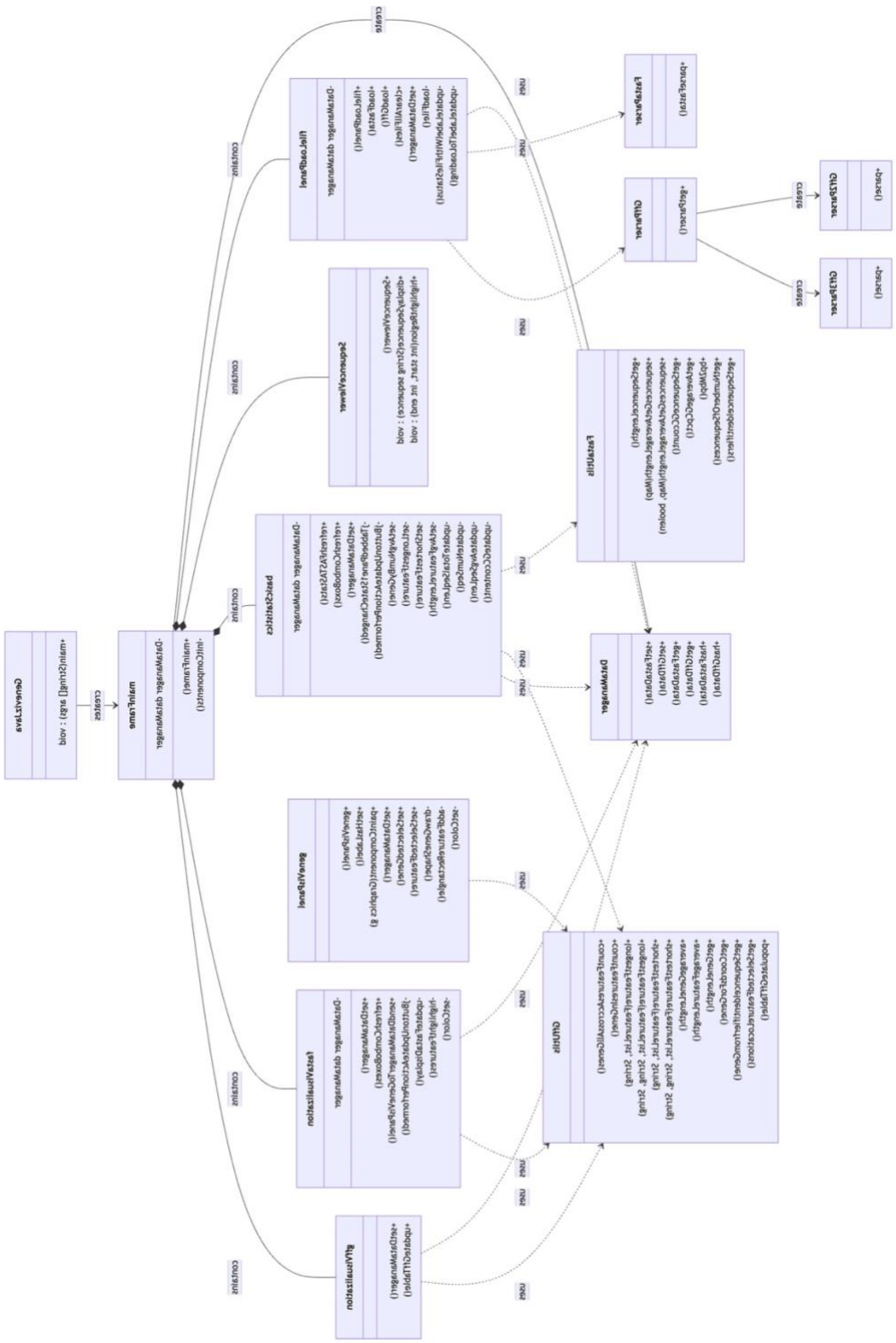
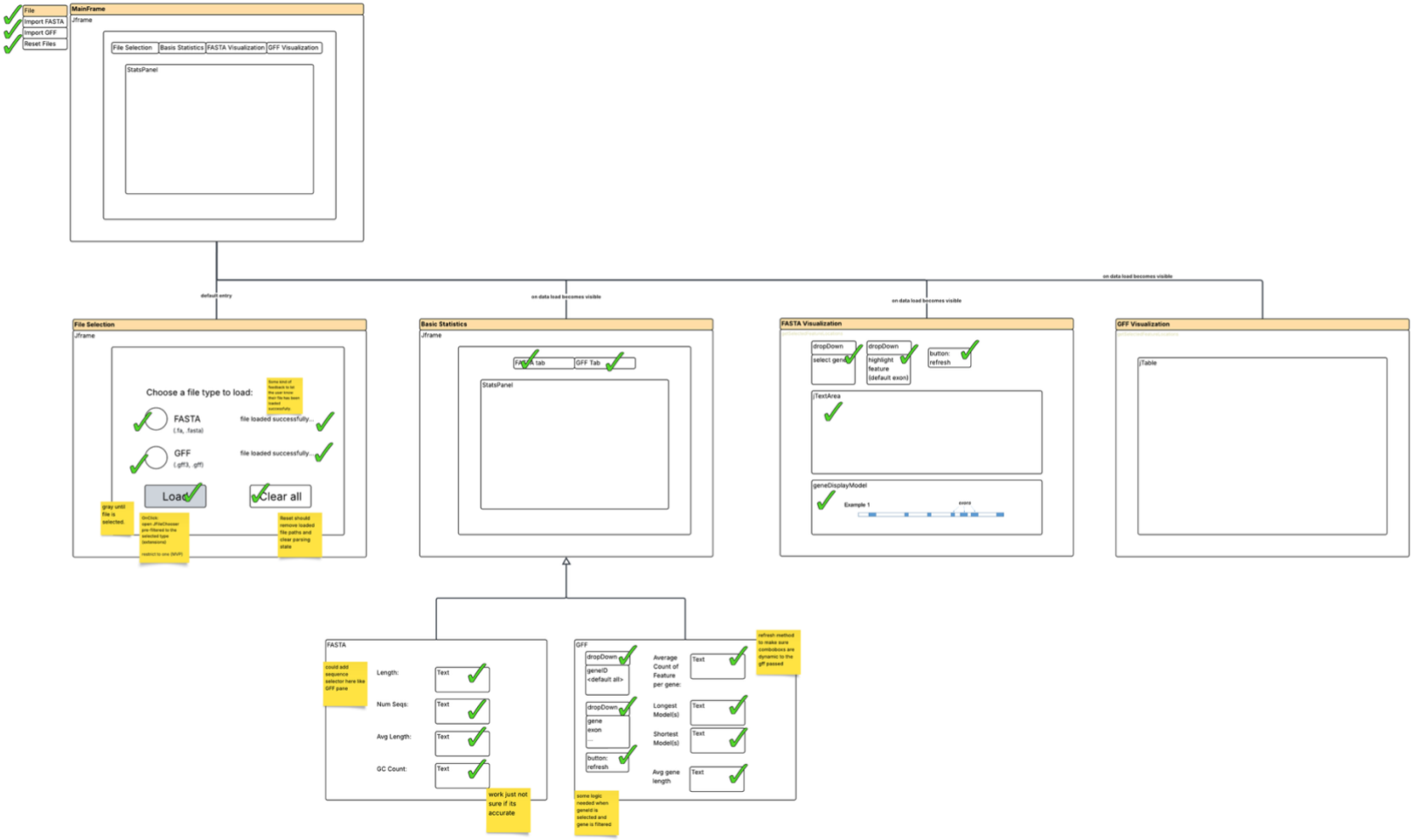
geneviz-java: Design Technical Documentation

1. UML




File
Import FASTA
Import GFF
Reset Files
MainFrame
Jframe
File Selection
Basis Statistics
FASTA Visualization
GFF Visualization
StatsPanel
default entry
on data load becomes visible
on data load becomes visible
on data load becomes visible
File Selection
Jframe
Choose a file type to load:
FASTA
(.fa, .fasta)
file loaded successfully...
GFF
(.gff3, .gff)
file loaded successfully...
Load
Clear all
gray until file is selected.
Reset should remove loaded file paths and clear parsing stats
Basic Statistics
Jframe
FASTA tab
GFF Tab
StatsPanel
FASTA Visualization
dropDown
select gene
dropDown
highlight feature (default exon)
button:
refresh
jTextArea
geneDisplayModel
Example 1
exon1
GFF Visualization
jTable
FASTA
could add sequence selector here like GFF pane
Length:
Text
Num Seqs:
Text
Avg Length:
Text
GC Count:
Text
work just not sure if its accurate
GFF
dropDown
geneID
<default all>
dropDown
gene
exon
...
button:
refresh
some logic needed when geneid is selected and gene is filtered
Average Count of Feature per gene:
Text
Longest Model(s)
Text
Shortest Model(s)
Text
Avg gene length
Text
refresh method to make sure comboboxs are dynamic to the gff passed

2. GUI

2.1 mainFrame

2.1.1 The main launcher for the application. It is called by the GeneVizJava main class, which is a conventional entry point into the application.

2.1.2 On initialization, the mainframe creates the four main panels — fileLoadPanel, SequenceViewer, fastaVisualization, and gffVisualization. The mainframe also sets the dataManager (pseudo-database class) to the relevant panels.

2.1.3 The mainframe also controls the visibility of the panels; only fileLoadPanel is always visible. The rest of the panels are visible only if the dataManager hasFastaData() and hasGffData(). If at any point these conditions are not met, the app will force the user back to the fileLoadPanel. This is controlled by a continuous timer loop every 1 second.

- *Design notes: I wanted a way to have my data-dependent tabs visible only once data is loaded. It became a bit complicated understanding what has access to what. The ComboBoxes in the statistics panel need the geneID from the GFF, so this panel needs to be loaded with dummy data first — and hidden. It can be unhidden once the data is loaded. But how do we check this event? There is a change listener to catch when the tab is opened, which could refresh the data, but this can only happen if the tab is visible. And what if the user clears the data using File > Clear All? Short answer: a bit of a wasteful solution — I have a rolling timer that checks every second if data is loaded and manages the tabs accordingly. It's a cheap comparison but unlikely to be best practice.*

2.1.4 The mainframe has a jTabbedPaneStateChanged listener. It's a tabbed JPanel object that holds the rest of the application in separate tabs. This state change listener allows the application to detect when the user changes tabs and deploy logic to update that tab upon refresh.

- *Design notes: Tabs were chosen to create a concise, clean-looking application. The app's UX design leaves a little to be desired, but it's simple and clean. It could use some color and more dynamic window resizing, but these were considered add-ons and did not make the MVP (minimum viable product).*

2.2 fileLoadPanel

2.2.1 Visually, this is a simple and clear tab. The user understands that this is where they load their data. They have two options: using the radio buttons or the file menu bar. Both use the same underlying method. User feedback is shown to indicate loading and file completion status. Feedback is also provided if an issue occurs at this stage. The user is only able to select compatible files at each point, so they cannot input any unsupported file types.

- *Design notes: GFF3 and GFF2 support was attempted, and there is a parser for both file types. However, GFF2 support was dropped since, even after loading into a BioJava FeaturesList, both objects required different methods to extract items. The workload for GFF2 support was then considered an add-on and did not make the MVP. (Lafita et al., 2019)*

2.2.2 The private loadFile method is the key method in this panel and does exactly what its name suggests. It checks which radio button is selected (from a button group in NetBeans) and loads a FileDialog that filters based on the selected file format. This method — which could be split into separate methods as it is too large — then attempts to parse the incoming file using the correct parsing object from the IO

class objects. For a FASTA file, a Map of DNASequence objects from BioJava, is created. For a GFF3 file, a FeatureList, also from BioJava, is created — more on these objects later (Lafita et al., 2019).

2.2.3 UI Labels are updated to indicate status and loading.

2.3 basicStatistics

2.3.1 The basicStatistics tab is probably one of the most heavy-lifting features of the application, with several user-configurable options. It relies heavily on two helper classes from utils called FastaUtils and GffUtils.

- *Design Note: On reflection, I could have extended the existing DNASequence class for FASTA manipulation and extended the FeatureList class for GFF manipulation. Instead, I have two large class objectsFastaUtils and GffUtils) that feel a bit stranded in utils. More on these objects later.*

2.3.2 UX design was intended to keep simple panels for each type of file statistics view. The basicStatistics tab is a JTabbedPane inside a JFrame (SequenceViewer — which is largely just a shell) that hosts a secondary JTabbedPane for a FASTA JPanel and a Features JPanel.

2.3.3 In a similar fashion to the mainframe, there is a StateChange listener to detect which tab is active and reload data if no data is already loaded.

2.3.4 The FASTA tab is relatively static and computes FASTA statistics for the entire file using the refreshFASTAStats method, which serves as a wrapper for several smaller, neatly packaged private methods—something the Java gods would surely approve of. These private methods are, by and large, wrappers themselves for the utils classes (which separate front-end code from back-end code) that perform the heavy lifting. Even that's not entirely accurate, as the utils methods are mostly wrappers around BioJava (Lafita et al., 2019) calls on DNASequence and FeatureList class objects… oh, Java?!

2.3.5 The GFF Features tab is more dynamic than the FASTA Statistics tab. It offers users the ability to interact with the gene model and display various statistics on subsets of data. Users can select a gene of interest (retrieved directly from the GFF file using a method called refreshComboBoxes) and the feature they are interested in — currently hardcoded, but this could be made more dynamic in future deployments. Users may also select all genes from the GFF file to view statistics at a higher level.

2.3.6 The application calculates the average count, the longest and shortest models, and the average length of models based on user-selected inputs. After making selections, the user clicks the Update button, which triggers an actionPerformed method. This private method calls several other private methods to compute and display the results.

2.4 fastaVisualization

2.4.1 The FASTA visualization tab contains a JTextArea to display a sequence from the FASTA file, a series of user-configurable inputs to control which gene and feature to display, and an Update button to refresh the page. This panel also contains the geneVisPanel — see section 2.5.

2.4.2 There is a setDataManager method, similar to previous modules, that allows the mainframe to pass the dataManager into this module.

2.4.3 There is a refreshComboBoxes method that populates the JComboBoxes for gene and feature with values — currently hardcoded.

2.4.4 The key method here is triggered when the user clicks the Update button — it calls jButtonUpdateActionPerformed(), which updates the JTextArea with the string of bases using code wrapped up in updateFastaDisplay(). This method calls GffUtils.getCoordsForGene() and makes use of the Location object from BioJava (Lafita et al., 2019). This object contains the start and end coordinates for the gene passed. These coordinates are then used to create a substring from the full FASTA sequence and display it in the JTextArea. After that, the highlightFeatures method is called, which attempts to paint over the text for the user-selected feature. The method calculates the coordinates from the selected gene and then extracts the coordinates for each feature in the gene using the getSelectedFeatureLocations method. This method returns a List of Location objects, which can then be looped over to plot onto the JTextAreaFastaDisplay using the Highlighter from javax.swing.text.

- *Design Note: getCoordsForGene was originally created, and then getSelectedFeatureLocations was subsequently needed. They perform similar functions, and getCoordsForGene is technically redundant. It could be replaced throughout the code with getSelectedFeatureLocations by passing in the gene.*

2.5 geneVisPanel

2.5.1 This panel contains the visualization used to draw a gene and its features.

2.5.2 It receives data from the fastaVisualization panel using the standard setDataManager methods defined in each module.

2.5.3 The main feature in this panel is the paintComponent method, which repaints itself each time the screen loads or is refreshed. This object extends the Graphics class from java.awt.Graphics. It first draws an empty gene shape based on the panel dimensions using drawGeneShape(). Then, if there is data available, it will loop through features in a similar manner to how the highlightFeatures method from fastaVisualization does. For each feature, it calls the addFeatureRectangle method, which recalculates the gene model on the screen and the gene length (in basepairs), and draws a rectangle on the gene image that is relative to its position in the FASTA file.

2.5.4 The direction of the strand is indicated by a text label that appears over the first rectangle rendered on the screen.

- *Design Note: This feature was a pain to create and needs to be refactored. It recalculates the gene drawing using the same approach as drawGeneShape — this could be packaged into a method. It calculates the gene start locations using the gene Location object passed, but in reality, I'm not sure it's useful to have seven different input parameters for a single function; it definitely needs refactoring. In addition, once I got it working, I realized that the model is effectively flipped for the reverse strand. For the MVP, I got it working and added a text label to indicate directionality, but ideally, the visualization should be flipped — points for effort!?*

2.6 gffVisualization

2.6.1 gffVisualization is a comparatively simple table that gets its data in the same way as the other panels, using setDataManager.

2.6.2 updateGffTable is a method wrapper for the GffUtils method populateGffTable. This method takes in a DefaultTableModel object from javax.swing.table and updates the table if it is not already populated, using the addRow method on the DefaultTableModel. The function loops over each feature in the GFF FeatureList, which can take time, but I see no way around this.

3. IO

3.1 FastaParser

3.1.1 Largely just a wrapper for the FastaReaderHelper object from org.biojava.nbio.core.sequence.io (BioJava) (Lafita et al., 2019).

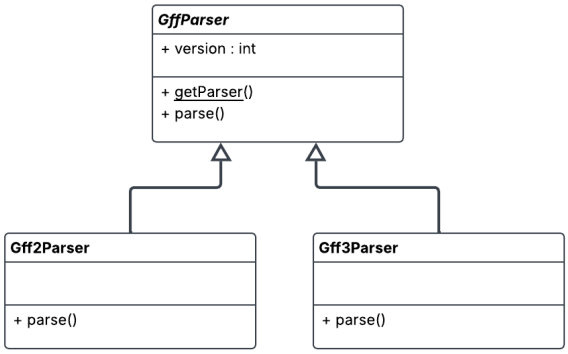
3.1.2 This class allows the user to take in a file and return a Map of DNA sequences.

3.2 GffParser

3.2.1 Largely just a wrapper for the objects GFF3Reader and GeneIDGFF2Reader from org.biojava.nbio.genome.parsers.gff, allowing the user to select a file and ultimately return a FeatureList, which is the BioJava class for handling feature files (Lafita et al., 2019).

3.2.2 GffParser is a factory class. It was originally made to return the correct parser depending on whether the user selected a GFF3 or GFF2 file.

- *Design Note: Even after parsing, the FeatureList for a GFF3 and GFF2 object looks different and would require different approaches to manage, so support for GFF2 was dropped and added to the backlog.*



4. Models

4.1 DataManager

4.1.1 DataManager is really just a pseudo-database that the application can use to set/get FASTA data in the form of a Map<String, DNASequence> and GFF data in a FeatureList.

4.1.2 The object is made in the mainframe and passed around the application where it's needed

4.1.3 There are some boolean methods to check whether data is already loaded.

5. Utils

- *Design Note: A general design note on my use of utils. Much of what I have in this folder consists of methods that extend the functionality of the Map<String, DNASequence> and FeatureList objects from BioJava (Lafita et al., 2019). In hindsight, these might not truly be utils and could be placed under models. Instead of creating a new class, I could actually extend the existing BioJava class objects and add my custom functionality. I could have extended the BioJava objects, make them instances (not static), and pass these to the various panels instead of using the data manager as a gofer. This occurred to me in a dream three days before submission, and so it will stay there.*

5.2 FastaUtils

5.2.1 FastaUtils contains generic helper methods for manipulating the FASTA DNASequence (and DNASequence Map) objects. Most of these are wrappers for existing BioJava methods (Lafita et al., 2019).

5.2.2 FastaUtils is heavily used in the BasicStatistics module.

5.3 GffUtils

5.3.1 Similar to FastaUtils, this class contains handler methods for manipulating FeatureList objects from BioJava (Lafita et al., 2019). It contains methods used in filtering the FeatureList — e.g., getSequenceIdentifierFromGene, getSelectedFeatureLocations. Methods for aggregating features include countFeaturesAcrossAllGenes, countFeaturesInGene, averageGeneLength, and averageFeatureLength. There are also methods for getting the shortest and longest feature — these methods are overloaded: if the user passes in a geneId, it will filter for the shortest/longest feature in that particular gene. If geneId is not provided, then it will find the shortest/longest feature in the GFF file. Other notable methods include helpers for calculating the coordinates from a gene (getCoordsForGene and getSelectedFeatureLocations), which are used in the FastaVisualization and GeneVisPanels.

6. Design Choices

6.1 Ant or Maven — **Decision**: Mavan

6.1.1 To help me decide on architecture, I need to understand if SQLite and BioJava are valid options. How easy are they to import, use, and share? I was able to make a dummy project using Maven and import SQLite and BioJava modules, running some boilerplate code from the GitHub README files of these packages. It seemed to work okay and produced the expected output. [not-included-in-submission]/Users/mspriggs/NetBeansProjects/programming_using_java/misc/javaWithMaven. The major bonus here is not having to recreate a parsing tool for GFF (v2 and v3). I'm not yet sure if I will need a database, but I know I have the option. I'm sure I can do this with Ant, but I recall the general advice being that Maven or Gradle is more advantageous if you have external dependencies, which I do. The XML logic seems easy enough. Ref: [https://www.youtube.com/watch?v=Fq05uX3G4b0 — accessed 15-Nov-25]

6.1.2 A consequence of using Maven is that, to include external dependencies, I need to create a fat JAR (though there are other ways).

6.2 BioJava or custom parser — **Decision**: BioJava

6.2.1 In its simplest form, I made a FASTA parser and a GFF3 parser in 71 lines [code not shown]. There is also a GeneIDGFF2Reader for GTF files, so I can allow my user to parse these too! There are options to parse large files using InputStreamProvider — consider this a feature request.

6.3 SQLite or custom class and storage module — **Decision**: custom class and storage module

6.3.1 At this point, I think a database might not be needed since the objects provided by BioJava seem to meet all my current needs. This may change as I get further into development (Lafita et al., 2019).

7. References

Lafita, A., Bliven, S., Prlić, A., Guzenko, D., Rose, P. W., Bradley, A., Pavan, P., Myers-Turnbull, D., Valasatava, Y., Heuer, M., Larson, M., Burley, S. K., & Duarte, J. M. (2019). BioJava 5: A community driven open-source bioinformatics library. *PLOS Computational Biology*, *15*(2), e1006791. https://doi.org/10.1371/JOURNAL.PCBI.1006791