Prof. Dr. Daniel Huson
Algorithmen der Bioinformatik
Fachbereich Informatik
Mathematisch-Naturwissenschaftliche Fakultät

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Advanced Java for Bioinformatics

Summer 2025

Assignment 4

Due: 22-May-2025

The goal of this assignment is to build a tree visualization using JavaFX shapes.

Your package `assignment4` should follow a structure similar to last week's assignment and should provide a main class called `TreeDrawer` that visualizes the part-of anatomy tree using two different standard layouts.

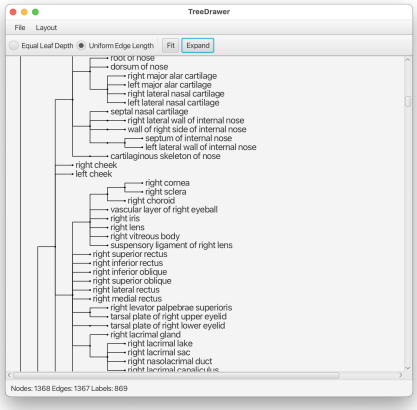
1 Designing the UI using Scene Builder (2 points)

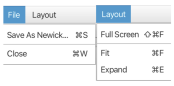
Use Scene Builder to design the user interface and save it to the file `Window.fxml`.

Assign `fx:id` values to all controls. For example, for the `Close` menu item, set `fx:id = closeMenuItem`.

In Scene Builder, set the controller class to `assignment4.window.WindowController`, and then use the `View` → `Show Sample Skeleton` menu item to generate starter content for the controller class.

The user interface should match the layout shown below:





Your UI should include menu items for:

- Saving the tree in Newick format (implementation is optional).
- Closing the application.
- Toggling full screen mode.
- Replicating the functionality of the `Fit` and `Expand` buttons.

The scene graph should use a border pane that has in the center a scroll pane that contains a stack pane that contains a single group that then contains three groups, one for edges, one for nodes and one for labels.

The bottom pane should display the number of nodes, edges, and leaves (i.e., labeled nodes, since only leaves should have labels in our visualization).

2 Cladogram Computations (4 points)

Implement a model class `Cladogram` that provides the following two methods:

1. `public static Map<ANode, Point2D> layoutEqualLeafDepth(ANode root);`
   This method computes coordinates for a left-to-right drawing where all leaves have the same x-coordinate.

   Use a post-order traversal with the following logic:

   - If v is a leaf, set $x(v) = 0$ and $y(v) = k$, where k is the number of leaves visited so far.
   - Otherwise, set:

     $$x(v) = \min\{x(w) \mid w \text{ is a child of } v\} - 1$$

     $$y(v) = \text{average}\{y(w) \mid w \text{ is a child of } v\}$$

2. `public static Map<ANode, Point2D> layoutUniformEdgeLength(ANode root);`
   This method computes coordinates so that all edges have the same length.

   Use a post-order to compute the y-coordinates (as in 1.).

   Use a pre-order traversal to compute x-coordinates:

   - Start with $x($`root`$) = 0$
   - For each node v, set $x(w) = x(v) + 1$ for all children w

3 Drawing the Cladograms (4 points)

The program should display either the "Equal Leaf Depth" layout or the "Uniform Edge Length" layout, based on user selection, coordinates computed as described above.

To this end, in a class called `DrawCladogram` define a method:
`public static Group apply(ANode root, Map<ANode, Point2D> nodePointMap, double width, double height)`
that takes as input the root of the tree, the mapping of nodes to points, and the desired width and height of the drawing. and produces a group of circles (for nodes), a group of paths (for edges) and a group of labels (for leaf labels) with x coordinates between 0 and `width` and y coordinates between 0 and `height`.

Put the three groups into a group and set the set of children of the stack pane equal to this group.

The font size of the labels should be set so that labels do not overlap (as a function of the number of leaves and the height).

Coordinates computed from the layout must be scaled to fit the visible drawing area. Implement two scaling strategies:

1. **Fit** – Scale the layout to fit the current visible area of the window, using:

   width = `controller.getScrollPane().getViewportBounds().getWidth()` and
   height = `controller.getScrollPane().getViewportBounds().getHeight()`

2. **Expand** – Scale the layout so that all labels are clearly readable, for example:

   width = `controller.getScrollPane().getViewportBounds().getWidth() - 400`
   height = `model.getNumberOfLeaves() * 16`

You can use this code to create a function that maps the original tree coordinates to coordinates that fit the given width and height:

```
public static Function<Point2D,Point2D> setupScaleFunction(Collection<Point2D> points, double width,
                            double height) {
  var xMin=points.stream().mapToDouble(Point2D::getX).min().orElse(0.0);
  var xMax=points.stream().mapToDouble(Point2D::getX).max().orElse(0.0);
  var yMin=points.stream().mapToDouble(Point2D::getY).min().orElse(0.0);
  var yMax=points.stream().mapToDouble(Point2D::getY).max().orElse(0.0);

  return (Point2D p) -> new Point2D((p.getX() - xMin) / (xMax - xMin) * width,
                       (p.getY() - yMin) / (yMax - yMin) *   height);
}
```

4 Save to Newick (optional)

Compute a Newick string representation of the tree, including labels only for the leaves.

The string should be computed in the model.

The implementation of the file chooser dialog and the actual writing to disk should be done in the presenter.

Should be reasonably straight-forward using this application of the Visitor pattern:

```
public static void inOrder(ANode v, Consumer<ANode> preVisitor, Consumer<ANode> betweenVisitor,
                                                                  Consumer<ANode> postVisitor) {
    preVisitor.accept(v);
    var remaining = v.children().size();
    for (var w : v.children()) {
      inOrder(w, preVisitor, betweenVisitor, postVisitor);
      if (--remaining > 0)
        betweenVisitor.accept(w);
    }
    postVisitor.accept(v);
}
```