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

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Advanced Java for Bioinformatics

Summer 2025

Assignment 2

Due: 8-May-2023

Due to the holiday on May 1^{st}, this is a $1 + \frac{1}{2}$ assignment (15 points rather than 10).

In this assignment, you are required to write a JavaFX program that can be used to explore the parts of the human body.

Later in the course, we will be working with three datasets:

- The "part-of" list of anatomy parts and a hierarchy of all "part-of" relationships.
- The 'is-a" list of anatomy parts and a hierarchy of "is-a" relationships.
- 3D models of all parts.

In this assignment, we will only work with the first item, the "parts" list.

1 Data loading (4 points)

There are three input files:

- `partof_parts_list_e.txt` - This contains a list of all parts (such as different organs, bones, muscles):

```
concept id representation id en
FMA3734 BP10374 aorta
FMA3736 BP10408 ascending aorta
FMA3768 BP10404 arch of aorta
FMA3784 BP10373 descending aorta
FMA3789 BP6896 abdominal aorta
...
```

- `partof_element_parts.txt` - for each part, this contains all the "file ids" that the part contains (these correspond to files that we will use at a later date to visualize the different parts):

```
concept id name element file id
FMA3734 aorta FJ1931
FMA3734 aorta FJ1932
FMA3734 aorta FJ3411
FMA3734 aorta FJ3413
FMA3734 aorta FJ3427
FMA3736 ascending aorta FJ3413
FMA3768 arch of aorta FJ3411
...
```

- `partof_inclusion_relation_list.txt` - this file defines parent - child relationships between the different parts. There is only one part that is not child of any other part, the root item "human body".

```
parent id parent name child id child name
FMA3734 aorta FMA3736 ascending aorta
FMA3734 aorta FMA3768 arch of aorta
FMA3734 aorta FMA3784 descending aorta
FMA3784 descending aorta FMA3789 abdominal aorta
FMA3784 descending aorta FMA87217 descending thoracic aorta
...
```

Setup a record class `ANode` like this:

```
public record ANode(String conceptId, String representationId, String name, Collection<ANode> children, Collection<String> fileIds) {
    public String toString() {
        return name + " (" + conceptId + ")";
    }
}
```

Write a class `TreeLoader` that has one static method that is used to parse the three files into a tree that you construct using `ANode` objects and returns the root:

```
public class TreeLoader {

    public static ANode load(String partsFile, String elementsFile, String relationsFile) throws IOException {
        // your code here...
    }
}
```

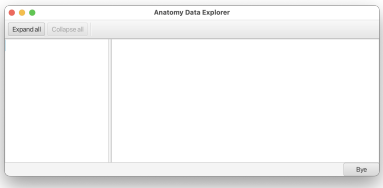
The method returns the root node of the anatomy tree. The root node contains a list of children. Each child contains a list of its children, etc. Also, each node has a list of file-ids that are strings, such as "`FJ2557`".

The `toString()` method of `ANode` ensures that, when the nodes are placed into "tree items" (task 3 below), then the labels used in the tree view will look like this: "`human body (FMA20394)`".

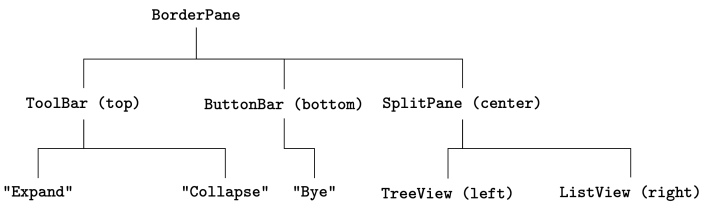
2 Anatomy Data Explorer basic program (2 points)

Setup your main class `AnatomyDataExplorer` that extends the JavaFX application class and implement the method `public void start(Stage stage)`.

Then, in the start method, setup and show a scene graph that has the following items:



In more detail, this should be the structure of the scene graph (the root node is a border pane):



3 Anatomy TreeView (4 points)

The goal is to show the anatomy tree using a TreeView, by implementing more code in your main class.

Proceed in three steps:

First, call your method `TreeLoader.load()` to load the anatomy tree.

Second, create the tree view (e.g using `var treeView=new TreeView<ANode>();`.

Third, populate the tree view as discussed in the lecture:

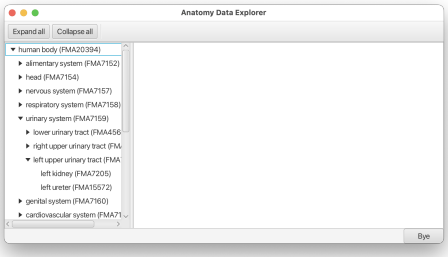
`var root=TreeLoader.load(`*files....*`);`

```
var rootItem= createTreeItemsRec(root);
var treeView=new TreeView<>(rootItem);

...

static TreeItem<ANode> createTreeItemsRec(ANode node) {
    var item = new TreeItem<>(node);
    for(var child : node.children()) {
        item.getChildren().add(createTreeItemsRec(child));
    }
    return item;
}
```

If this is done correctly, then you should see the anatomy hierarchy displayed in your tree view:



Note that each tree item constructed in this way contains the corresponding ANode as its value, which can be accessed using the method `getValue()`.

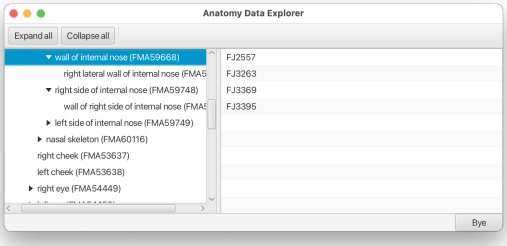
4 ListView (3 points)

Each `ANode` contains a list of file-ids. We want these to be displayed in the list view.

To implement this, you need to add a "change listener" to the selected item of the tree view (as discussed in the lecture):

Whenever the selection changes, then if the new selected item is null, clear the list view. Otherwise, populate the list view with all the file-ids associated with the ANode that is the value of the selected tree view item (one per line).

For example, if you select the part "wall of internal nose" in the tree view, then in the list view, you should see the file ids: FJ2557, FJ3263, FJ3369 and FJ3395:



5 Expand all, collapse all and bye buttons (2 points)

Implement the three buttons.

- When the user presses “Bye” the program should quit.
- When the user presses “Collapse all” the tree should collapse to the root node.
- When the user presses “Expand all” then *all* nodes should be expanded.

Optional: The “`Collapse all`” button should only be enabled when the tree is not collapsed to the root node.