**Github**

**Exploratory Projects**

Two folders within this repository.

**Revision control:**

Within projects is 3 different folders (independent, semi, integrated).

It shows three different methods of the same page.

**Independent:** Is a variety of classes which each do their entirely own thing.  When our page consists of buttons, they are called by the Button method’s and added into a revisionLogButton ArrayList then transferred over to the revisionLog page.

**Semi:** Takes away the use of ArrayList but still has a Button Class where the buttons are initialized, However, they are declared in the revisionLog class.

**Integrated:** Integrated has everything in one class. Where there are methods consisting of button, textfield, label initializations that are then called from the revisionLog method.

**Projects:**

Consists of code for:

**Help:**

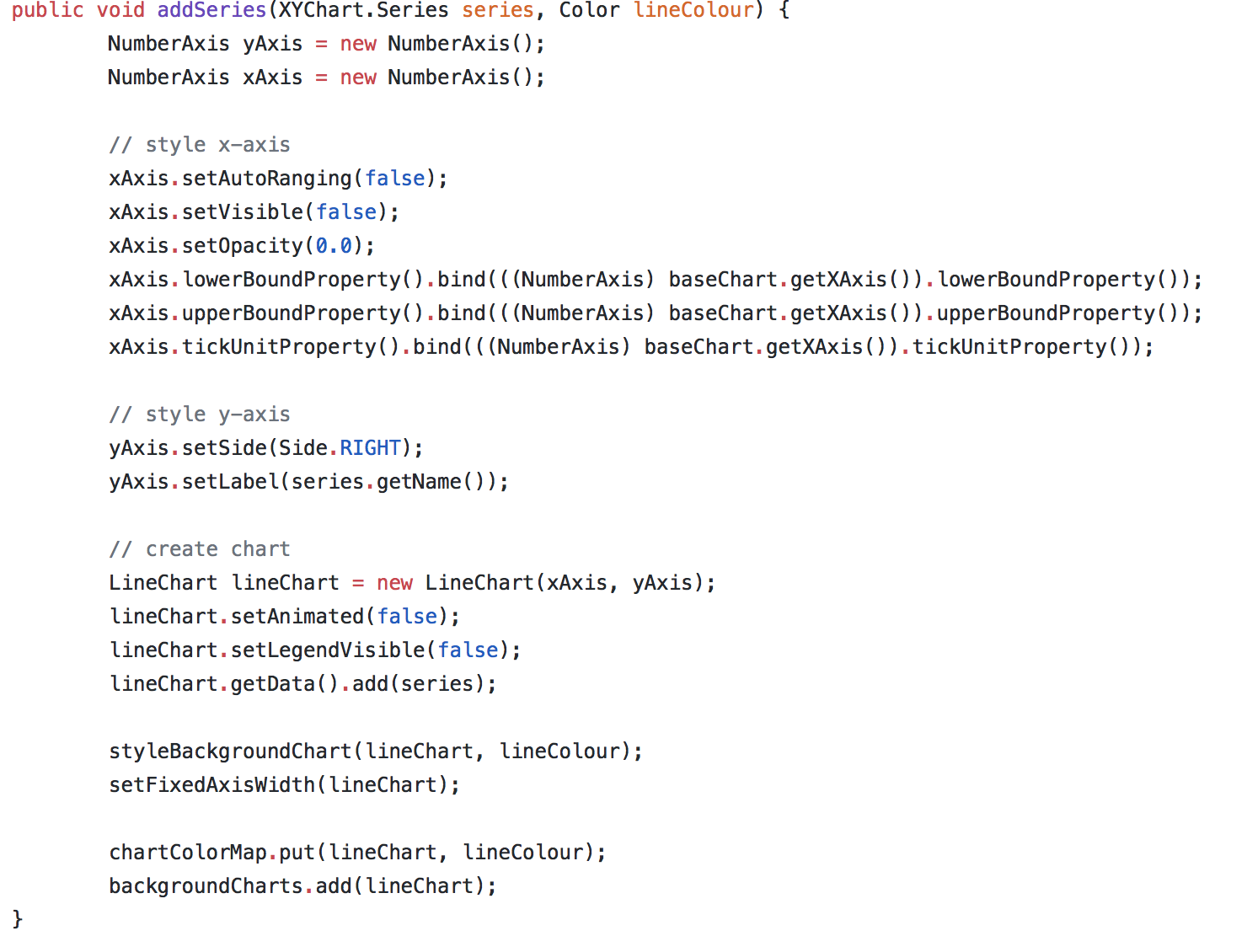
Opens up a new window that contains definitions for everything on the current page.

As well as information on rules and special events that occur on that page.

**Print:**

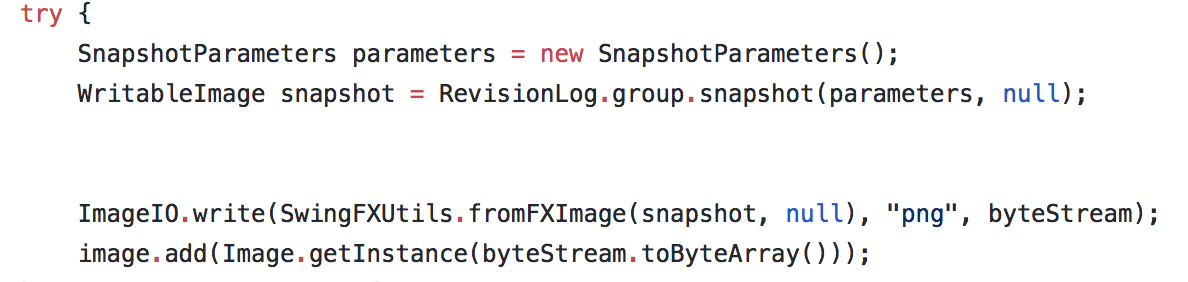
On button action, opens up a page setup which allows the user to select page orientation, size, and printer.

**STD Config:**

Consists of a class that demonstrates the methods used to put multiple Y-axis on a chart

**Save as PDF:**

Takes a snapshot of the current scene (full screen snapshot), and puts the snapshot on a pdf Document and saves it to a folder on the desktop.



**Zoom:**

Consists of a folder with revisionLog zoom. For semi distributed code(different classes).

As, well as several classes which were used for scaling CSS buttons which all have zoom code implemented as well as code which scales the buttons at every zoom implementation.

**Main**

One folder within this repository

**STD. Config**

Contains one folder

**STD Log Finished**

**jars**

Contains every .jar file needed to run the complete code without error.

**activation.jar**

**itextpdf-5.2.1.jar**

**mail.jar**

**mysql-connector-java-8.0.11.jar** is needed to connect to the MySQL database as well as if changes are to be made to the database through application actions.

**Steps to run**

This is a step-by-step readme file that goes over how to setup whatever is needed and to run the application. It goes over how to setup the MySQL database needed to run and what variables within the code that requires change.

**test.jar**

This runnable jar file fetches data points from the database and creates a line chart based off those points. It also creates KPFs within the chart by grabbing the KPF values in the database.

**toRun**

The main application. It contains both the data input (zoomRevisionLog.jar) and line chart (test.jar) pages.

**zoomRevisionLog.jar**

This runnable jar file allows you to input data for the line chart. Once you have filled out every text field and submit it, the backend makes calculations to create x and y values and KPF values. These values get sent through to another table within the database.

**Challenges:**

**Zoom**

Zoom is a massive class that can consist of many ways to zoom into a page (Example, specific buttons, scroll, scroll into an object,...etc).

We did implement a zoom function that was able to zoom into where the mouse was pointing ( it could also scale the objects that it’s zooming as well).

However, adding on the other functionalities that were mentioned above would take time.

**Scroll**

Scroll was a challenge encountered while creating the help pop-up. At first it seemed like a complex task. However, using the scrollpane layout, we were able to bind the scrollpane to the group layout and add it to the scene. Then using unique objects to scrollpane we were able to set it to our preferences (vertical/ horizontal visible).

**Print**

Print is a challenge we have yet to find a solution for. Our problem is that the we cannot scale our scene compared to the printer scene. So everytime we print, ¼ of the right side of the page is not being printed off. We made a post on StackOverFlow that can be found on the excel document on public where we explain the problem in more detail.

**SaveAsPDF**

SaveAsPDF can be seen in the projects folder. We take a snapshot of the scene, create a pdf document and put the snapshot in the pdf document then save it in a pre-defined folder on the desktop.

**Email**

Email was a challenge at the beginning but ended up being a fun solution. We created a dsp email to use to send the emails. Username = “dsppracticeemail@gmail.com” Password = “Kepstrum2005”. When a revision was released (on button action) the function would then iterate through the database and find anyone who is a lead on the project and would send them an appropriate email regarding the release of the line.

**Database Connection**

To be able to use the database we had to put it on the local server. One particular problemwe had here was putting in the line “autoReconnect=true&SSL=false”. Most of the time we found SSL to be true and the connection would not work. However, since we found this solution we didn’t deal with this problem again.

**Insert Into Database**

Properly inserting into the database at first was difficult. We had to do it correctly to avoid sql injection which would allow people to hack into our database. However, using preparedStatements and online forums we were able to find a solution to this.

**Creating .jar file**

We did a lot of investigation into GlassFish to create a .jar file without including our dependencies. However, after some investigation we found that glassfish may only be used for web applications. We then exported our java class as a .jar file and it worked just fine.

**Adding a field**

Adding a field seemed like an easy task at first, just adding several textfields on a button action. However, adding textfields with no name seemed to cause a problem when putting that into the textfieldwhich is impossible using javafx. We created a counter variable in the database to count the amount of fields already in the database to then insert (numberOfPreviouslyDefinedFields + 1) into the database

**Updating fields**

Updating fields was solved when adding a field was solved because then we could use that variable in adding and updating fields everytime something is clicked in the revisionLog Class.

**Creating line charts with the given data**

Creating a chart was luckily made easy due to there being a built-in library for charting. Where difficulties started to arise was when we made the chart into a multi-axis chart without changing the scale of the rest of the line charts.

**KPF Lines (finding screen pixel)**

Due to having a graph on-top of a scene, we struggled to find the pixel position.