# The Expressive Power of JAVA Programming: Creating User Interface Elements

Azrul Hazri Jantan, Vinod Ramanathan, and Mazura Shareena

**Abstract**—One of the success factors in software application usability is to provide intuitive Graphical User Interface (GUI) according to the user preferences and system functionalities. Many graphical authoring tools are available in the current market, but programming language such as Java also becoming very important as a programming tool in terms of creating a more generic with reusable functions and modules for the application development. However, the efforts of designing and creating GUI elements using Java can be quite a discouraging experience which requires deep understanding of some large libraries, complex functions, and fairly advanced aspects of Java programming. This paper will describe Java programming language with its GUI programming capabilities. The Java components that are being used in GUI and event-driven programming are shown and described.

**Index Terms**— JAVA programming; Graphical User Interface (GUI); Abstract Windowing Toolkit (AWT); Swing component; and user interface elements.

## 1 INTRODUCTION OF JAVA GUI PROGRAMMING

Graphical User Interface (GUI) software is often large and complex, and the correctness of GUI's code is essential to the correct execution of the overall software [1]. Virtually, all GUI applications today are built using toolkits and interface builders enabling a rapid and iterative process of prototyping and usability testing, crucial for achieving high-quality user interfaces [2]. However, in the standard of Java programming language, including its descriptive components, the elements of user interfaces are built in a form of hierarchical structure using GUI components. Each component resided on the interface layout can be defined in hierarchical nested structure, depending on the various related GUI components such as buttons, menu, text areas, combo box, and so on. All the components then must be divided into particular panels, frames, or windows. In addition, interactions between users and application can took place by implementing so called event-driven programming. The basic examples of event-driven are such as pressing a button, selecting item in combo box, and displaying multiple windows. All of these actions cause an event to be invoked, which will be reported by the system to a class named event listener. Each event listener owns event handler method for handling the events, thus it will provide appropriate responses of the user's actions**.**

All GUI components belong to a frame and a window. To locate those components in a window, Java allows designer and programmer to work with two windowing toolkit approaches, namely Abstract Windowing Toolkit (AWT) and Swing. AWT was originally being used as Java windowing standards and it was designed to design the "look and feel" of the underlying window system in which they are displayed. However, with the advent of Java 2 platform, AWT is not much widely used anymore – it was gradually replaced by a new Swing toolkit approach. Swing components are designed to have a fixed "look and feel" on all operating system platforms – making it more flexible and easy to manage GUI components. They are less prone to windowing system dependency bugs. Swing is built on AWT components, and also uses its event model. While AWT provides heavyweight components, Swing provides lightweight components and adds advanced controls such as tables because it does not require the use of native resources within the operating system [3]. One of the biggest challenges using Java Swing is that developing GUI applications can be quite a discouraging experience which requires deep understanding of some large libraries, complex functions, and fairly advanced aspects of Java programming. Wide range of GUI programs can be written by using a small subset of the Swing package. This can made possible by constructing three small classes that simplifies three of the most complex aspects of graphics programming such as 2D-graphics, layout of components, and event-handling [2][3]. Besides GUI components, Java also allows graphics drawing. The paintComponent method is supplied with a Graphics2D object (a subclass of Graphics), which contains a much richer set of drawing operations. It includes pen widths, dashed lines, shapes, image and gradient color, fill patterns, the use of arbitrary local fonts, a floating point coordinate system, and a number of coordinate transformation operations.


- *Azrul H.J. is a senior lecturer at FSKTM, Universiti Putra Malaysia, Malaysia, Selangor 43400. E-mail: azrulhazri@upm.edu.my*
- *Vinod R. is a Master student at Multimedia Dept. Universiti Putra Malaysia, Malaysia, Selangor 43400. E-mail: astrovinodsarna@ gmail.com*
- *Mazura S. is a Master student at Multimedia Dept., University Putra Malaysia, Malaysia, Selangor 43400. E-mail: mazurashareena@yahoo.com*

This paper is organized as follows. Section II will briefly describe the anatomy of a Java Swing Application. Section III will provide information on Java GUI API. In this section we will clearly elaborate four major distinctions of Java GUI, namely Container Class, Component Class, Helper Class and Layout Manager. Then, Section IV will describe how Java works with event-driven programming to support user interface components with user inputs. And finally we will conclude and summaries our discussions in Section V.

## 2 THE ANATOMY OF A SWING APPLICATION

One of the big obstacles using Abstract Windowing Toolkit (AWT) in GUI programming is to develop comprehensive GUI projects; instead of simple GUI applications. Besides, AWT is prone to platform specific bugs, because its peer-based approach relies heavily on the underlying platform. Thus, to overcome these limitations, Java 2 introduced a more versatile and flexible functions and libraries known as Java Swing [5]. Java Swing package are less dependent on the target platform and use less of the native GUI resource. All components in this package are derived from JComponent, which deals with the pluggable look and feel, keystroke handling, action object, borders and accessibility. A typical Swing application will consist of a main window with frames, user interface elements, and contents. The main shell is represented as a JFrame. Within the JFrame, an instance of JRootPane acts as a container for all other components in the frame [3]. Fig. 1 illustrates how root pane holds user interface components in a Swing application.

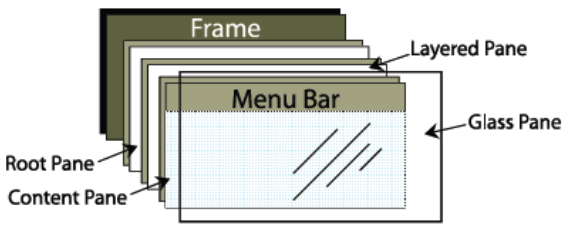


Fig.1: JRootPane's Container

Root pane can be classified into four classes; namely glass pane, layered pane, content pane, and optional menu bar. Glass pane is invisible by default that reacts like a sheet of glass to accommodate all components in the pane. Layered pane reacts to position and locate all contents on the interface layout, which consists of content pane and optional menu bar. Content pane is the other class of root pane where it is the container of the root pane's visible components. And finally, optional menu bar allows users to create and locate a menu bar on the appropriate pane layout. In the next section we will describe Swing components (user interface elements) that could reside in the frame or pane layouts.

## 3 JAVA APPLICATION PROGRAMMING INTERFACE

The GUI API in Java Swing contains the essential classes to locate both container and user interface (UI) components in a window application [3]. The GUI API classes can be divided into four groups, namely container classes, component classes, and helper classes (using Abstract Windowing Toolkit package). All UI components located in a frame can be arranged according to any positions by layout manager's class. Layout manager reacts as a medium of logical graphical presentation of page layout to define components' positions.

### 3.1 Java Container Classes

The main objective of container classes is to hold and partition user interface components. There are four major container classes in Java Swing package; namely JFrame, JPanel, JApplet, and JDialog. All classes and their subclasses are grouped in the javax.swing package. The most common container in Java is JFrame. It is the basic Java Swing container to hold user interface components in Java GUI application. To create a frame, developer can simply use and declare JFrame class to construct the windows.

```
public class newFrame {
    public static void main(String[] args) {
        JFrame frame = new JFrame ("Frame");
        frame.setSize(200,200);
        frame.setVisible(true);
    }
}
```

The descriptions of containers are described in Table 1.

TABLE 1. List of Java Swing Containers

| Container Class | Descriptions |
|---|---|
| JFrame | Container of a window that holds other Swing user interface components such as button, text field, combo box, etc. |
| JPanel | Invisible container that holds Swing user interface components. It can be nested and resided in other container. Also capable to hold graphics components. |
| JApplet | Subclass of Applet application. Need to extend JApplet to create a Swing-based Java Applet application. |
| JDialog | Normally used as a popup window or message box to receive or display additional information. It could also provide notification that an event has occurred. |

### 3.2 Java Component Classes

Component is the superclass of all Java GUI components. JComponent is an abstract superclass of all the lightweight Java Swing components. Therefore all Swing user interface components should become as an instance of JComponent. User interface components are important in GUI applications as they are all integrated and functioned as the medium of input/output interactions between users and the application. All components in Java Swing have no differences compare to other application tools components. Thus, the implementation of user interface components in Java Swing is also capable to handle and manage information manipulation similar to other common GUI applications. Table 2 summaries most common component classes resided in Java Swing component.

TABLE 2. List of Java Swing Components

| User Interface Categories | Swing Components |
|---|---|
| Menu Elements | JMenu; JToolbar; JTabbedPane; JFileChooser; |
| Form Elements | JCheckBox; JComboBox; JRadioButton; JTextField; JPasswordField; JTextArea; JList; JTextPane; JTable; |
| Navigation Elements | JButton; JSlider; JTree; JSpinner; JLabel; |

According to Table 2, we categorized all components to their common uses in general GUI applications. Components might be used for menu purposes, or as form elements, or navigation elements to navigate from one interface layout to another. These components have their own look and feel properties that can be changed throughout the uses of Java Helper Classes; and they also can be arranged or positioned into predefined interface layout (logical presentation layout) through Java Layout Manager. We will discuss these in the next sections.

### 3.3 Java Helper Classes

The main objective of Java Helper Classes is to provide the look and feel property of GUI components. The helper classes are not in Swing package, but they are defined in java.awt package. Therefore, any Swing components with different setting of properties should import java.awt package. Examples of Java Helper Classes are summaries in the following Table 3.

TABLE 3. List of Java Helper Classes

| Helper Class | Descriptions |
|---|---|
| Graphics | An abstract class that provides a graphical context for drawing strings, lines, and simple shapes. |
| Color | Set the properties of colors of GUI components such as foreground and background colors of any shapes. |
| Font | Specifies different setting of fonts for the text or caption in GUI components such as font type, size, and styles. |
| Font Metrics | An abstract class used to get the properties of the fonts. |
| Dimension | Encapsulate the size of width and height of a component in a single object. |
| Layout Manager | The layout of interface in logical presentation. It specifies how GUI components are positioned in a container. |

All Java Helper Classes in Table 3 can be used as additional layout setting for GUI components. However, developers are still capable to implement some basic functions on GUI applications without the uses of these classes.

### 3.4 Java Layout Managers

Despite of using hard-coded pixel measurements of GUI components arrangements, Java Layout Manager allows developer to manage and handle GUI components in a simpler way. In general, Java Layout Manager is an abstract level of programming to maps all user interface elements on all window system to their predefined layout positions. Each of GUI components is placed in a container such as JFrame or JPanel, so that it can easily managed by the container's layout manager. According to the layout manager setting, Java will place GUI components at the required positions. The positions are all depend on the category of layout manager. Each layout is set in containers using the setLayout (LayoutManager) method. In general, there are five common layout managers; namely flow layout, grid layout, border layout, card layout, and box layout.

1) Flow Layout: Flow layout is the simplest and default layout manager for java.awt and javax.swing components. GUI components are arranged in the container from left to right in the order in which they are added from the source codes. After a line (the current row) was filled, then the layout manager moves onto a newer line. The allignment direction is determined by three constants: FlowLayout.LEFT, FlowLayout.CENTER, and FlowLayout.RIGHT. It describes all GUI components can be alligned either from left, center, or right.

2) Grid Layout: The main consideration of grid layout is the number of rows and columns in which how it can hold and accommodate GUI components. This layout will arranges components in a grid or matrix presentation. The container reacts as a rectangular grid. Then it will be divided into equal-sized rectangles, and GUI components are placed in each rectangle, from left to right, and from current row to another row.
3) Border Layout: The basic principle of border layout manager is to partition a window into five main areas; called East, West, North, South, and Center. The position of each GUI components is defined by using adding component syntax. For example, developer can locate component either at BorderLayout.EAST, BorderLayout.WEST, BorderLayout.NORTH, BorderLayout.SOUTH, or BorderLayout.CENTER of the window. If the GUI components are resided at east and west areas, they can be stretch vertically, while components at north and south can be stretch horizontally.
4) Card Layout: Compare to previous layout manager, card layout reacts as an organisation of stacked components on a container, with only one card being visible at a time (at the top surface). The first component added is the visible component when the container is first displayed. Methods exist to go through the stack sequentially or to access a particular card.
5) Box Layout: Box layout manager allows multiple components to be laid out vertically; y-axis or horizontally; x-axis. All GUI components do not wrap on the window, so when the frame is resized the components remain in their initial arrangement. Components are arranged in the order in which they are added throughout the source codes.

## 4 EVENT DRIVEN PROGRAMMING WITH JAVA

In previous section, we have described how Java provides user interface elements for designing and developing GUI applications. Those elements are provided in Java Swing package; systematically hard-coded and they are all the same as user interface elements provided in many authoring tools. All GUI components can be made possible for displaying objects and information in a specific frame of window. However, the applications look so static since there is no execution from an event source. Event source is important in order to invoke input information and let application runs and generates outputs according to the event type.

In this section, we will describe how Java works with event-driven programming. In general, event-driven programming will allow application to receive an input from triggered event such as mouse clicked, to generate and produce necessary output to the users. Different actions from users will result different event types. Therefore, developer must distinguish every single input and output operation from the events so that the Java program could know and execute the events correctly. Table 4 summaries common GUI components, user actions, and event handler types.

TABLE 4. Examples of Event Triggered and Event Handler Types.

| GUI components / User actions | Event Handler Type |
|---|---|
| JMenuItem<br>Selecting menu item | ActionEvent |
| JButton<br>Clicking a button | ActionEvent |
| JTextField<br>Pressing enter on a text field | ActionEvent |
| JRadioButton<br>Clicking a radio button | ActionEvent;ItemEvent |
| JComboBox<br>Selecting an item | ActionEvent;ItemEvent |
| JCheckBox<br>Clicking a check box | ActionEvent;ItemEvent |

To manage an event, Java program must deal with listener; an object that will handle the whole event. Java provides several number of listener interface for every type of GUI event. The listener interface is usually named XListener for XEvent. For example, the corresponding listener interface for ActionEvent is ActionListener. Then, for each listener interface, there would be a listener method that needed to be implemented. The method for ActionListener interface is actionPerformed(ActionEvent). Another important consideration is to register method of the source object. The method registration is depends on the event type. For example, if ActionEvent was invoked, the source object would named as addActionListener(). The following Java codes show how listener interface handle and manage GUI event.

```
JButton btn = new JButton("Click Me!");
JTextField txt = new JTextField(20);
btn.addActionListener(new ActionListener() {
    public void actionPerformed (ActionEvent e) {
        txt.setText("IJIDM 2013");
    }
});
```

The output from the program above should display the following frame layout.

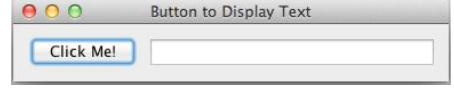


Fig. 2: Example of Event-driven Display

In Fig.2, we only provide a simple example of interface layout that consists of a button and text field. Once user click the button, actionPerformed method was invoked by ActionListener interface to display "IJIDM 2013" text on the text field components. This example has shown how Java could handle and support event-driven programming for its Java GUI components. The following table summarizes how Java deals with other event categories.

TABLE 5. Mouse Event, Key Event, and Timer Event

| Event Categories | Descriptions |
|---|---|
| Mouse Event | Handle any event input from mouse clicked, mouse pressed, mouse released, or mouse dragged on components. There are two listener interface; MouseListener to manage actions as clicked, pressed, released. MouseMotionListener is to handle actions such as mouse dragged and moving cursor. |
| Key Event | Enable the uses of keyboard to provide event input such as key pressed, key released, and key typed. KeyEvent is the handler that will return key code by getKeyChar() or getKeyCode() methods. |
| Timer Event | Mainly used to control animations in an application. It is implemented under ActionEvent with ActionListener interface to handle an object called Timer. Timer has several methods to manage the event such as start(), stop(), and delay(). |

Java provides a powerful mechanism to handle event-driven programming for its GUI applications. From the information given in this section, it is obvious that Java can support all types of events that fired by users; including from keyboard events, mouse events, and timer events. All event handlers in listener interface gave efficient and dynamic approaches on how developer could implement and develop GUI application according to the event types.

## 5 DISCUSSIONS AND SUMMARIES

In the last few years, the features offered in Abstract Windowing Toolkit (AWT) have provided Java Developer with tremendous GUI facilities to design and develop comprehensive GUI applications [6]. However, because of some limitations on portability issues and heavyweight components of AWT, Java has introduced a better version of GUI components, called Java Swing [5]. All the components in Swing environment are built on AWT components, and also use its event-driven model. While AWT provides heavyweight components, Java Swing provides more lightweight components and adds advanced controls such as tables because it does not require the use of native resources within the operating system. Besides, the numbers of GUI components offered in Swing have made Java GUI programming becoming more dynamic and comprehensive module to be used.

Creating a user interface components will always be a challenging task for Java developer [4]. In one aspect, Java GUI programming requires deep understanding of some large libraries, complex functions, and fairly advanced aspects of Java programming. Besides, the efforts of developing GUI applications involve creative process that requires significant and deep technical skills and talent from the developer sides. Developer must always create and produce a good and effective user interface according to the WYSIWYG (What You See Is What You Get) philosophy. Users of the application should easily see what they need when they need it.

Although the benefits of Java GUI programming are clear, but not all of us realised that the efforts of designing and developing GUI applications are not only focusing on the internal strength of the Java GUI programming, but few considerations must be placed into account [4]. We believe that there are some other factors that made Java GUI programming becoming a much tougher and complicated programming language for designing and developing GUI applications. The following are current challenges of efforts for Java developer to design and develop a more complex and complicated GUI applications:

1) GUI programming for tasks and user behavior
   The inherent complexity of the tasks and applications are the most challenging factor to Java Developer. Besides, it is also really hard to anticipate the behaviour of application users in performing actions in the application. Despite of considering the internal application complexity, Java developer needs a deeper look on how the application processes can be supported by GUI programming.
2) GUI programming for user personalization
   The existences of various levels of user background increase the challenges on how applications should be built according to their usability preferences and navigation capabilities. It is hard on how Java developer can distinguish different classes of users according to their needs while using the GUI components in the application.
3) GUI programming for real-time applications
   Some applications such as virtual reality and intelligent applications need a fast feedback with different modules of GUI layouts. Complex multimedia animations and simulations in GUI applications also have made the real-time programming for programming language becomes much harder. Thus, the best way of how programming language can better control and manage timing (temporal factor) for real-time applications is still a big challenge for many programmers.

4) Design guidelines and methodology for GUI programming language are not sufficient
   Most of design methods, such as hypermedia design methods and web design methods have provided full guidelines principles with the main focus of application domain and structures. There are still lacks of attentions on how GUI should be designed and developed effectively. In addition, the factor of creative GUI process must also take into consideration. While logical GUI presentation layout can be done easily, the physical GUI presentation must then depends on the creativity skills of the Java developer which does not covered in any design guidelines.
5) Increasing complexity of GUI in current applications
   Currently, there are many huge and large scopes of GUI applications available in the market. Each application considers GUI as the vital part in the application in order to attract user attentions. Thus this has becoming another challenge to all current programming languages. For example, could Java GUI programming produce the abovementioned applications? Could Java GUI programming been chosen as another alternative for higher level programming language compares to other programming language or authoring tools?

In overall, the use of Java GUI programming in many GUI applications which can be implemented in various operating system platforms has provides good facilities for the developer to create and develop a simple yet comprehensive user interface. Conclusion can be made that Java is a platform independent, effective, robust, simple, and secure programming language. In the future, a tool which is easier to learn by the designer and developer could be developed and hence, it can extensively increase the efficiency of designing the GUI layout. We can expect fast innovations in GUI software design in the future.

## ACKNOWLEDGMENT

The author would like to acknowledge the funding support for this research proceeding from the Research Management Centre (RMC), Universiti Putra Malaysia, 43400 Selangor DE, Malaysia, under the grant of Research University Grant Scheme (RUGS) – Project 05-01-11-1249RU. The author would also like to thank the anonymous reviewers for their valuable feedbacks and comments.

## REFERENCES

[1] B. A. Myers, "UIMSs, Toolkits, Interface Builders", Human Computer Interaction Institute, Carnegie Mellon University, May 1996.

[2] B. A. Myers, S. E. Hudson, and R. Pausch, "Past, present, and future of user interface software tools", ACM Transactions on Computer-Human Interaction (TOCHI), Volume 7, Issue 1, pp. 3-28, March 2000.

[3] J. W. Cooper "A Beginners Introduction to Java 2", IBM Thomas J Watson Research Center, August 2000.

[4] Brad Myers, "Challenges of HCI Design and Implementation", ACM Interactions, vol. 1, no. 1, p. 73-83, Jan 1994.

[5] D. Agushinta R., A. Tarigan, E. Wisnu Moyo, F. H. Siburian, and S. Widiyanto, "The use of JAVA Swing's component to develop a widget," International Journal of Human Computer Interaction (IJHCI), vol. 1, no. 4, 2011.

[6] T. Mariusz, "GUIs without pain – the declarative way," ISBN 978-953-307-084-1, p. 270, 2010.

**Azrul Hazri Jantan** obtained his bachelor degree in Computer Science from Universiti Teknologi Malaysia and Master degree in Information Technology from University of Adelaide, Australia. He received his PhD degree in Universiti Sains Malaysia, Penang in 2009. He is working as Senior Lecturer in Faculty of Computer Science and Information Technology (Multimedia Department), Universiti Putra Malaysia. His current research interests include human computer interaction, web engineering (design), and hypermedia design and development.