# INFO0027: Project 2 (20% – Groups of 2)
File Explorer

Prof. Laurent Mathy - Gaulthier Gain

Submission before Friday May 17

## 1 Introduction

In this project, you will implement the **logic** of a graphical file explorer in `Java`. This application is only a simulation of a file explorer. In other words, you do not have to deal with real files and folders. The objective is to let you apply design patterns that you have seen in the theoretical course.

## 2 File Explorer

The application shown in Figure 1 consists of a graphical interface which is itself composed of two main components:

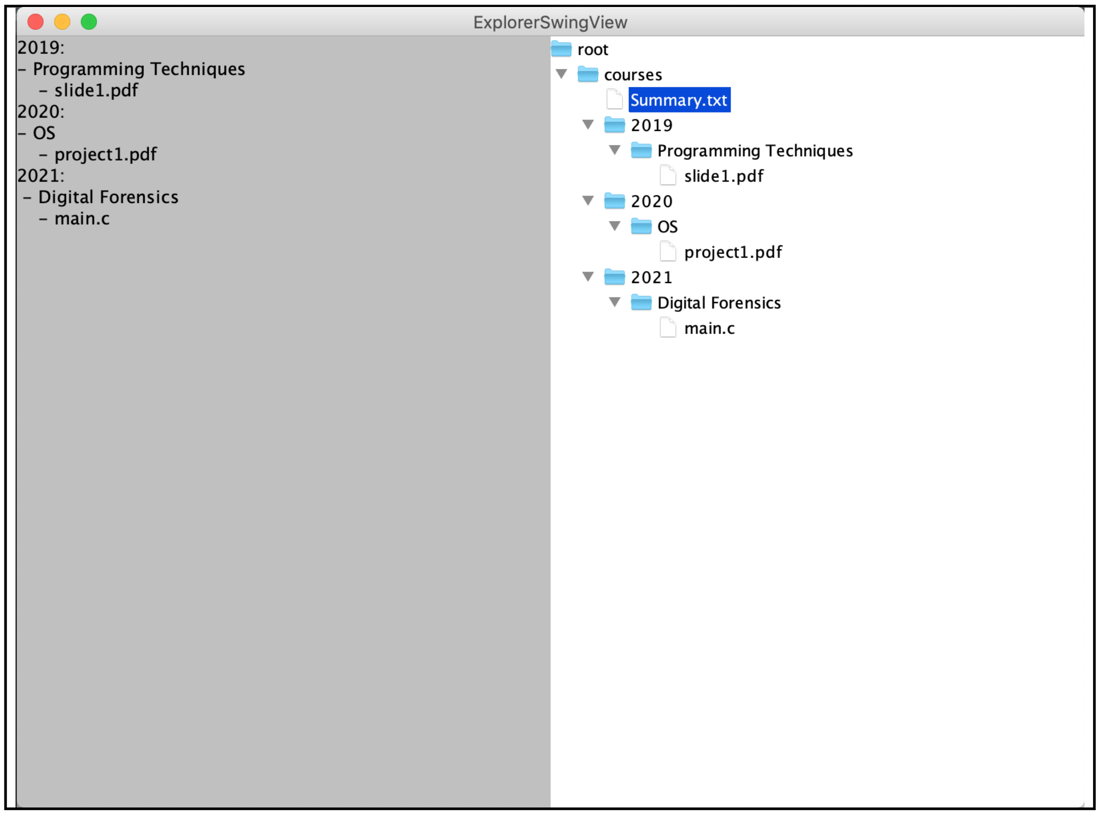
1. A `JTree` component that represents the hierarchical file system. You must interact with this component to add new files and folders.
2. A `JTextArea` component that allows to display the content of a file/folder.

In addition, to these two components, a context menu is displayed when the user performs a right click on the `JTree` component. This popup menu contains the following menu options: create a file, create a folder, copy a file/folder, make an alias and compress a folder. These options will be discussed in Section 3.

The GUI is to be handled by a library that will be made available to you (in other words, you do not have to program it by yourself). This library contains mechanisms for signaling that the user performs some actions, to which your program should respond in the appropriate way. For instance, when the user create a new file, the program receives a notification, which should then trigger operations such as displaying the right input dialog to enter the name of the file, adding it to the hierarchical tree explorer, and finally refreshing the GUI display.

The library discussed above is provided as a file called `graphics.jar`[1]. The classes contained in this file are only given as bytecode (in other words, their

[1]`http://www.montefiore.ulg.ac.be/~gain/courses/info0027.php#projects`



```
2019:
- Programming Techniques
   - slide1.pdf
2020:
- OS
   - project1.pdf
2021:
 - Digital Forensics
   - main.c
```

Figure 1: File explorer

source code is not available). The documentation of this library is given in the Appendix.

## 3 Requirements

Concerning the logic of your code, you must handle the features defined below. Depending the functionality, design patterns are not always necessary. Note that for all the actions, the root of the `JTree` must have been defined before. This list of feature is only an overview, additional information can be found in Appendix.

Note: if something is not explicitly explained, you are free concerning its implementation.

## 3.1 Create a file/folder

The first action concerns the file/folder creation. When a file/folder is created, this one is directly added below the current selected node. Before creating the node, you must specify some information depending if it is a file or a folder.

1. If a file is created, you must specify its name and its content (always text content). A graphical popup has been designed to handle this case. This one is `fileMenuDialog()` and will return an array of string containing the filename and its content (always as a text).
2. If a folder is created, you must only specify its name. The graphical popup associated is `folderMenuDialog()` that will simply return its name.

For both, you must simply displayed their name in the tree. In addition, file(s) and folder(s) can have the same name/content.

## 3.2 Make an alias (file only)

User can also create an alias of a file. An alias can be seen as a special node that is explicitly associated to a given file. When an alias is created, this one provides the same interface as their associated file (same name, same operations, ...) without duplicating the contents of this file. In other words, we can compare it as a pointer to its underlying file.

When an alias is created, this one must be added to the same (tree) level than the original node. The name of its underlying file following by the string `(alias)` must be displayed in the tree.

Note: you can't make an alias of the root node.

## 3.3 Copy a file/folder/archive

The user can also copy a file, a folder or an archive. In this case, it's a complete copy (unlike an alias).

When a copy is performed, the resulting element must be added to the same (tree) level than the copied element. The name of the copied node following by the string `(copy)` must be displayed in the tree.

Note: you can't copy the root node as well as the aliases.

## 3.4 Compress a folder into an archive (folder only)

Another feature of the program is to compress (in a fictive way) a folder. In this case, the archive will be built step by step using the adequate graphical popups namely `displayArchiveWindow1()`, `displayArchiveWindow2()` and `displayArchiveWindow3()`. They will respectively return the name, the extension and the compression level of the archive.

When an archive is created, it must be added to the same (tree) level than the folder. The name as well as the extension must be displayed in the tree.

Note: you can't compress the root node.

## 3.5 Display the content of a file/folder

When the user performs a double click, it must display some information in the `JTextArea` component:

- If the double click is performed on a file/alias, its content must be displayed. The same behaviour must be handled for a copy or an alias.
- If the double click is performed on a folder or an archive, all the contained files/folders must be displayed[2].

Note: if the content of a file/folder/archive is empty you must show nothing (empty text).

## 3.6 Log user activity

To keep a track of the user actions, a custom logger module must be implemented. This module will handle textual messages emitted by the system and will either record them by printing them on the console, **or** be writing them in a file. The module must be customizable. In others words, it must be possible to concatenate, in any order, additional information to any message. Moreover, it should be easy to add new approaches to record textual messages or/and unimplemented information.

Your program must thus record the following information: the current date and time, the name of the user that uses the program, the current operating system, and the user actions (actions related to the callbacks). By default, it will log information in the standard output (console). Nevertheless, if the `log` argument is provided to the program, this one will only log messages in a file (called `log.txt`). The following message is an example of log:

```
Tue Apr 01 10:27:33 CEST 2019 - gaulthiergain - Mac OS X - eventExit
```

Hint: use the `System`[3] class to get such information and the `PrintWriter`[4] class to write in a file.

[2] Only the name, not the content.
[3] `https://docs.oracle.com/javase/8/docs/api/java/lang/System.html`
[4] `https://docs.oracle.com/javase/8/docs/api/java/io/PrintWriter.html`

## 4 Implementation

A skeleton[5] is provided. This one contains two classes:

1. `Main.java`: main class of the program which contains the entry point. It instantiates a `GuiHandler` object to deal with the graphical components. This class **can't** be modified.

2. `GuiHandler.java`: class that will handle the graphical events and will provide callbacks. You will mainly work on this class.

Concerning the implementation, the following guidelines must be respected:

1. You will implement the program using Java >= 1.8, with the following packages only: `java.lang`, `java.io`, `java.util` and of course, `graphics`.

2. You will not cluster your program in packages or directories. All your source java files should be found in the same directory (default package).

3. You will ensure that your program is fully functional (i.e. compilation and execution). You can test it on the student's machines in the lab (ms8xx.montefiore.ulg.ac.be).

4. You will only use the provided graphical interface. Developing your own GUI instead of the provided library, or using any classes provided by other graphical libraries (such as Swing, AWT, JavaFX, ..) is prohibited!

5. By not considering the skeleton, you are free regarding the implementation of your the program however you must provide a well structured code where each part has a well-defined role ("library" or "business" code). In other words, it is strictly forbidden to have a single file containing all the code.

6. You must respect the oriented-object paradigm and some of its concepts should be used.

7. Your program can take a maximum of one argument (see Section 3.6).

8. Exceptions or invalid actions must be displayed through popup dialogs (see Appendix).

## 5 Bonus (1 point)

During a copy, the keyword `(copy)` is added after the name of the file/folder. If there are several copies of a file, the following result will be obtained: `file(copy)(copy)(copy)(copy)`. Although this output is acceptable, it can be improved by a bonus feature. Instead, you can count the number of copies of a file:

[5] `http://www.montefiore.ulg.ac.be/~gain/courses/info0027/Project2/Code`

1. file
2. file(copy_1)
3. file(copy_2)
4. file(copy_3)
5. file(copy_4)

## 6 Report

You must then realize a report explaining your software architecture and the design patterns that you have used in your program. For each design pattern, you must describe why you have used it, and what are the advantages/drawbacks. You should use static diagrams to represent the interactions between the different classes. Your report must have **maximum** 5 pages. Note that static diagrams can be placed in appendix.

We would appreciate an estimation of the time you passed on the assignment, and what the main difficulties that you have encountered were.

## 7 Criteria

Concerning the criteria, we will primarily evaluate you on the understanding of design patterns. Nevertheless, we will also consider other criteria such as the structure of your code, the respect of the defined API and the correctness. Your report will also be evaluated. Note that no automatic test will be done on the submission platform.

## 8 Submission

Projects must be submitted before the Friday May 17, 11:59pm. After this time, a penalty will be applied to late submissions. This penalty is calculated as a deduction of $2^N - 1$ marks (where $N$ is the number of started days after the deadline).

Submissions will be made, as a `.zip` archive, on the submission system. At the root of the archive, you must place:

- the `Java` source files (**without** any folder).
- A max five-pages report (in English) explaining your architecture and the design patterns that you have considered.

**Bon travail...**

## Appendix to the Project Statement

The GUI library is provided on the exercises website[6], as a `graphics.jar` file to be included into your project. This GUI is able to render all the graphical elements of the file explorer in a window, to handle click events, and to detect when the user closes the window.

### Handling events

To handle graphical events, callbacks listeners have been defined:

```
void doubleClickEvent(Object selectedNode);
```

Method informing program using this library that the user has performed a double click on the node which is selected.

```
void createFileEvent(Object selectedNode);
```

Method informing program using this library that the user has clicked on the create file menu to create a new file which will be the child of the selected node.

```
void createFolderEvent(Object selectedNode);
```

Method informing program using this library that the user has clicked on the create folder menu to create a new folder which will be the child of the selected node.

```
void createAliasEvent(Object selectedNode);
```

Method informing program using this library that the user has clicked on the alias menu to make an alias of the selected node.

```
void createCopyEvent(Object selectedNode);
```

Method informing program using this library that the user has clicked on the copy menu to make a copy of the selected node.

```
void createArchiveEvent(Object selectedNode);
```

Method informing program using this library that the user has clicked on the create archive menu to compress the selected node.

[6] `http://www.montefiore.ulg.ac.be/~gain/courses/info0027.php#projects`

```
void eventExit();
```

Method informing program using this library that the graphical user interface has been closed.

### The `JTree` component

The hierarchical file system is managed thanks to a `JTree` component. This one provides the following interfaces/methods:

```
void setRootNode(Object newRootNode) throws
    RootAlreadySetException;
```

Method allowing to insert a new root to the tree. This method must be called **before** all the other ones.

```
boolean isRootNodeSelected();
```

Method allowing to check if the current selected node is the root of tree.

```
void refreshTree();
```

Method allowing to refresh the tree and to expand all its nodes. This method must be invoked in order for modifications to become visible to the user.

```
void addNodeToSelectedNode(Object newNodeObject) throws
    NoSelectedNodeException;
```

Method allowing to insert a new element/node to the current selected node.

```
Object addNodeToParentNode(Object newNodeObject) throws
    NoSelectedNodeException, NoParentNodeException;
```

Method allowing to insert a new element/node to the parents of the current selected node. It returns the parents node's user object. This method can be used, for example, for the alias feature.

```
void addNodeToLastInsertedNode(Object newNodeObject, int
    level) throws NoPreviousInsertedNodeException,
    LevelException;
```

Method allowing to insert a new element/node directly to a node that has just been inserted before. This method can be used, for example, for the copy feature in a recursive way. The `level` argument determines the level of the new inserted node(s)/element(s). The level must always be greater than zero, otherwise an exception is thrown.

### The `JTextArea` component

To manage the display, a `JTextArea` component has been used. This one is wrapped within the `TextAreaManager` class and must be retrieved before using its methods.

```
TextAreaManager getTextAreaManager();
```

Method allowing to get a `TextAreaManager` object. This object allows to display/clear a text in the `JTextArea` graphical component.

When you have retrieved the `TextAreaManager` object, you can directly work on it by using its two methods:

```
void appendText(String text);
```

Method allowing to add text to the `JTextArea` component.

```
void clearAllText();
```

Method allowing to clear all the text of the `JTextArea` component.

## Popup dialog box

In addition to the main frame, there exist also several types of popup dialogs:

1. The first type is related to the creation of files and folders.
2. The second one is related to the creation of an archive.
3. The last one concerns dialog box that must be used to handle exceptions or invalid actions.

#### File/Folder dialog box

```
String folderMenuDialog();
```

Method allowing to display the folder manager window. It displays the folder name input and it returns the name of the folder as a `String`. If the user has clicked on "cancel", it returns `null`.

```
String[] fileMenuDialog();
```

Method allowing to display the file manager window. It displays the filename and content inputs and it returns an array of `String` where the first element ([0]) represents the filename as a `String` and the second element ([1]) represents the file's content (as a `String` too). If the user has clicked on "cancel", it returns `null`.

**Archive dialog box**

```
String displayArchiveWindow1();
```

Method allowing to display the first window of the archive manager. It displays the archive name input and it returns the name of the archive as a `String`. If the user has clicked on "cancel", it returns `null`.

```
String displayArchiveWindow2();
```

Method allowing to display the second window of the archive manager. It displays the archive extension input and it returns the extension of the archive as a `String`. If the user has clicked on "cancel", it returns `null`.

```
int displayArchiveWindow3();
```

Method allowing to display the third window of the archive manager. It displays the archive compression input and it returns the compression level of the archive as an integer. If the user has clicked on "cancel", it returns -1.

**Info/Error dialog box**

```
void showPopup(String message);
```

Method allowing to show an information popup message where the argument represents the desired info message. This method is available if you want to print some information but it is not mandatory to use it.

```
void showPopupError(String messageError);
```

Method allowing to show an error popup message where the argument represents the desired error message.

## Exceptions

The methods above and the constructor of the `ExplorerSwingView` class can throw the following exceptions:

| **Exception** | **Context** |
|---|---|
| `RootAlreadySetException` | the user tries to insert a new root at the tree. |
| `NullHandlerException` | handler is not initialized. |
| `NoParentNodeException` | the user tries to insert a new element to a node that doesn't have parents. |
| `NoPreviousInsertedNodeException` | the user tries to insert a new element to a node that doesn't exist. |
| `NoSelectedNodeException` | the user tries to insert a new element without having selected a node. |
| `LevelException` | exception launched when the level of the inserted node is not a positive integer. |

Table 1: Exceptions within the graphics library

## Table for permissions

The following table shows the different limitations that apply on a specific object according to the feature.

| Feature/From | Root | Folder | File | Alias | Copy | Archive |
|---|---|---|---|---|---|---|
| create a file | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ |
| create a folder | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ |
| create a copy | ✗ | ✓* | ✓ | ✗ | ✓ | ✓ |
| create an alias | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ |
| create an archive | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ |

Table 2: Table for permissions

*copy the entire folder.