The Use of Java Swing's Components to Develop a Widget

Dewi Agushinta R. dewiar@staff.gunadarma.ac.id
Computer Science and Information Technology Faculty
Gunadarma University
Pondok Cina, Depok 16424,
West Java, Indonesia

Avinanta Tarigan avinanta@staff.gunadarma.ac.id
Industrial Technology Faculty
Gunadarma University
Pondok Cina, Depok 16424,
West Java, Indonesia

Egy Wisnu Moyo eguy@student.gunadarma.ac.id
Informatics Department
Gunadarma University
Pondok Cina, Depok 16424,
West Java, Indonesia

Fitria Handayani Siburian dearest_v3chan@student.gunadarma.ac.id
Informatics Department
Gunadarma University
Pondok Cina, Depok 16424,
West Java, Indonesia

Sigit Widiyanto ddxq_cuayang@student.gunadarma.ac.id
Informatics Department
Gunadarma University
Pondok Cina, Depok 16424,
West Java, Indonesia

Abstract

Widget is kind of application that provides a single service such as a map, news feed, simple clock, and battery-life indicators. It is developed to facilitate user interface (UI) design. A user interface function may be implemented using different widgets developed on different UI platforms. This article presents a comprehensive review on Java Swing as a platform to develop widgets. It is a platform that is generally used in various applications, such as in desktop, web browser, and mobile phone. Furthermore, we also describe UI elements of Java Swing's components used to establish widgets. At the end, this article discusses comparison between Java Swing and several commonly used UI platforms.

Keywords: Desktop, Java Swing, Web Browser, Widget

1. INTRODUCTION

Since 2003, a new kind of application has been significant proliferation onto both the desktops of computers and web-enabled portable devices like mobile phones [1]. This kind of application is generally referred to by developers as a widget engine. A piece of software that is able to run other smaller applications known as widgets or gadgets [2].

One of the biggest challenges of widget development is writing multiple sets of computer codes so that a widget will be compatible with multiple operating systems and types of devices. Companies considering new mobile widgets should evaluate and then deploy applications according to the following criteria: business model, distribution model, server-side application framework and the run-time environment.

Many widgets have been developed to facilitate user interface design. The user interface (UI) of an application provides an interactive environment for a user to better communicate with the system [3]. A user interface (UI) function may be implemented using different widgets with the same function. Most designers do not develop UI components in house because many widgets engines are available on the market. Widgets can be described as a type of single-purpose application, which rather than operate in a completely stand alone fashion instead is deployed within a framework that handles basic functions and services.

Swing is a GUI application programming interface that is included in the Java development kit [4]. Swing is built on top of Abstract Window Toolkit (AWT). So, many AWT features are found in Swing. In particular, Swing uses AWT's event handling model and layout managers. In other side Swing adds new components, event types, and layout managers. Moreover, Swing includes replacements for most AWT components. The name of a replacement component begins with a 'J' followed by the name of the corresponding AWT component.

Swing is a part of GUI technologies so-called Java Foundation Classes (JFC). JFC provides classes that support devices for the physically and mentally, that customize the appearance and behavior of GUI components. Thus, Swing provides enhanced 2D graphics support, and that provide drag and drop support.

Since Java is an object-oriented language, understanding a component's inheritance hierarchy and the corresponding interfaces that it implements are important to understand where the component gets its functionality. Some of JComponent elements include support for borders, component sizing, component painting, mouse over tool tip text, actions (which is a combination of a GUI component and an action event listener), binding keystrokes to actions, and double buffering (which prevents a component from flickering when it is continuously painted as in animation).

This article presents fundamental concepts of a widget application that is able to produce a customized product based on the circumstances and standards. We present a review on Java platform Swing compared with other platform as a widget developer. We will introduced the Java literature in section 2. We will explain about widget types and how to deploy it. Section 3 presents Java Swings as a widget. Discussion about comparation of Java Swing and several platform presents in section 4. Last section of this article is conclusion and future work.

2. LITERATURE

Java is very particular about the order in which actual parameter values are included in constructions and method invocations. All users have to make sure they place the parameters in the required order whenever they write a construction involving multiple actual parameters. Java provides extensive mechanisms to enable a programmer to tell it how the components of a program’s interface should arranged.

Most current computer programs interact with their users through mechanisms like buttons, menus, and scroll bars [5]. These mechanisms are called Graphical User Interface Components, or GUI components. The actual details of how GUI components should behave in a Java program are described as part of the Swing library.

The Swing component set is part of the Java Foundation Classes (JFC). The JFC is a collection of Java classes which are used to develop software based on GUI [6]. JFC also has classes that are used to add the functions and the ability to create a variety interaction of Java programming languange. JFC contains a number of GUI components (buttons, textfields, scrollpanes, etc.) that are intended to be direct replacements of the corresponding AWT GUI components. Once users understand the basic form of a construction, it is easy to learn how to construct GUI components of other types. All users need to know is the name assigned by Swing to the type of component you want to create and the actual parameters to use to specify the details of the components. This information is provided in the documentation for the Swing libraries that is available online.

Many widgets have been developed to facilitate user interface design. The Swing gives the programmer a more powerful interface to the widgets. Employing a technology called the Model/View/Controller (MVR) architecture, the Swing has given users the ability to control how widgets look, how it respond to input, and, for some more complex widgets, how data is represented. MVC is a design pattern often used in building user interface (UI) [7].

A UI function may be implemented using different widgets with the same function. A UI Component is specialized into three categories: StaticDisplay, ActionInvoker and InteractionControl [8]. The StaticDisplay category is relevant at those UI components providing visual information, such as labels. The ActionInvoker category is relevant at those UI components receiving user triggers, such as a button. The InteractionControl category is relevant at those UI components receiving user options concerning navigation through the UI, such as a menu. The graphical components that make up the user interface not only deliver information to the user, but also are the sources of the user-generated events that provide input. When the user performs an action such as pressing a button, the component in which the action takes place generates an event [9]. An event listener is an object that responds to or handles a particular kind of event. The event listener registers with the component generates an applicable event. The listener responds by performing some appropriate action.

Desktop Widgets

The desktop widget is a type of widget that has interactive virtual tools providing single-purpose services such as showing the user the latest news, the current weather, the time, a calendar, a dictionary, a map program, a calculator, desktop notes, photo viewers, or even a language translator, among other things. The desktop widget is commonly just called widgets [10]. Figure 1 shows an example of desktop widget.

FIGURE 1: Desktop widget example.

Examples of widget engines include dashboard widgets of Apple Macintosh, Microsoft gadgets in Windows Vista and in the Windows Live system, Plasmoids (widgets in Plasma the workspace for the KDE desktop environment), Porlets in Google Desktop, Yahoo! Widgets, gdesklets, adesklets, and Screenlets in Linux and also Opera widgets on all platforms (desktop, mobile TVs, gamming consoles) using the Opera browser's rendering engine.

Web Widgets

Web browsers can also be used as widget engine infrastructures. While widgets on desktop have attracted the most attention initially, there has also been a parallel development of widgets for web. It means that small chunk of web functionality that has facility to be embedded in other web applications. For example, adding an instant messaging tool to a web log to enable live interaction with visitors. Figure 2 illustrates example of web widget.

FIGURE 2: Web widget example.

Web widgets are entirely server-side entities that are integrated into other web pages [11]. When the server generates a page, it insters the widget code, typically implemented in Flash or JavaScript, into the web page being created based on the user's preferences. By manipulating these preferences, the users can place mini web applications into their personalized web page.

In most cases web widgets can be developed by users of a framework. Since the introduction of web widgets, thousands of them have been created for a wide variety of tasks. Some examples are used to display the weather, to show news headlines and to play games. Often framework offer users a catalog where they can download predefined web widgets [12].

Web widgets have unleashed some commercial interest, due their perceived potential as a marketing channel, mainly because they provide interactively through social networks. The first known web widget, Trivia Blitz, was introduced in 1997. It was a game applet offered by Uproar.com (the leading online game company from 2000-2001) that appeared on over 35,000 websites ranging from Geocities personal pages to CNN and Tower Records.

Collaboration Widgets

Desktop widget and their supporting engines have been developed in response to the need from users' discrete chunks of functionality in a simple and fun way. It is exclusively from a single-user viewpoint. A number of collaboration widgets to be developed are Gabbly chat tools and 3Bubbles.

More Widgets

Blidgets are desktop widgets that connect the user to a blog.

Widget draft standard is used to standardize some aspects of widgets. The Opera browser is the first widget engine to adopt this draft W3C standard.

Mobile Widgets are like desktop widgets, but for a mobile phone. Applications for a single specific purpose and with low functional complexity are appropriate to be developed as mobile widgets [13]. Mobile widgets can maximize screen space use and may be especially useful in placing live data-rich applications on the device idle-screen/ home-screen/"phone-top". While widgets are a convenience in online world, they can be looked at as near-essential in mobile world. The reason is because mobile device is small and interface is often challenging. Their user interface can be kept simple, which allows easy and intuitive interaction though the limited input capabilities of mobile devices.

TV set widgets is also available for TV's. Yahoo! Widget Engine is announced as a component of the next generation TV sets.

Deploying A Widget

The widget can be described as a Graphical User Interface (GUI), which can interact with user, so the developer has to attend on designing. It has been described the initial findings on using a lightweight approach to addition of small applications ('widgets') to the palette of options available for Learning Designing environments. There are two models architecture. Firstly, the model uses a local framework for the instantiation of tools within a managed environment. Second, the model use of wider framework to incorporate tools distributed that usually is used on web based development.

In other development, qualities of Model View Controller (MVC) have to base on widgets standardization. The MVC is a modern user interface framework for Smalltalk [14]. The MVC Triad can produce an "Observer" based framework that is easy to use and more flexible than currently available in other Smalltalk.

Widget can be implemented on many kinds application, such as mobile application. Users can create a personal widget by themselves without programming. When deploy the widget, we have to respect on standardizing of widget development. It is described arguing the continued proliferation of a class of software application known as widgets onto desktop computers and mobile phones has resulted in incompatibilities across most widget and related software.

There are various kinds of widget which are related software provided in software development. One of them is Standard Widget Toolkit (SWT), which provided by Java [15]. SWT is the software component that delivers native widget functionality for the Eclipse platform in an operating system independent manner. The other part is standard for Google web application called Google Web Toolkit (GWT) [16]. The standard GWT distribution comes with a wide range of widgets for use in our application. GWT have implemented a strong hierarchy of Java classes in order to provide an element of consistency across widgets where that consistency naturally exists.

Control Flow of a GUI Program

Figure 3 is the control flow of a GUI program. The start and end steps are basically the same as the ones for a text-based menu program. After a GUI component (widget) is created, it is positioned on screen by a layout manager. Examples of GUI components include buttons, menus, and windows. Before we explain the next step in the flow chart, we need to provide information on events.

An event is, for a lack of a better word, "something" that can happen to a GUI component. An event may or may not result in a change in the component's appearance or state. For example, usually, a mouse click on a button changes the appearance of the button so that it looks depressed. However, usually, when the mouse pointer is moved over a button, the button does not change its appearance. Nevertheless, this is still considered an event. The reason is that the appearance change is not caused by the event. It is caused by the thing that responds to the event.

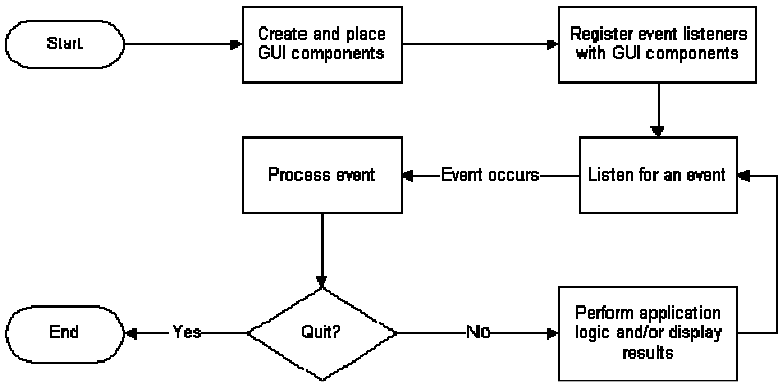

FIGURE 3: GUI Program Flow Control
(http://www.cs.ubc.ca/~ramesh/cpsc304/tutorial/JDBC/Swing/jdbc_swing.html).

Each event has its own set of triggers that result in the firing (creation) of the event. Generally, user input act as the triggers. For example, when a mouse button is clicked while the mouse pointer is over a button, a mouse event is triggered. This results in the button firing a mouse event. The event handling tutorial at Sun has a list of the types of events that each Swing component can fire. Anything that can fire an event is an event source. Not all event sources are GUI components, but most are.

An event that is fired will be "uneventful" if there is nothing to respond to it. The things that respond to events are event listeners (knows as event handlers or callbacks). Generally, you need to register an event listener with an event source so that event source can inform the listener about any events. This is the third step in the flow chart. Some GUI components have pre-registered event listeners that handle standard events like example of a mouse click changing the appearance of a button.

In the next step, program simply waits for events. When an event occurs, registered event listeners are notified, so that they can process the event. If the user decided to quit, an event listener should terminate program. Otherwise the event listener that corresponds to the selected command will be invoked. After the listener performs its task, the program waits for another event.

Model-View-Controller Architecture

The model encapsulates an object's data. The view represents the appearance of object. It renders data in model. When the model's data changes the model notifies the view, so that the view can update itself. The controller represents behavior of the object. When the user interacts with view, the input is transferred from the view to the controller. The controller then updates the model's data appropriately based on semantics of the user input. The purpose of a model-view-controller architecture is to decouple an object's data, appearance and behavior. To change the interface ("look and feel") of an object, we only have to change the view and controller. Figure 4 shows how the model-view-controller architecture applies to Swing components.

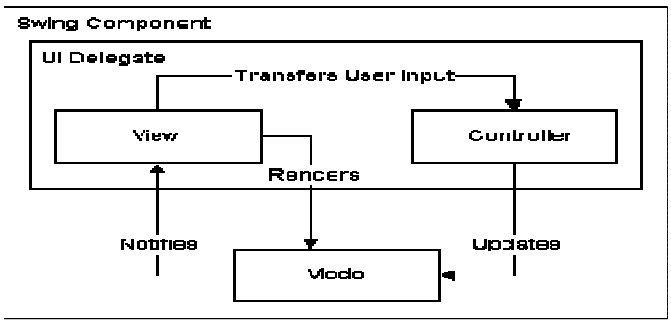

FIGURE 4: MVC Architecture.

3. JAVA SWINGS AS A WIDGET

3.1 Basic Controls

JButton

JButton is one of the ordinary buttons. We can make a JButton be the default button as figure 5. At most one button in a top-level container can be default button. The default button typically has a highlighted appearance and acts clicked whenever the top-level container has the keyboard focus and the user presses Return or Enter key. The exact implementation of default button feature depends on the look and feel. For example, the default button changes to whichever button has the focus, so that pressing Enter clicks the focused button. When no button has the focus, the button originally specified as the default button becomes the default button again.

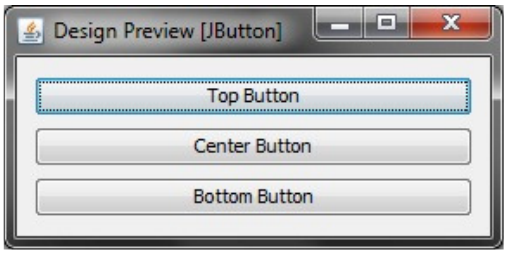

FIGURE 5: Button component.

JCheckBox

JCheckBox class provides support for check box buttons. We can also put check boxes in menus, using the JCheckBoxMenuItem class. Because JCheckBox and JCheckBoxMenuItem inherit from AbstractButton, Swing check boxes have all the usual button characteristics. For example, we can specify images to be used in check boxes.

Check boxes are similar to radio buttons but their selection model is different, by convention. Any number of check boxes in a group can be selected, such as none, some, or all, illustrated in figure 6. A group of radio buttons, on the other hand, can have only one button selected. A check box generates one item event and one action event per click. Usually, we listen only for item events, since they let we determine whether the click selected or deselected the check box.

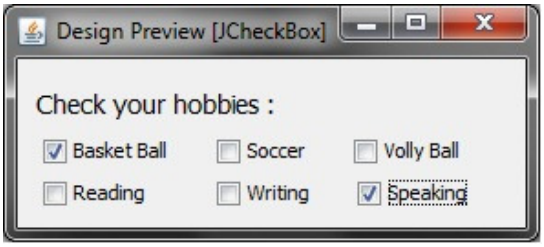

FIGURE 6: CheckBox component.

JComboBox

A JComboBox, which lets user choose one of several choices, can have two very different forms. The default form is uneditable combo box, which features a button and a drop-down list of values. Second form, called editable combo box, features a text field with a small button abutting it. The user can type a value in the text field or click the button to display a drop-down list.

Combo boxes require little screen space, and their editable (text field) form is useful for letting user quickly choose a value without limiting user to displayed values. Other components that can display one-of-many choices are groups of radio buttons and lists. Groups of radio buttons are generally the easiest for users to understand, but combo boxes can be more appropriate when space is limited or more than a few choices are available. Lists are not terribly attractive, but they are more appropriate than combo boxes when the number of items is large, over 20, or when selecting multiple items might be valid. Editable and uneditable combo boxes are so different. An uneditable combo box in figure 7(a) contains an array of strings, but we could just as easily use icons instead. To put anything else into a combo box or to customize how the items in a combo box look, we need to write a custom rendered. An editable combo box illustrated in figure 7(b) would also need a custom editor. A combo box uses a combo box model to contain and manage the items in its menu. When we initialize a combo box with an array or a vector, the combo box creates a default model object for we. As with other Swing components, we can customize a combo box in part by implementing a custom model which is an object that implements the ComboBoxModel interface.

(a) An Uneditable (b) An Editable

FIGURE 7: ComboBox component.

An editable combo box fires an action event when user chooses an item from the menu and when the user types Enter. Note that the menu remains unchanged when user enters a value into combo box. If we want, we can easily write an action listener that adds a new item to combo box's menu each time the user types in a unique value.

JList

A JList presents user with a group of items, displayed in one or more columns, to choose from as in figure 8. Lists can have many items, so they are often put in scroll panes. In addition to lists, the following Swing components present multiple selectable items to the user, such as combo

boxes, menus, tables, and groups of check boxes or radio buttons. Because list is in single-selection mode, this code can use getSelectedIndex to get index of the just-selected item. JList provides other methods for setting or getting the selection when the selection mode allows more than one item to be selected. If we want, we can listen for events on the list's list selection model rather than on the list itself.

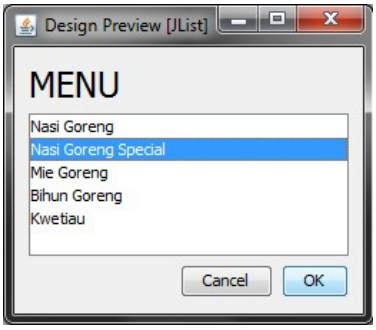

FIGURE 8: List component.

JMenu

A menu provides a space-saving way to let user choose one of several options, as in figure 9. Other components with which user can make a one-of-many choice include combo boxes, lists, radio buttons, spinners, and tool bars. Menus are unique in that, by convention, they are not placed with other components in the UI. Instead, a menu usually appears either in a menu bar or as a popup menu. A menu bar contains one or more menus and has a customary, platform-dependent location. It is usually along the top of a window. A popup menu is a menu that is invisible until user makes a platform-specific mouse action, such as pressing the right mouse button, over a popup-enabled component. The popup menu then appears under the cursor.

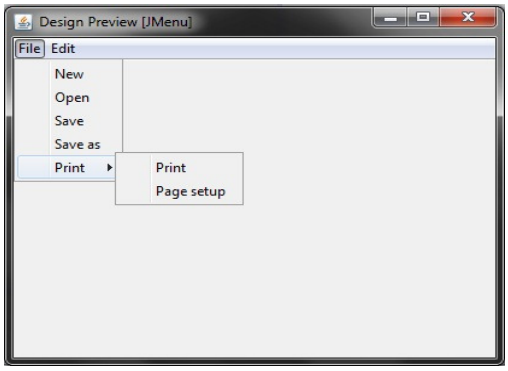

FIGURE 9: Menu component.

JRadioButton

Radio buttons are groups of buttons in which, by convention, only one button at a time can be selected. The Swing release supports radio buttons with the JRadioButton and ButtonGroup classes. To put a radio button in a menu, use JRadioButtonMenuItem class. Other ways of displaying one-of-many choices are combo boxes and lists. Radio buttons look similar to check boxes, but, by convention, check boxes place no limits on how many items can be selected at a time. Because JRadioButton inherits from AbstractButton, Swing radio buttons have all the usual button characteristics. For instance, we can specify the image displayed in a radio button, as in figure 10.

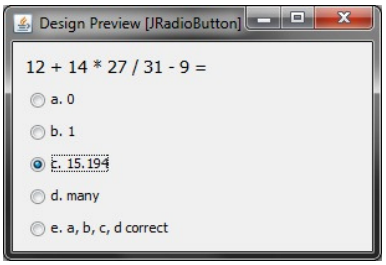

FIGURE 10: RadioButton component.

JSlider

A JSlider component is intended to let user easily enter a numeric value bounded by a minimum and maximum value. If space is limited, a spinner is a possible alternative to a slider, as in figure 11. By default, spacing for major and minor tick marks is zero. To see tick marks, we must explicitly set the spacing for either major or minor tick marks (or both) to a non-zero value and call the setPaintTicks(true) method. However, we also need labels for our tick marks. To display standard, numeric labels at major tick mark locations, set the major tick spacing, then call the setPaintLabels(true) method.

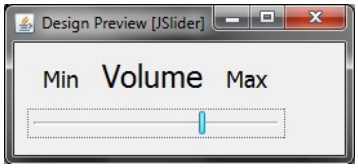

FIGURE 11: Slider component.

JSpinner

Spinners are similar to combo boxes and lists in that they let user choose from a range of values. Like editable combo boxes, spinners allow user to type in a value. Unlike combo boxes, spinners do not have a drop-down list that can cover up other components. Because spinners do not display possible values, they are often used instead of combo boxes or lists when set of possible values is extremely large. However, spinners should only be used when the possible values and their sequence are obvious. A spinner is a compound component with three subcomponents: two small buttons and an editor. The editor can be any JComponent, but by default it is implemented as a panel that contains a formatted text field. The spinner's possible and current values are managed by its model, illustrated in figure 12.

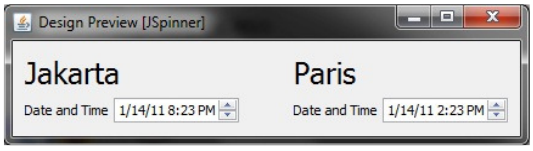

FIGURE 12: Spinner component.

JTextField

A text field is a basic text control that enables user to type a small amount of text. When user indicates that text entry is complete which is usually by pressing Enter, the text field fires an action event. If we need to obtain more than one line of input from the user, use a text area. The use of JTextField's getText method is to retrieve the text currently contained by text field. The text returned by this method does not include a new line character for the Enter key that fired the action event. Because JTextField class inherits from JTextComponent class, text fields are very flexible and can be customized almost any way you like. For instance, we can add a document listener or a document filter to be notified when text changes, and in the filter case we can modify

text field accordingly. Information on text components can be found in Text Component Features. Figure 13 shows the textfield component.

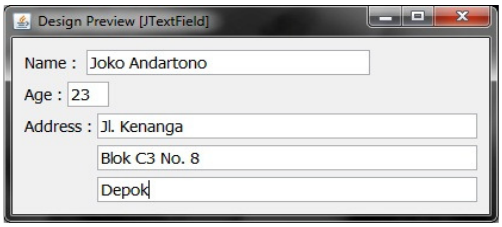

FIGURE 13: TextField component.

JPasswordField

The JPasswordField class, a subclass of JTextField, provides specialized text fields for password entry. For security reasons, a password field does not show the characters that user types, shown in figure 14. Instead, the field displays a character different from the one typed, such as an asterisk '*'. As another security precaution, a password field stores its value as an array of characters, rather than as a string. Like an ordinary text field, a password field fires an action event when the user indicates that text entry is complete, for example by pressing the Enter button.

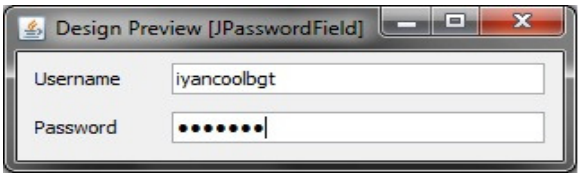

FIGURE 14: PasswordField component.

3.2 Interactive Displays of Highly Formatted Information

JColorChooser

A color chooser is a component that we can place anywhere within our program GUI. The JColorChooser API also makes it easy to bring up a dialog (modal or not) that contains a color chooser. Figure 15 is a picture of an application that uses a color chooser to set the text color in a banner.

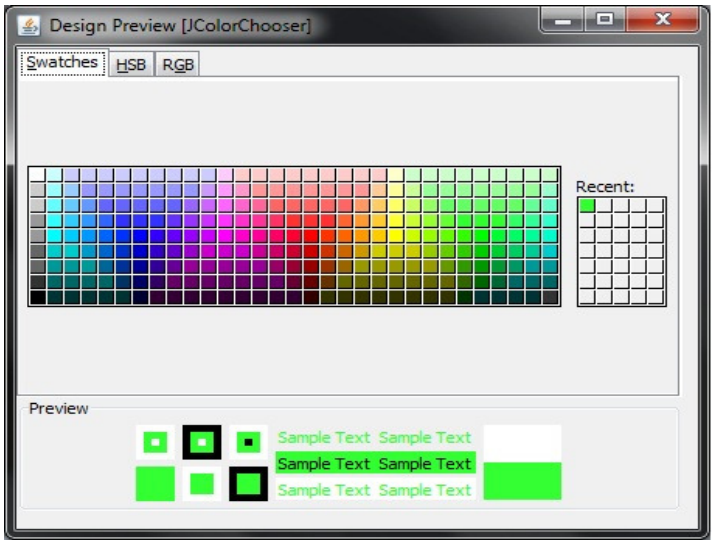

FIGURE15: ColorChooser component.

The color chooser consists of everything within the box labeled ChooseTextColor. It contains two parts, a tabbed pane and a preview panel. The three tabs in the tabbed pane select chooser panels. The preview panel below the tabbed pane displays the currently selected color.

JFileChooser

File choosers provide a GUI for navigating the file system, and then either choosing a file or directory from a list, or entering the name of a file or directory as in figure 16. To display a file chooser, we usually use JFileChooser API to show a modal dialog containing file chooser. Another way to present a file chooser is to add an instance of JFileChooser to a container. The JFileChooser API makes it easy to bring up open and save dialogs.

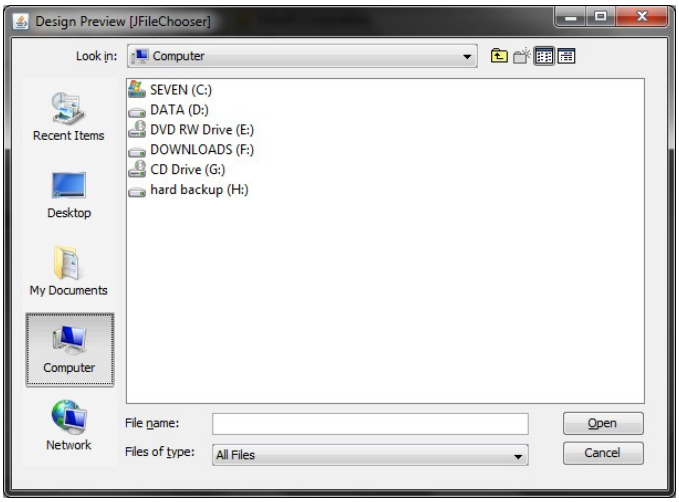

FIGURE 16: FileChooser component.

JEditorPane and JTextPane

Two Swing classes support styled text which is JEditorPane and its subclass JTextPane, illustrated in figure 17. The JEditorPane class is the foundation for Swing's styled text components and provides a mechanism through which we can add support for custom text formats. If we want unstyled text, use a text area instead.

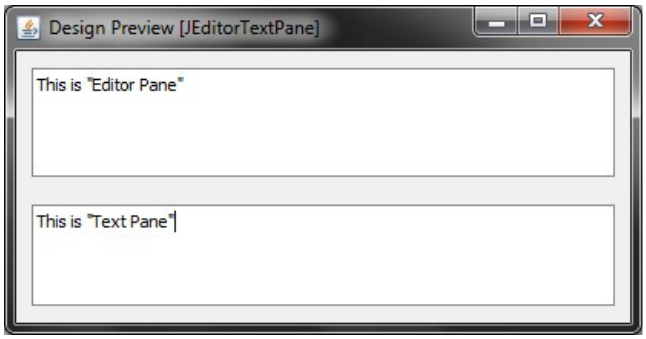

FIGURE 17: Menu component.

JTables

With the JTable class we can display tables of data, optionally allowing user to edit data. JTable does not contain or cache data; it is simply a view of the data. Figure 18 illustrated a picture of a typical table displayed within a scroll pane.

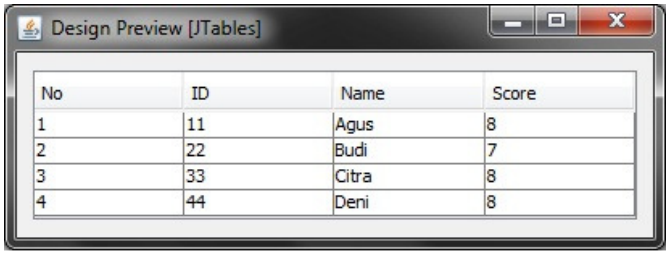

FIGURE 18: Tables component.

To set and change column widths, all columns in a table start out with equal width by default, and the columns automatically fill the entire width of table. When table becomes wider or narrower (which might happen when user resizes the window containing the table), all column widths change appropriately. When user resizes a column by dragging its right border, then either other columns must change size, or the table's size must change. By default, the table's size remains the same, and all columns to the right of the drag point resize to accommodate space added to or removed from the column to the left of the drag point.

JTextComponents

Swing text components display text and optionally allow user to edit the text. Swing provides six text components, along with supporting classes and interfaces that meet even the most complex text requirements. In spite of their different uses and capabilities, all Swing text components inherit from same super class, JTextComponent, which provides a highly-configurable and powerful foundation for text manipulation. The following figure 19 shows JTextComponent hierarchy.

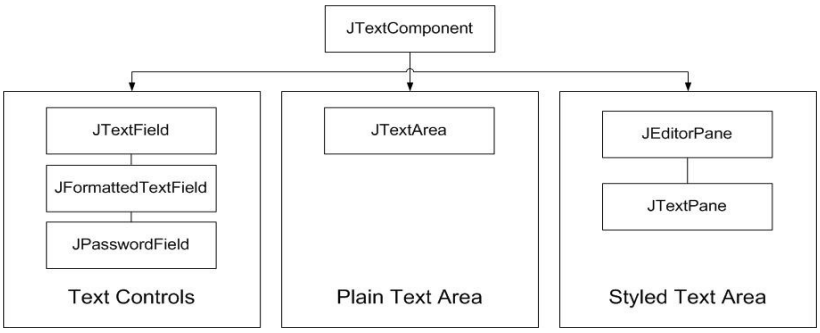

FIGURE 19: Text component hierarchy.

JTree

With JTree class, we can display hierarchical data. A JTree object does not actually contain our data; it simply provides a view of data. Like any non-trivial Swing component, the tree gets data by querying its data model. JTree displays its data vertically. Figure 20 is a picture of a tree.

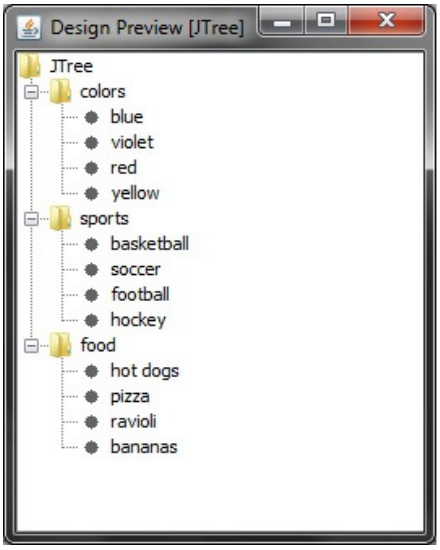

FIGURE 20: Tree hierarchy.

JLabel

With JLabel class, we can display unselectable text and images. If we need to create a component that displays a string, an image, or both, we can do so by using or extending JLabel. If the component is interactive and has a certain state, use a button instead of a label. The following figure 21 introduces a label demo that displays three labels. The window is divided into three rows of equal height; the label in each row is as wide as possible.

FIGURE 21: Label component.

JProgressBar
Sometimes we can't immediately determine the length of a long-running task, or the task might stay stuck at same state of completion for a long time. We can show work without measurable progress by putting the progress bar in indeterminate mode. A progress bar in indeterminate mode displays animation to indicate that work is occurring. As soon as the progress bar can display more meaningful information, we should switch it back into its default, determinate mode. ProgressBar showed in figure 22.

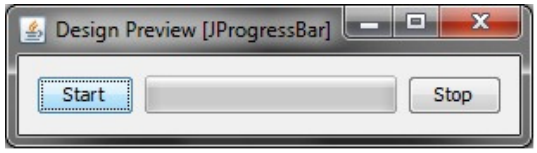

FIGURE 22: ProgressBar component.

JSeparator
The JSeparator class provides a horizontal or vertical dividing line or empty space. It is most commonly used in menus and tool bars. In fact, we can use separators without even knowing that a JSeparator class exists, since menus and tool bars provide convenience methods that create and add separators customized for their containers. Separators are somewhat similar to borders, except that they are genuine components and, as such, are drawn inside a container, rather than around the edges of a particular component. Figure 23 showed a menu that has three separators, used to divide the menu into four groups of item.

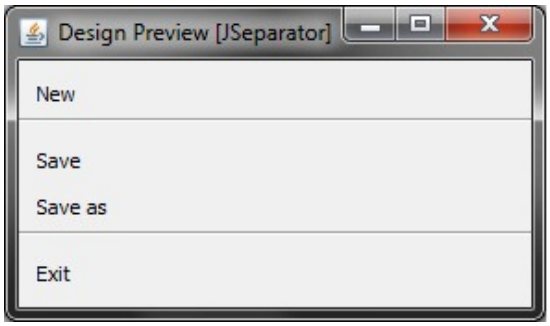

FIGURE 23: Separator component.

JToolTip
Creating a tool tip for any JComponent object is easy. Use the setToolTipText method to set up a tool tip for the component. When user of program pauses with cursor over any of the program's buttons, the tool tip for the button comes up. We can see this by running the ButtonDemo example, which is explained in How to Use Buttons, Check Boxes, and Radio Buttons. Figure 24

showed tool tip that appears when the cursor pauses over the left button in the ButtonDemo example.

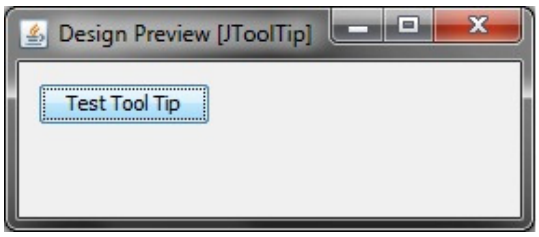

FIGURE 24: ToolTip component.

3.3 Top-Level Containers

JApplet

JApplet is a subclass of java.applet.Applet that is a "Top-Level Swing Container". Because of that, each Swing applet has a root pane. The most noticeable effects of the root pane's presence are support for adding a menu bar and the need to use a content pane. As described in Using Top-Level Containers, each top-level container such as a JApplet has a single content pane. JApplet also provides getImage method for loading images into an Applet. The getImage method creates and returns an Image object that represents the loaded image.

JDialog

A Dialog window is an independent sub window meant to carry temporary notice apart from the main Swing Application Window as in figure 25. Most Dialogs present an error message or warning to a user, but Dialogs can present images, directory trees, or just about anything compatible with the main Swing Application that manages them. Every dialog is dependent on a Frame component. A Dialog can be modal. When a modal Dialog is visible, it blocks user input to all other windows in the program. JOptionPane creates JDialogs that are modal. To create a non-modal Dialog, we must use the JDialog class directly.

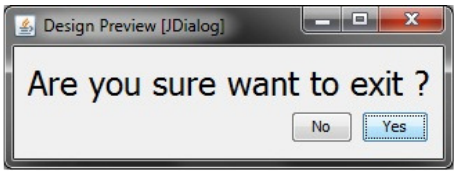

FIGURE 25: Dialog component.

JFrame

A Frame is a top-level window with a title and a border. The size of the frame includes any area designated for the border. The dimensions of border area may be obtained using the getInsets method. Since the border area is included in overall size of the frame, the border effectively obscures a portion of frame. A frame, implemented as an instance of the JFrame class, is a window that has decorations such as a border, a title, and supports button components that close or iconify the window. Applications with a GUI usually include at least one frame. Applets sometimes use frames, as well. To make a window that appears within another window, use an internal frame. Figure 26 showed the demo of frame.

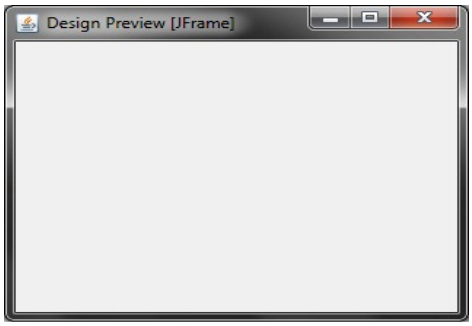

FIGURE 26: Frame component.

3.4 General-Purpose Containers

JPanel

The JPanel class provides general-purpose containers for lightweight components. By default, panels do not add colors to anything except their own background. However, we can easily add borders to them and otherwise customize their painting. We can change a panel's transparency by invoking the setOpaque method. A transparent panel draws no background, so that any components underneath show through. Figure 27 illustrated a demo of panel.

FIGURE 27: Panel component.

JScrollPane

A JScrollPane provides a scrollable view of a component. When screen real estate is limited, use a scroll pane to display a component that is large or one whose size can change dynamically. Other containers used to save screen space include split panes and tabbed panes.

A scroll pane uses a JViewport instance to manage the visible area of the client. The viewport is responsible for positioning and sizing the client, based on the positions of the scroll bars, and displaying it. A scroll pane also may use two separate instances of JScrollBar for scroll bars. The scroll bars provide interface for user to manipulate the visible area. The following figure 28 shows the three areas of a scroll bar: the knob (sometimes called the *thumb*), the (arrow) buttons, and the track.

FIGURE 28: ScrollPane component.

JSplitPane

A JSplitPane displays two components, either side by side or one on top of the other as in figure 29. By dragging the divider that appears between components, user can specify how much of split pane's total area goes to each component. The user can use them to make the divider move completely to one side or the other. Instead of adding the components of interest directly to a split pane, we often put each component into a scroll pane. We then put the scroll panes into the split pane. This allows the user to view any part of a component of interest, without requiring the component to take up a lot of screen space or adapt to displaying itself in varying amounts of screen space.

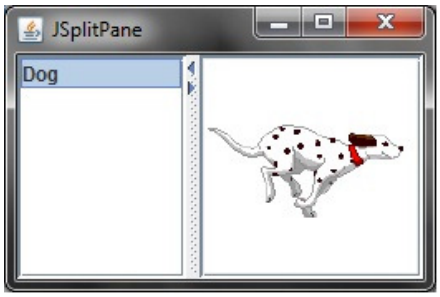

FIGURE 29: SplitPane component.

JTabbedPane

With JTabbedPane class, we can have several components, such as panels; share the same space, illustrated in figure 30. The user chooses which component to view by selecting the tab corresponding to desired component. If we want similar functionality without the tab interface, we can use a card layout instead of a tabbed pane.

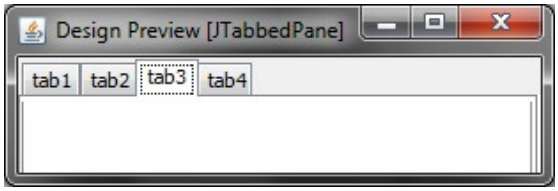

FIGURE 30: TabbedPane component.

JToolBar

A JToolBar is a container that groups several components, such as buttons with icons, into a row or column. Often, tool bars provide easy access to functionality that is also in menus. By default, user can drag tool bar to another edge of its container or out into a window of its own. The component that the tool bar affects is generally in the center of container. The tool bar must be

the only other component in the container, and it must not be in the center. Figure 31 showed the demo of toolbar.

FIGURE 31: ToolBar component.

3.5 Special-Purpose Containers

JInternalFrame

With JInternalFrame class we can display a JFrame that looks like window within another window. Usually, users add internal frames to a desktop pane. The desktop pane is an instance of JDesktopPane, which is a subclass of JLayeredPane that has added API for managing multiple overlapping internal frames. Internal frames are not windows or top-level containers, however, which makes them different from frames, shown in figure 32. Because internal frames are implemented with platform-independent code, they add some features that frames cannot give us. One such feature is that internal frames give us more control over their state and capabilities than frames do.

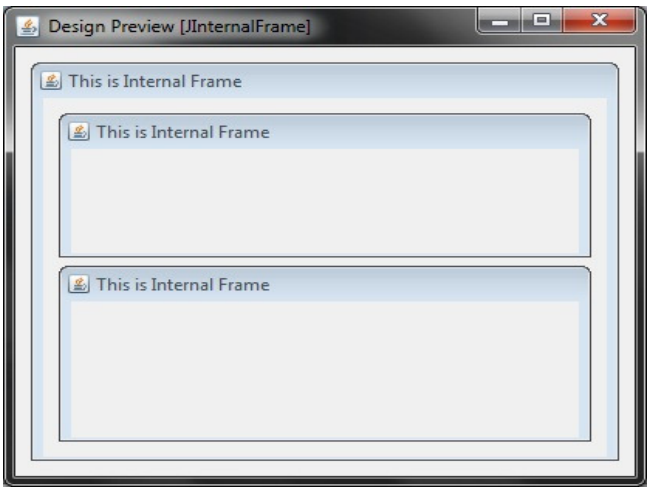

FIGURE 32: InternalFrame component.

JLayeredPane

Swing provides two layered pane classes. The first, JLayeredPane, is class that root panes use and is class used here. The second, JDesktopPane, is a JLayeredPane subclass that is specialized for the task of holding internal frames. A layered pane is a Swing container that provides a third dimension for positioning components: *depth*, also known as *Z order*. When adding a component to a layered pane, we specify its depth as an integer. The higher the number, the closer the component is to the "top" position within the container. If components overlap, the "closer" components are drawn on top of components at a lower depth. The relationship between components at the same depth is determined by their positions within the depth. The demo of layeredpane showed in figure 33.

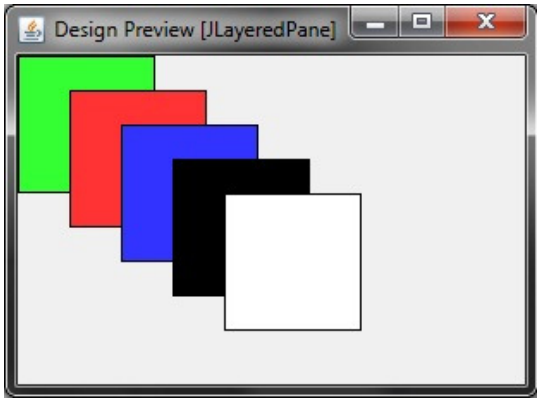

FIGURE 33: LayeredPane component.

Using Top-Level Containers tells us that the basics of using root panes — getting the content pane, setting its lower manager, and adding Swing components to it. The glass pane is useful when we want to be able to catch events or paint over an area that already contains one or more components. For example, we can deactivate mouse events for a multi-component region by having the glass pane intercept the events. Or we can display an image over multiple components using the glass pane**.** The layered pane serves to position its contents, which consist of the content pane and the optional menu bar. It can also hold other components in a specified Z order. The container of the root pane's visible components, excluding the menu bar. The optional menu bar, home for the root pane's container's menus. If the container has a menu bar, we generally use the container's setJMenuBar method to put the menu bar in the appropriate place.

4. DISCUSSION

Swing is a Tool Kit that is a development of the AWT aiming to overcome shortages in AWT. For example, many of GUI components are absent in AWT e.g. Table components which can be used to create the reports interface from database objects. Moreover, as an UI platform AWT lacks of flexibility. AWT components are derived directly from the Component class. Unlike AWT, Swing components are derived from the Container class. That is, Swing has greater flexibility to fulfill the needs of desktop applications development.
Furthermore, AWT is built as native application where the look of application follows the setting of operating system. This is caused by native subroutine call in order to display the application on the screen. Whereas Swing is built not as native application, so the look of application interfaces is independent from operating system and may be customized. Swing is written pure 100% in Java programming language without using the wrapper to call native routines code via JNI (Java Native Interface). Look A Feel facility support enables the ability to switch the display instantly. Thus, the program interface can be easily altered in accordance to the desire and goals. For instance, to have fancy UI or to have consistent looks in all circumstances. However, when native look a like interface is desired, Swing is not able to give better results from AWT. For example, fonts rendering are lees smooth from those of SWT.

In addition, Swing already supports 2D technology provided by its library. Hence, it is possible to use it for visual data processing, such as image processing, 2D object, painter, animated, or even OpenGL using JOGL. These capabilities added value to the Swing, so applications will look more beautiful and professional.

The following description compare commonly used UI development environment using five parameters, namely cross-platform ability, usability, effectiveness, efficiency, and user satisfaction. Table 1 shows summary of the discussion.

Parameter	Swing	Silverlight	GTK	HTML 5	QT
Cross-Platform Ability	✓	✓	✓	✓	✓
Usability	3	4	4	3	3
Effectiveness	3	4	3	4	4
Efficiency	4	4	3	4	2
Satisfaction	4	4	3	4	3

*The range of value is 1 - 5.

TABLE 1: Advantages and disadvantages some of the standard widget.

Swing and Silverlight runs on many platforms. Both could be used to develop the desktop, web, as well as mobile widget. Though, Silverlight is currently supported on Windows Mobile phones. Besides desktop widget, QT Widget could be used to develop the mobile widget. Its performance has been seen in mobile applications developed for Nokia's Symbian OS. GTK is mostly used in developing desktop widget for Linux/ Unix environment. HTML5 combined with Ajax and Java Script can be used to create powerful web widgets. HTML 5 can be rendered and runs on any platforms as long as the browser supports it.

Silverlight and GTK are easy to use than the others. Silverlight application can be written in C++, Basic, and Java language environment. In progress, Silverlight is also developed for ASP.NET. In addition to C++, GTK applications can also be written in other programming language. The abandoned PHP-GTK project showed an example that GTK can also be used with the most popular web programming language to develop desktop applications.

HTML and GTK have consistency on their goal which is GTK as Desktop Widget and HTML 5 as Web Widget. However Silverlight, QT, and HTML 5 are more effective to solve the real problem in widget area. For example, the web applications become more attractive and interactive.

The other parameters are efficiency and satisfaction. These parameters are two idealisms that always equal. For instance, the password field widget that we needed to protect or authorized our system. Swing, Silverlight and HTML are provided this function. So, It is only needed a function to call this widget. A programmer only needs a few times to develop the field to password input. Details of all comparison can be seen in the table 2 below.

Standard Widget's Name	Hybrid Widget			Desktop Widget	Web Widget
	Swing	Silverlight	QT	GTK	HTML 5
Basic Concept					
Button	JButton	Button Control	Qbutton	GtkButton	<command>
Check Box	JcheckBox	CheckBox Control	QcheckBox	GtkCheckButton	<command checked>
Combo Box	JcomboBox	ComboBox Control	QcomboBox	GtkComboBox	
List	Jlist	ListBox Control	Qlist	GtkListStore	<datalist>
Menu	Jmenu		Qmenu	GtkMenu	<menu>
Radio Button	JradioButton	RadioButton Control	QradioButton	GtkRadioButton	<command radiogroup>
Slider	Jslider	Slider Control	Qslider	GtkScale	
Spinner	Jspinner		Qspin	GtkSpinButon	
Text Field	JtextField	TextBox Control	QeditText	GtkEntry	<input>
Password Field	JpasswordField	PasswordBox Control			<input pass>
Interactive Display					
Color Chooser	JColorChooser			GtkColorDialog	
File Chooser	JFileChooser			GtkFileChooserDialog	
Pane	JEditorPane & JTextPane		QPane	GtkTextView	
Tables	JTables	Table Control	QTables	GtkTable	<table>
Text Component	JTextComponents				
Tree	JTree		QTree	GtkTreeView	<datagrid>
Label	JLabel	Label Control	QLabel	GtkLabel	<label>
Progress Bar	JProgressBar	ProgressBar Control	QProgressBar	GtkProgressBar	<progress>
Separator	JSeparator		QLine		
Tool Tip	JToolTip				
Top Level Containers					
Applet	JApplet				
Dialog	Jdialog		Qdialog	GtkMessageDialog	<dialog>
Frame	Jframe		Qframe		<iframe>
General-purpose Containers					
Panel	JPanel	Stack, Dock, Wrap Panel	QPanel		
Scroll Pane	JScroolPane	ScrollViewer Control	QHScroll & QVScroll	GtkScrolledWindow	
Split Pane	JSplitPane		Q Split	GtkSplit	
Tabbed Pane	JTabbedPane	Tab Control	QTab		
Tool Bar	JToolBar		QToolBar	GtkToolBar	
Special-purpose Containers					
Internal Frame	JInternalFrame				
Layered Pane	JLayeredPane		QPane	GtkPaned	

TABLE 2: Comparison of the component standard widgets.

In the table 2 above, the completeness component of all standard widget were compared to the Swing widget. Desktop Widget has more function than Web Widget. Desktop widget is used as interactive tools on single-purpose problem. Hence developer could develop the widget by small limited problem. This kind of widget could be developed as free as possible which is not depended on network problem. Contrast on Web Widget that need wider environment and higher access. Developing of this widget will need limited. Widgets platform have difference on main function based on kind of widget platform. Desktop Widget Standard Development in this discussion is Swing, GTK, and Silverlight. Swing Widget Standard is under Java Platform [4] [5], which is provided by NetBeans. Whereas GTK as a Standard Widget expanded by C programming language. GTK runs on the Linux Operating System. Silverlight is Microsoft products, which usually is integrated on the ASP.NET Environment.

Back on the completeness of the component widget, Swing has complete aspect. This kind of widget could be used as the Desktop Widget and Web Widget, moreover mobile widget development. The components detail is much more than in the table. Every widget has special widget which proves the main function of the standard widget. Swing develops a web environment use JApplet and support on developing the mobile widget, because it is able to support on Java ME. Based on ISO 19241 Part 14 and 15, this widget satisfies the Menu Dialogs and Command Dialogs [17].

Another kind of the widget is Silverlight. This widget could be developed on the .NET Framework. In addition this widget could be support on ASP.NET as desktop widget. This widget is able to integrate with web service such as, JSON, SOAP, REST, RSS, and ATOM [9]. The performance samples of Silverlight were seen on the Olympic Design Web site (http://nbcolympics.com) and Hard Rock Café Memorable Web site (http://memorabilia.hardrock.com) which are utilized a new imaging technology based on Silverlight. Silverlight also approved a library of websites that are registered early adopters can be reviewed by visiting the Silverlight website at http://silverlight.net/Showcase/.

Little bit different with the GTK (Gnome Tool Kit). It is a pure desktop widget. It could be run under Linux platform. To develop widget using this standard widget, have to adapt to C programming language. Nevertheless, this GTK has been expands to be GTK+ version which is support on Microsoft Windows, BeOS, Solaris, Mac OS X, and others [18]. The interface result by this standard widget is like Swing Widget.

The latest of HTML version is HTML 5. This version has a new vision compatibility, usability, interoperability, and universal access. This version is not at all changed from the version before. Generally, the improvement is to keep everything working smoothly. A lot of effort has been put into researching common behavior. For example, Google analyzed millions of pages to discover the common ID names for DIV tags and found a huge amount of repetition. This improvement is to overcome the problem of web widget which has a high access by user. The standard widget that is developing is devoted to the text processing and image graphic and it is all about a real problem [19].

The mobile widget which is compared in this discussion is QT Widget. The widget in QWidget (QT Widget) is the atom of the user interface: it receives mouse, keyboard and other events from the window system, and paints a representation of itself on the screen. Every widget is rectangular, and they are sorted in a Z-order. A widget is clipped by its parent and by the widgets in front of it. All of components of this widget are defined in class. QT needs “gcc” machine in the windows layered to compile the code [20].

5. CONCLUSION AND FUTURE WORK

A widget is often small part of a bigger user interface, it needs component that is simple, but nevertheless provides the users with easy navigation and rich interactivities. The using of Java

Swing enables developer to develop widgets that can run on wide range of platforms, from desktop computers, web applications, to mobile phone or gadgets. Moreover, Java Swing provides rich of components that are useful in developing widgets. This advantage has been discussed in this article along with a comprehensive review of each components mentioned.

In order to gain higher usability, specific user behavior in operating widgets should be taken into account. How user navigates in making specific action in such limited screen space can be brought into a design of a framework. The forthcoming work emphasizes in developing this framework. It will take the advantage of the richness of Java Swing's components and user interface system that are optimized for developing multi-platforms widgets.

6. REFERENCES

1. Caeceres, S. M., *"Standardising Widgets – Improving various aspects of client-side web applications".* PhD dissertation, Queensland University of Technology Brisbane Australia, 2007

2. Manjunath, G., Thara S. & Bosamiya H. "*Creating Personal Mobile Widgets without Programming"*, 18th International World Wide Web Conference, Madrid, Spain, 20-24, 2009

3. Zhao X., & Zou Y. *"A Framework for Incorporating Usability into Model Transformations".* MDDAUI07@Models Conference. - Nashville, Tennessee, USA, 2007

4. Java-Oracle. *"Using Java Component*". http://java.sun.com/docs/books/tutorial/uiswing/components/index.html (22 April 2010)

5. Murtagh T. P. *"Programming with Java, Swing and Squint".* Williams College, 2007

6. Bima, I., *"Materi Pelatihan Java Swing"*. http://projecttemplate.googlecode.com./files/swing-excerpt.pdf, 2003 (15 January 2011)

7. Zukowski J. *"Fundamentals of JFC/Swing: Part II"*. Java Developer Connection, Training Course, MageLang Institute, 1999

8. Wu, J.H., Shin, S.S., Chien, J.L., Chao, W.S., and Hsieh, M.C., *"An Extended MDA Method for User Interface Modeling and Transformation".* The 15th European Conference on Information Systems, St. Gallen, Swiss, June 7-9, pp 1632 – 1642, 2007

9. Horn, S. "*Microsoft Silverlight 3: A Beginner's Guide*". McGraw-Hill, New York, 2010

10. Wilson, S., Shaples, P. & Griffiths, D. "*Extending IMS Learning Design Services Using Widgets: Initial findings and proposed architecture".* Inter-science Enterprises, Vol. 2, No. 4, 1-11, 2006

11. Paller G. *"Widget Technologies on different mobile platforms"*

12. Bezemer, Cor-P. *"Automated Security Testing of AJAX Web Widget Interactions".* Master's Thesis, Delft University of Technology Delft, the Netherlands, 2008

13. Kaar, C. *"An Introduction to Widgets with particular emphasis on Mobile Widgets".* Technical Report Number 06/1/0455/009/02, Hagenberg University of Applied Sciences, October 2007

14. Bower, A., & McGlashan. B. "*Twisting the Triad – The evolution of the Dolphin Smalltalk MVP application framework".* Tutorial Paper for ESUG, 2000

15. Ritchie, Simon. *"SWT – The Standard Widget Toolkit".* Technical Report, 2002

16. Hanson R., & Tacy A., *"GWT in Action – Easy Ajax with the Google Web Toolkit"*, Chapter 4, Manning Publications, 2007.

17. Travis, D. *"Bluffers' Guide to ISO 9241".* User-Focus, UK, 2004

18. Krause, A. "*Foundation of GTK+ Development*". APRESS, USA, 2007

19. David, M., "HTML5: Designing Rich Internet Applications", Focal Press, USA, 2010

20. Summerfield, M. *"Rapid GUI Programming with Python and Qt:The Definitive Guide to PyQt Programming"*. Prentice-Hall, Michigan, 2007