("regular");

    //MultiLayerNetwork model take an input of a a INDArray
    //so, I load 14 grades from the arrayList into an array
    double[] arr_1d = new double[14];
    for (int i = 0; i < grades.size(); i++) {
        arr_1d[i] = grades.get(i);
    }

    //then from an double[] array to an INDArray
    INDArray x = Nd4j.zeros(1, arr_1d.length);
    for (int i = 0; i < arr_1d.length; i++) {
        x.putScalar(0, i, arr_1d[i]);
    }

    //out will use the loaded model and the data in INDArray x to predict a value between
    //1.07~.9
    INDArray output = model.output(x);

    //Rounding the decimal to two decimal places
    return Math.round(output.getDouble(0, 0)*100.0)/100.0;

}
```

Technique: Serialization and Deserialization
Link To Success Criterion/Criteria: - The system should automatically save data so that the system does not reset every time the application is closed
Source:(Liguori, Robert, and Patricia Liguori.)

Justification:

The client can close the application and and reopen it without having to re-enter all grades. All accounts, grades, classes, objects will be saved to data is retained; this achieves the success criteria and increases the usability and convenience of the system to the client.

Explanation:

The classes **Student**, **Class**, **Teacher**, implement the interface *Serializable*. The classes **StudentManager**, **TeacherManager**, **LoginController**, **CreateAccountController**, each have a **save**() function to save an **ArrayList** of objects to a file, for instance the array containing all Students, Teachers, Login Details, Classes. When the system is loaded, all ArrayLists are loaded using the **load**() method and all data is retained. I deserialize them back into objects using **casting**

Example:

```java
public static void load() {
    ArrayList<Student> t = null;

    try {

        FileInputStream fi = new FileInputStream("AllStudents.txt");
        ObjectInputStream oi = new ObjectInputStream(fi);

        // Read objects
        allStudents = (ArrayList<Student>) oi.readObject();

        oi.close();
        fi.close();

    } catch (FileNotFoundException e) { //https://www.mkyong.com/java/how-to-read-and-write-java-object-to-a-file/
        System.out.println("File not found");

    } catch (IOException e) {
        System.out.println("Error initializing stream");

    } catch (ClassNotFoundException e) {

        e.printStackTrace();

    } catch (Exception e) {

    } finally {

    }

    }
```

```java
public static void save() {

    try {
        FileOutputStream f = new FileOutputStream(new File("AllStudents.txt"));
        ObjectOutputStream o = new ObjectOutputStream(f);

        // Write objects to file
        o.writeObject(allStudents);

        o.close();
        f.close();

    } catch (IOException e) {
        System.out.println("Error initializing stream");
    } finally {
        return;
    }

}
```

Technique: Transient Variables
Link To Success Criterion/Criteria: - The system should automatically save data so that the system does not reset every time the application is closed - The system Java Application must be less than 5GB total.
Source:(Liguori, Robert, and Patricia Liguori.)

Justification:

Programming **transient** variables allowed me to only have what is needed in the system, and increased security of inactive users' data because when objects are Serialized, transient variables are not. Also, less information is stored, so less storage is required, achieving the next success criteria

Explanation:

The Person Class has two **transient** variables: username and password. Upon serialization of Student or Teacher objects, username and password aren't serialized. When a user logs on, all objects are then deserialized, and only current registered user's object's username and password are loaded. All other object's username-password are null Strings.

Person Class:

```
public abstract class Person implements Serializable {

    private String firstname;
    private String surname;
    protected String schoolName;
    private int id;
    private transient String password;
    private transient String username;
```

Use of Transient Variables in User Login System Code (Details of Mechanics):

```
if (tokens[0].compareToIgnoreCase("Teacher") == 0) {
    int id = Integer.parseInt(tokens[3]);

    Teacher h = TeacherManager.findTeacher(id);
    h.setPassword(tokens[2]);
    h.setUsername(tokens[1]);

    GRADE_PREDICTION.primaryTeacher = h;
    GRADE_PREDICTION.primaryStudent = null;

    System.out.println(GRADE_PREDICTION.primaryTeacher);

    //Teacher Account Switches to Teacher Page --> Search.fxml
    appStage = (Stage) submit.getScene().getWindow();
    root = FXMLLoader.load(getClass().getResource("Search.fxml"));
    Scene scene = new Scene(root);
    appStage.setScene(scene);
    appStage.show();
    return;
}
if (tokens[0].compareToIgnoreCase("Student") == 0) {
    int id = Integer.parseInt(tokens[3]);

    Student h = StudentManager.findStudent(id);
    h.setPassword(tokens[2]);
    h.setUsername(tokens[1]);

    GRADE_PREDICTION.primaryTeacher = null;
    GRADE_PREDICTION.primaryStudent = h;

    System.out.println(GRADE_PREDICTION.primaryStudent);

    //Student Account Switches to Student Page --> StudentPage.fxml
    appStage = (Stage) submit.getScene().getWindow();
    root = FXMLLoader.load(getClass().getResource("StudentPage.fxml"));
    Scene scene = new Scene(root);
    appStage.setScene(scene);
    appStage.show();

}
```

If the Account is a teacher Account

The Teacher with the the ID *Id* is loaded into the Teacher object *Teacher h*. Then, that Teacher, *h*, who is the current user of the system has their username and password stored in the object. These were the transient variables that weren't saved when all Teacher objects were Serialized.

Hence, only the current user has their username and password loaded into the system; this increases security and enables second layer data hiding.

If the Account is a Student Account

The Student with the the ID *Id* is loaded into the Student object *Student h*. Then, that Student, *h*, who is the current user of the system has their username and password by getting the text from the TextFields, just like the Teachers.

Again, just like the Teacher accounts, only the current Student user has their username and password loaded into the system because the Transient variables didn't allow for username and password to be Serialized; this increases security and enables second layer data hiding.

Technique: Functional Interface with Lambda Expressions
Link To Success Criterion/Criteria: - The system should be able to accurately predict the grade of a student in a specific subject, and allow the teacher to have some influence
Source:(Liguori, Robert, and Patricia Liguori.)

Justification:

The **Functional Interface** made the code more modular. Moreover, it's easier to create use the prediction algorithms when necessary because each algorithm was a separate instantiation of the predictable interface.

Explanation:

The functional interface, **Predictable**,contains one method, **predict**(), so that many instatiations of the predictable class will have different **predict**() methods, allowing for the creation of many prediction algorithms. Predictable has three instantiation, each containing its own unique **predict**(): a unique prediction algorithm. I used **Lambda Expressions** to create the methods as Predictable is a single method FI: increase code readability.

Example:

```
import java.util.ArrayList;
import java.io.*;

@FunctionalInterface
public interface Predictable {
//there are many algorithms for prediction,
//so im using lambda expressions to and an interface to make different prediction algorithms

    public double predict(ArrayList<Double> grades, ArrayList<Double> attributes, Student s);

    default void viewTrend(ArrayList<Double> grades) {
        System.out.println("Nothing");
    }

}
```

```
public ArrayList<Double> predictGrade(int studentID, Class c, ArrayList<Double> g) throws IOException {
    //returns an arraylist of the different predicted grades individually. eg {5.6, 6. 7.1}
    ArrayList<Double> predicteds = new ArrayList<Double>();

    Student s = c.findStudent(studentID);

    if (s == null) {
        return null;
    }
```

```
//NN Predict
Predictable NNpredict = (grades, attributes, h) -> {
    double grade = 0;
    try {
        NNModel model = new NNModel();
        ArrayList<Double> temp = new ArrayList<Double>();
        int temp1 = grades.size();
        for (int i = 0; i < 14; i++) {
            temp.add(grades.get(i % temp1));
        }
        grade = model.Dense(temp);
        return grade;
    } catch (Exception e) {
        System.out.println("error1");
        return grade;
    }

};
```

First instantiation of Predictable, using lambda expression to define the single predict() method because Predictable is a functional interface. It has a unique algorithm

```
//TA Predict
Predictable TeacherAverage = (grades, attributes, h) -> {
    ArrayList<Double> gradeList = grades;
    int size = gradeList.size();
    if (size == 0) {
        return 0;

    }
    if (size <= 3) {
        return gradeList.get(size - 1);
```

Second instantiation, another unique algorithm

```
Predictable gradientPredict = (grades, attributes, h) -> {

    // compute the gradients
    ArrayList<Double> gradients = new ArrayList<Double>();
    double sf1 = 0.5;
    double outlier = 1.5;
    System.out.println(grades);
    for (int i = 0; i < grades.size() - 1; i++) {
        double result = grades.get(i + 1) - grades.get(i);
        System.out.println(result);
        if (result > outlier) {// to keep outliers in check
            result *= sf1;
        }
        System.out.println(result);
        gradients.add(result);
    }

    LinReg r = new LinReg(gradients, 10);

    ArrayList<Double> thing = r.predict();

    double pred = 0;

    pred += thing.get(0) * gradients.size(); // this finds the next point on the line.

    pred += thing.get(1);
    double sf2 = 0.2; //this is to limit the chnage in gradient.

    if (h != null) {
        h.gradient = pred * sf2;

    }

    return pred;

};
```

```
    ArrayList<Double> wow = new ArrayList<Double>();
    for (Double d : g) {
        if (d != 0.0) {
            wow.add(d);
        }
    }
    if (wow.size() == 0) {
        for (int i = 0; i < 3; i++) {
            predicteds.add(0.0);
        }
        return predicteds;
    }

    predicteds.add(NNpredict.predict(wow, null, s));

    predicteds.add(LinRegPredict.predict(wow, null, s));

    predicteds.add(TeacherAverage.predict(wow, null, s));

    if (wow.size() > 1 && wow.size() <= 5) {

        gradientPredict.predict(wow, null, s);

    }
    if (wow.size() > 5) {

        ArrayList<Double> gdg = new ArrayList<Double>();
        for (int i = wow.size() - 3; i < wow.size(); i++) {
            gdg.add(wow.get(i));
        }
        gradientPredict.predict(gdg, null, s);

    }

    return predicteds;

}
```

Implementation of each lambda expression to create a combines prediction by storing all three predictions in an arraylist. Each call predict, but their predict functions are unique because they were defined differently due to the use of lambda expressions to define them.

Technique: GUI with Lambda Expressions
Link To Success Criterion/Criteria: - The system Java Application must be less than 5GB total. - The client must be able to provide fluent transition between pages: within 3 clicks a user should locate anywhere - The client must be able to enter, edit, and delete student grades - The system should be able to display a trend graph of student grades against time
Source:(Liguori, Robert, and Patricia Liguori.) Note: I saw NetBeans' Examples on JavaFX code and learnt from there. ***All code is original***

Justification:

To appeal (aesthetically) to users who find it difficult to use a menu driver or a command line interface. GUI can display a trendline graph of student's grades; text can be inputted easily; WIMP make navigation fluid, increasing system usability.

Explanation:

The GUI was coded using **JavaFX**. The layout of the files were created using an application called SceneBuilder, which created .**FXML** files. Each **.FXML** had a **controller** class with static variables as the GUI elements. Each controller has a Button to switch between different **Scenes**, and transfer the **Parent stage** to a different one, making the GUI a fully connected layout. A new **Window** only appears when the system prompts to reload grades of a student that was removed from a class. Here, I use the function **.showAndWait()** so that the program does not continue until an option is selected. I incorporated elements such as **Labels** to display text, **LineCharts** to display trends, **Buttons** to initiate actions, **TextField** to allow user input. The use of **lambda expressions, and even nested lambda expressions** was vital as all **.SetOnAction()** for **Events** and **.OnKeyPressed()** were coded using **LE's** to perform unique actions increasing the ease to programming the system. For instance, a live search bar where every key pressed will trigger a search for a list of students containing a **TextField**'s content in their name.

Method that is called by a button when Adding a Student to a class:

This method is run when the Submit button is pressed on the AddStudent page, where the user adds a student to a class.

```
@FXML
private void ButtonHandler(ActionEvent event) throws Exception { // this switchs scene not stage

    if (tempClass.getClassStudents().contains(currentStudent) == false) {
        System.out.println(currentStudent.getGrades().keySet());

        currentStudent.getGrades().keySet().forEach(c -> {
            if (c.getID() == tempClass.getID()) { // if there are multiple classes, which one should we pick.
                try {
                    ViewClassInfoController.s = currentStudent;
                    ViewClassInfoController.c = tempClass;

                    System.out.println("hello");
                    Stage newStage = new Stage();
                    Parent root = FXMLLoader.load(getClass().getResource("ViewClassInfo.fxml"));
                    Scene scene = new Scene(root);
                    newStage.setScene(scene);
                    newStage.showAndWait();
                } catch (Exception e) {
                    System.out.println("hi");
                    e.printStackTrace();
                }

            }

        });

        if (transfer == false) {
            currentStudent.getGrades().remove(tempClass);
        }

        tempClass.addStudent(currentStudent);

        appStage = (Stage) submit.getScene().getWindow();
        root = FXMLLoader.load(getClass().getResource("Search.fxml"));
        Scene scene = new Scene(root);
        appStage.setScene(scene);
        appStage.show();
    }

}
```

Example of lambda expression to increase code readability

.showAndWait() to allow the new screen ViewClassInfo to finish running. If a student has a record of being in the class, the system will ask to reload the grades from before.

First, I'm declaring the Stage to be changed at the stage that houses the current button. Then I'm loading the other Scene I want to travel to, and instantiating a new scene with that root. Then, I change my previously defined Stage and make is display the new Scene, thereby switching Scenes, but not creating a new window: In this case, I switched from AddStudent to Search.

ViewClassInfo.fxml:

```
@FXML
private void ButtonHandlerYes(ActionEvent event) throws Exception {

    //if the Yes button is selected, the variable "transfer" in the AddStudent page is set to true
    //the student's old grades will be loaded back into the class
    AddStudentController.transfer = true;
    Stage stage = (Stage) yes.getScene().getWindow();
    stage.close();
}
@FXML
private void ButtonHandlerNo(ActionEvent event) throws Exception {

    //if the Yes button is selected, the variable "transfer" in the AddStudent page is set to false
    //the students grades will not be retained
    AddStudentController.transfer = false;
    Stage stage = (Stage) no.getScene().getWindow();
    stage.close();

}
@Override
public void initialize(URL url, ResourceBundle rb) {
    String text = "The student " + s.getFirstname() + " already has record of attending this class. "
            + "\nWould you like to load his previously attained grades?";
    prompt.setText(text);
}
```

** When a student is removed from the class, the student object is removed from the class' student list, but the class and grades aren't removed from the student. If a student is added back into a class they were deleted from, the ViewClassInfo screen is displayed. On this page, there is a "Yes" and "No" button, prompting "Would you like to load previously attained grades?" The "Yes" button runs ButtonHandlerYes(), and the "No" button runs ButtonHandlerNo().*

GUI LineChart:

```
String[] labels = {"11Q1-1", "11Q1-2", "11Q2-1", "11S1", "11Q3-1", "11Q3-2", "11Q4-1", "11S2",
    "12Q1-1", "12Q1-2", "12Q2-1", "12S1", "Mock", "I.A"};
lc.setTitle(currentStudent.getFirstname() + "s Trend");
XYChart.Series series = new XYChart.Series();
series.setName("Trend");
ArrayList<Integer> indexs = new ArrayList<Integer>();
XYChart.Data temp = new XYChart.Data();
XYChart.Data temp2 = null;
for (int i = 0; i < grades.size(); i++) {
    if (grades.get(i) != 0.0) {
        //skips all spaces that are null
        series.getData().add(new XYChart.Data(labels[i], grades.get(i)));
        XYChart.Data temps = new XYChart.Data(labels[i], grades.get(i));
        temp2 = new XYChart.Data(labels[i], grades.get(i));
        temp = temps;

        indexs.add(i);

    }
}

lc.getData().add(series);
```

Adds the series to the graph and displays it

In this for loop, data points are being added to the series, which is essentially a collection of x-coordinates and y-coordinates. On the x-axis, there is the name of the assessment, and on the y-axis there are the grades. Each index in the grades array is mapped to an index in the labels array, for instance, if *i=2*, then the third grade will be paired with the label "11Q2-1". Adding "labels[i],grades.get(i)" does just that. This is possible, because empty grades are denoted by 0.0, so if the grade in the array is 0.0, the for-loop *counter i* ,will be skipped! There can be some gaps in the grades array.
This allows for flexibility in grade entering because not every grade needs to be entered in succession.

Active searchBar (every time a key is pressed, the search items update):

This method is the active search bar method. It's purpose is to allow the teacher to search for students, but every time a key is pressed the search options narrow according to the text in the search bar.

```
@Override
public void initialize(URL url, ResourceBundle rb) {

    searchBar.setOnKeyPressed(event -> {

        ObservableList<Student> items = FXCollections.observableArrayList();
        items.setAll(StudentManager.findStudents(searchBar.getText()));
        lv.setItems(items);

        ListView list = lv;

        list.setOnMouseClicked(new EventHandler<MouseEvent>() {

            @Override
            public void handle(MouseEvent me) {
                //Check wich list index is selected then set txtContent value for that index
                list.getItems().forEach((c) -> {
                    if (list.getSelectionModel().getSelectedIndex() == list.getItems().indexOf(c)) {
                        currentStudent = (Student) c;
                    }
                });

            }

        });

        lv.refresh();

    });
}
```

First lambda expression to code the .setOnKeyPressed() method

First, I declare an ObservableList of Students because that is what I am search for: Students. Then, I search for all of the student's who's names contain the searchBar's text. The findStudent() method searches through all students and returns an ArrayList of all students who have a certain string in their name. Then, I set the searchBars items to that ArrayList of students to only display them.

The nested Lambda Expression is for each of the items (the students with the substring in their name). So, I figured if the ID if the student pressed is equal to the student object's ID, then that object must be the intended selected object. Then, if this condition was met, I set the currentStudent equal to that object.

Then I refresh the list to display the students

Technique: Aggregation using HashMaps
Link To Success Criterion/Criteria: - The client must be able to enter, edit, and delete student grades - The client should be able to add all their Classes and Students to classes, with data separate from each student.
Source: *Java HashMap*, www.w3schools.com/java/java_hashmap.asp.

Justification:

Aggregation allows the current user, if a Teacher, to access all Objects in the system: all Students, Classes and Grades. This fulfills both success criteria because it allows for a more efficient, connected system for the Teacher: as everything is accessible, the Teacher can edit grades of any student, add students to any class.

Explanation:

The **Student** class contains a **HashMap** that aggregates all the classes that the **student** objectives in. Each **Student** has an **ArrayList** of trends, grades, probabilities(from neural network), and a predicted grade for each class. When a student is added into a class, a **Map** is *put* into each of the Student's HashMaps with a **key:Class**, and an **associated value** of an ArrayList. Teacher has an **Arraylist** of all classes he/she teaches. The **Class** class aggregates an **ArrayList** of all students in the class, and 'has a' class **Teacher**.

Example:

```
public class Teacher extends Person implements Serializable {

  public class Class implements Serializable {

      private static final long serialVersionUID = 2L;

      private Teacher classTeacher;
      private ArrayList<Student> classStudents = new ArrayList<Student>();
      private int classID;
      private Subject classSubject; //includes name
```

```
public class Student extends Person implements Serializable {

    private static final long serialVersionUID = 3L;

    private ArrayList<Double> trend = new ArrayList<Double>();

    // private ArrayList<Double> grades;
    private Map<Class, ArrayList<Double>> grade = new HashMap<Class, ArrayList<Double>>(); // this is a hashmap...
    private Map<Class, ArrayList<Double>> trends = new HashMap<Class, ArrayList<Double>>();
    private Map<Class, ArrayList<Double>> probabilities = new HashMap<Class, ArrayList<Double>>();
    private Map<Class, Double> predictedGrades = new HashMap<Class, Double>();
```

Technique: Validation Exception Handling using Try-Catch blocks
Link To Success Criterion/Criteria: - The system should be able to prevent the input of invalid data, incorrectly formatted data, or wrong data: there should be proper error management when data is inputted
Source: *Java HashMap*, www.w3schools.com/java/java_hashmap.asp.

Justification:

Validation will prevent any abrupt breaks in the system, allowing the success criteria to be achieved because Try-Catch will manage all errors and allow me to deal with them appropriately.

Explanation:

Certain methods throw IOException, but **Try-Catch** blocks are often present to not only **catch**, but act on caught Exceptions. For instance, the **StudentManager** Class, the function **save**() throws an exception to prevent the chance of an application class due to an error in initializing the IO stream between the code and the file "AllStudent.txt". In fact, the first time the application is run, where the file is empty, the Exception is caught and dealt with.

```
public static void save() {

    try {
        FileOutputStream f = new FileOutputStream(new File("AllStudents.txt"));
        ObjectOutputStream o = new ObjectOutputStream(f);

        // Write objects to file
        o.writeObject(allStudents);

        o.close();
        f.close();

    } catch (IOException e) {
        System.out.println("Error initializing stream");
    } finally {
        return;
    }

}
```

Technique: Inheritance with Abstract Class
Link To Success Criterion/Criteria: - The client must be able to enter, edit, and delete student grades - The client must be able to contact a student regarding any matter
Source: Liguori, Robert, and Patricia Liguori.

Justification:

To make my code more **modular**. **Person** is **abstract** (super class) because it shouldn't be instantiated; my system depends on the distinction between a **Student** and a **Teacher**: both inherit common traits from Person. In order for a Teacher to contact students, the teacher needs to have their own data.

Explanation:

I used **abstraction** to design the template of a person. The **Person** class has common attributes and functions: **name**, **id, password**, **username**. **Student** and **Teacher** inherit these common traits from Person. **toString()** in **Person** is also an **abstract** function to ensure there is a toString defined in **Student** and **Teacher**. This establishes code readability.

```java
public abstract class Person implements Serializable {

    private String firstname;
    private String surname;
    protected String schoolName;
    private int id;
    private transient String password;
    private transient String username;

    protected Person(String firstname, String surname, String schoolName, int id) {
        this.firstname = firstname;
        this.surname = surname;
        this.schoolName = schoolName;
        this.id = id;
    }
```

```java
public class Teacher extends Person implements Serializable {

    private static final long serialVersionUID = 1L;

    private ArrayList<Class> classes = new ArrayList<Class>();

    public Teacher(String firstname, String surname, String schoolName, int id) {
        super(firstname, surname, schoolName, id);
    }
```

```java
public class Student extends Person implements Serializable {

    private static final long serialVersionUID = 3L;

    private ArrayList<Double> trend = new ArrayList<Double>();

    // private ArrayList<Double> grades;
    private Map<Class, ArrayList<Double>> grade = new HashMap<Class, ArrayList<Double>>(); // this is a hashmap...
    private Map<Class, ArrayList<Double>> trends = new HashMap<Class, ArrayList<Double>>();
    private Map<Class, ArrayList<Double>> probabilities = new HashMap<Class, ArrayList<Double>>();
    private Map<Class, Double> predictedGrades = new HashMap<Class, Double>();

    double gradient = 0;

    public Student(String firstname, String surname, String schoolName, int id) {
        super(firstname, surname, schoolName, id);
    }

    public Student() {
        super();
    }
```

Technique: Binary Search
Link To Success Criterion/Criteria: The client should be able to obtain the grades of a student by searching through firstname, surname, or ID
Source: Drien, Marcos.

Justification:

Search for Students by their ID. $O(log\ n)$is very efficient.

Explanation:

The Teacher and Class classes have a function to search for students.

Example:

```
public Student findStudent(int id) {
    boolean found = false;
            int low = 0;
            int high = this.classStudents.size();
            int place = -1;
            while(!found && (low<high)){
                int index = (low+high)/2;
                if(this.classStudents.get(index).getID()==id){
                    found = true;
                    place = index;
                    return this.classStudents.get(place);
                }
                else{
                    if(id<this.classStudents.get(index).getID()){
                        high = index-1;
                    }
                    else{
                        low = index+1;
                    }
                }
            }

    return null;
}
```

Technique: Sorting
Link To Success Criterion/Criteria: The client should be able to obtain the grades of a student by searching through firstname, surname, or ID
Source: Liguori, Robert, and Patricia Liguori. Drien, Marcos

Justification:

Binary Search needs a sorted Array so it helps the success criteria. Provides structure to listing students, so usability is increased.

Explanation:

The Class class sorts using insertion sort.

Example:

```
public void sortClasses(int setting) {

    int csp = 0;
    while (csp < this.classes.size()) {
        int i = csp;
        Class se = this.classes.get(i);
        int j = i + 1;
        while (j < this.classes.size()) {

            switch (setting) {

                //sort by ID
                case (0):
                    if (this.classes.get(j).getID() < se.getID()) {
                        i = j;
                        se = this.classes.get(j);
                    }
                    break;

                //sort by subject name
                case (1):
if (this.classes.get(j).getClassSubject().getSubjectName().compareToIgnoreCase(se.getClassSubject().getSubjectName()) < 0) {
                        i = j;
                        se = this.classes.get(j);
                    }
                    break;

            }
            j++;
        }
        this.classes.set(i, this.classes.get(csp));
        this.classes.set(csp, se);
        csp++;
    }

}
```