Introduction To GUI

GUIs stand Graphical User Interfaces that allows users to interact with electronic devices through graphical icons and visual indicators, instead of text-based user interfaces, typed command labels or text navigation.

You can *reuse* the graphics classes provided in Java for constructing your own Graphical User Interface (GUI) applications. Using these graphic classes are not so difficult, if you follow the API documentation, samples and templates provided.

There are current three sets of Java APIs for graphics programming: AWT (Abstract Windowing Toolkit), Swing and JavaFX.

1. AWT API was introduced in JDK 1.0. Most of the AWT components have become obsolete and should be replaced by newer Swing components.
2. Swing API, a much more comprehensive set of graphics libraries that enhances the AWT, was introduced as part of Java Foundation Classes (JFC) after the release of JDK 1.1. JFC consists of Swing, Java2D, Accessibility, Internationalization, and Pluggable Look-and-Feel Support APIs. JFC has been integrated into core Java since JDK 1.2.

   We have been using JDK 1.8 or JDK 1.9 which has since been named JDK 8 and JDK 9 respectively.

3. The latest JavaFX, which was integrated into JDK 8, is meant to replace Swing.

AWT Packages

AWT is huge! It consists of 12 packages of 370 classes (Swing is even bigger, with 18 packages of 737 classes as of JDK 8). Fortunately, only 2 packages - `java.awt` and `java.awt.event` - are commonly-used.

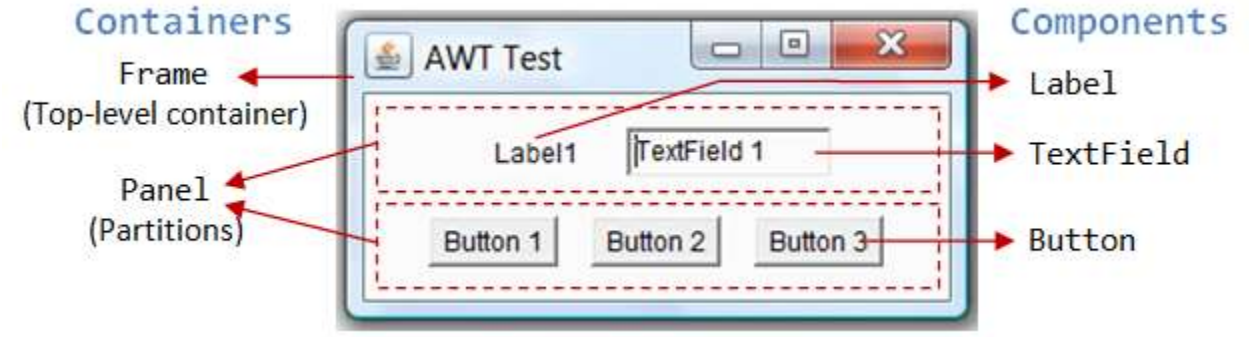
1. The `java.awt` package contains the *core* AWT graphics classes:
   - GUI Component classes, such as `Button`, `TextField`, and `Label`.
   - GUI Container classes, such as `Frame` and `Panel`.
   - Layout managers, such as `FlowLayout`, `BorderLayout` and `GridLayout`.
   - Custom graphics classes, such as `Graphics`, `Color` and `Font`.

2. The `java.awt.event` package supports event handling:
   - Event classes, such as `ActionEvent`, `MouseEvent`, `KeyEvent` and `WindowEvent`,
   - Event Listener Interfaces, such as `ActionListener`, `MouseListener`, `MouseMotionListener`, `KeyListener` and `WindowListener`,
   - Event Listener Adapter classes, such as `MouseAdapter`, `KeyAdapter`, and `WindowAdapter`.

AWT provides a *platform-independent* and *device-independent* interface to develop graphic programs that runs on all platforms, including Windows, Mac OS X, and Unixes.

Introduction To GUI

Containers and Components



There are two types of GUI elements:

1. *Component*: Components are elementary GUI entities, such as `Button`, `Label`, and `TextField`.
2. *Container*: Containers, such as `Frame` and `Panel`, are used to *hold components in a specific layout* (such as `FlowLayout` or `GridLayout`). A container can also hold sub-containers.

In the above figure, there are three containers: a `Frame` and two `Panels`. A `Frame` is the *top-level container* of an AWT program. A `Frame` has a title bar (containing an icon, a title, and the minimize/maximize/close buttons), an optional menu bar and the content display area. A `Panel` is a *rectangular area* used to group related GUI components in a certain layout. In the above figure, the top-level `Frame` contains two `Panels`. There are five components: a `Label` (providing description), a `TextField` (for users to enter text), and three `Buttons` (for user to trigger certain programmed actions).

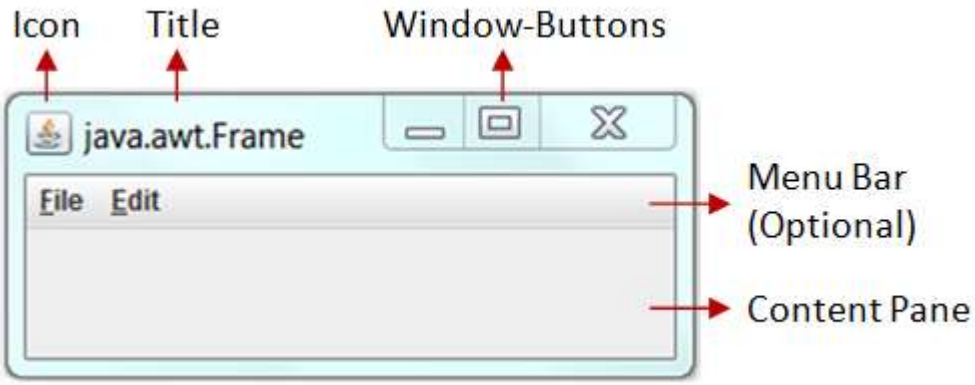
In a GUI program, a component must be kept in a container. You need to identify a container to hold the components. Every container has a method called `add(Component c)`. A `container` (say `c`) can invoke `c.add(aComponent)` to add `aComponent` into itself. For example,

```
Panel pnl = new Panel();          // Panel is a container
Button btn = new Button("Press"); // Button is a component
pnl.add(btn);                     // The Panel container adds a Button component
```

AWT Container Classes

Top-Level Containers: Frame, Dialog and Applet

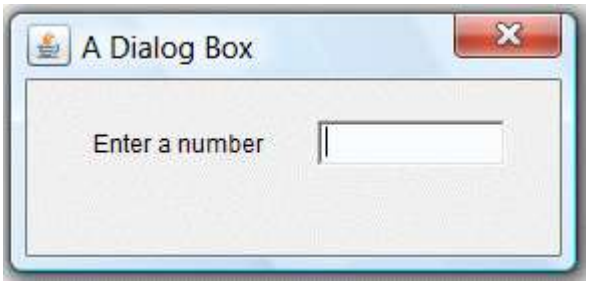
Each GUI program has a *top-level container*. The commonly-used top-level containers in AWT are `Frame`, `Dialog` and `Applet`:



A `Frame` provides the "main window" for your GUI application. It has a title bar (containing an icon, a title, the minimize, maximize/restore-down and close buttons), an optional menu bar, and the content display area. To write a GUI program, we typically start with a subclass extending from `java.awt.Frame` to inherit the main window as follows:

```
import java.awt.Frame;  // Using Frame class in package java.awt

// A GUI program is written as a subclass of Frame - the top-level container
// This subclass inherits all properties from Frame, e.g., title, icon, buttons, content-pane
public class MyGUIProgram extends Frame {

   // private variables
   ......

   // Constructor to setup the GUI components
   public MyGUIProgram() { ...... }

   // methods
   ......
   ......

   // The entry main() method
   public static void main(String[] args) {
      // Invoke the constructor (to setup the GUI) by allocating an instance
      MyGUIProgram prog = new MyGUIProgram();
   }
}
```

Introduction To GUI



An AWT `Dialog` is a "*pop-up window*" used for interacting with the users. A `Dialog` has a title-bar (containing an icon, a title and a close button) and a content display area, as illustrated.
An AWT `Applet` (in package `java.applet`) is the top-level container for an applet, which is a Java program running inside a browser.

Secondary Containers: `Panel` and `ScrollPane`

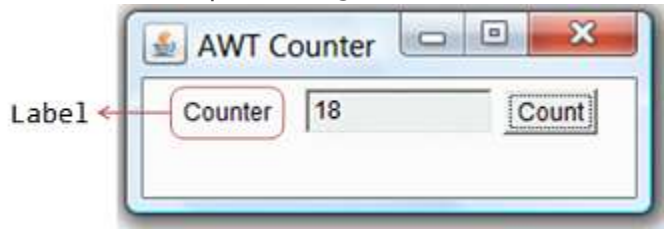
Secondary containers are placed inside a top-level container or another secondary container. AWT provides these secondary containers:

`Panel`: a rectangular box used to *layout* a set of related GUI components in pattern such as grid or flow.
`ScrollPane`: provides automatic horizontal and/or vertical scrolling for a single child component.

AWT `Component` Classes

AWT provides many ready-made and reusable GUI components in package `java.awt`. The frequently-used are: `Button`, `TextField`, `Label`, `Checkbox`, `CheckboxGroup` (radio buttons), `List`, and `Choice`, as illustrated below.

AWT GUI Component: `java.awt.Label`



A `java.awt.Label` provides a descriptive text string. Take note that `System.out.println()` prints to the system console, NOT to the graphics screen. You could use a `Label` to label another component (such as text field) to provide a text description.

Introduction To GUI

Check the JDK API specification for `java.awt.Label`.

Constructors

```
public Label(String strLabel, int alignment); // Construct a Label with the given text String, of the text alignment
public Label(String strLabel);                // Construct a Label with the given text String
public Label();                               // Construct an initially empty Label
```

The `Label` class has three constructors:

1. The first constructor constructs a `Label` object with the given text string in the given alignment. Note that three `static` constants `Label.LEFT`, `Label.RIGHT`, and `Label.CENTER` are defined in the class for you to specify the alignment (rather than asking you to memorize arbitrary integer values).
2. The second constructor constructs a `Label` object with the given text string in default of left-aligned.
3. The third constructor constructs a `Label` object with an initially empty string. You could set the label text via the `setText()` method later.

Constants (`final static` fields)

```
public static final LEFT;    // Label.LEFT
public static final RIGHT;   // Label.RIGHT
public static final CENTER;  // Label.CENTER
```

These three constants are defined for specifying the alignment of the `Label`'s text, as used in the above constructor.

Public Methods

```
// Examples
public String getText();
public void setText(String strLabel);
public int getAlignment();
public void setAlignment(int alignment);  // Label.LEFT, Label.RIGHT, Label.CENTER
```

The `getText()` and `setText()` methods can be used to read and modify the `Label`'s text. Similarly, the `getAlignment()` and `setAlignment()` methods can be used to retrieve and modify the alignment of the text.

Constructing a Component and Adding the Component into a Container

Three steps are necessary to create and place a GUI component:

1. Declare the component with an *identifier* (*name*);
2. Construct the component by invoking an appropriate constructor via the `new` operator;
3. Identify the container (such as `Frame` or `Panel`) designed to hold this component. The container can then add this component onto itself via *`aContainer`*`.add(`*`aComponent`*`)` method. Every container has a `add(Component)` method. Take note that it is the container that actively and explicitly adds a component onto itself, NOT the other way.

Example

```
Label lblInput;                     // Declare an Label instance called lblInput
lblInput = new Label("Enter ID");   // Construct by invoking a constructor via the new operator
add(lblInput);                      // this.add(lblInput) - "this" is typically a subclass of Frame
lblInput.setText("Enter password"); // Modify the Label's text string
lblInput.getText();                 // Retrieve the Label's text string
```

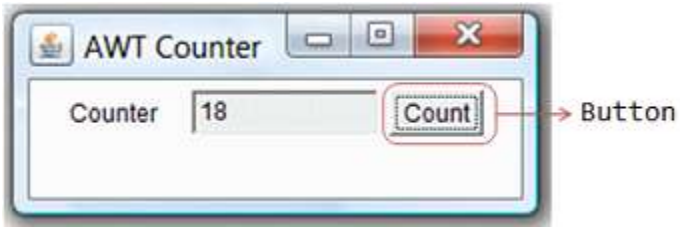
An Anonymous Instance

You can create a `Label` without specifying an identifier, called *anonymous instance*. In the case, the Java compiler will assign an *anonymous identifier* for the allocated object. You will not be able to reference an anonymous instance in your program after it is created. This is usually alright for a `Label` instance as there is often no need to reference a `Label` after it is constructed.

Example

```
// Allocate an anonymous Label instance.
// "this" container adds the instance.
// You CANNOT reference an anonymous instance to carry out further operations.
add(new Label("Enter Name: ", Label.RIGHT));

// Same as
Label xxx = new Label("Enter Name: ", Label.RIGHT)); // xxx assigned by compiler
add(xxx);
```

AWT GUI Component: `java.awt.Button`



A `java.awt.Button` is a GUI component that triggers a certain programmed *action* upon clicking.

Constructors

```
public Button(String btnLabel);
   // Construct a Button with the given label
public Button();
   // Construct a Button with empty label
```

The `Button` class has two constructors. The first constructor creates a `Button` object with the given label painted over the button. The second constructor creates a `Button` object with no label.

Public Methods

```
public String getLabel();
```

```
   // Get the label of this Button instance
public void setLabel(String btnLabel);
   // Set the label of this Button instance
public void setEnable(boolean enable);
   // Enable or disable this Button. Disabled Button cannot be clicked.
```

The `getLabel()` and `setLabel()` methods can be used to read the current label and modify the label of a button, respectively.

Note: The latest Swing's `JButton` replaces `getLabel()/setLabel()` with `getText()/setText()` to be consistent with all the components. We will describe Swing later.

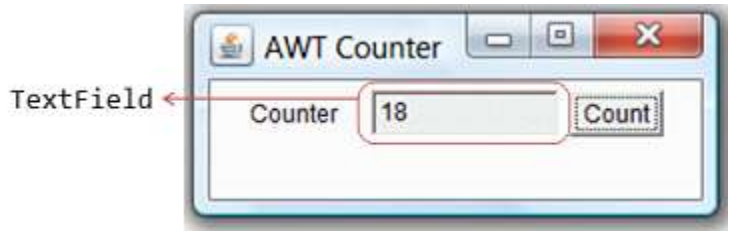
Event

Clicking a button fires a so-called `ActionEvent` and triggers a certain programmed action. I will explain event-handling later.

Example

```
Button btnColor = new Button("Red"); // Declare and allocate a Button instance called btnColor
add(btnColor);                       // "this" Container adds the Button
...
btnColor.setLabel("Green");          // Change the button's label
btnColor.getLabel();                 // Read the button's label
...
add(Button("Blue"));    // Create an anonymous Button. It CANNOT be referenced later
```

AWT GUI Component: `java.awt.TextField`



A `java.awt.TextField` is single-line text box for users to enter texts. (There is a multiple-line text box called `TextArea`.) Hitting the "ENTER" key on a `TextField` object fires an `ActionEvent`.

Introduction To GUI

Constructors

```
public TextField(String initialText, int columns);
   // Construct a TextField instance with the given initial text string with the number of columns.
public TextField(String initialText);
   // Construct a TextField instance with the given initial text string.
public TextField(int columns);
   // Construct a TextField instance with the number of columns.
```

Public Methods

```
public String getText();
   // Get the current text on this TextField instance
public void setText(String strText);
   // Set the display text on this TextField instance
public void setEditable(boolean editable);
   // Set this TextField to editable (read/write) or non-editable (read-only)
```

Event

Hitting the "ENTER" key on a `TextField` fires a `ActionEvent`, and triggers a certain programmed action.

Example

```
TextField tfInput = new TextField(30); // Declare and allocate an TextField instance called tfInput
add(tfInput);                          // "this" Container adds the TextField
TextField tfResult = new TextField();  // Declare and allocate an TextField instance called tfResult
tfResult.setEditable(false) ;          // Set to read-only
add(tfResult);                         // "this" Container adds the TextField
......
// Read an int from TextField "tfInput", square it, and display on "tfResult".
// getText() returns a String, need to convert to int
int number = Integer.parseInt(tfInput.getText());
number *= number;
// setText() requires a String, need to convert the int number to String.
tfResult.setText(number + "");
```

Take note that `getText()/SetText()` operates on `String`. You can convert a `String` to a primitive, such as `int` or `double` via `static` method `Integer.parseInt()` or `Double.parseDouble()`. To convert a primitive to a `String`, simply concatenate the primitive with an empty `String`.

Introduction To GUI

Sample program that creates a window and components

```
import javax.swing.*; //make sure you import this library!
import java.awt.*;
import java.awt.event.*;

 class IntroToSwing
{
    public static void main(String[] args)
    {
        /* Lets start learning about JFrames!
         * A JFrame is a top-level window with a title and a border.
         */

        //1. Create the JFrame.
        //we are creating an object from the JFrame class called frame and passing it the value FrameDemo
        JFrame frame = new JFrame("FrameDemo");

        //2. Optional: What happens when the frame closes?
        //this line of code causes the entire program to close when the window closes
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


        //3. Create components and put them in the frame.
        //...create emptyLabel...
        //create object emptylabel from class JLabel and pass it nothing
        JLabel sampleLabel = new JLabel("My Label");
        JButton sampleButton = new JButton("The Button");

        // get the container of the frame
        Container content = frame.getContentPane();
        content.setLayout(new FlowLayout());

        // add this label to the container
        content.add(sampleLabel);
        content.add(sampleButton);

        //4. Size the frame.
        //this code adjusts the size of the frame and fit the components within the frame.
        frame.setPreferredSize(new Dimension(500,500));
         frame.pack();

        //5. Display the JFrame.
        frame.setVisible(true);
    }
}
```

What is a container?
A container is a parent class that holds one or more child components. A container has a layout that determines how the child components are arranged within the container.

Imagine the container as being a garage, which can hold many things. Some things that this garage can hold are tool boxes (like menu bars), or boxes to hold more items (like content panes)

Action Listeners are used to define what should be done when a user performs certain operation.
An action event occurs, whenever an action is performed by the user. Examples: When the user clicks a button, chooses a menu item, presses Enter in a text field. The result is that an actionPerformed message is sent to all action listeners that are registered on the relevant component.