Documentation and Notes

For

General Telescope Control GUI

John Tran and Emily Cook

Last Updated: 8/10/16

Mentor: Waid Schlaegel

**Table of Contents**

1. Introduction
2. Usage
3. Organization
4. Known Bugs and Issues
5. Software References
6. Things that aren’t finished and Waid’s/Rast’s Desires
7. What to do when the code is completely finished
8. Contact

**Introduction**

The main reason this software was developed was to have a general telescope control GUI that can easily be used and portable across multiple operating systems such as Windows and Linux. It is designed to be able to control the iOptron Minitower Pro mount. The program was written in Java using the jssc library, a third party library. It includes features such as time, positional control, and display of current position. The program was initially developed in Linux, Ubuntu 16.04.1 LTS so it is advised that you do the same. Then test and debug on newer and different operating systems such as Windows or Ubuntu 18. Currently it is incomplete, however it is hoped that other features will be added by future scholars as time permits. The development of this software was used primarily using the command line and a text editor. Not a single ide.

**Usage**

To use this program in its current state, you must:

1. Download and install the Java Development Kit (jdk). You can download it from the Oracle website. On Windows, you’ll have to set the path in the environment variables to the jdk. Google this. On Linux, it’s just a few simple install commands you can also Google. This will allow you to use javac and java commands for development. Make a simple hello world program to see if “javac HelloWorld.java” and “java HelloWorld” works before continuing. Of course if you already have the jdk, then you can skip this step.
2. Download and install the jssc library (allows for serial COM), Google it if the link is broken <https://code.google.com/archive/p/java-simple-serial-connector/>

On Linux, you have to add the jssc.jar to the ext file (or something similar) In my case:

*/usr/lib/jvm/java-1.8.0-openjdk-amd64/jre/lib/ext*

On Windows, you have to add the jssc.jar to the ext file (or something similar) In my case:

*C:\Program Files\Java\jdk1.8.0\_60\jre\lib\ext*

1. Plug in the telescope mount
2. Go to the TelescopeGUI folder
3. If using Windows, you’ll have to run this command every time since jssc.jar is a 3rd party library.

*set classpath=*path to jssc.jar*;*

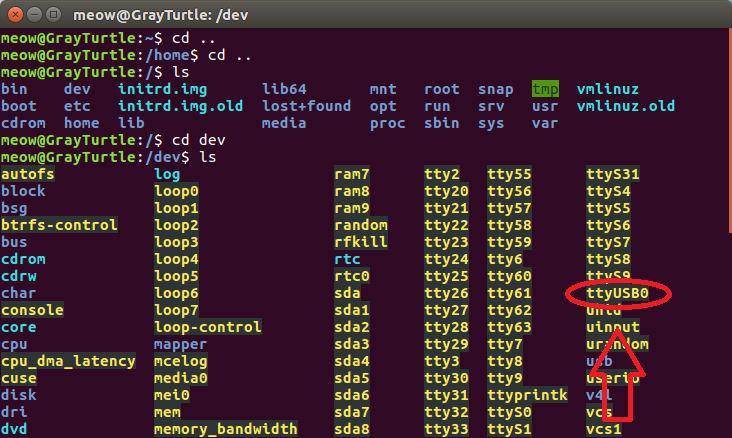
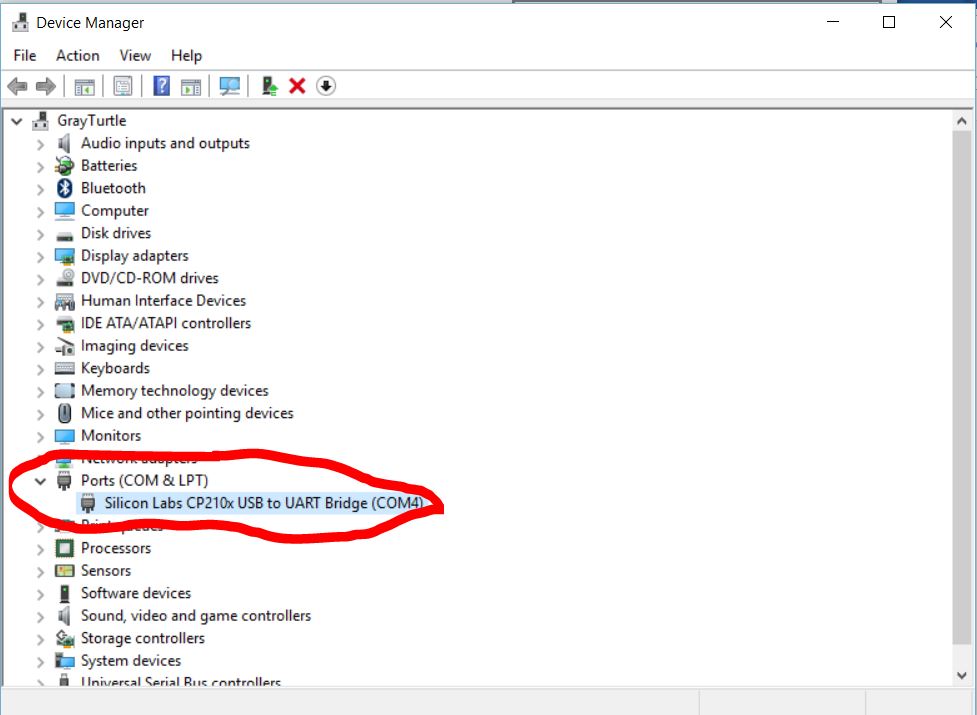
example:

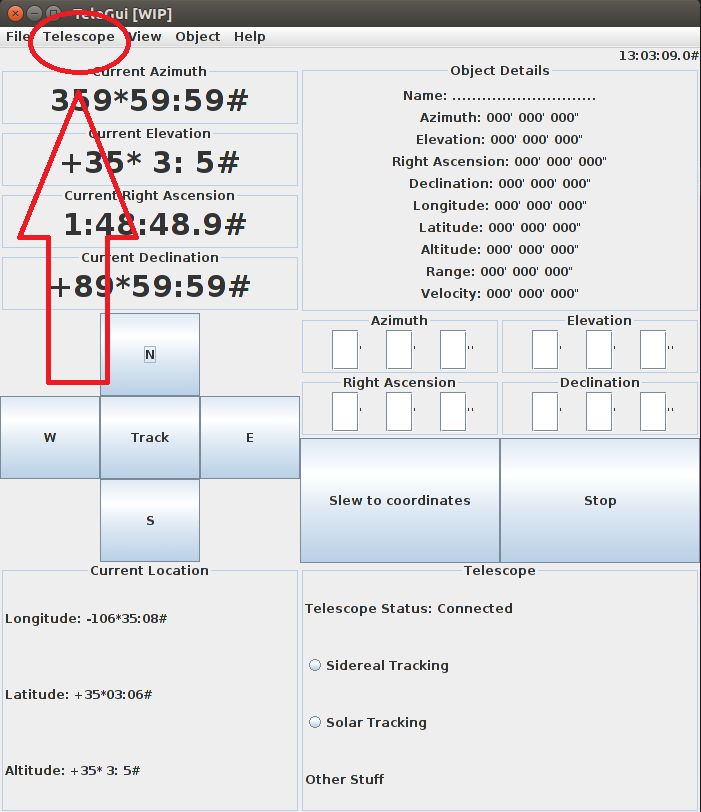
*set classpath=C:\Program Files\Java\jdk1.8.0\_60\jre\lib\ext\jssc.jar;*

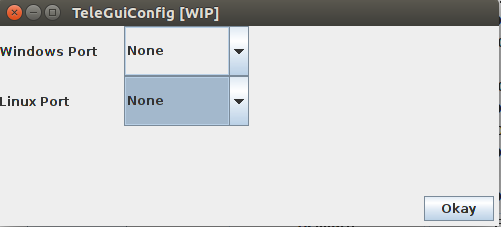
1. Run *javac Tele.java* then *java Tele*. This should run the GUI.
2. Check to see what port the mount shows up on.

If on Windows: Go to the device manager and go to Ports or whatever pops up that is similar. You may have to update your device driver for this to communicate properly.

If on Linux: Go to your /dev/ in your root folder and look for the ttyUSB#



1. Go to “Telescope” in the menu of the frame and choose the appropriate port.



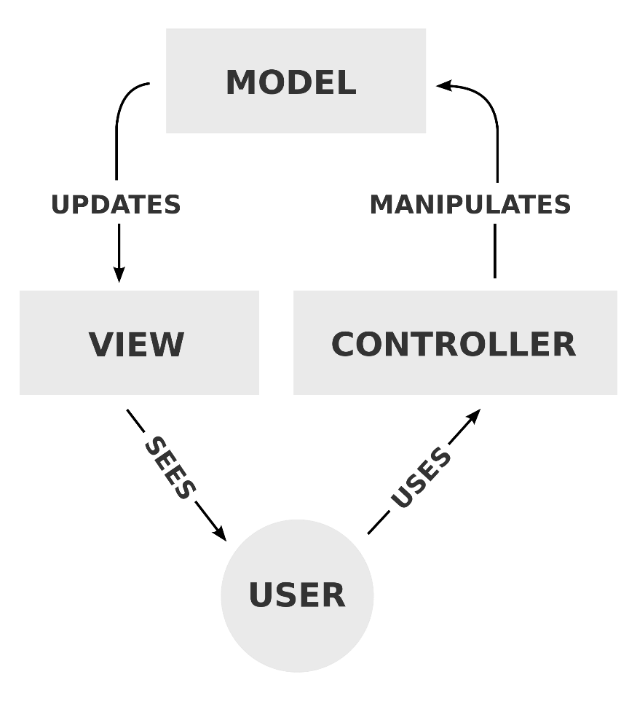
1. The GUI should be working now.
2. In order to send more commands in the code such as change the tracking rate to sidereal, refer to the page 42 in the PDF of the mount manual in this folder. TelePort.java is where the commands are being sent to the mount.

**Organization**

The program is organized in model view controller architecture.

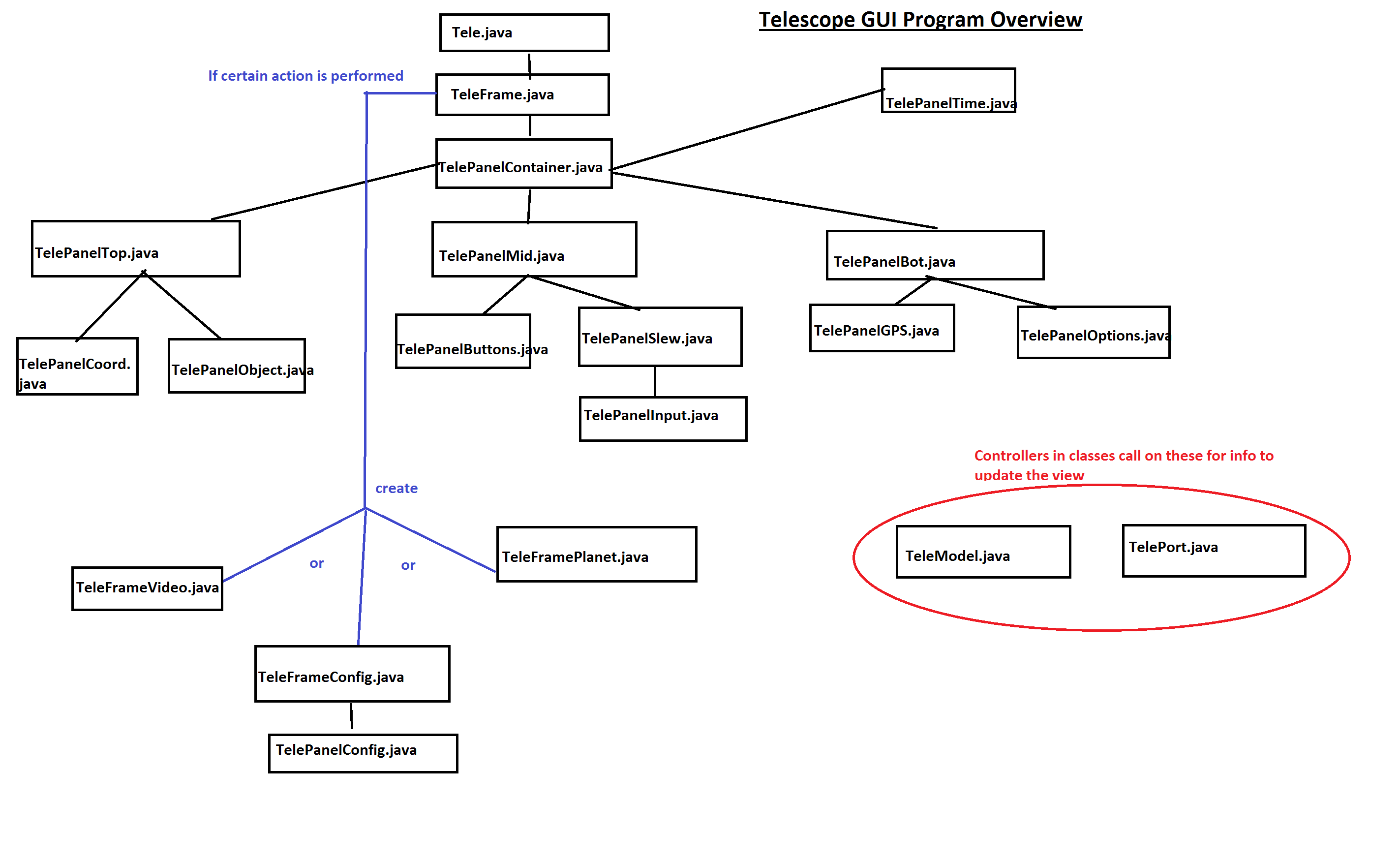
“Model–view–controller (MVC) is a software architectural pattern for implementing user interfaces on computers. It divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user.

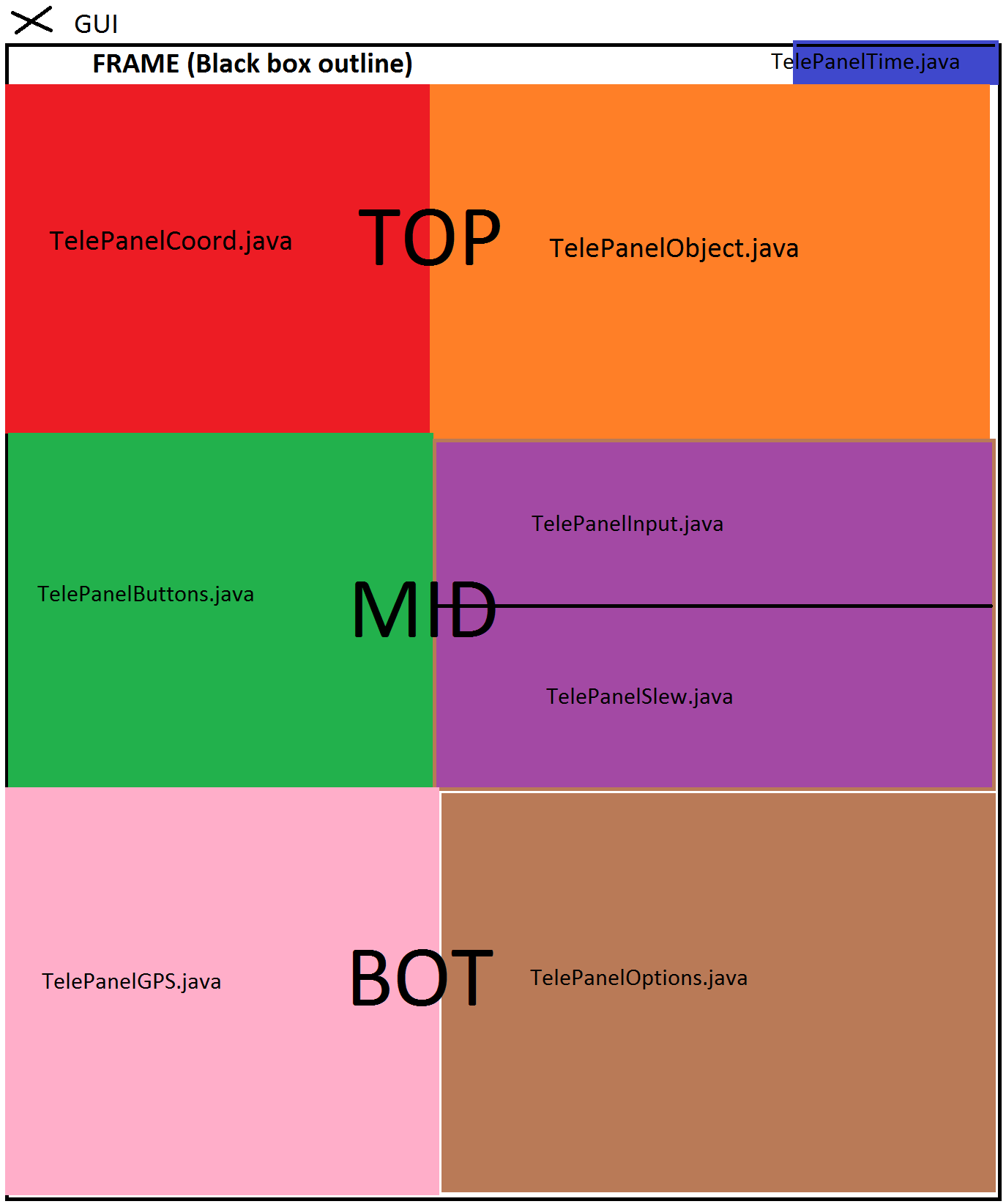
* A *model* stores data that is retrieved according to commands from the controller and displayed in the view.
* A *view* generates new output to the user based on changes in the model.
* A *controller* can send commands to the model to update the model's state (e.g. editing a document). It can also send commands to its associated view to change the view's presentation of the model (e.g. by scrolling through a document)” (Wikipedia).

TeleModel.java and TelePort.java are purely for calculations and interpretation aka the model. The rest of the classes are views and controllers.

The program is also organized in such a way where there are panels within panels within panels. The Layouts are mainly GridBagLayout and GridLayout. Here’s how the GUI creates itself:

1. The main method in Tele.java creates TeleFrame.java which creates and adds TelePanelContainer.java (this will contain all of the components).
2. TelePanelContainer.java creates and adds TelePanelTime.java, TelePanelTop.java, TelePanelMid.java, and TelePanelBot.java (the clock and TOP, MID, BOT sections of the GUI).
3. TelePanelTop.java creates and adds TelePanelCoord.java and TelePanelObject.java
4. TelePanelMid.java creates and adds TelePanelButtons.java and TelePanelSlew.java
5. TelePanelSlew.java also adds in TelePanelInput.java
6. TelePanelBot.java creates and adds TelePanelGPS.java and TelePanelOptions.java
7. Once the “Configure Telescope” option under “Telescope” in the frame menu is clicked, it creates TeleFrameConfig.java and that code adds TelePanelConfig.java.
8. When the “Video” option under “View” in the frame menu is clicked, it creates TeleFrameVideo.java.
9. When the “Planetarium” option under “View” in the frame menu is clicked, it creates TeleFramePlanet.java





**Known Bugs and Issues**

* When the program opens on a smaller screen resolution, it is too large. A way to fix it is to have the GUI resize relative to the screen size.
* Choosing a second port after the first has been opened brings up a “port busy” exception from the jssc library.
* Response time is slow, it might be Java or the way it is programmed
* When the mount slews to a certain defined point, it starts spinning in circles. This is most likely an issue with the mount itself.

**Software References**

* Satellite Tracker
* TeleTrack

**Things that aren’t finished and Waid’s/Rast’s Desires**

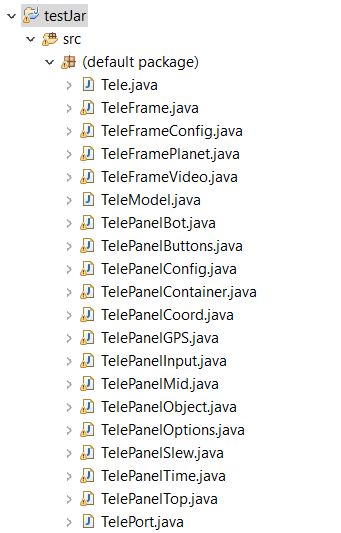
* In the menu of the frame: File, View, Object, and Help
* User Input of Azimuth, Elevation, Right Ascension, and Declination
* Telescope tracking: Sidereal and Solar tracking
* Night Mode. In the frame menu: View > Night Mode
* General formatting of presented data on GUI
* Waid wants video to work with a webcam. In the frame menu: View > Video
* Waid wants separate Planetarium software to be called on. View > Planetarium
* Rast wants input of several goto points to track a satellite (this is a theory on how to put a tracking option on the mount)

**What to do when the code is completely finished**

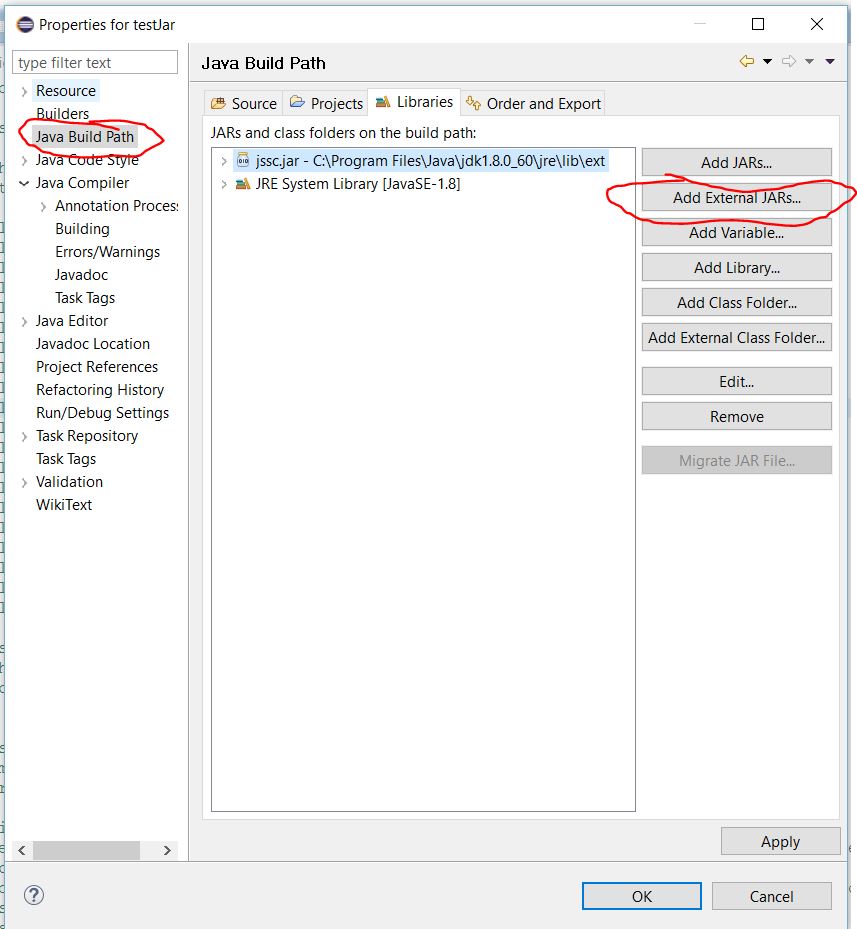
1. Update all documentation, including the comments in the code.
2. Make a general user manual for normal people (plebs)
3. Export the GUI to a .jar file (or try to figure out other executable files, but .jar should suffice)

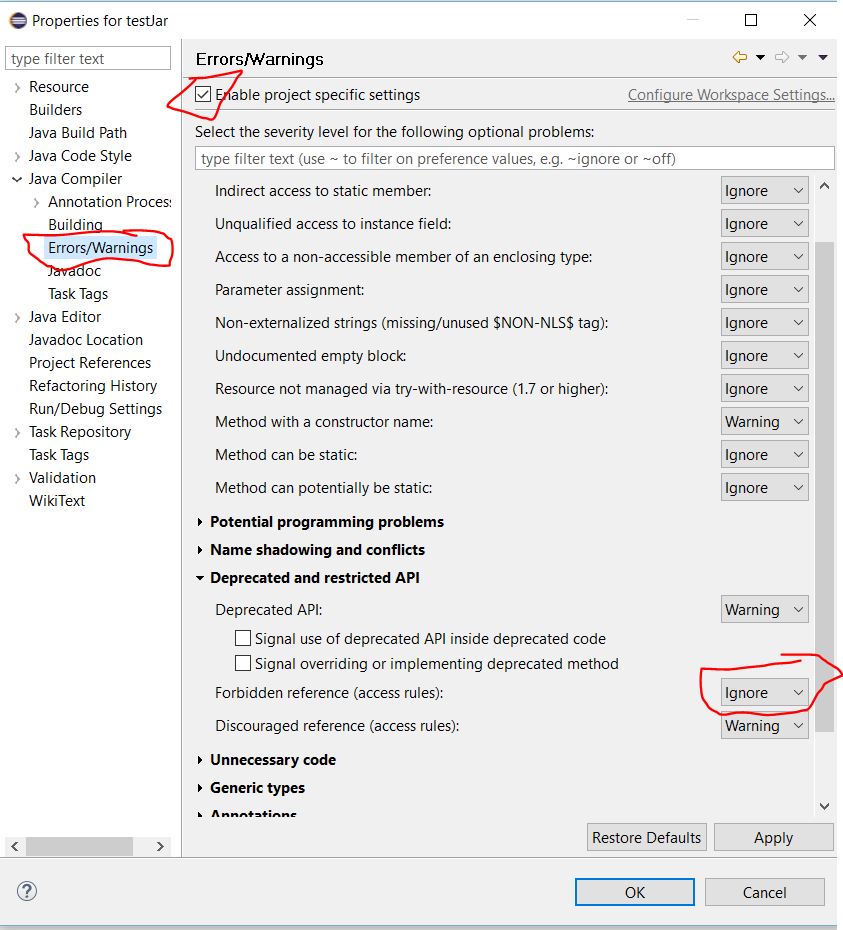
The only way that I know how to do this is using an IDE (surprise). We’ll be using Eclipse.

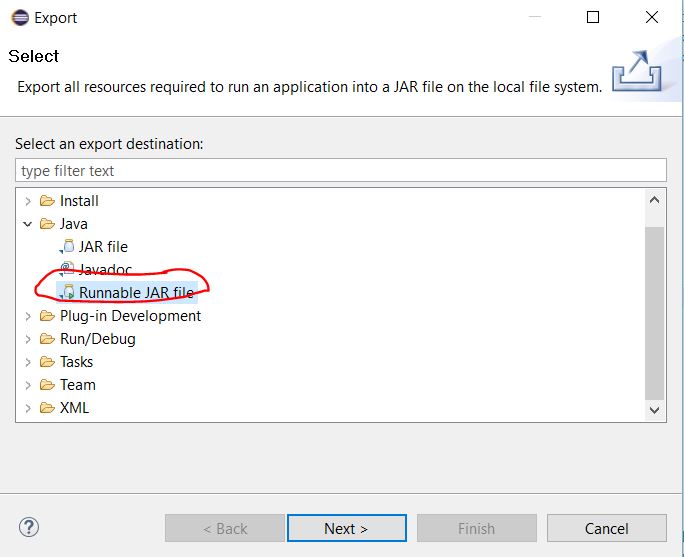
1. Go to File > New > Java Project. Then copy and paste the GUI files into the src folder in the project (I named it testJar), and it should make a default package. Here what it looks like for me:

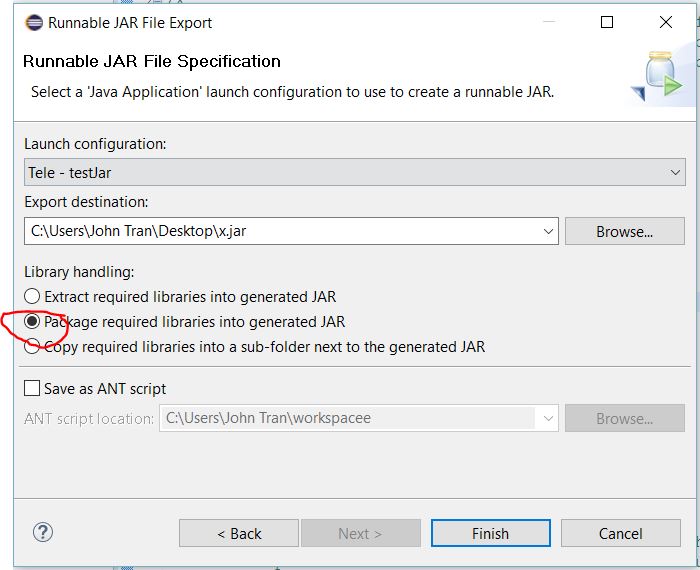


1. Right-click on your project and go to Properties > Java Build Path. Then click on the Libraries tab. Go to Add External JARs… and add in your jssc.jar file. This should add in the jssc library to the final .jar and the user will never need to go through the pains of installing jssc.



1. ****You might have noticed some errors that Eclipse picked up about the jssc library. We don’t care about this. The reason Eclipse is acting this way because it doesn’t like third party API. Obviously, in this case we need it. To disable this error right-click on your project and go to Properties > Java Compiler > Errors/Warnings and click Enable project specific settings. Then go to Forbidden reference (access rules) and choose ignore.
2. Finally, it is time to export the GUI to a single .jar file. Right-click on your project and go to Export…. Then go to Java > Runnable JAR file. Choose the main class (Tele) in your Launch Configuration, choose export destination and name the final GUI. Then choose Package required libraries into generated JAR. That should be it!





1. **CELEBRATE.** The program should be able to run on both Windows and Linux at this point. Contact John Tran about your success.

**Contact**

John Tran – [tran.john1337@gmail.com](mailto:tran.john1337@gmail.com) or [johtr@okstate.edu](mailto:johtr@okstate.edu) , 918-851-8587