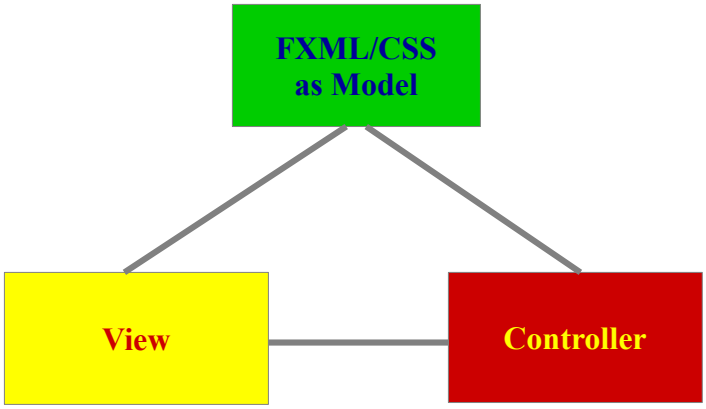
Model-View-Controller
MVC
with
SWING

I. Introduction

Desktop GUI development is becoming more and more complex. Development time is getting longer and longer. There are numerous proposed solutions, but the MVC approach is proving to be the most popular and widely accepted. With MVC, developers can improve prototyping work and shorten the software development process. JAVA offers developers JFX or JavaFX models based on the MVC framework. Unfortunately, with SWING, developers have to resort to third-party software. The most popular SWING MVC is Spring Boot. However, Spring Boot is very complex and requires intensive training. For example, the configuration alone is a very time-consuming process. For this reason, I tried to develop a new approach with SWING based on the JFX concept.

II. SWING MVC

Java SWING MVC leans on the JFX model



Instead of FXML/CSS, plain text files are used (without cascading style sheets) and the syntax is similar to XML syntax, case sensitive. Nevertheless, the JFX principles remain unchanged. Namely:

- The **model** is based on the SWING (javax) objects in plain text with XML-like syntax. Example: <panel>name=anyName file=componentFile.txt size=400,600 location=400,0 </panel>, where the <panel> element stands for JPanel.
- The **view** is the main application where the standard method **static void main(String[] args)** is implemented. Similar to JFX, main() "starts" the JFX application, while start() loads the model using FXMLLoader(). Here the loader is the **SWINGLoader()**.
- The **controller** is the part where dependencies are injected or actions and events are handled. .

That's the whole similarity between JFX and SWING MVC.

III. Java SWING MVC

I developed this Java SWING MVC to simplify the MVC development process and reduce its complexity, focusing on the active GUI SWING objects, while **JFrame**, **JPanel** and **JDialog** are the main frame (base) for the other SWING objects (J**Button**, **JLabel**, **JTextField**, etc.). **JFileChooser** or **JOptionPane**, are user-interactive and therefore so user-dependent that they belong to the implementation part of the controller and not to the model or view.

The **SWINGLoader** API

	Meaning
SWINGLoader(String model)	Constructor without Controller
SWINGLoader(byte[] model, String directory)	Constructor without Controller
SWINGLoader(String model, String controller)	Constructor with Controller
SWINGLoader(String model, String controller, String[] parameters)	Constructor with Controller and Parameters.
public Object load() throws Exception	Load, instantiate and return the model to the View
public ArrayList<String> nameList()	Return a List of all names of instantiated SWING objects (i.e. referenced names)
public HashMap<String, Object> getComponentMap()	Return a map of all instantiated SWING objects. The map is passed as the sole parameter to the Controller.

The controller as the second parameter (see 3rd and 4th constructors above) is instantiated by the SWING loader and must exist, even if it is empty or not implemented. By convention, it always has at least ONE parameter: **HashMap<String, Object>**. The third parameter of the controller is the passed string array. The string model is a text-file name in which the model descriptions and all dependencies can be found (abs. path. In the case of the rel. path, the current directory is used).

The **HashMap** contains all instantiated Java SWING objects (e.g. JButton, JTextArea, etc.). The SWING objects only need to be "injected" with actions (e.g. event listening) or all kinds of dependencies and attributes. Note that the names of the SWING objects must be unique.

Example:

```
public class ProtoController {
  /**
  Predefined Constructor for Controller
  @param map HashMap with String as keys (defined in the model) and Object as values (J Components)
  */
  public MyController(HashMap<String, Object> map) {
      ....
  }
// or with parameters
  public MyController(HashMap<String, Object> map, String[] parms) {
      ....
  }
```

Programming example:

The View: **MySWING.java**

```
import javax.swing.JFrame;
import javax.swing.UIManager;
// SWING Model-View-Controller
import joeapp.mvc.SWINGLoader;
// Joe Schwarz (C)
public class MySWING {
  public MySWING( ) throws Exception {
     // load the model MySwing.txt and MyController
     SWINGLoader sl = new SWINGLoader("MySwing.txt", "MyController");
     ((JFrame) ml.load()).setVisible(true);
   }
  public static void main(String... args) throws Exception {
     UIManager.setLookAndFeel("com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel");
     new MySWING( );  // start MVC
  }
}
```

The invocation is **java MySWING**

The **Model** which is in plain text (note: **<!--** starts a comment and always ends with **--!>**. Same XML syntax). The model: **mySwing.txt** (containing JTree description **tree.txt**)

```
<!-- JFrame for MySWING --!>
<frame>name=JFrame tittle=MySWING-MVC close=true size=350,400 location=200,20 </frame>
<tree>name=jTree nodes=tree.txt size=300,300 location=20,20 color=cyan</tree>
<label>name=Lab1 text="Your Selection:-", size=300,12 location=20,330</label>
<!-- end of JFrame --!>
```

The controller: **MyController.java**

```
import javax.swing.*;
import java.awt.event.*;
import java.util.HashMap;
// Joe Schwarz (C)
public class MyController {
 public MyController(HashMap<String, Object> map) {
  JTree tree = (JTree) map.get("jTree");
  JLabel lab = (JLabel) map.get("Lab1");
  tree.getSelectionModel().addTreeSelectionListener(e -> {
    lab.setText("Your Selection::"+e.getPath().toString());
  });
 }
}
```

The subfile **tree.txt** for JTree

```
<!-- + for sub-trees follow, - no subtree. --!>
<Earth:+:Fruits,Species,Trees/>
<Fruits:-:Orange,Banana,Pineapple,lemon/>
<Species:+:Carnivores, Herbivores/>
<Carnivores:+:Birds,Animals,Fishes/>
<Birds:-:Eagle,Hawk,Owl/>
<Animals:-:Lion,Tiger,Hyena/>
<Fishes:-:Shark,Ocra,Piranha/>
<Herbivores:-:Deer,Horse,Elephant/>
<Trees:-:Fir,Maple,sycamore/>
```

Note: The names provided (e.g. JFrame, jTree and Lab1) are mandatory as they are referenced in the controller to access the instantiated J-Components (see red marking). The path to the model file can be either relative (current or user directory) or absolute. This directory is then the root for ALL subsequent files found in the model (if they are all relative, of course). The displayed MySWING-MVC

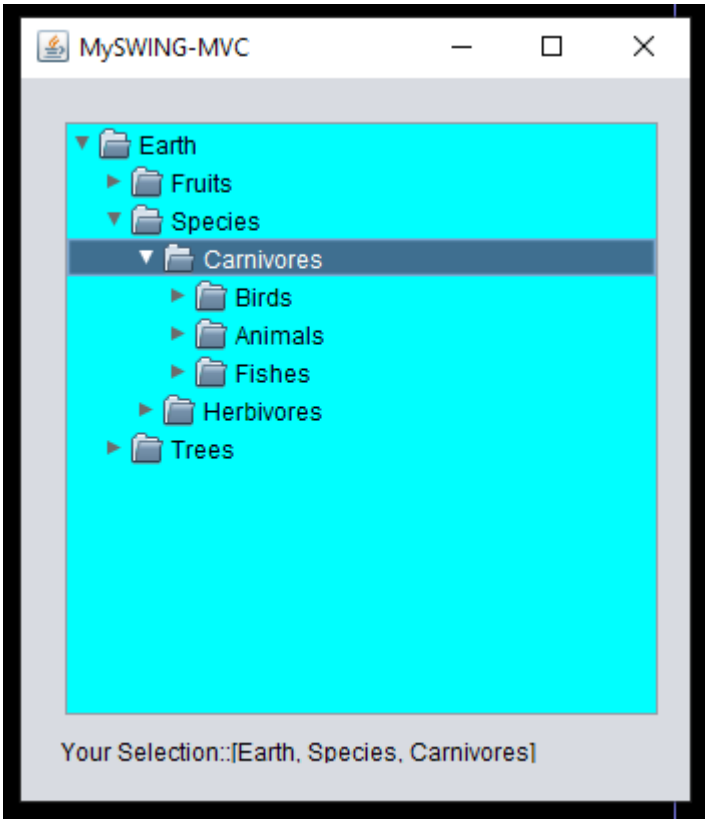


Fig.1

IV. The Modeling Syntax in plain text

The syntax is simple. There are only **a few** rules as following:

1. Comment starts with **<!--** and always ends with **--!>**
2. **<keyword>** element_1 …. element_n **</keyword>**
3. Case sensitive
4. continue in the next lines is allowed
5. JFrame and JDialog must start as the first line. Same for JPanel if it itself serves as the base.
6. Embedded J-components for another J-component or an item list must be declared in a supplementary file (same syntax). Example: See **nodes=tree.txt** above. Each line (plus continuation) must **start with** < and is **terminated with** />
7. No space between keyword and value (e.g. **name=anyName**).
8. Strings can be specified with/without quotes. However, strings with embedded spaces must be enclosed in quotes.
9. Chain of values must be separated by commas. No space between values.
10. The size and location of each J component must be specified by user. AWT tools (like BorderLayout, GridLayout, etc.) are NOT supported.
11. Customized SWING objects must contain an **empty constructor**.
12. A **plus** (+) indicates that a subfolder follows, a **minus** (-) means that items follow. See the tree.txt above

All keywords are derived from the SWING component names - **without the J** - and all in lowercase. Example: button for JButton, textfield for JTextField, etc.

V. Programming with SWING MVC

Model

The model is a simple text file (with suffix **.txt**). The first line must be the JFrame or JDialog line. If JPanel is used as a base, it must also be the first line. Other JComponents can be coded in any order. It is important that size and position do not overlap (see e.g. MySwing.txt on page 3). A straightforward coding algorithm. Example:

```
<!-- the viewing frame with embedded JDialog --!>
<frame> name=MyFrame title="MVC with JDialog" size=600,600 location=20,20
     close=true resize=false</frame>
<button>name=Start text="Start Dialog" size=100,100 location=230,230 color=yellow</button>
<dialog>name=MyDialog  load=dPanel.txt  size=500,500  location=0,0  bgimage=RicePaddy.jpg
</dialog>
<!-- end of frame --!>
```

View

Developing the view is relatively simple if you don't want to do any extra work before making the view VISIBLE (see for example: MySWING.java on page 2). The general structure is:

```
Import joeapp.mvc.SWINGLoader:
public class MyApp {
   public MyApp(String model, String controller) {  // always expecting 2 parameters
     SWINGLoader sl = new SWINGLoader(model, controller);
     JFrame jf = (JFrame) sl.load();
     … // do some extra works
     jf.setVisible(true);
   }
   public static void main(String... args) {
     … // your initialization works
   }
}
```

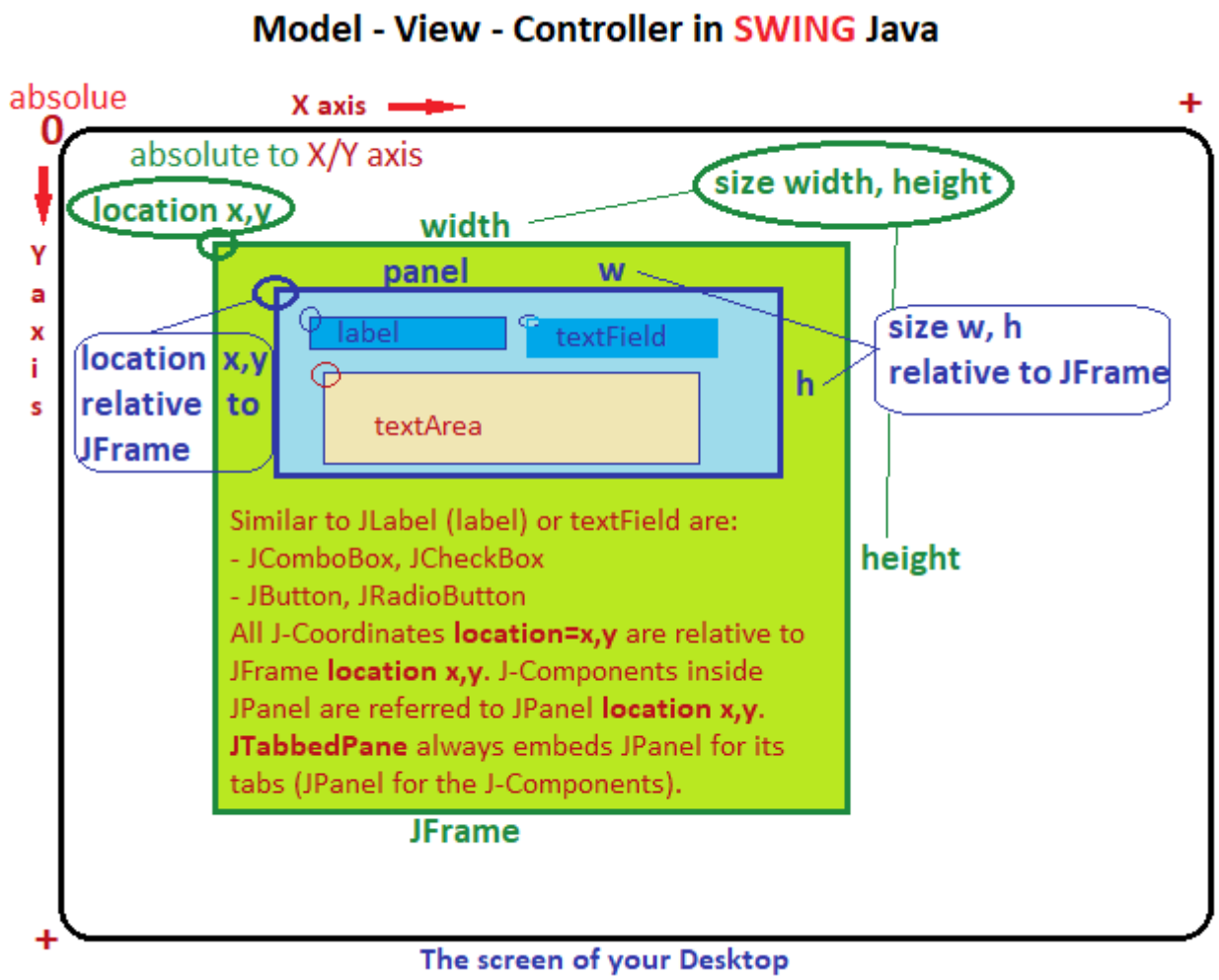
Controller

The controller always has ONE parameter: HashMap<String, Object>. The HashMap contains all instantiated J components specified in the model file. They are referenced by their declared name (see, for example, MyController.java on page 3). Inside the controller, "dependencies" are inserted (or implemented). Usually these are the event listeners. If you are an experienced developer, you can implement any fancy things you want to do here. Note: You need to access the J components from the map and cast them to the correct SWING objects you are working with. The access keys are the names declared in the model with the element: name=anyName. Example: Jbutton

```
JButton but = (JButton) map.get("Start_Button");  // fetch the JButton under the name Start_button
```

The general structure is:

```
Import java.util.HashMap;
// Joe Schwarz (C)
public class MyController {
   public MyController(HashMap<String, Object> map) {
      JXYZ xyz = (JXYZ) map.get("MyXYZ"); // e.g. JButton, JTextField, etc.
      … // insert dependency, e.g. addActionListener(e -> { … });
   }
   … // other methods or internal classes invoked/used by Constructor
}
```



The image shows you the coordinates of SWING objects within a view. JFrame and JDialog refer to the screen coordinates (absolute), while the other SWING coordinates refer to JFrame/JDialog (relative). The coordinates of JPanel's SWING children refer to JPanel. JFrame and JDialog can have SWING children directly (i.e. without JPanel). The layout of all JComponents is determined by their size and their position.

VI. SWING Elements

As mentioned before, SWING J component names are used for SWING MVC modeling elements – without the J prefix and in lowercase. Example: for JTextArea it is <textarea>. Each model element contains some attributes that are used by SWINGLoader to instantiate the corresponding J component. These attributes are divided into two parts:

- **Mandatory**: Mandatory attributes are obligatory and must be specified.
- **Optional**: Optional attributes are those that can be omitted.

Each chain of items in the supplement file must be embedded between < … />

Note: **Case sensitive**. File names can be without (relative) or with path (absolute). If the path is relative, the current user directory is used as ROOT for all subsequent files with relative names. Otherwise, the root directory is the path of the model file. Only icon files can be URLs.

Another URL case: see JEditorPane and JtextPane.

The following SWING MVC elements are implemented (note: active J components like JFileChooser, JOptionPane, etc. are interactive components that usually trigger a user action - hence they belong to the controller). Note that the 3 keywords name, size and location are mandatory.

JFrame

```
<frame>name=anyName tittle="anyTitle" size=width, height bgimage=imgFile location=x,y
          resize=false close=false</frame>
optional: bgimage, location (default 0,0 or upper left corner of screen) and resize,
          tittle, close (default: true), nolayout (default true)
```

Explanation:

- name=... is name to be accessed from HashMap by this name
- tittle=.....is the tittle of JFrame
- bgimage=... defines the background image of JFrame. It could be an URL.
- size=w,h set JFrame with the width=w and the height=h
- location=x,y positions JFrame at the coordinate x,y on the screen
- resize=true/false allows JFrame to be resized (true)
- close=true/false allows JFrame to be closed (true) at the click on the upper right corner of JFrame

JPanel

```
<panel>name=name size=w,h location=x,y</panel>
```

Explanation:

- name=... is name to be accessed from HashMap by this name
- size=w,h set JPanel with the width=w and the height=h
- location=x,y positions JPanel at the coordinate x,y relating to JFrame/JDialog

If JPanel must be embedded inside another J Components it must be declared as following and its included J Components are declared in an extra text file (or supplement file.)

```
<panel>name=anyName file=modelFile.txt size=w,h location=x,y</panel>
```

- file=... is the file that contains the J Components for THIS JPanel

Example:

```
<frame> name=Embedded title="MVC with JFrame -embedded JPanel-" size=800,600 location=20,20
      bgimage=RicePaddy.jpg close=true resize=false</frame>
<panel>name=EmbeddedJPanel file=panel.txt size=400,600 location=400,0</panel>
```

The file panel.txt:

```
<panel> name=pan size=400,500 location=200,0</panel>
<!-- label and button --!>
<label>name=Lab_1 text="Please click", size=100,12 location=10,25</label>
<button>name=But1 text="START" size=100,50 location=90,5 color=yellow</button>
<!-- label and textfield --!>
<label>name=Lab_2 text="Name:", size=100,12 location=10,75</label>
```

```
<textfield>name=TxtField text="Your Name" size=250,30 location=90,65 color=yellow</textfield>
```

JDialog

```
<dialog>name=anyName tittle="anyTitle" load=panel.txt size=w,h location=x,y close=false</dialog>
optional: load (panelModel file), tittle, close (default: true)
```

Explanation:

- name=... is name to be accessed from HashMap by this name
- tittle=.....is the tittle of JDialog
- load=... defines loading JPanel for this JDialog
- size=w,h set JDialog with the width=w and the height=h
- location=x,y positions JDialog at the coordinate x,y on the screen
- close=true/false allows JDialog to be closed (true) at the click on the upper right corner of JDialog

JTabbedPane

```
<tabbedpane>name=name tabs=panel_1.txt,...,panel_X.txt tabtext="text_1",...,"text_X"
        size=w,h location=x,y</tabbedpane>
optional: tabs (where panel_X.txt is the loading JPanel for tab_X), tabtext (header for tab_X)
```

Explanation:

- name=... is name to be accessed from HashMap by this name
- tabs=.....gives the list of embedded JPanel for each Tab (in sequence 1,2,...x). For the panel_X.txt: see **JPanel.**
- tabtext=... a list of Tab names according to the **tabs** attribute
- size=w,h set JTabbedPane with the width=w and the height=h
- location=x,y positions JTabbedPane at the coordinate x,y relating to Jframe/JDialog

JButton

```
<button>name=anyName text=anyText size=w,h location=x,y icon=imageFile font=fontName fontSize=n
 fontType=anyType color=anyColor textColor=anyColor</button>
optional: color, textColor, font, fontSize, fontType
```

Explanation:

- name=... is name to be accessed from HashMap by this name
- text=.....is the name on JButton
- icon=... the Icon image file for JButton
- color=... is the background color (**java.awt.Color**) Example: color=blue.
- textColor=... is the Text Color (see color)
- font=... is the font. Example: font=Verdana
- fontSize=... is the font size (a number). Example: fontSize=13
- fontType=... is the font type. Example: fontType=bold
- size=w,h set JButton with the width=w and the height=h
- location=x,y positions JButton at the coordinate x,y relating to JFrame/JPanel/JDialog

JRadioButton

```
<radiobutton>name=anyName text=anyText size=w,h location=x,y  font=fontName fontSize=n
 fontType=anyType color=anyColor textColor=anyColor icon=imageFile color=anyColor</radiobutton>
optional: color, textColor, font, fontSize, fontType
```

Explanation: see JButton

JToggleButton

```
<togglebutton>name=anyName text="anyText" size=w,h location=x,y  font=fontName fontSize=n
 fontType=anyType color=anyColor textColor=anyColor icon=imageFile color=anyColor</button>
optional: color, textColor, font, fontSize, fontType
```

Explanation: see JButton

JLabel

```
<label>name=anyName text="anyText" size=w,h location=x,y icon=fName font=fontName fontSize=n
 fontType=anyType color=anyColor textColor=anyColor opaque=true color=anyColor</label>
optional: opaque (default false), color, textColor, font, fontSize, fontType
```

Explanation:

- name=... is name to be accessed from HashMap by this name
- text=.....is the JLabel text
- icon=... the Icon image file for JLabel
- color=... is the color as a text given by **java.awt.Color**. Example: color=blue.
- opaque=true/false overlaps the background color
- size=w,h set JLabel with the width=w and the height=h
- location=x,y positions JLabel at the coordinate x,y relating to JFrame/JPanel/JDialog

JTextField

```
<textfield>name=anyName text="anyText" size=w,h location=x,y column=n color=anyColor font=fontName
 fontSize=n fontType=anyType textColor=anyColor</textfield>
optional: text, color, textColor, font, fontSize, fontType
```

Explanation:

- name=... is name to be accessed from HashMap by this name
- text=.....is the JTextField text
- color=... is the color as a text given by **java.awt.Color**. Example: color=blue.
- column=... defines the number of columns
- size=w,h set JTextField with the width=w and the height=h
- location=x,y positions JTextField at the coordinate x,y relating to Jframe/JPanel/JDialog

JFormattedTextField

```
<formattedtextfield>name=name text="anyText" size=w,h location=x,y  font=fontName
  fontSize=n fontType=anyType textColor=anyColor column=n color=anyColor</formattedtextfield>
optional: text, color, textColor, font, fontSize, fontType
```

Explanation: see JTextField

JPasswordField

```
<passwordfield>name=name text="anyText" size=w,h location=x,y
                    column=n color=anyColor</passwordfield>
optional: text, color
```

Explanation: see JTextField

JTextArea

```
<textarea>name=anyName text="anyText" size=w,h location=x,y edit=false font=fontName
  fontSize=n fontType=anyType textColor=anyColor column=n color=anyColor row=n
  column=n color=anyColor</textarea>
optional: text, color, row, column, edit (default true), textColor, font, fontSize, fontType
```

Explanation:

- name=... is name to be accessed from HashMap by this name
- text=.....is the JTextArea text
- color=... is the color as a text given by **java.awt.Color**. Example: color=blue.
- column=... gives the number of columns
- rows=... gives the number of rows
- edit=true/false is either editable (true) or not.
- size=w,h set JTextArea with the width=w and the height=h
- location=x,y positions JTextArea at the coordinate x,y relating to JFrame/JPanel/JDialog

JTextPane and JEditorPane

```
<textpane>name=anyName content=plain/html size=w,h location=x,y text=text.txt font=fontName
  fontSize=n fontType=anyType textColor=anyColor</textpane>
<editorpane>name=anyName content=plain/html size=w,h location=x,y text=text.txt font=fontName
  fontSize=n fontType=anyType textColor=anyColor</editorpane>
optinal: content (support: plain or html), text (the file contains the content) or an URL, textColor, font, fontSize,
         fontType
```

Explanation:

- name=... is name to be accessed from HashMap by this name
- text=.....is the content text for JTextPane or JEditorPane. If it's a web **URL** the content of this site will be downloaded into the pane
- content=... defines the content format. Either **plain** or **html**
- size=w,h set JTextPane or JEditorPane with the width=w and the height=h
- location=x,y positions JTextPane/JEditorPane at the coordinate x,y relating to JFrame/JPanel/JDialog

JList

```
<list>name="name" items="string_1",..,"string_x" color="color" size=w.h location=x,y</list>
<list>name="name" items=file.txt color="color" size=w.h location=x,y</list>
optional: items, color
note: only String or icon-file (.gif, .png, .jpg) as items. Other things must be set by controller with add(item)
```

Explanation:

- name=... is name to be accessed from HashMap by this name
- items=.....is the item-list or a file.txt that contains the list
- color=... is the color as a text given by **java.awt.Color**. Example: color=blue.
- size=w,h set JList with the width=w and the height=h
- location=x,y positions JList at the coordinate x,y relating to JFrame/JPanel/JDialog

JComboBox

```
<combobox>name=anyName items="string1",..,"stringX" color=anyColor size=w.h location=x,y
 font=fontName fontSize=n fontType=anyType textColor=anyColor </combobox>
optional: items, color, textColor, font, fontSize, fontType
only String or icon-file (.gif, .png, .jpg) as items. Other things must be set by controller with add(item)
```

Explanation:

- name=... is name to be accessed from HashMap by this name
- items=.....is the list of items of a **file.txt** containing the items
- color=... is the color as a text given by **java.awt.Color**. Example: color=blue.
- size=w,h set JComboBox with the width=w and the height=h
- location=x,y positions JComboBox at the coordinate x,y relating to JFrame/JPanel/ JDialog

JCheckBox

```
<checkbox>name=anyName text="text" icon=imagefile selicon=imagefile disicon=imagefile
            selected=true color=anyColor size=w,h location=x,y</checkbox>
optional: color, selected, selicon, disicon
```

Explanation:

- name=... is name to be accessed from HashMap by this name
- text=... the text for JCheckBox
- icon=.....the icon image file
- selicon=... icon if selected
- disicon=... icon if disselected
- selected=true/false (default: false) preset selection
- color=... is the color as a text given by **java.awt.Color**. Example: color=blue.
- size=w,h set JCheckBox with the width=w and the height=h
- location=x,y positions JCheckBox at the coordinate x,y relating to JFrame/JPanel/ JDialog

JprogressBar and **JSlider**

```
<slider>name=anyName size=w,h location=x,y minmax=a,b orient=vertical/horizontal<slider>
<progressbar>name=anyName size=w,h location=x,y minmax=a,b orient=vertical/horizontal</progressbar>
optional minmax, orient (default horizontal)
```

Explanation:

- name=... is name to be accessed from HashMap by this name
- minmax=... the min and max value for JProgressBar

- orient=vertical/horizontal
- size=w,h set JProgressBar with the width=w and the height=h
- location=x,y positions JProgressBar at the coordinate x,y relating to JFrame/JPanel/ JDialog

JTable

```
<table>name=anyName size=w,h location=x,y roco=n,m color=anyColor table=table.txt</table>
optional roco, table, color
Note: Vector and TableModel are NOT supported. Only String for Object. Format for table.txt
    <col_1,..,col_x/>
    <row_1_1,..,row_1_x/>
    ...
    <row_x_1,..,row_x_x/>
Where col_1 ... col_x must be the first line
```

Explanation:

- name=... is name to be accessed from HashMap by this name
- roco=... gives the number of rows and columns
- table=.....the file that contains the table
- color=... is the color as a text given by **java.awt.Color**. Example: color=blue.
- size=w,h set JTable with the width=w and the height=h
- location=x,y positions JTable at the coordinate x,y relating to JFrame/JPanel/JDialog

JTree

```
<tree>name=anyName nodes=tree.txt size=w.h location=x,y color=anyColor</tree>
optional: items, color
note: only list of String as nodes or a tree file containing the nodes (currently NO icon)
    format:
    <Root:?:node_1, ... ,node_n/>
    <node_1:?X:item_1 ... node_1:item_X/>
    ...
    <node_n:?:item_1 ... node_n:item_X/>
Where ? stands for + (subtrees follow), - (no substree)
```

Explanation:

- name=... is name to be accessed from HashMap by this name
- nodes=.....the file that contains the nodes
- color=... is the color as a text given by **java.awt.Color**. Example: color=blue.
- size=w,h set JTree with the width=w and the height=h
- location=x,y positions JTree at the coordinate x,y relating to JFrame/JPanel/JDialog

JFXPanel (connection to JFX components)

```
<fxpanel>name=anyName size=w,h location=x,y </fxpanel>
```

Explanation:

- name=... is name to be accessed from HashMap by this name
- size=w,h set JFXPanel with the width=w and the height=h

- location=x,y positions JFXPanel at the coordinate x,y relating to Jframe/JPanel/JDialog

JMenuBar

```
<menubar>name=anyName size=w,h location=x,y file=menu.txt</menubar>
obligatory: name, size, location, file
optional: -
note: menu.txt format
    menu:anyName,keyEvent,item:anyName,keyEvent,item:anyName,keyEvent ...
    ...
    menu:anyName,keyEvent,checkbox:anyName,keyEvent (e.g. VK_F)
Continue in the next line with a + after the last item (separated by a blank)
```

Explanation:

- name=... is name to be accessed from HashMap by this name
- file=.....the file that contains the menu
- size=w,h set JMenuBar with the width=w and the height=h
- location=x,y positions JMenuBar at the coordinate x,y relating to JFrame/JPanel/JDialog

JPopupMenu

```
<popupmenu>name=anyName text=anyText size=w,h location=x,y file=items.txt</popupmenu>
obligatory: name, size, location, file
optional: text
Note: items.txt contains the menu items either in text strings or in image-file
    example: "Item_1",image.jpg,...
```

Explanation:

- name=... is name to be accessed from HashMap by this name
- file=...a file contains the list of items
- text=.....the text for JPopupMenu
- size=w,h set JPopupMenu with the width=w and the height=h
- location=x,y positions JPopupMenu at the coordinate x,y relating to JFrame/JPanel/JDialog

JToolBar

```
<toolbar>name=anyName text=anyText orientation size=w,h location=x,y </toolbar>
optinal: text, orientation (HORIZONTAL/VERTICAL)
```

Explanation:

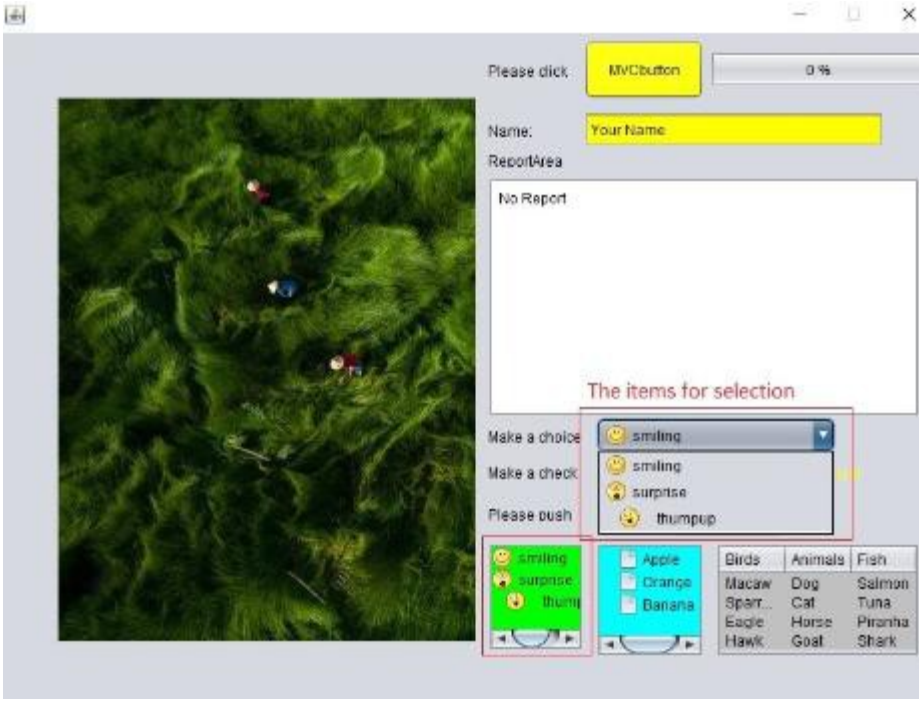
- name=... is name to be accessed from HashMap by this name
- text=.....the text for JToolBar
- size=w,h set JToolBar with the width=w and the height=h
- location=x,y positions JToolBar at the coordinate x,y relating to JFrame/JPanel/JDialog

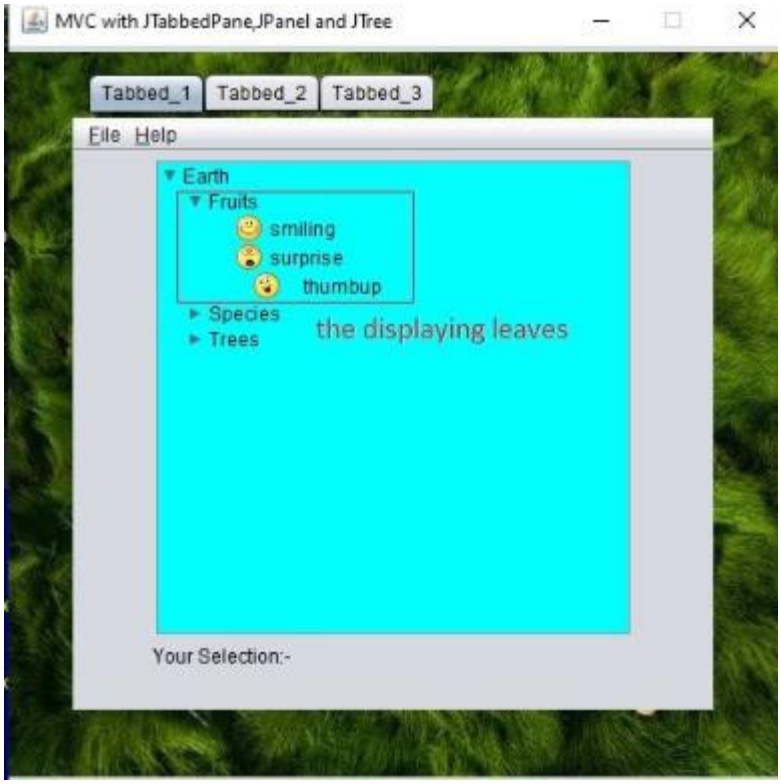
Note: JToolBar is normally populated with different GUI items such as JButton, JComboBox, etc. Because of numerous possibilities JToolBar should be populated in the controller.

Some hints for **JComboBox, JList, J Table** and **JTree**: if icons (and text) are used as items the **entry format** is as following:

text@imageFile or @imageFile if text is unnecessary text@URL the URL could be a link to an image on the WEB (Note: only for Image).
Example: - For JTree: The node (here: Fruits) stands before the item list. The mnemonic + (plus) for the subnodes that follow and - (minus) indicates that items follow <Fruits:-:smiling@smiling.gif,surprise@surprise.gif,thumbup@thumbUp.gif/> - For JComboBox or JList: <smiling@smiling.gif,surprise@surprise.gif,thumpup@thumbUp.gif/>

And the displaying results:





VII. SWINGLoader

SWINGLoader works similarly to **FXMLLoader**. Using the descriptions, SWINGLoader can create and instantiate the specified J components and store their references in a HashMap. Using their unique name (or ID in FXML), J components can be easily accessed and processed. And this is how a controller works. Instead of an annotation like **@FXML** for each component or each method, the SWING controller can directly access each J component by its name using **HashMap.get(name)**. And this makes the correction work easier during the development phase. Developers can focus either on the model (in plain text) or on the controller. Adding some new listeners to a J component can be done immediately inside the controller.

SWINGLoader is a parser and compiler in one. It reads the model.txt file, parses the content and checks it for syntactical correctness before compiling the pseudocode into Java code. Exceptions are only thrown at runtime. For simplicity, the generic Exception class is used to represent the problem that occurred as clearly as possible. Example: Reading a list of items from the JMenuBar.

```
String file = SWING.getItem("file=", s);
if (file == null) throw new Exception("Expected file=... @line:"+s);
String[] M = getArray(SWING.getFile(file));
…
public static ArrayList<String> getFile(String file) throws Exception {
```

```
    File fm = new File(file);
    if (!fm.exists()) throw new Exception("Invalid file: "+file);
    String mod[] = (new String(Files.readAllBytes(fm.toPath()))).split(System.lineSeparator());
     ...
  }
```

The SWING MVC jar file (**joemvc.jar**) is about 46 KB in size. It is relatively small, clean and fast. All you need to do is include joemvc.jar in your (global) CLASSPATH.

Note: If you need to include some JFX components in your SWING app, JFXPanel in your controller must be populated with JFX components. SWINGLoader provides an instantiated JFXPanel. Example:

```
public class JFXController {
  public JFXController(HashMap<String, Object> map) {
    JButton but = (JButton) map.get("But");
    but.addActionListener(e -> {
      JFXPanel jfx = (JFXPanel) map.get("jfx");    // retrieve JFXPanel
      Group root = new Group();
      .... // JFX components
      Scene scene = new Scene(root, 500, 500);
      jfx.setScene(scene);
    });
  }
}
```

The SWING-MVC Package:

API	Comment
SWINGLoader	The main SWINGLoader
SWING	All tools used by SWINGLoader
ICell	Icon Cell used for JTree, JList, JComboBox, JTable
TreeRenderer	The TreeCellRenderer for JTree and JTable
ComboRenderer	The JComboBox Renderer
ListRenderer	The JList Renderer
TableRenderer	The JTableRenderer

SWING API -the tools

static Object[] asObject(String s, String dir, Class<?> cls)	Object array from the Model Jcomponent
static String[] getArray(String pat,String s,String dir,Class<?> cls)	String array of the given pattern from Class cls with the root dir
static int[] getBounds(String s)	get the values w/h and x/y
static Color getColor(String color)	get JAVA color the given string color

static ArrayList<String> purify(String cont)	purify the content of model
static ArrayList<String> getFile(String file, Class<?> cls)	get the file content
static ImageIcon getIcon(String icon, String dir)	create an ImageIcon
static String getItem(String pat, String s)	get the value of the given pattern
static int[] getValues(String pat, String s)	get the twin values (of location or size)
static boolean isImage(String img)	true if the file is an image file
static int mnemonic(String mne)	get JAVA Key mnemonic
static byte[] getJARFile(String fName, Class<?> cls)	read the content of a JARred file from the JARred class cls
static Font getFont(String s)	Check for the Font and return an instantiated font if it is specified

SWINGLoader API

SWINGLoader(String model)	Constructor without Controller
SWINGLoader(String model, String controller)	Constructor with Controller
SWINGLoader(String model, String controller, String[] parameters)	Constructor with Controller and Parameters.
SWINGLoader(byte[] model, String directory)	Constructor without Controller
SWINGLoader(String model, Class<?> class)	Constructor with Caller or Controller class
Object load()	load the model and instantiate the given J components
ArrayList<String> nameList()	NameList of J Components in the model

Example: SWINGLoader with parameters:

```
SWINGLoader sloader = new SWINGLoader("my_model.txt", "MyController", new String[] { "Hello" });
```

Renderer API (implicitly used by SWINGLoader to generate the Modeling SWING J components when images are involved as symbols. Users should not tamper with the renderers. See the next page for details)

ComboRenderer	Implementation of DefaultListCellRenderer
ListRenderer	See ComboRendere
TableRenderer	Implementation of.DefaultTableCellRenderer
TreeRenderer	Implementation of TreeCellRenderer

Example: JComboBox. Only the content elements define the elements to be displayed (text or symbols). Note: The additional file **icons.txt** is for clarity.

The model:

```
<combobox>name=cBox items="Apple","Orange","Banana","Guava" size=200,30 location=100,320</combobox>
```


```
<combobox>name=cBox items=icons.txt size=200,30 location=100,320</combobox>

The icons.txt:
<smiling@smiling.gif,surprise@surprise.gif,thumpup@thumbUp.gif/>
```

Since you do not work directly with the renderers and their ICells, your controller accesses the ICell depending on the modeling. For example: by querying the selected object.

```
JComboBox<String> cbx = (JComboBox) map.get("cBox");
cbx.addActionListener(e -> {
  Object obj = cbx.getSelectedItem();
  if (obj instanceof ICell) System.out.println("\nYou've selected:"+((ICell)obj).getText());
  else System.out.println("\nYou've selected:"+(String)obj);
});
```

Note: the element "text@imageFile" could be "text@https://image.com/smiling.gif". However, you should not use this case often due to the long "download time". In other words, it slows down your app. It is better to use the local image files. Similar to JComboBox, it is the same with JTree, JTable and JList.

When you work with renderer, your elements are wrapped by ICell and when an element is selected, its values can be retrieved by ICell getters. All renderers have ONE method, getICell(), and this method returns the corresponding ICell of the selected cell.

ICell API	Comment
public ICell(String text@imageFile, String root)	Constructor with the string of format text@imageFile

	and the ROOT directory
public JLabel getCell()	Return the JLabel containing the Text and the Icon
public ImageIcon getIcon()	Return the ImageIcon of imageFile
public String getFile()	Return the given imageFile (without @)
public String getText()	Return the given text
public void setIcon(ImageIcon icon)	Set a new Icon
public void setText(String text)	Set a new text

Example:

```
JList<String> jlst = (JList) map.get("jlist");
jlst.addListSelectionListener(e -> {
  Object obj = jlst.getSelectedValue();
  if (obj instanceof ICell) System.out.println("\nSelected from JList::"+((ICell)obj).getText());
  else  System.out.println("\nSelected from JList::"+(String)obj);
});
```

```
JTree jtree = (JTree) map.get("jTree");
jtree.getSelectionModel().addTreeSelectionListener(e -> {
  DefaultMutableTreeNode dn = (DefaultMutableTreeNode)e.getPath().getLastPathComponent();
  ICell ic = (ICell) dn.getUserObject();
  System.out.println("From JTree:"+ic.getText());
});
```

Customized SWING objects

For simplicity, SWINGLoader only instantiates customized SWING objects with an empty constructor (without parameters). If parameters are needed, they should be handled with setters. Example:

Original:

```
public class SysMonSWING extends JPanel implements Runnable {
  public SysMonSWING(int width, int height) {
    this.width = width;
    this.height = height;
    setPreferredSize(new Dimension(width, height));
    ax = new ArrayList<Integer>(20);
    mem = new ArrayList<Integer>(20);
    pTi = new ArrayList<Integer>(20);
    pLo = new ArrayList<Integer>(20);
    cpu = new ArrayList<Integer>(20);
    swp = new ArrayList<Integer>(20);
    vir = new ArrayList<Integer>(20);
  }
  ...
```

Work-Around

```
public class SysMonSWING extends JPanel implements Runnable {
  public SysMonSWING( ) {
    super();
  };
  public SysMonSWING(int width, int height) {
    super();
    setLayout(width, height);
  }
  public void setLayout(int width, int height) {
    this.width = width;
    this.height = height;
    setPreferredSize(new Dimension(width, height));
    // create the lists
    ax = new ArrayList<Integer>(20);
    mem = new ArrayList<Integer>(20);
    pTi = new ArrayList<Integer>(20);
    pLo = new ArrayList<Integer>(20);
    cpu = new ArrayList<Integer>(20);
    swp = new ArrayList<Integer>(20);
    vir = new ArrayList<Integer>(20);
  }
  ...
```

And the declaration in the model for this customized JPanel is as usual:

```
<SysMonSWING> name=sysmon size=500,500 location=80,7 </SysMonSWING>
```

Problems and Exceptions

All SWINGLoader exceptions are thrown by JAVA. The exceptions are usually:

Misspelling error in the file name of the model or its subfiles. Example: genericPane.txt: missing l in genericPanel.txt

```
<panel>name=EmbeddedJPanel file=genericPane.txt size=400,600 location=400,0</panel>

C:\JoeApp\mvc\example>java GenericView
Exception in thread "main" java.lang.NullPointerException
        at joeapp.mvc.SWING.readJAR(Unknown Source)
        ....
```

Mistyping the keywords. Example: <fram> unstead of <frame> (missing e of frame)

```
<fram> name=Embedded title="MVC with JFrame -embedded JPanel-" size=800,600 location=20,20
      bgimage=RicePaddy.jpg close=true resize=false</frame>

C:\JoeApp\mvc\example>java GenericView
Exception in thread "main" java.lang.Exception: Model must start with <frame> or <panel> or <dialog>
        at joeapp.mvc.SWINGLoader.load(Unknown Source)
        ....
```

Missing a mandatory element (syntax). Example: name=anyName

```
<panel> file=genericPanel.txt size=400,600 location=400,0</panel>
```

```
C:\JoeApp\mvc\example>java GenericView
Exception in thread "main" java.lang.Exception: Missing name=... @line:<panel> file=genericPanel.txt
size=400,600 location=400,0 </panel>
        at joeapp.mvc.SWING.validate(Unknown Source)
        ....
```

Syntax problems. Example: missing the closed element </table>

```
<table>name=jTable size=170,95 location=200,425 table=tableX.txt color=lightgray
```

```
C:\JoeApp\mvc\example>java GenericView
Exception in thread "main" java.lang.Exception: Missing "</table>" @line:<table>name=jTable size=170,95
location=200,425 table=tableX.txt color=lightgray
        at joeapp.mvc.SWING.validate(Unknown Source)
        ....
```

The more coding problems there are, the more obscure the exceptions are. As you can see in the first example (wrong typo on the file name), the exception is a NullPointerException. It is difficult to be precise with such problems because such an exception could be a non-instantiated object (e.g. String) or something else. Also, SWINGLoader considers the root directory to be the directory where the model is located. Two scenarios: The root directory is

- either the directory of the model file when it's given with the **absolute** path,
- or the working directory (**user.dir)** when it's a **relative** path. NullPointerExeption is here the biggest problem.

Missing root in case of embedded SWINGLoader. Example: JFrame invokes JDialog and the JDialog model is with **relative** path. And the Exception message is, of course, misleading

```
JButton start  = (JButton) map.get("Start");
start.addActionListener(e -> {
  try {
    SWINGLoader ml = new SWINGLoader("jdialog.txt");
    JDialog jd = (JDialog) ml.load();
    ...
});
```

```
C:\JoeApp\mvc\example>java DirectMVC
java.lang.Exception: Model must start with <frame> or <panel> or <dialog>.
Found:jdialog.txt
        at joeapp.mvc.SWINGLoader.load(Unknown Source)
        ....
```

The confusing message is due to the use of the FIRST SWINGLoader constructor, which is used to execute the embedded SWINGLoader. This means: its root directory is independent of the root directory of the calling SWINGLoader. This works if the JDialog file is specified as an absolute path. Example

```
JButton start  = (JButton) map.get("Start");
start.addActionListener(e -> {
  try {
    SWINGLoader ml = new SWINGLoader("c:/JoeApp/mvc/example/model/jdialog.txt");
```

```
    JDialog jd = (JDialog) ml.load();
    ...
});
```

More: Compare the examples between DialogMVC and DirectMVC (the last two examples) to see how to work with the embedded SWINGLoader.

Incorrect referenced name: The NullPointerException can occur if the referenced name is misspelled or unknown (returned as NULL by HashMap). Example (declared as "Start" in the model)

```
JButton start  = (JButton) map.get("Button");
start.addActionListener(e -> {
    ...
});
```

```
C:\JoeApp\mvc\example>java DirectMVC
java.lang.NullPointerException
        at DirectController.<init>(Unknown Source)
        ....
```

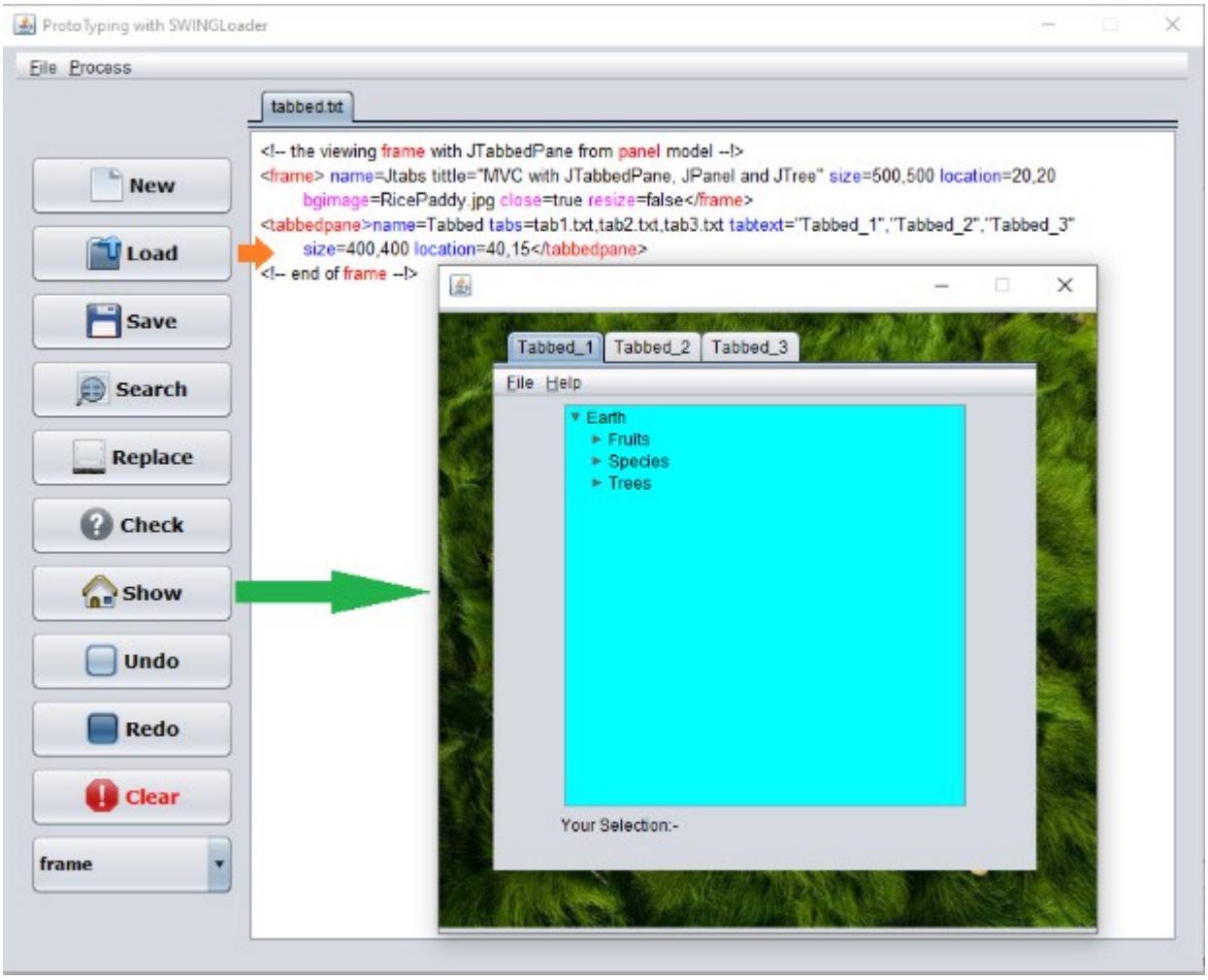
Same NullPointerException if the starting model is unknown. Example:

```
C:\JoeApp\mvc\example>java GenericView jfx.txt JFXController
Exception in thread "main" java.lang.NullPointerException
        at joeapp.mvc.SWING.readJAR(Unknown Source)
```

Correct is: c:\JoeApp\mvc\example\model\jfx.txt

VIII. Prototyping Editor

This simple **SWING MVC ProtoTyping editor** allows you to interactively design and create a model. The keywords are showed in **red**, mandatory words in **blue** and optional words in **pink**. Invocation: **java -jar joemvc.jar**. The appearance of the user interface:



The SWING MVC Editor allows users to prototype and display a model without having to create a corresponding controller first. With the JCombobox, users only need to select the SWING objects they want. And then they just need to customize the SWING objects according to their planned layout. Example:

```
<button>name=anyName text=anyText size=w,h location=x,y icon=imageFile color=anyColor
textColor=anyColor font=Arial fontType=PLAIN fontSize=s</button>
```

```
<button>name=new text=New size=150,45 location=20,80 font=Verdana
    fontType=BOLD fontSize=14 icon=new.png</button>
```

And the result is the New button with **Verdana font, bold, size 14** along with the **new.png** icon (see image above). As you can see on the edit panel (here: JTextPane), the keywords (e.g. frame, tabbedpane) are red, the mandatory words (name, size, location) are blue and the optional words are magenta (text, icon, etc.). The buttons are self-explanatory. If the loaded

model (Load button) is valid (Check button), it can be “displayed” (Show button) without the need to create an empty controller (see inner image).
The **Prototyping** models:

The main model _prototype_.txt

```
<frame> name=frame title="ProtoTyping with SWINGLoader" size=900,720 location=10,10
      close=true resize=false</frame>
<menubar>name=menubar size=862,20 location=10,5 file=_menu_.txt</menubar>
<button>name=new text=New size=150,45 location=20,80 font=Verdana
      fontType=BOLD fontSize=14 icon=new.png</button>
<button>name=load text=Load size=150,45 location=20,130 font=Verdana
      fontType=BOLD fontSize=14 icon=load.png</button>
<button>name=save text=Save size=150,45 location=20,180 font=Verdana
      fontType=BOLD fontSize=14 icon=save.png</button>
<button>name=search text=Search size=150,45 location=20,230 font=Verdana
      fontType=BOLD fontSize=14 icon=search.png</button>
<button>name=replace text=Replace size=150,45 location=20,280 font=Verdana
      fontType=BOLD fontSize=14 icon=replace.png</button>
<button>name=check text=Check size=150,45 location=20,330 font=Verdana
      fontType=BOLD fontSize=14 icon=check.png</button>
<button>name=show text=Show size=150,45 location=20,380 font=Verdana
      fontType=BOLD fontSize=14 icon=show.png</button>
<button>name=undo text=Undo size=150,45 location=20,430 font=Verdana
      fontType=BOLD fontSize=14 icon=undo.png</button>
<button>name=redo text=Redo size=150,45 location=20,480 font=Verdana
      fontType=BOLD fontSize=14 icon=redo.png</button>
<button>name=clear text=Clear size=150,45 location=20,530 textColor=red
      font=Verdana fontType=BOLD fontSize=14 icon=clear.png</button>
<combobox>name=cBox items=_protolist_.txt size=150,45 location=20,580
      font=Verdana fontType=BOLD fontSize=12</combobox>
<tabbedpane>name=tabbed size=685,630 location=180,30 tabs=_tab1_.txt tabtext="New_Model"
      color=gray</tabbedpane>
```

The _tabe_.txt for the Tab

```
<textpane>name=area color=gray size=685,600 location=0,0</textpane>
```

The _menu_.txt for the JMenuBar

```
<menu:New,VK_N,item:Load,VK_L,item:Save,VK_S/>
<menu:Process,VK_P,item:Find,VK_F,item:Replace,VK_R/>
```

The _protolist_.txt for the JComboBox

```
<frame,panel,dialog,button,radiobutton,togglebutton,label,
combobox,checkbox,slider,progressbar,table,tree,fxpanel,textfield,
formattedtextfield,passwordfield,textarea,textpane,editorpane,
tabbedpane,list,toolbar/>
```

IX.Some examples

Three following examples show you how SWING MVC works. First of all: a generic View.

GenericView.java

```
import javax.swing.*;
// SWING MVC
import joeapp.mvc.SWINGLoader;
// Joe Schwarz (C)
public class GenericView {
  public GenericView(String[] p) throws Exception {
    SWINGLoader ml = new SWINGLoader(p[0], p[1]);
    ((JFrame) ml.load()).setVisible(true);
  }
  private static String[] parms;
  public static void main(String... a)  throws Exception {
    if (a.length == 2) parms = a;
    else {
      parms = new String[2];
      parms[0] = "generic.txt";
      parms[1] = "GenericController";
    }
    UIManager.setLookAndFeel("com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel");
    new GenericView(parms);
  }
}
```

The invocation is simple: java GenericView model.txt controllerName

The model for ALL available J Components: **generic.txt**

```
<frame> name=Embedded title="MVC with JFrame -embedded JPanel-" size=800,600 location=20,20
      bgimage=RicePaddy.jpg close=true resize=false</frame>
<panel>name=EmbeddedJPanel file=genericPanel.txt size=400,600 location=400,0</panel>
```

The **genericPanel.txt**

```
<panel> name=pan size=500,600 location=200,0</panel>
<label>name=Lab1 text="Please click", size=100,12 location=10,25</label>
<button>name=But1 text="MVCbutton" size=100,50 location=90,5 color=yellow</button>
<progressbar>name=pBar size=180,30 location=195,15 minmax=0,100 orient=horizontal</progressbar>
<label>name=Lab2 text="Name:", size=100,12 location=10,75</label>
<textfield>name=TxtF1 text="Your Name" size=250,30 location=90,65 color=yellow</textfield>
<label>name=Lab3 text="ReportArea", size=100,12 location=10,100</label>
<textarea>name=TxtA1 text="No Report" size=360,200 location=10,120 scroll=true</textarea>
<label>name=Lab4 text="Make a choice", size=100,12 location=10,330</label>
<combobox>name=cBox items="Apple","Orange","Banana","Guava" size=200,30 location=100,320</combobox>
<label>name=Lab5 text="Make a check", size=100,12 location=10,360</label>
<checkbox>name=kBox1 text="Ripe" color=red selected=true size=50,30 location=100,352</checkbox>
<checkbox>name=kBox2 text="Green" color=green size=60,30 location=160,352</checkbox>
<checkbox>name=kBox3 text="Never mind" color=yellow size=100,30 location=235,352</checkbox>
<label>name=Lab6 text="Please push", size=100,12 location=10,395</label>
```

```
<radiobutton>name=Radio1 text="push" size=100,50 location=100,375 color=red</radiobutton>
<togglebutton>name=toggle1 text="Toggle" size=100,40 location=160,380 color=red</togglebutton>
<list>name=jlist items="Apple","Orange","Banana","Guava" color=green size=80,90 location=10,425</list>
<tree>name=jTree size=90,95 location=100,425 color=cyan nodes="Apple","Orange","Banana"</tree>
<table>name=jTable size=170,95 location=200,425 table=table.txt color=lightgray</table>
```

The **GenericController.java**

```
import java.awt.*;
import java.util.*;
import javax.swing.*;
import javax.swing.event.*;
// Joe Schwarz (C)
public class GenericController {
  public GenericController(HashMap<String, Object> map) {
    // get JButton, JTextField and JTextArea
    JButton but = (JButton) map.get("But1");
    JTextField jtf = (JTextField) map.get("TxtF1");
    JTextArea jta  = (JTextArea) map.get("TxtA1");
    but.addActionListener(e -> {
      but.setBackground(on? Color.yellow:Color.green);
      jta.append("\nButton was clicked. Color changed to:"+(on?"YELLOW":"GREEN")+
              "\nProcessBar starts....");
      on = !on;
      // start pb Thread
      (new Fill((JProgressBar) map.get("pBar"), jta)).start();
    });
    // read JTextField and write into JTextArea
    jtf.addActionListener(e -> {
      jta.append("\nName is:"+jtf.getText());
    });
    JComboBox<String> cbx = (JComboBox) map.get("cBox");
    cbx.addActionListener(e -> {
      jta.append("\nYou've selected:"+(String)cbx.getSelectedItem());
    });
    c1 = true;
    c2 = c3 = false;
    JCheckBox cb1 = (JCheckBox) map.get("kBox1");
    cb1.addItemListener(e -> {
      if (c1) jta.append("\nYour choice: Ripe.");
      c1 = !c1;
      cb1.setForeground(c1?Color.red:Color.blue);
      cb1.setText(c1?"Ripe":"gone");
    });
    JCheckBox cb2 = (JCheckBox) map.get("kBox2");
    cb2.addItemListener(e -> {
      c2 = !c2;
      if (c2) jta.append("\nYour choice: Green.");
      cb2.setForeground(c2?Color.blue:Color.green);
      cb2.setText(c2?"gone":"Green");
    });
    JCheckBox cb3 = (JCheckBox) map.get("kBox3");
    cb3.addItemListener(e -> {
      c3 = !c3;
```

```
      if (c3) jta.append("\nYour choice: NeverMind.");
      cb3.setForeground(c3?Color.blue:Color.yellow);
      cb3.setText(c3?"gone":"Yellow");
    });
    JRadioButton rad = (JRadioButton) map.get("Radio1");
    rad.addItemListener(e -> {
      br = !br;
      jta.append("\nYou've "+(!br?"Radio popped":"pushed"));
      rad.setForeground(br?Color.blue:Color.red);
      rad.setText(br?"pop":"push");
    });
    JToggleButton tog = (JToggleButton) map.get("toggle1");
    tog.addItemListener(e -> {
      jta.append("\nYou've Toggled");
      tog.setBackground(tog.isSelected()?Color.red:Color.blue);
    });
    JList<String> jlst = (JList) map.get("jlist");
    jlst.addListSelectionListener(e -> {
      jta.append("\nSelected from JList:"+jlst.getSelectedValue());
    });
    JTree jtree = (JTree) map.get("jTree");
    jtree.getSelectionModel().addTreeSelectionListener(e -> {
      jta.append("\nSelected from JTee:"+e.getPath().toString());
    });
    JTable jtable = (JTable) map.get("jTable");
    jtable.getSelectionModel().addListSelectionListener(e -> {
      int row = jtable.getSelectedRow();
      int col = jtable.getSelectedColumn();
      jta.append("\nSelected from JTable:"+jtable.getValueAt(row, col));
    });
  }
  private class Fill extends Thread {
    public Fill(JProgressBar pb, JTextArea jta) {
      this.jta = jta;
      this.pb = pb;
    }
    private JProgressBar pb;
    private JTextArea jta;
    public void run() {
      try {
        for (int i = 0; i <= 100; i += 10) {
          pb.setValue(i);
          pb.setIndeterminate(false);
          // delay the thread
          Thread.sleep(400);
        }
        jta.append("\nProgressBar is full");
      } catch (Exception ex) {ex.printStackTrace();}
    }
  }
  private boolean on = false, c1, c2, c3, br = false;
}
```

And the presentation of **GenericView**

The invocation: java GenericView

The **JTabbedPane**: the model tabbed.txt, tab1.txt, tab2.txt, tab3.txt, menu.txt and popup.txt

```
<!-- the viewing frame with JTabbedPane from panel model --!>
<frame> name=Jtabs tittle="MVC with JTabbedPane, JPanel and JTree" size=500,500 location=20,20
     bgimage=RicePaddy.jpg close=true resize=false</frame>
<tabbedpane>name=Tabbed tabs=tab1.txt,tab2.txt,tab3.txt tabtext="Tabbed_1","Tabbed_2","Tabbed_3"
     size=400,400 location=40,15</tabbedpane>
<!-- end of frame --!>
```

```
<panel> name=Pane1_1 size=400,500 location=200,20 </panel>
<menubar>name=menubar size=400,20 location=0,0 file=menu.txt</menubar>
<tree>name=jTree nodes=tree.txt size=300,300 location=50,25 color=cyan</tree>
<label>name=Lab1 text="Your Selection:-", size=300,12 location=50,330</label>
```

```
<panel> name=Panel_2 size=400,500 location=200,20 </panel>
<label>name=Lab4 text="Please click", size=100,12 location=10,25</label>
<button>name=But2 text="MVCbutton" size=100,50 location=90,5 color=yellow</button>
<label>name=Lab5 text="Name:", size=100,12 location=10,75</label>
<textfield>name=TxtF2 text="Your Name" size=250,30 location=90,65 color=yellow</textfield>
<label>name=Lab6 text="ReportArea", size=100,12 location=10,100</label>
<textarea>name=TxtA2 text="No Report" size=360,200 location=10,120 scroll=true</textarea>
<label>name=Lab7 text="Make a choice", size=100,12 location=10,330</label>
<combobox>name=cBox items="Apple","Orange","Banana","Guava"
          size=200,30 location=100,320</combobox>
```

```
<panel> name=Panel_3 size=400,500 location=200,20 </panel>
<popupmenu>name=Popup text="popup" size=120,80 location=10,10 file=popup.txt</popupmenu>
<editorpane>name=MyEditor content=plain
          text=https://docs.oracle.com/javase/9/docs/api/javax/swing/JEditorPane.html
          size=300,300 location=50,30 </editorpane>
```

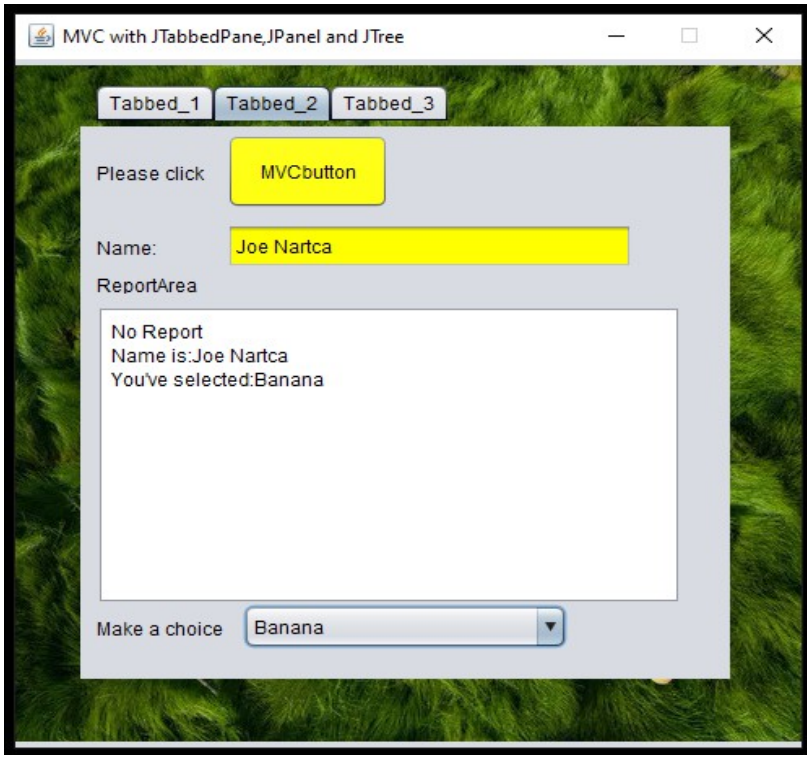
```
<menu:File,VK_F,item:Open,VK_O,item:Print,VK_P,item:Copy,VK_C/>
<menu:Help,VK_H,checkbox:About,VK_A,item:Search,VK_S/>
```

```
<item:smiling,smiling.gif,item:surprise,surprise.gif,item:thumbup,thumbUp.gif/>
```

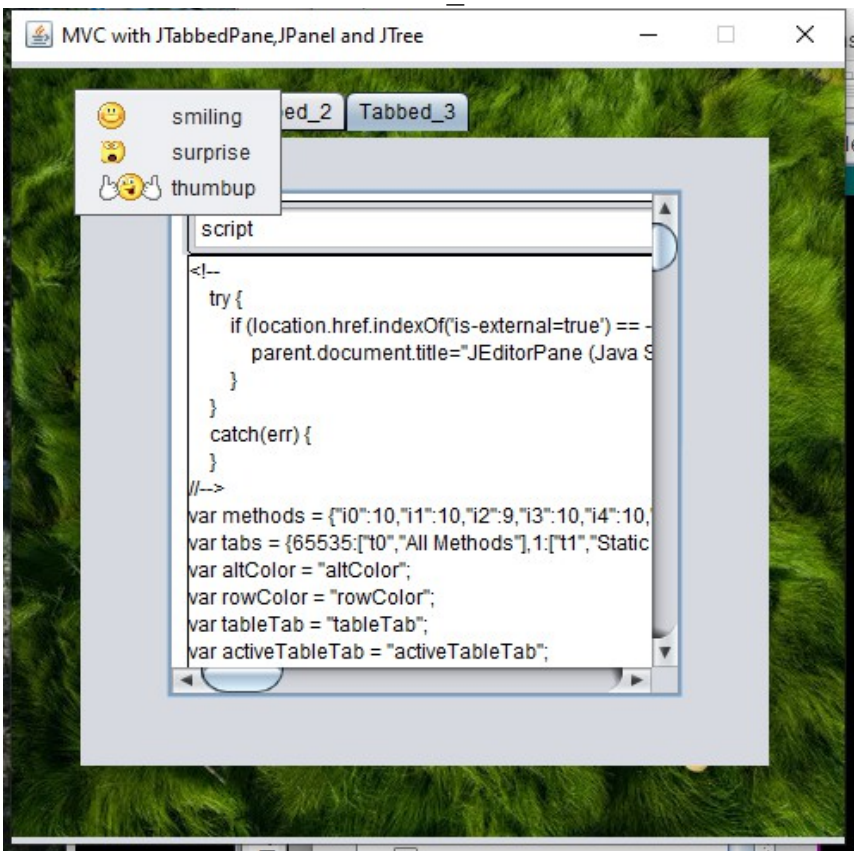
The **TabbedController.java**

```
import java.awt.*;
import java.util.*;
import javax.swing.*;
import java.awt.event.*;
//
import joeapp.mvc.*;
// Joe Schwarz (C)
public class TabbedController {
  public TabbedController(HashMap<String, Object> map) {
    JPanel jp = (JPanel) map.get("Pane3");
    JTree tree = (JTree) map.get("jTree");
    JLabel lab = (JLabel) map.get("Lab1");
    JPopupMenu pop = (JPopupMenu) map.get("Popup");
    pop.addMouseListener(new MouseListener() {
      public void mouseEntered(MouseEvent e) {
        pop.show(jp , e.getX(), e.getY());
      }
      public void mouseClicked(MouseEvent e) { }
      public void mouseExited(MouseEvent e) { }
      public void mousePressed(MouseEvent e) { }
      public void mouseReleased(MouseEvent e) { }
    });
    tree.getSelectionModel().addTreeSelectionListener(e -> {
      lab.setText("Your Selection::"+e.getPath().toString());
    });
    // Tab_2
    JButton but2 = (JButton) map.get("But2");
    JTextField jtf2 = (JTextField) map.get("TxtF2");
    JTextArea jta2  = (JTextArea) map.get("TxtA2");
    but2.addActionListener(e -> {
      but2.setBackground(on2? Color.yellow:Color.green);
      jta2.append("\nButtonTab_2 was clicked. Color changed to:"+(on2?"YELLOW":"GREEN"));
      on2 = !on2;
    });
    // read JTextField and write into JTextArea
    jtf2.addActionListener(e -> {
      jta2.append("\nName is:"+jtf2.getText());
    });
    JComboBox cbx = (JComboBox) map.get("cBox");
    cbx.addActionListener(e -> {
      jta2.append("\nYou've selected:"+(String)cbx.getSelectedItem());
    });
  }
  private boolean on2 = false;
}
```

And the presentation with the invocation java GenericView tabbed.txt TabbedControllerTab_1



Tab_2



Tab_3

MVC with **JDialog**

1. **Indirect JDialog**: JDialog is started (START button). **dialog.txt** and **dPanel.txt**

```
<frame> name=MyFrame tittle="MVC with JDialog" size=600,600 location=20,20
      close=true resize=false</frame>
<button>name=Start text="Start Dialog" size=100,100 location=230,230 color=yellow</button>
<dialog>name=MyDialog load=dPanel.txt size=500,500 location=0,0 bgimage=RicePaddy.jpg </dialog>
```

```
<!-- panel for Dialog --!>
<panel> name=Pane1 size=400,400 location=40,25 </panel>
<label>name=Lab1 text="Please click", size=100,12 location=10,25</label>
<button>name=But text="MVCbutton" size=100,50 location=90,5 color=yellow</button>
<label>name=Lab2 text="Name:", size=100,12 location=10,75</label>
<textfield>name=TxtF text="Your Name" size=250,30 location=90,65 color=yellow</textfield>
<label>name=Lab3 text="ReportArea", size=100,12 location=10,100</label>
<textarea>name=TxtA text="No Report" size=360,200 location=10,120 scroll=true</textarea>
<label>name=Lab4 text="Make a choice", size=100,12 location=10,330</label>
<combobox>name=cBox items="Apple","Orange","Banana","Guava"
               size=200,30 location=100,320</combobox>
<!-- end of pane1--!>
```

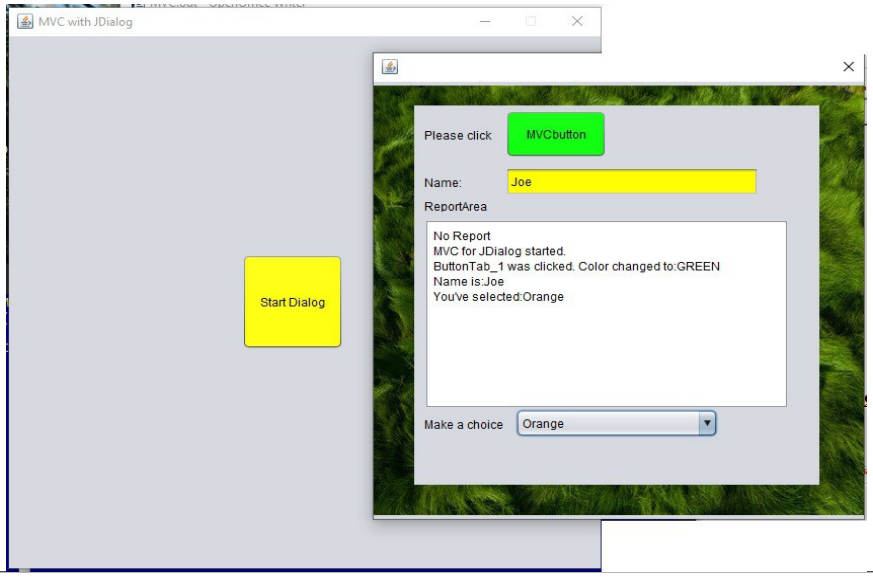
The DialogController

```
import java.awt.*;
import java.util.*;
import javax.swing.*;
import java.awt.event.*;
// Joe Schwarz (C)
public class DialogController {
  public DialogController(HashMap<String, Object> map) {
    // get JButton, JTextField and JTextArea
    JButton start  = (JButton) map.get("Start");
    JButton but    = (JButton) map.get("But");
    JTextField jtf = (JTextField) map.get("TxtF");
    JTextArea jta  = (JTextArea) map.get("TxtA");
    start.addActionListener(e -> {
      JDialog jd = (JDialog) map.get("MyDialog");
      // because location is not given -> setLocationRelativeTo.
      jd.setLocationRelativeTo((JFrame)map.get("MyFrame"));
      jta.append("\nMVC for JDialog started.");
      jd.setVisible(true);
    });

    but.addActionListener(e -> {
      but.setBackground(on? Color.yellow:Color.green);
      jta.append("\nButtonTab_1 was clicked. Color changed to:"+(on?"YELLOW":"GREEN"));
      on = !on;
    });
    // read JTextField and write into JTextArea
    jtf.addActionListener(e -> {
      jta.append("\nName is:"+jtf.getText());
    });
    JComboBox cbx = (JComboBox) map.get("cBox");
    cbx.addActionListener(e -> {
```

```
    jta.append("\nYou've selected:"+(String)cbx.getSelectedItem());
   });
 }
 private boolean on = false;
}
```

Invocation: java GenericView dialog.txt DialogController

2. **Direct JDialog** is loaded and started by main Controller: direct.txt




```
<frame> name=MyFrame tittle="MVC with JDialog" size=400,310 location=20,20
      close=true resize=false</frame>
<button>name=Start text="Start Dialog" size=100,40 location=130,10 color=yellow</button>
<label>name=Lab text="ReportArea", size=100,14 location=150,60</label>
<textarea>name=TxtA text="No Report" size=350,180 location=15,80 scroll=true edit=false</textarea>

<panel> name=Pane1 size=400,170 location=40,25 </panel>
<button>name=But text="Close Dialog" size=100,50 location=145,5 color=yellow</button>
<label>name=Lab2 text="Name:", size=100,12 location=10,75</label>
<textfield>name=TxtF text="Your Name" size=250,30 location=90,65 color=yellow</textfield>
<label>name=Lab4 text="Make a choice", size=100,12 location=10,100</label>
<combobox>name=cBox items="Apple","Orange","Banana","Guava"
                size=200,30 location=100,110</combobox>
```

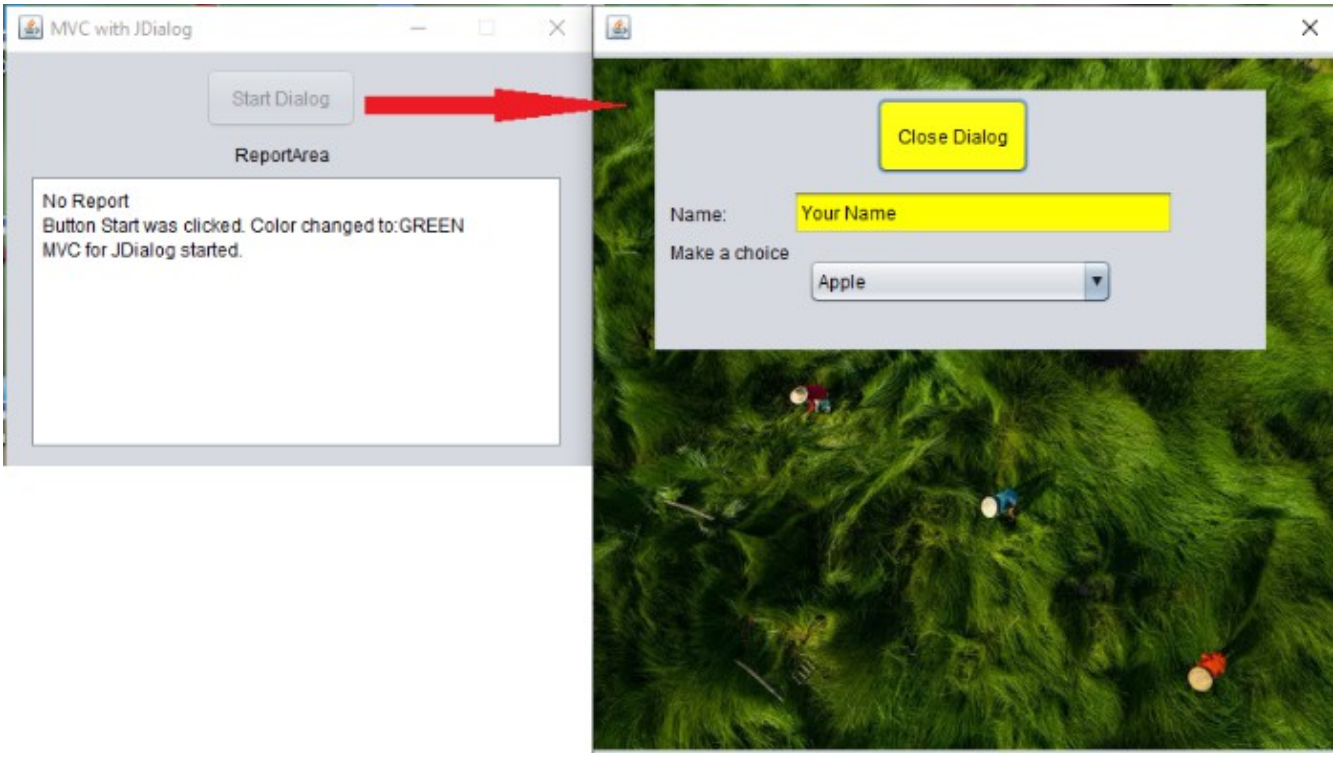
And a separate JDialog: **jdialog.txt**

```
<dialog>name=MyDialog load=diPanel.txt size=500,500 location=0,0 close=false
          bgimage=RicePaddy.jpg </dialog>
```

The Controller: DirectDialog.java

```
import java.awt.*;
import java.util.*;
import javax.swing.*;
import java.awt.event.*;
//
import joeapp.mvc.SWINGLoader;
// Joe Schwarz (C)
```

```
public class DirectController {
  public DirectController(HashMap<String, Object> map) {
    // get JButton, JTextField and JTextArea
    JButton start  = (JButton) map.get("Start");
    JTextArea jta  = (JTextArea) map.get("TxtA");
    start.addActionListener(e -> {
      if (dialog) return;
      start.setEnabled(false);
      start.setBackground(on? Color.yellow:Color.green);
      jta.append("\nButton Start was clicked. Color changed to:"+(on?"YELLOW":"GREEN"));
      on = !on;
      try {
        SWINGLoader ml = new SWINGLoader("c:/JoeApp/mvc/example/model/jdialog.txt");
        JDialog jd = (JDialog) ml.load();
        HashMap<String, Object> mp = ml.getComponentMap();
        JButton but = (JButton) mp.get("But");
        but.addActionListener(e1 -> {
          jta.append("\nDialogButton was clicked. Dialog closed");
          start.setEnabled(true);
          dialog = false;
          jd.dispose();
        });
        JTextField jtf = (JTextField) mp.get("TxtF");
        jtf.addActionListener(e2 -> {
          jta.append("\nFrom Dialog\nName is:"+jtf.getText());
        });
        JComboBox cbx = (JComboBox) mp.get("cBox");
        cbx.addActionListener(e3 -> {
           jta.append("\nFrom Dialog:\nYou've selected:"+(String)cbx.getSelectedItem());
        });
        jd.setLocationRelativeTo((JFrame)map.get("MyFrame"));
        jta.append("\nMVC for JDialog started.");
        jd.setVisible(true);
      } catch (Exception ex) {
        jta.append("\nCan't start SWINGLoader. Reason:"+ex.toString());
      }
    });
  }
  private boolean on = false, dialog = false;
}
```



IX. Similarity between SWING MVC and JFX MVC

To show you the similarity between the two MVCs two apps which work exactly as the same, but one is developed in **SWING** and the other in **JFX** with FXML.

The main model

```
<frame> name=frame title="ODBServer" size=680,600 location=10,10
        close=true resize=false</frame>
<tabbedpane>name=Tabbed tabs=_tab1_.txt,_tab2_.txt,_tab3_.txt,_tab4_.txt

tabtext=AdminManagement,UserMaintenance,ODBMaintenance,SystemMonitoring
        size=680,600 location=0,5</tabbedpane>
```

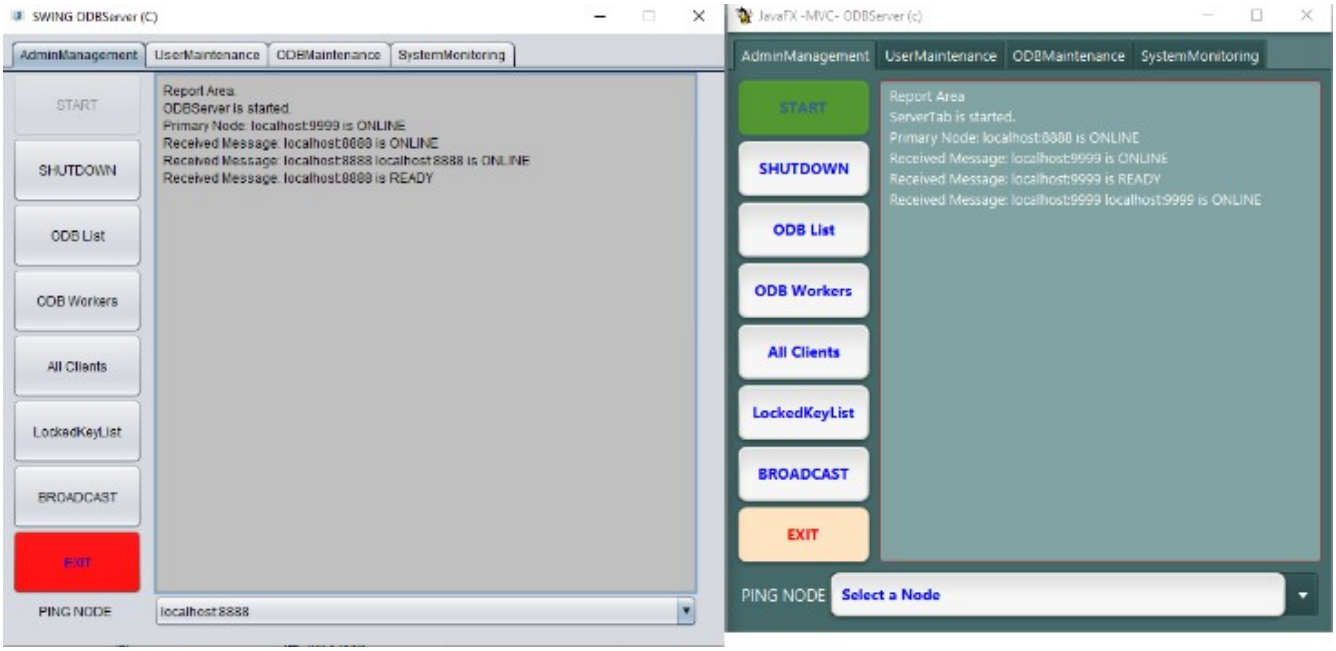
```
<?xml version="1.0" encoding="UTF-8"?>
<?import java.net.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.geometry.*?>

<?import jfx.SysMonJFX?>

<AnchorPane fx:id="root" maxHeight="-Infinity" maxWidth="-Infinity"
          minHeight="-Infinity"    minWidth="-Infinity"    prefHeight="550.0"
prefWidth="555.0"
  xmlns="http://javafx.com/javafx/9"
  xmlns:fx="http://javafx.com/fxml/1"
  fx:controller="JFXController">
 <children>
   <TabPane fx:id="tabPane"
       tabClosingPolicy="UNAVAILABLE">
     <tabs>
       <Tab fx:id="serverTab" text="AdminManagement">
          <content>
            <fx:include fx:id="Server" source="ServerTab.fxml" />
          </content>
       </Tab>
                                <Tab    fx:id="userTab"    text="UserMaintenance"
onSelectionChanged="#event">
          <content>
            <fx:include fx:id="User" source="UserTab.fxml" />
          </content>
       </Tab>
```

	<Tab fx:id="odbTab" text="ODBMaintenance" onSelectionChanged="#event"> <content> <fx:include fx:id="ODB" source="ODBTab.fxml" /> </content> </Tab> <Tab fx:id="moniTab" text="SystemMonitoring"> <content> <VBox fx:id="vbox" alignment="center" spacing="10" > <padding><Insets top="5"/></padding> <children> <SysMonJFX fx:id="sysmon" width="520.0" height="503.0" /> </children> </VBox> </content> </Tab> </tabs> </TabPane> </children> <stylesheets> <URL value="@joe.css" /> </stylesheets> </AnchorPane>

The first Tab

<panel> name=Pane1_1 size=680,500 location=0,0 </panel> <button>name=start text=START size=120,60 location=10,0 color=green</button> <button>name=stop text=SHUTDOWN size=120,60 location=10,60</button> <button>name=list text="ODB List" size=120,60 location=10,120</button> <button>name=workers text="ODB Workers" size=120,60 location=10,180</button> <button>name=clients text="All Clients" size=120,60 location=10,240</button> <button>name=keys text=LockedKeyList size=120,60 location=10,300</button> <button>name=bcast text=BROADCAST size=120,60 location=10,360</button> <button>name=exit text=EXIT size=120,60 location=10,420 color=red textColor=blue</button> <textarea>name=area1 color=lightgray size=500,480 location=140,0</textarea> <label>name=lab text="PING NODE" size=100,30 location=35,480</label> <combobox>name=ping items=PING size=500,30 location=140,480</combobox>	<?xml version="1.0" encoding="UTF-8"?> <?import javafx.scene.control.*?> <?import javafx.scene.layout.*?> <?import javafx.scene.control.ComboBox ?> <?import javafx.scene.text.Font?> <?import javafx.geometry.*?> <AnchorPane fx:id="serverPane" prefHeight="500.0" prefWidth="520.0" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1" fx:controller="ServerTab"> <children> <VBox alignment="center" spacing="10" > <HBox spacing="10" > <padding><Insets top="10" right="10" left="10"/></padding> <VBox alignment="center" spacing="6" > <children> <Button fx:id="Start" onAction="#start" text="START" prefHeight="50.0" prefWidth="120.0" /> <Button fx:id="Shutdown" onAction="#shutdown" text="SHUTDOWN" prefHeight="50.0" prefWidth="120.0" /> <Button fx:id="Odblist" onAction="#odblist" text="ODB List" prefHeight="50.0" prefWidth="120.0" /> <Button fx:id="Odbworker" onAction="#odbworker" text="ODB Workers" prefHeight="50.0" prefWidth="120.0" /> <Button fx:id="Allclients" onAction="#allclients" text="All Clients" prefHeight="50.0" prefWidth="120.0" /> <Button fx:id="Lockedkeylist" onAction="#lockedkeylist" text="LockedKeyList" prefHeight="50.0" prefWidth="120.0" /> <Button fx:id="Broadcast" onAction="#broadcast" text="BROADCAST" prefHeight="50.0" prefWidth="120.0" /> <Button fx:id="Exit" onAction="#exit" text="EXIT" prefHeight="50.0" prefWidth="120.0" /> </children> </VBox> <TextArea fx:id="report" prefWidth="405.0" wrapText="true" editable="false"/> </HBox> <HBox alignment="center" spacing="5" > <Label fx:id="PingLab" text="PING NODE"> <Font size="13.0" /> </Label> <ComboBox fx:id="Pingnode" onAction="#pingnode" promptText="Select a Node" prefHeight="40.0" prefWidth="447.0" /> </HBox> </VBox> </children> </AnchorPane>

The same for the other Tabs and the GUI appearances are as following:

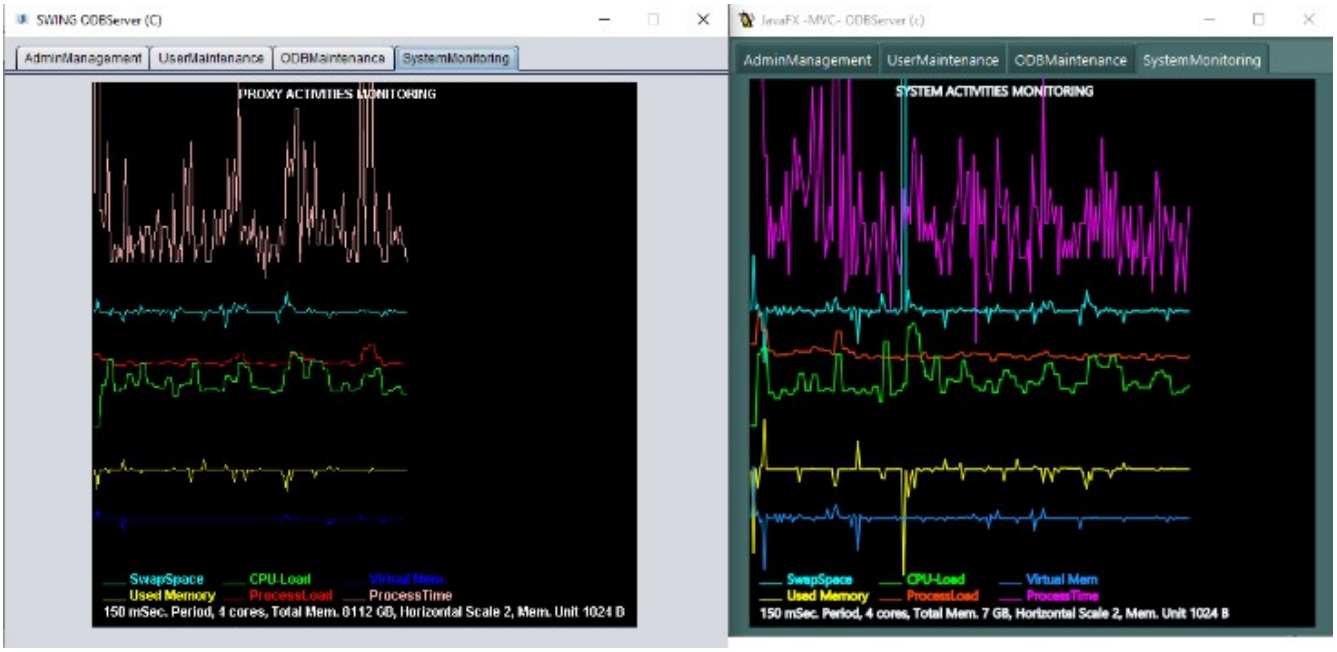


and the fourth Tab

```
<panel>name="panel_4" size=680,500 location=0,0</panel>
<SysMonSWING> name=sysmon size=500,500 location=80,7 </SysMonSWING>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.geometry.*?>
<?import jfx.SysMonJFX?>

<AnchorPane fx:id="sysmonPane" prefHeight="503.0" prefWidth="520.0"
          xmlns="http://javafx.com/javafx/8"
          xmlns:fx="http://javafx.com/fxml/1"
          fx:controller="SysMonTab">
  <children>
    <VBox fx:id="vbox" alignment="center" spacing="10" >
      <padding><Insets top="5"/></padding>
      <children>
        <SysMonJFX fx:id="sysmon" width="520.0" height="503.0" />
      </children>
    </VBox>
  </children>
</AnchorPane>
```

And the result is similar too:



Joe T. Schwarz