Assignment 5: Treemaps

1. Introduction

We continue our study of recursive algorithms and also gain familiarity with building graphical user interfaces (GUIs) and querying your computer's file system. *We will ask you for the time spent on this assignment, so keep trac*k.

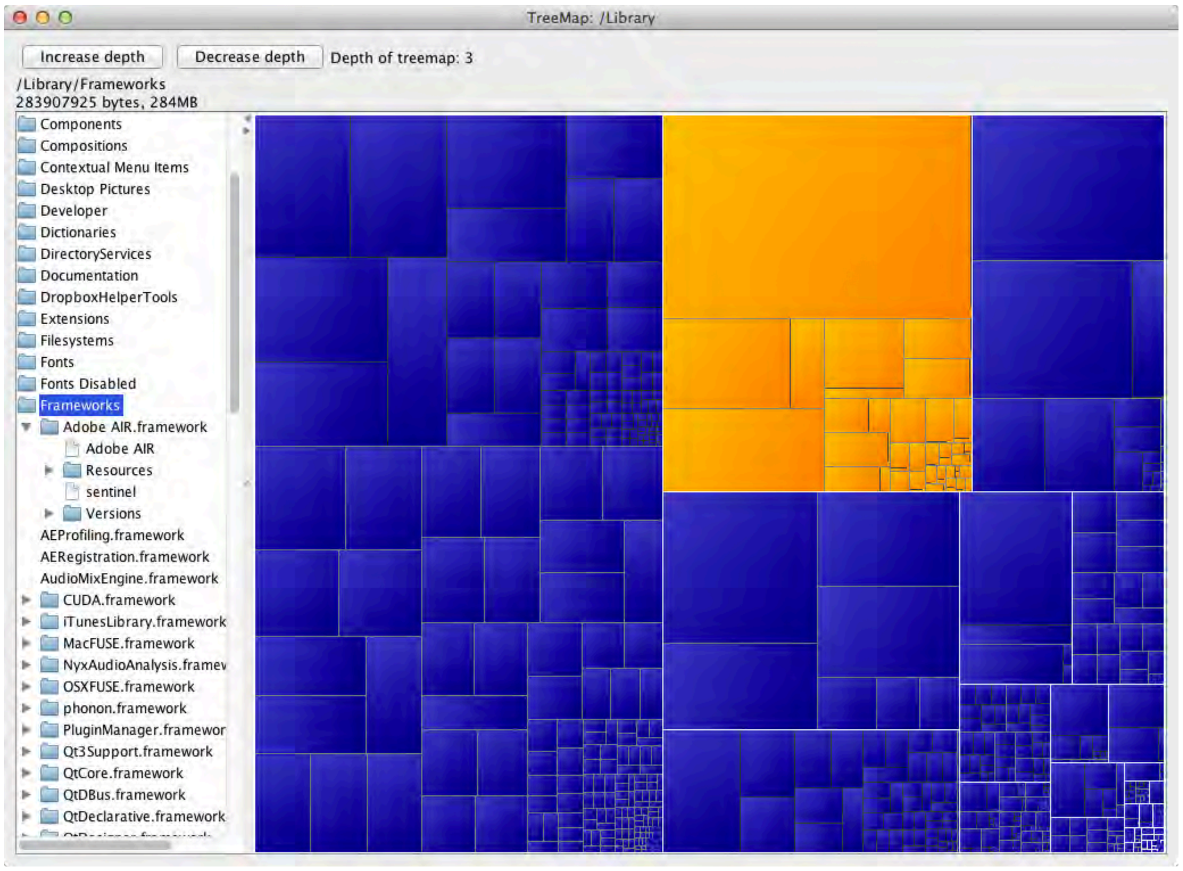
The goal of the program is to display the structure of a file system using a visualization known as a *treemap*. Each file or directory is displayed as a rectangle, whose area is proportional to the size of the object. If a file/directory is contained within another directory, the rectangle corresponding to the first is nested within the rectangle corresponding to the second. Thus, a treemap is a hierarchical structure, just like the file system itself, and it shows at a glance the relative sizes of different objects in the file system.

You can read about treemaps at www.cs.umd.edu/hcil/treemap-history/ and see some nice animations at https://visualign.wordpress.com/2011/11/09/implementation-of-treemap/

To whet your appetite, we show in the diagram below what your completed program could produce. On the left is part of the hard-drive file structure. On the right is the treemap of that part of the hard drive, to recursion depth 3. A directory is selected on the left, its path and size are displayed above the file structure, and the corresponding rectangle is displayed in orange on the right. The biggest objects in a directory are nested at the top left of the containing rectangle, the smallest at the bottom right.

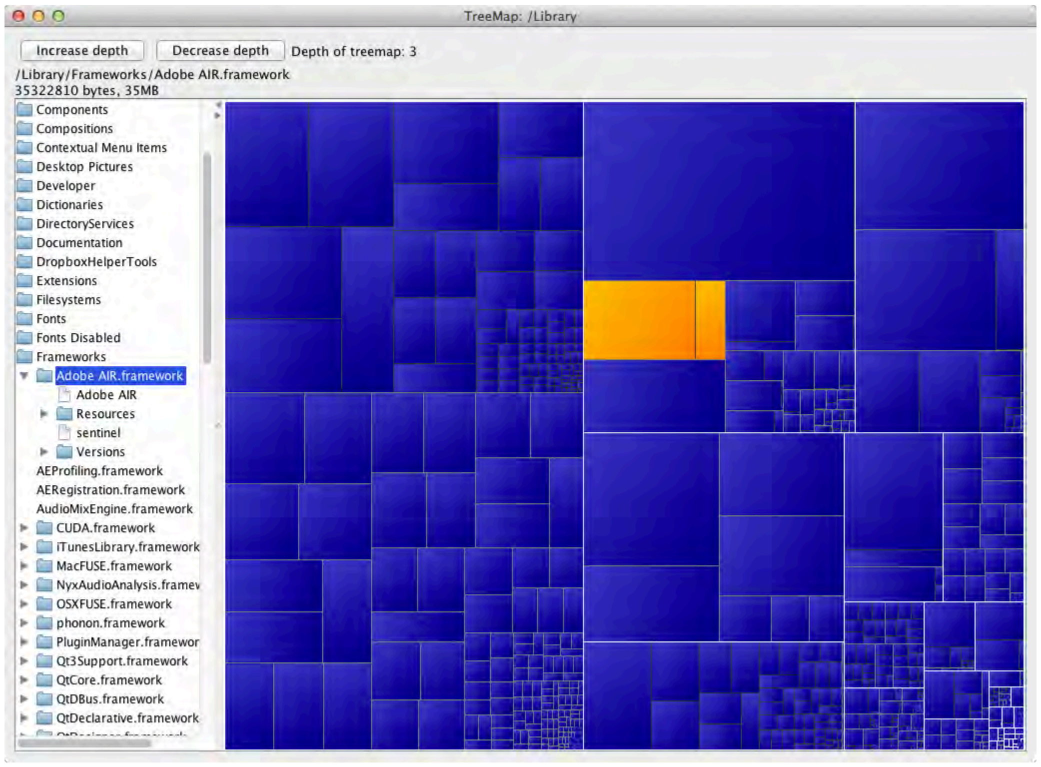
Click another directory or file in the left column, and its rectangle will be highlighted in yellow, as shown in the image on the next page. Click on a rectangle in the tree map, and it will turn yellow and its name will be selected in the left column, as shown in the second image on the next page. Note that each orange rectangle is a complete treemap, nested within a larger treemap!

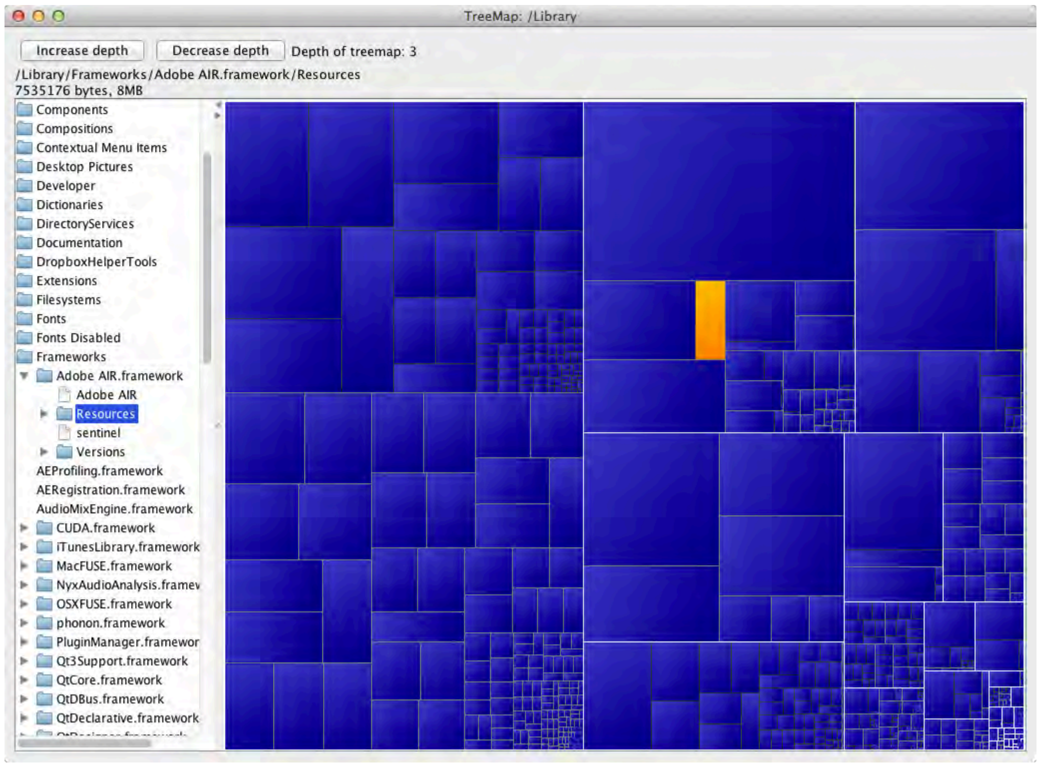
You can buy apps that implement this functionality. You can also write the program yourself.



/Library/Frameworks
283907925 bytes, 284MB

```
Components
Compositions
Contextual Menu Items
Desktop Pictures
Developer
Dictionaries
DirectoryServices
Documentation
DropboxHelperTools
Extensions
Filesystems
Fonts
Fonts Disabled
Frameworks
▼ Adobe AIR.framework
    Adobe AIR
  ▶ Resources
    sentinel
  ▶ Versions
  AEProfiling.framework
  AERegistration.framework
  AudioMixEngine.framework
▶ CUDA.framework
▶ iTunesLibrary.framework
▶ MacFUSE.framework
▶ NyxAudioAnalysis.framev
▶ OSXFUSE.framework
▶ phonon.framework
▶ PluginManager.framewor
▶ Qt3Support.framework
▶ QtCore.framework
▶ QtDBus.framework
▶ QtDeclarative.framework
```


TreeMap: /Library
Increase depth
Decrease depth
Depth of treemap: 3
/Library/Frameworks/Adobe AIR.framework
35322810 bytes, 35MB
Components
Compositions
Contextual Menu Items
Desktop Pictures
Developer
Dictionaries
DirectoryServices
Documentation
DropboxHelperTools
Extensions
Filesystems
Fonts
Fonts Disabled
Frameworks
Adobe AIR.framework
Adobe AIR
Resources
sentinel
Versions
AEProfiling.framework
AERegistration.framework
AudioMixEngine.framework
CUDA.framework
iTunesLibrary.framework
MacFUSE.framework
NyxAudioAnalysis.framev
OSXFUSE.framework
phonon.framework
PluginManager.framewor
Qt3Support.framework
QtCore.framework
QtDBus.framework
QtDeclarative.framework


TreeMap: /Library
Increase depth
Decrease depth
Depth of treemap: 3
/Library/Frameworks/Adobe AIR.framework/Resources
7535176 bytes, 8MB
Components
Compositions
Contextual Menu Items
Desktop Pictures
Developer
Dictionaries
DirectoryServices
Documentation
DropboxHelperTools
Extensions
Filesystems
Fonts
Fonts Disabled
Frameworks
Adobe AIR.framework
Adobe AIR
Resources
sentinel
Versions
AEProfiling.framework
AERegistration.framework
AudioMixEngine.framework
CUDA.framework
iTunesLibrary.framework
MacFUSE.framework
NyxAudioAnalysis.framev
OSXFUSE.framework
phonon.framework
PluginManager.framewor
Qt3Support.framework
QtCore.framework
QtDBus.framework
QtDeclarative.framework

2. Grading

Solutions will be graded on correctness, the quality of the algorithms, and style. A correct program compiles without errors and behaves according to the requirements given in this handout and the comments of the code. A program with good style is clear, concise, and easy to read. Follow the style guidelines given for this course on the course website.

3. Collaboration policy and academic integrity

You may do this assignment with one other person. If you are going to work together, then, as soon as possible —and at least by the day *before* you submit the assignment— get on the CMS for the course and do what is required to form a group. Both people must do something to form the group: one proposes, the other accepts.

If you do this assignment with another person, you must *work together*. It is against the rules for one person to do some programming on this assignment without the other person sitting nearby and helping. You should take turns "driving" —using the keyboard and mouse.

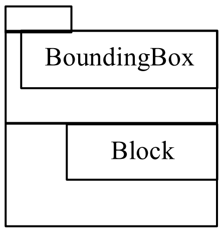
With the exception of your CMS-registered partner, you may not look at anyone else's code, in any form, or show your code to anyone else, in any form. You may not show or give your code to another person in the class. While you can talk to others, your discussions should not include writing code and copying it down.

4. Getting help

If you don't know where to start, if you are lost, etc., please SEE SOMEONE IMMEDIATELY —a course instructor, a TA, a consultant. Do not wait. A little in-person help can do wonders.

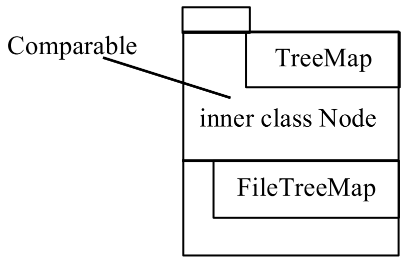
5. The structure of the program

We discuss the classes and what each one does. Spend some time perusing the class files to get a sense of the program. Reading well-presented code, being aware of how it is written, is a good idea.



Class Boundingbox: an instance describes a rectangle in the plane, by giving its lower left point and its higher right point. Take a look at it. It is immutable, and its two fields are public. One can reference them directly, without getter methods. You don't have to code anything in it.

Class Block: an instance is a BoundingBox used in a particular context; it represents the rectangle for a file or directory in the treemap. You don't have to code anything here. It has a method *paint*(*Graphics2D*), which draws the rectangle. Take a look.



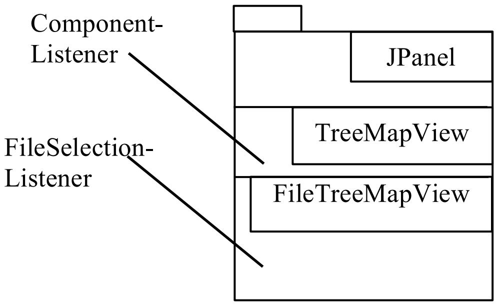
Classes TreeMap and FileTreeMap. TreeMap implements a general treemap, using objects of inner class Node to describe each node of the treemap. Note that Node implements interface Comparable, based on the "weight" or "size" of the nodes. The fields of TreeMap are the root node of the tree, the selected node (if any), and the width/height assigned to the tree.

TreeMap contains recursive method *sliceAndDice*, which constructs the treemap. You will write this method.

FileTreeMap extends TreeMap in order to particularize the treemap for a file system. It contains lookup tables to map file paths to treemap nodes, and vice versa.

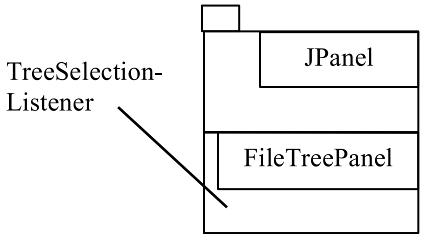
The only connection to the GUI in these two classes is recursive method *TreeMap.Node.paint*(*Graphics, …*), which paints the tree whose root is that node using the Graphics parameter. Separating computation from communicating results of that computation to a user, is an important part of good design.

Classes TreeMapView and FileTreeMapView. TreeMapView is the JPanel on which the treemap is painted. One field of this class is the treemap itself, of class TreeMap. The class listens to the resizing of the component —i.e. the JPanel— in which case the whole treemap has to be recomputed again. This class is short and simple.



Subclass FileTreeMapView particularizes TreeMapView to this special implementation of a treemap for a file system. Its major functionality is to provide procedure *selection-Changed*, which is required by interface FileSelection-Listener. Given a path for a file or directory (selected from the panel on the left), this procedure finds it in the treemap, selects it, and then repaints so that its corresponding rectangle in the treemap will become orange.

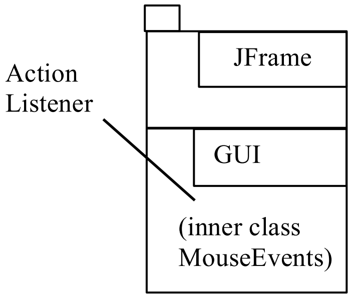
Class FileTreePanel. This subclass of JPanel contains the files and directories that appear in the left part of the GUI. We found this class on the web; it was written by Kirill Grouchnikov. We changed it slightly to fit our needs. We don't understand all of it; we just use it.



This class implements TreeSelectionListener so that it can respond to single clicks to select a file or directory and double clicks to open or close a directory.

This class has procedure *main*, so it can be run as an application. We used this procedure when we first began using the class to see how it worked and to modify it slightly.

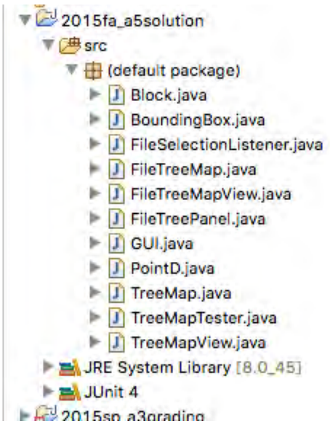
Class GUI. This class extends JFrame, so an instance is associated with a window on your monitor. It has method *main*, to create an instance of the GUI and start the program running. It places all components in the JFrame, as you might expect, and it implements ActionListener so that it can listen to clicks on the two buttons. When either of the buttons increase-recursion-depth and decrease-recursion-depth are pressed, method *actionPerformed* causes the treemap to be recomputed by calling *recomputeTreeMap*.



Class GUI has an inner class, MouseEvents —as discussed in lecture. Its method *mouseClicked* is called to process a click in the treemap (to select a file). That method calls function *getNodeContaining* to retrieve the node and select it in the file tree.

Looking at the whole program as one big entity can be daunting. There is a lot there. But choose one piece of it, as shown above, and study it carefully, and it begins to make sense. Break a large problem into smaller pieces, making each piece as independent as possible —i.e. making the interfaces between them small. That is the way to solve larger problems.

6. Your task



Download a5release.zip and add the enclosed files to a new project in Eclipse. Your project structure should look like the image on the right.

Note: You are responsible for information put into pinned Piazza note @665, *A5 FAQs*, which already has stuff in it. We have posted explanations of parts of the assignment on @665. *Look* at @665 before *asking a question, please!*

You will be making changes to FileTreeMap.java, TreeMap.java, and GUI.java, in that order. When you begin working on each method, *first look in Piazza note @655 for information and explanations*. Let's take them one by one.

Class FileTreeMap: function *size*(*int d*, *File f*)

This class specializes a treemap to display a file system. You need to write function *size*(*int d*, *File f*), which computes the size of file or directory *f*, up to a maximum recursion depth of *d*. The purpose of having you write this function is to learn about how one deals with class File.

Class TreeMap: functions *.getSplit* and *sliceAndDice*

- Your job is to code the "slice-and-dice" algorithm for computing a treemap (see Section 7). The methods you need to complete are:
- *TreeMap.getSplit* and *TreeMap.sliceAndDice*

You can do some testing using a JUnit testing class, and we suggest you do this at least for *Node.getSplit*.

Procedure *TreeMap.main* is there for you to be able to test TreeMap independently of other parts of the code. Manually create some TreeMap objects and either print them (*TreeMap.printTree*) or, even better, figure out how to create a simple JFrame and display the treemap (*TreeMap.paint*) on it. We will not look at it.

Class GUI: functions *GUI.MouseEvents.mouseClicked* and *GUI.actionPerformed*

This class contains the code for the main JFrame, including the bit that processes mouse clicks on the treemap: *GUI.MouseEvents.mouseClicked*. You have to write this method. To do this, you need to find out which leaf node of the treemap was clicked and select the corresponding file/directory in the hierarchy on the left.

Method *GUI.actionPerformed* is called when a GUI button is clicked to increase or decrease the recursion depth. You have to write this method. You need to figure out which button was clicked and recompute the treemap appropriately.

The purpose of having you write these methods is to get you to work with GUIs a bit.

7. Building a treemap

A treemap is built recursively, as shown below. It is given a rectangle r in which the treemap will go.

```
 // Lay out a treemap for directory d in onscreen rectangle r
Algorithm TREEMAP(d, r):
    List all the immediate children (files and subdirectories) of d in descending
        order by size (stop if the maximum desired recursion depth
        has been reached or there are no children)
    Call procedure LAYOUT(children, 0, #children -1, r), which assigns to each child a rectangle of
          area proportional to its size, nested within r
    For each child c with assigned rectangle s: Call TREEMAP (c, s)
```

You will write the LAYOUT procedure. It can be implemented in many different ways. The one we use is called "slice-and-dice". This is itself a recursive algorithm! We give a high-level explanation of it below (*SplitRatio* is a predefined constant between 0 and 1). But also see the comments in the method in the code itself, which gives a step-by-step set of instructions to implement.

```
// Partition rectangle r into subrectangles corresponding to nodes. The area of
// each subrectangle is proportional to the size of the associated node in node[m..n].
// Precondition: list nodes is sorted in descending order by size
Algorithm LAYOUT-SLICEANDDICE(nodes, m, n, r):
    If nodes[m..n] is empty, do nothing and return
    If nodes[m..n] s has just one element, allocate it the entire rectangle r and return
    Select the smallest h such that the sum of te sizes of
            nodes[m..h] is at least SplitRatio times the total size of nodes[m..]
    Split r into two parts s and t according to SplitRatio along
            its longer side (to avoid very narrow shapes)
    Assign nodes[m..h] to split-off part s and nodes[h+1..n] to rectangle t
    Call LAYOUT-SLICEANDDICE(nodes, m, h, s)
    Call LAYOUT-SLICEANDDICE(nodes, h+1, n, t)
```

8. What to submit

In the comment at the top of file GUI.java, replace *hh* and *mm* by the hours and minutes you spent on this assignment. Replace BOTH of them with integers. Please be careful in doing so. A program extracts this data, and if you change anything other than hh and mm, you screw up the program. Put in your name and Netid and tell us what you thought about the assignment. Submit a .zip file that contains all the .java programs in the src directory of the project.