SE/ComS 319 Homework 4: Swing

Homework must be individual's original work. Collaborations and discussions of any form with any students or faculty members other than Samik Basu are not allowed. If you have any questions/doubts/concerns, post your questions/doubts/concerns on the blackboard LS and/or contact the TAs or instructor.

The homework is due on October 24, 2012 at 11:59PM. Submission guidelines are as follows:

1. Submit via Blackboard
2. You are provided XText1.java. Update class name appropriately such that your top-level Java class is XText1⟨yournetid⟩ (your Java file is, therefore, named XText1⟨yournetid⟩.java)
3. You must submit just XText1⟨yournetid⟩.java, i.e., all class definitions should be present in one file XText1⟨yournetid⟩.java; all classes that you write should be named ⟨someclassname⟩⟨yournetid⟩
4. Your submission XText1⟨yournetid⟩.java must compile and execute in Java 6 or 7, without requiring any external libraries or files
5. Write your name at the top of XText⟨yournetid⟩.java and mention the Java version you have used (Java 6 or 7)
6. Submit a short report in pdf format: hw3.Fall12.report⟨yournetid⟩.pdf explaining how you updated the given code to solve this homework assignment. Your report must not be more than a page. Include at least the following items:
   (a) Important changes in the existing methods
   (b) Addition of new methods

Violation of any of the above requirements will result in automatic 10pt deduction from your homework assignment.

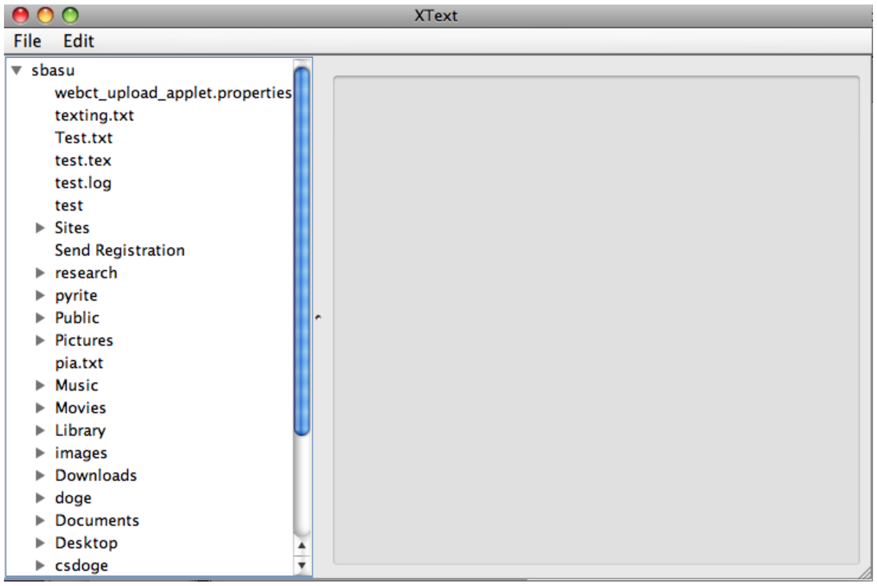
Start with the XText1.java file given to you as a solution to the homework 3 assignment.

1. **Basic Re-organization.** Split the text editor pane into two: left and right panes, using JSplitPane. The right pane now will provide tabbed view (you are given some code that you can use). The left pane will show the tree-view of the file system (use JTree). The tree consists of root and intermediate nodes that are directories and the leaf-nodes that are files. The tree structure can be used to open files (with extensions .txt and .text) in the right pane area.
2. **On-the-fly Construction of Tree.** You should not construct the entire tree in one shot; tree nodes should be added as and when needed. In other words, if a node is expanded in a tree, only its children are added to the tree and shown in the tree-view. Consider the example, where the directory structure is as follows: Directory D1 contains two files F1, F2 and a sub-directory D2 (i.e. D1 has three children F1, F2, D2). D2 in turn contains file F3 and a directory D3.

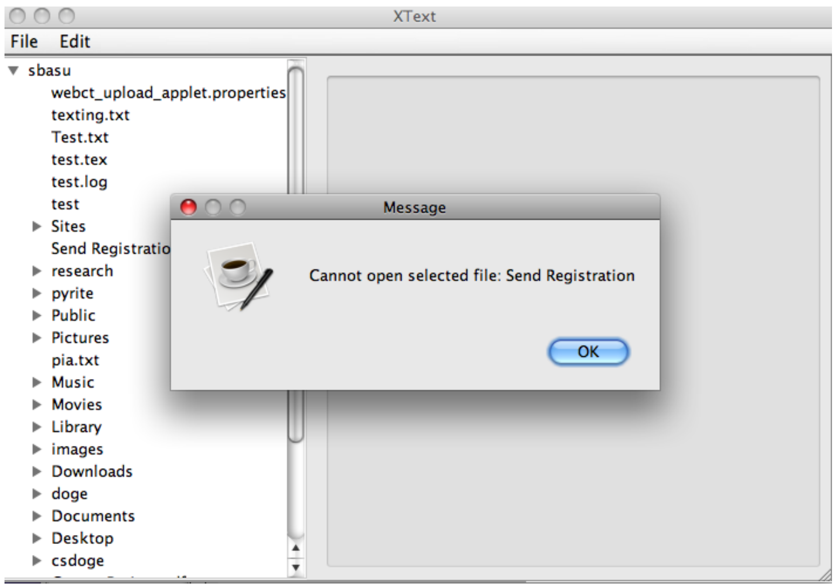
   Let initially the tree view is showing D1 and not its children (In tree view directories will be shown as nodes that are expandable). The tree model at this point should not contain the sub-tree rooted at D1; in other words, the tree model should not contain F1, F2, D2, F3, D3. When D1 is expanded, the tree model should get updated via inclusion of F1, F2, D2 and the tree view will be appropriately updated.
3. **View-only Tree.** You are not required to allow user-update directly on the tree. That is, users cannot rename file, add file to the tree directly.

4. **Use Cases**

(a) Initially, the left pane contains the tree-view starting from the user's home directory (the view of the home directory is unexpanded) and the right pane is empty. See below.

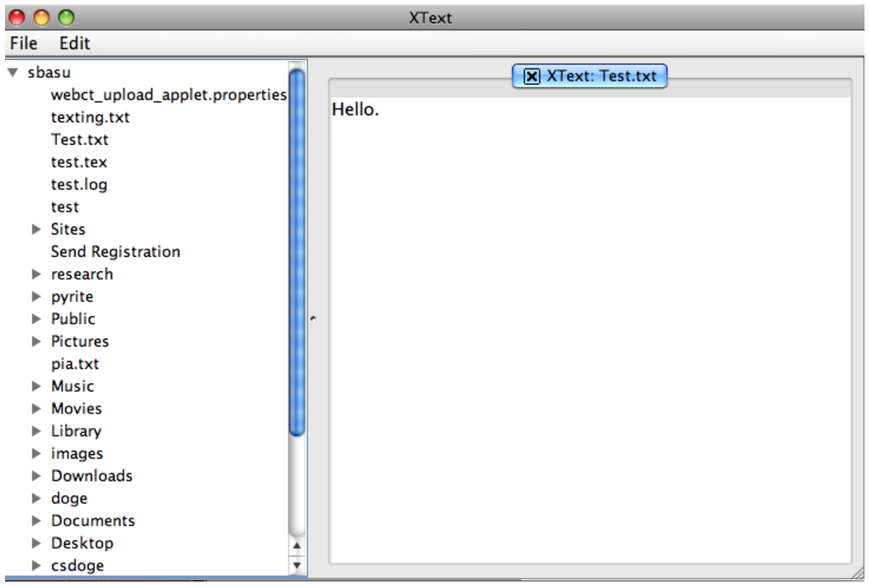


(b) When the user clicks on a file in the tree-view that cannot be opened (i.e., the file does not have .txt or .text extensions), a message stating that the file cannot be opened is displayed. See below.



Cannot open selected file: Send Registration

(c) When the user clicks on a file in the tree-view that can be opened (i.e., the file has .txt or .text extensions), a new tab item composed of a text area is added. The text area contains the data of the file that has been clicked. This text area becomes visible in the right pane. The following shows the new state of the editor after the user clicks on `Text.txt` file in the file-tree.



(d) When the user clicks on a different file in the tree-view that can be opened, another tab item is added and a new text area containing the file data appears in the right pane.

(e) User can navigate between the files already open in the tab by clicking on the corresponding tab.

If the user tries to open a file F either via the tree or via the File menu, and if F is already open in some tab, then the view of the right pane changes (if needed) to show the tab that contains the file F.

If the user tries to open a file F either via the tree or via the File menu, and if F is not already open in some tab, then a new tab containing F is added to the right pane.

(f) The user can close a tab item by clicking on the "cross-mark" in the tab . If the file associated to the tab has been changed, the user will get an option to save the changes (use the same specification as in Homework 3 assignment).

If the user closes the editor via File menu exit item, then for all files in the tab *that have been changed*, the user will get an option to save the changes for each file.

(g) Every time the user clicks on `New` in File menu, a new tab containing an empty text area opens.

(h) Every tab has a title of the form: `XText:<openedFileName>` or `XText:New`.

(i) If the save menu item is clicked, then text area being viewed in the right pane will be the one that will be saved.

(j) All edit operations (either invoked Edit menu or via Popup) should work on the text area being viewed in the right pane.

5. **Your code must be reasonably commented**

   (a) Every class must have a description. If the class is responsible for creating an user-interface, the description should state that. If the class is responsible for realizing some storage and computation, the description should state that.
   Sample comment:

   ```
   /**
    *  ButtonPanel contains three buttons and
    *  implements the action listener that listen
    *  to the button events
    */
   ```

   (b) Every method that has more than one program statement must be commented.
   Sample comment:

   ```
   /**
    *    checkRowWin method: to check whether a row has a wining combination
    *    @param index: row index of the gridElement
    *    @return 0 if player zero has winning combination in row index
    *            1 if player one has wining combination in row index
    *           -1 otherwise
    */
   ```

   (c) If you are using Eclipse, you can just type "/**" followed by ENTER to create the comments.

6. **You can use any code-snippets from the public domain** if you find anything useful. Provide a reference for all such code-snippets at relevant places in your source.

   Some example Code is presented in the next page for you to review and use (if you want).

You are given following code snippet for creating close-icons for the tabs. You can use them in your solution.

```
class CloseTabIcon implements Icon {
  private int x_pos;
  private int y_pos;
  private int width;
  private int height;
  private Icon fileIcon;

  public CloseTabIcon(Icon fileIcon) {
    this.fileIcon=fileIcon;
    width=16;
    height=16;
  }

  public void paintIcon(Component c, Graphics g, int x, int y) {
    this.x_pos=x;
    this.y_pos=y;

    Color col=g.getColor();

    g.setColor(Color.black);
    int y_p=y+2;
    g.drawLine(x+1, y_p, x+12, y_p);
    g.drawLine(x+1, y_p+13, x+12, y_p+13);
    g.drawLine(x, y_p+1, x, y_p+12);
    g.drawLine(x+13, y_p+1, x+13, y_p+12);
    g.drawLine(x+3, y_p+3, x+10, y_p+10);
    g.drawLine(x+3, y_p+4, x+9, y_p+10);
    g.drawLine(x+4, y_p+3, x+10, y_p+9);
    g.drawLine(x+10, y_p+3, x+3, y_p+10);
    g.drawLine(x+10, y_p+4, x+4, y_p+10);
    g.drawLine(x+9, y_p+3, x+3, y_p+9);
    g.setColor(col);
    if (fileIcon != null) {
      fileIcon.paintIcon(c, g, x+width, y_p);
    }
  }

  public int getIconWidth() {
    return width + (fileIcon != null? fileIcon.getIconWidth() : 0);
  }

  public int getIconHeight() {
    return height;
  }

  public Rectangle getBounds() {
    return new Rectangle(x_pos, y_pos, width, height);
  }
}
```

You can create a class (possibly an inner class for easier accessibility) which extends `JTabbedPane` and implements `MouseListener`. One of the methods in `MouseListener` that will be of interest is the `mouseClicked(MouseEvent e)`. The MouseEvent will contain information of the coordinates of the click event. These coordinates can be used to find out whether the “X” mark in the object of type `CloseTabIcon` has been clicked or not.