Version control gives people the ability to record changes made to a digital product over a period which allows the person accessing the digital product to go back and reference a past version of the digital product from the future. Version control tools, like Git, allow for certain actions to be taken to manage the saved states of digital assets. This tool monitors your files of any changes made and the different stages the file goes through to be accepted into the next save state or abandoned.

Without a version control tool, it would be extremely difficult to manage multiple version of a file with the same name. You would not have the ability to see what changes have occurred on the file without having the two documents side-by-side. And you would not have the ability to go back and see what changes had occurred in the past and why from the future.

Version control tools help organize software projects because it allows multiple people to collaborate on a single project where each person is working on different features. When a person is complete with a feature, they can move it to where the main project lives, and it can be a part of production without relying on sending their finished feature to everyone working on the project.

Version control helped in managing my source code in developing the Calculator application by using a remote repository to host my final saved state where I can push all my changes. Further, it allows me to make local changes to my files without any concern for the remote source code. The changes I make locally do not show up on the remote repository until I take my changes through the workflow of stages which helps me manage what changes I want to push to the remote server.

Further benefits version control provided was in developing features. When a new feature for the Calculator was put on the backlog, the new feature was created in a branch. This branch was used to separate the new code and the original code in the master branch. Doing this would prevent any unwanted changes to be made to the master branch. If the feature was developed but not tested well then, the new branch would not be pushed to the master branch until the branch has been approved.

When creating commits using git, you can view the history of commits and which branches they occurred. This helps in monitoring the progress of features being build for the application which is helpful because some features are dependent on other features being built. History lets you see the author, committer, committed date, message, and commit id.

Pushing to remote shared repositories from local working files helps teams collaborate from a distance and make a program go from a local program to a global program. The functionality of git repositories allows for this type of collaboration to happen. Git allows collaborators pull information into their local repositories that were pushed to the remote repository from other programmers local working files.

Pushing local repositories allows for a divide and conquer, agile development methodologies to take place. Separating the local and remote repos removes any influence that happens on the local to influence the remote. Functionally, there are other gates that promote this local and remote repo relationship. When pushing a local branch to the remote or merging with other branches, conflicts in the code, where the code in the local overlaps the repo and branch being pushed or merged to, can be used to shows where the code overlaps and differences are and thus the code can be fixed to fix. In the reverse direction, previous commits can be combined, split, or removed. All this takes place in the local repository and other programmers will not have to pull in code at a high frequency because of one person’s commits. Configuration adds to the functionality of repositories by allowing the user to insert a name and an email where they can be identified if there are any questions. Further, configuration allows for coloring the UI in the CLI when a merge conflicts appears to be able to see the differences easier.

My previous experience and view of branching has been very little. I did know why branches were needed. After learning and implementing branches, they are used to silo the development of new features and ideas in the local environment without impacting the code in the remote repository until the new features have been tested committed and pushed to production. When developing the calculator app, each new mathematical function was a new feature that added to the usability of the calculator. Each new function had its own branch where the changes made to one feature did not impact the other until you had to merge branches.

When the developed feature was complete, for example, when the developer is merging two branches together there could be conflicts in the branches. A conflict occurs when there is different information in a line of code from either the modification, addition, or subtraction of code. The merge view appears, and you can see visually the difference between the intended merge destination and the branch code. The developer must resolve the difference which allows for a clean merging of branches.

An additional feature for checking the history of changes in the repository is to use ‘git blame’. This feature allows you to get a more comprehensive view with more details about the detail of the commit and who the author is so you can potentially contact them if you have any questions. Also, it shows the time of the commit if you need that information as well.

These actions come at different stages of the feature’s lifecycle. The repository is first branched to create a feature that does not influence the remote repos code. Then the new code can be stage for later committing or committed immediately to the local repository. If there are conflicts in the file, them you will encounter an alert that prompts you to resolve the issues. After your conflicts have been resolved you can then begin to commit the code again. Creating history for that branch that was committed to.

Performing code reviews provided me with the extra motivation to develop well written code to help the person who is reviewing the code to be able to comment and review the code much quicker. Knowing that my code will be criticized reminded me to make sure that each line of code is precise and had a purpose to the overall application. An example of this was to make sure that all my code appeared clean when developing the code. This consisted of making sure the proper tabs were represented for nested code and comments were aligned in the proper places that were consistent. Another due diligence was to develop the unit test to make sure there were no error in the code and the proper results were returned to the user.

Code review feedback should ideally criticize your code and not the person developing the code. Although it could be hard to separate the two since a human is developing the code that is being reviewed. Feedback should only discuss the algorithms chosen to solve the problem, the cleanliness of the code and its alignment to make it east to read, and more code specific related issues. To prevent negative feedback, I intentionally tried to align all my right had side comments and nested code tabs to ensure maximum readability. Further, I confirmed the results of the student name unit test to make sure that the information was coded and is returned correctly.

A way to resolve an issue brought up in a code review is to collect the reviews and conduct a severity analysis to determine which issues are most important to solve that will make a large impact on the overall project. Severities can be ranked from ‘Low’, ‘Medium’, to ‘High’, where ‘High’ issues are tackled first because they have the most severe impact on other parts of the code base. Areas of high severity could be redundant code or a security flaw.

When in the development process, git provides the ability to see certain aspects of the codebase. Important information can be displayed like the commit message from previous commits, the person who submitted the commit, and details of the code that has been submitted versus other branches. History view is useful to know what the latest update was to the branches feature and the progress that was made along the way. This is extremely useful in a profession setting where the project manager can see the progress of a feature and determine the changes that need to happen if the desired progress is not being made.

In preparation of producing code in the codebase, there has to be communication between those whom are coding in it. If you are working in a company and you want to begin working on a piece of code you must communicate this to your manager and your peers so your manager can organize who is working on which feature and not waste valuable time and resources resulting in two people solving the same problem. The git workflow has due diligence build into the workflow. Before you begin working on your feature, you pull in the latest code from the codebase to ensure you have all the latest commits that could potentially influence your development. Then, after making sure I have the latest code, it is important to remember to create a new branch for each new feature being developed to keep the commits relevant to the overall project and to help other people understand where you are in development.

While developing code, I made sure to incrementally test to code to ensure I get the expected result and the program does not crash which helps to double check the work.

# CS 310 Calculator Changelog

## Version 5.0.0 2019-07-17

Note: Tutorial Four – Work on Branches in Parallel

Major Changes:

* Created new branch Javadoc to hold stashed commits

A screenshot of a cell phone

Description automatically generated

Minor Changes:

* Added @since version in Binary Operations

A screenshot of a social media post

Description automatically generated

Bug Fixes:

* Not Applicable

## Version 4.0.0 2019-07-17

Note: Tutorial Four – Work on Branches in Parallel

Major Changes:

* Created branch for multiply operation
* Created Class Multiply with multiply logic

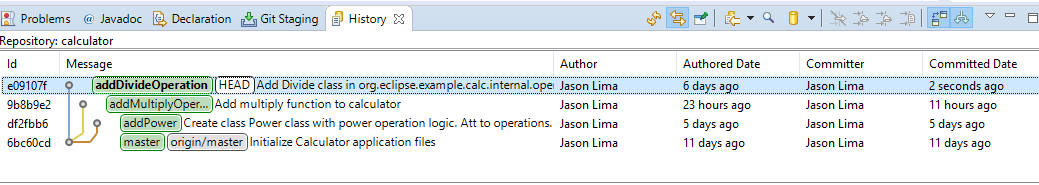
A screenshot of a social media post

Description automatically generated

* Imported Multiple class
* Added Multiply() class to standard operations

Minor Changes:

* Combined two commits in the addDivideOperation branch



Bug Fixes:

* Not Applicable

## Version 3.0.0 2019-07-12

**Note**: Remember that a changelog works in chronological order from newest to oldest. See section Version 1.0.0 for example data from the first tutorial. Add your own data to section Version 2.0.0 with details from your work in Tutorial Two and section Version 3.0.0. with your work from Tutorial Three.

Note: Tutorial Three – Inspecting the History

Major Changes:

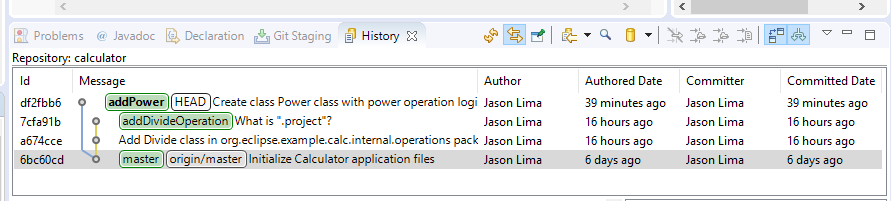
* Not applicable

Minor Changes:

* Not applicable

Bug Fixes:

* Not applicable



## Version 2.0.0 2019-07-12

Note: Tutorial Two – Develop Another Feature and Use Staging View

Major Changes:

* Created a new branch called “addPower”
* Created logic for power operator

A screenshot of a social media post

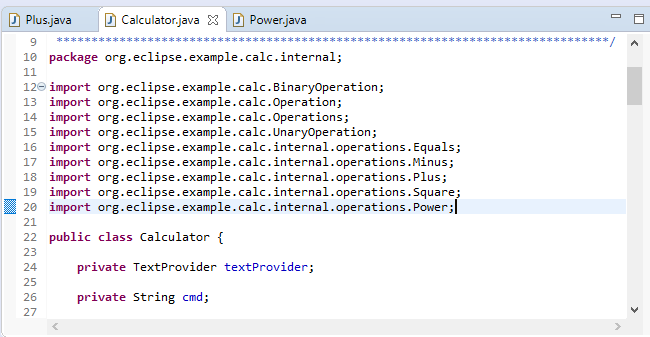
Description automatically generated

* Added Power class to setupDefaultOperations

A screenshot of a social media post

Description automatically generated

* Import Power Class into the Calculator package



Minor Changes:

* Committed changes to local repository

Bug Fixes:

* Not Applicable

## Version 1.0.0 2019-07-11

Note: Tutorial One – Develop a Feature Bug Fix

Major Changes:

* Created new feature branch “addDivideFeature”
* Add Divide class in org.eclipse.example.calc.internal.operations package with division logic.

A screenshot of a social media post

Description automatically generated

* Add Divide class method to default operations in calculator class.

A screenshot of a social media post

Description automatically generated

* Import Divide Class to use class functions into calculator class

A screenshot of a cell phone

Description automatically generated

* <Enter each major change from Tutorial Two here as its own bullet point.>
  + <Include at least one screenshot for each major change.>

Minor Changes:

* Committed changes to local repository

Bug Fixes:

* Not applicable

# CS 310 Jukebox Changelog

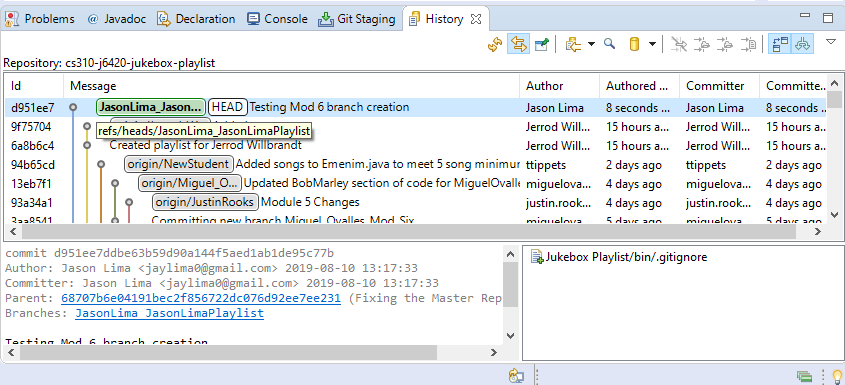
## Version 3.0.0 2019-08-11

**Note**: Remember that a changelog works in chronological order from newest to oldest. See section Version 1.0.0 for example data from the first tutorial. Add your own data to section Version 2.0.0 with details from your work in Tutorial Two and section Version 3.0.0. with your work from Tutorial Three.

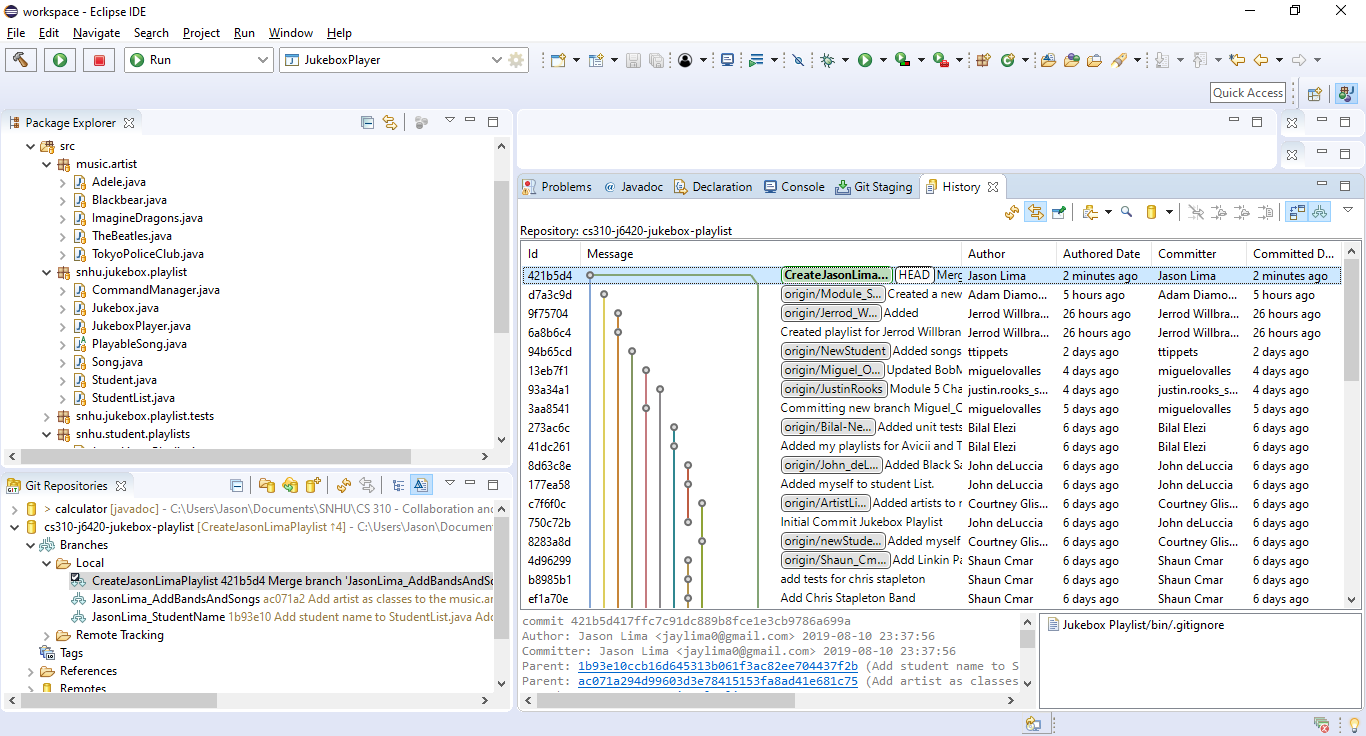
Note: Tutorial Three – Creating student playlist

Major Changes:

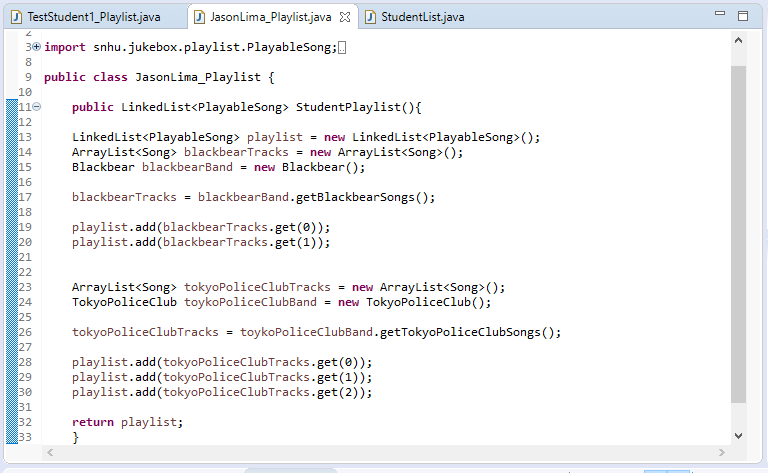
* Created a new branch for the new feature called JasonLimaPlaylist



* Merged previously created branches, adding he student name and the music

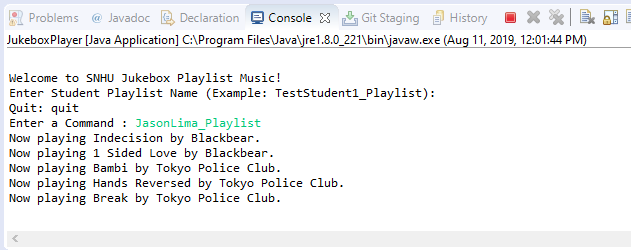


* Create class JasonLima\_Playlist to hold the students’ playlist information
* Added Band and their music to the JasonLima Playlist



* Created a JasonLima\_Playlist option in the application menu



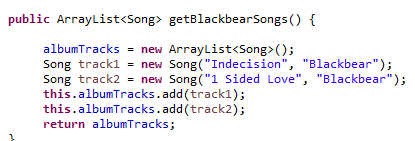


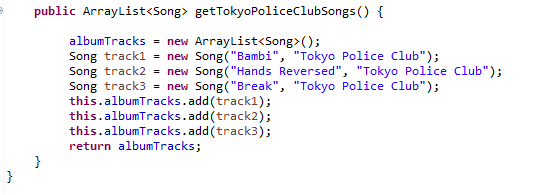
Minor Changes:

* Committed changes to artist classes
* Committed changes to add Jason lima’s playlist to the switch statement option menu
* Committed the creation of the Jason Lima playlist to pull in artist and their songs

Bug Fixes:

* Fixed get method for each new artist that was added



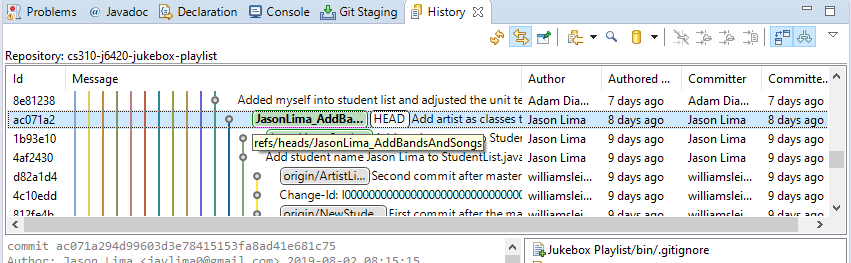


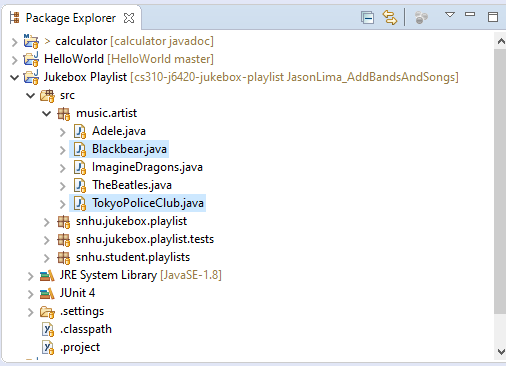
## Version 2.0.0 2019-08-02

Note: Tutorial Two – Develop and Add Music Artist

Major Changes:

* Create a new branch JasonLima\_BandAndSongs to add artist and their music to a student’s playlist



* Created a new Class for artist Blackbear and added songs
* Created a new Class for artist Tokyo Police Club and added songs

Minor Changes:

* Committed Blackbear artist class
* Committed Tokyo Police Club artist class

Bug Fixes:

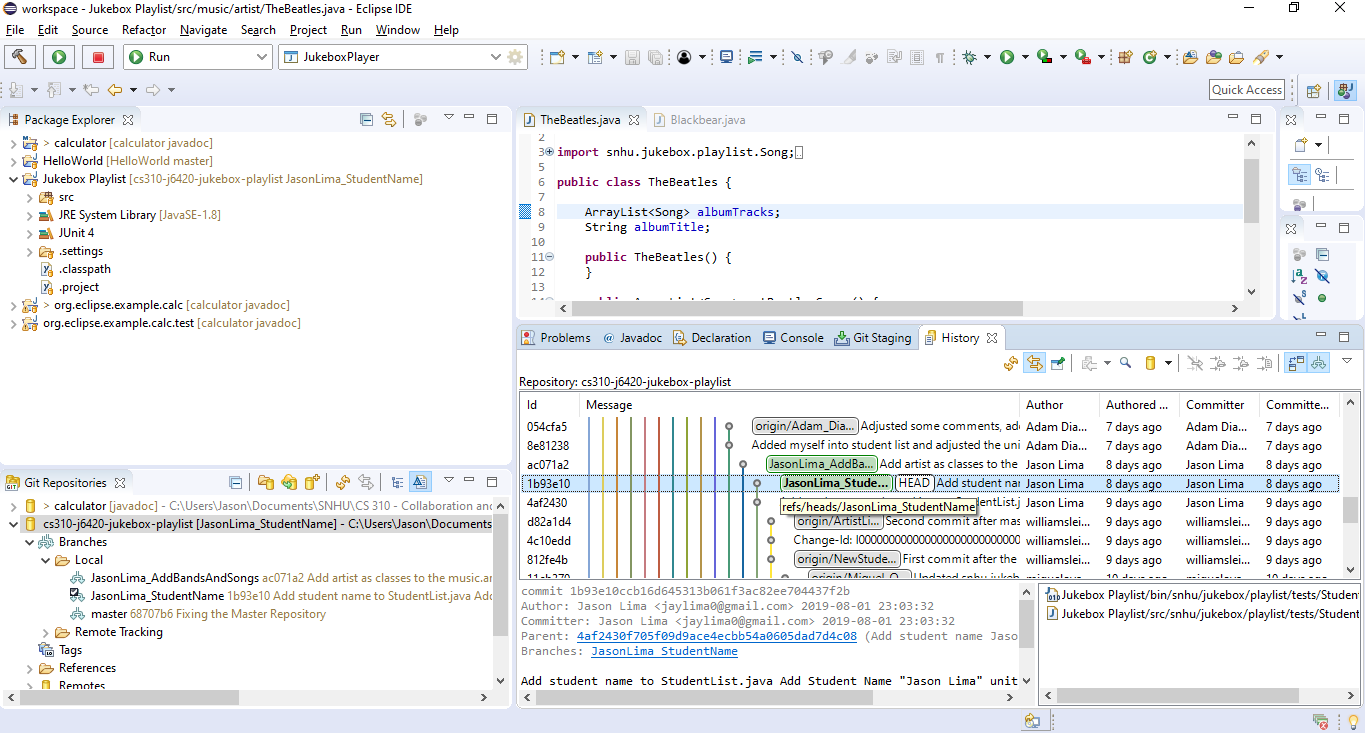
* N/A

## Version 1.0.0 2019-08-02

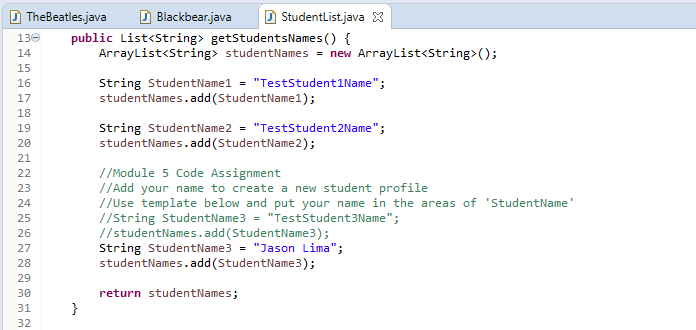
Note: Tutorial One – Add Student Name

Major Changes:

* Created a new branch JasonLima\_StudentName to add a new feature to the code



* Add Student Name to the student list to be able to search the students playlist



Minor Changes:

* Added student name to unit test
* Committed the new student name after successfully completing a unit test

Bug Fixes:

* N/A