

LESSON XII. GUI and Event Programming (1/2)

Trịnh Tuấn Đạt

1

Objectives

- After this lesson, students (learners) can:
 - Understand the concept of “GUI Programming”
 - Understand the concepts of “Container” and “Component”
 - Know how to create AWT containers and AWT components
 - Know how to organize AWT components inside an AWT container.
 - Understand how to handle AWT events, using different ways
 - Write many demo AWT applications.

2

Content

I. Introduction

- II. Programming GUI with AWT
- III. AWT Event-Handling

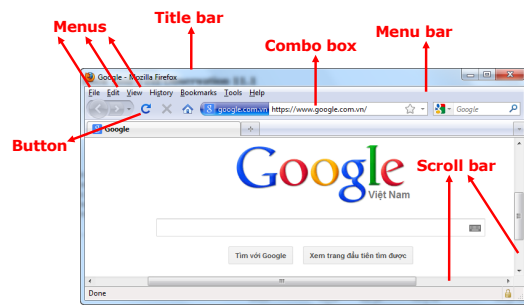
3

I. Introduction

- A graphical user interface - GUI (pronounced "GOO-ee"):
 - is a type of user interface
 - allows users to interact with electronic devices using images rather than text commands
- Why use term GUI?
 - The first interactive user interfaces to computers were not graphical

4

1. Introduction



5

Java APIs for graphics programming

- Two core sets of Java APIs for graphics programming:
 - AWT (Abstract Windowing Toolkit)
 - Swing
- AWT:
 - introduced in JDK 1.0
 - should be replaced by newer Swing components
- Swing:
 - enhances AWT
 - integrated into core Java since JDK 1.2
- Others:
 - Eclipse's Standard Widget Toolkit (SWT)
 - Google Web Toolkit (GWT)
 - 3D Graphics API such as Java bindings for OpenGL (JOGL) and Java3D.

6

Content

- I. Introduction
- II. Programming GUI with AWT**
- III. AWT Event-Handling

7

II. Programming GUI with AWT

- 2.1 AWT Packages
- 2.2 Containers and Components
- 2.3 AWT Container Classes
- 2.4 AWT Component Classes
- 2.5. Layout Managers

8

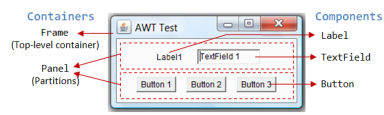
2.1. AWT Packages

- Huge: there are 12 packages.
 - Only 2 packages: java.awt & java.awt.event are commonly-used
 - Platform-independent & device-independent
- Core graphics classes of java.awt:
 - GUI Component classes (such as Button, TextField, and Label),
 - GUI Container classes (such as Frame, Panel, Dialog and ScrollPane),
 - Layout managers (such as FlowLayout, BorderLayout and GridLayout),
 - Custom graphics classes (such as Graphics, Color and Font).
- java.awt.event package supports event handling
 - Event classes (such as ActionEvent, MouseEvent, KeyEvent and WindowEvent),
 - Event Listener Interfaces (such as ActionListener, MouseListener, KeyListener and WindowListener),
 - Event Listener Adapter classes (such as MouseAdapter, KeyAdapter, and WindowAdapter).

9

2.2. Containers and Components

- Two types of GUI elements:
 - *Component*: elementary GUI entities (Button, Label, TextField.)
 - *Container* (Frame, Panel and Applet): *hold components in a specific layout*. A container can also hold sub-containers



10

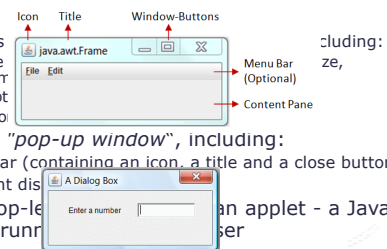
2.3. AWT Container Classes

- **Top-level AWT containers: Frame, Dialog and Applet.**
- **Frame:**
 - provides "main window" for GUI application, including:
 - a title bar (containing an icon, a title, the minimize, maximize/restore-down and close buttons)
 - an optional menu bar
 - the content display area.
- **Dialog:** a "pop-up window", including:
 - a title-bar (containing an icon, a title and a close button)
 - a content display area
- **Applet:** top-level container for an applet - a Java program running inside a browser

11

2.3. AWT Container Classes

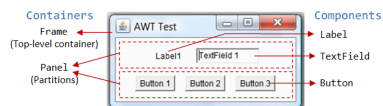
- **Top-level AWT containers: Frame, Dialog and Applet.**
- **Frame:**
 - provides
 - a title bar (containing an icon, a title, the minimize, maximize/restore-down and close buttons)
 - an optional menu bar
 - the content display area.
- **Dialog:** a "pop-up window", including:
 - a title-bar (containing an icon, a title and a close button)
 - a content display area
- **Applet:** top-level container for an applet - a Java program running inside a browser



12

2.3. AWT Container Classes

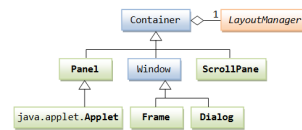
- **Secondary Containers: Panel, ScrollPane**
 - are placed inside a top-level container or another secondary container
- **Panel:**
 - a rectangular box (partition) under a higher-level container,
 - used to *layout* a set of related GUI components
- **ScrollPane:** provides automatic horizontal and/or vertical scrolling for a single child component



13

2.3. AWT Container Classes

- **Hierarchy of the AWT Container Classes**



14

2.4. AWT Component Classes

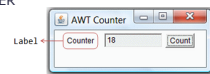
- AWT provides many GUI components:
 - Button, TextField, Label, Checkbox, CheckboxGroup (radio buttons), List, and Choice



15

2.4. AWT Component Classes

- **java.awt.Label:** provides a text description message.
- **Constructors:**
 - // Construct a Label with the given text String, of the text alignment
 - `public Label(String strLabel, int alignment);`
 - `public Label(String strLabel);` // Construct a Label with the given text
 - `public Label();` // Construct an initially empty Label
- **Constants:**
 - `public static final LEFT; // Label.LEFT`
 - `public static final RIGHT; // Label.RIGHT`
 - `public static final CENTER; // Label.CENTER`
- **Public methods:**
 - `public String getText();`
 - `public void setText(String strLabel);`
 - `public int getAlignment();`
 - `public void setAlignment(int alignment);`



16

2.4. AWT Component Classes

- To construct a Component and add into a Container:
 - Declare the component with an *identifier*
 - Construct the component
 - Identify the container designed to hold this component. Use `add` method:
 - Ex: `aContainer.add(aComponent)`

- Example:

```
Label lblInput;
lblInput = new Label("Enter ID");
this.add(lblInput);
lblInput.setText("Enter password");
lblInput.getText();
```

17

2.4. AWT Component Classes

- java.awt.Button**: triggers a certain programmed *action* upon clicking.

- Constructors:

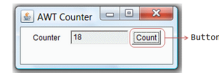
- `public Button(String buttonLabel);`
- `public Button(String buttonLabel);`

- Public Methods

- `public String getLabel();`
- `public void setLabel(String buttonLabel);`
- `public void setEnabled(boolean enable);`

- Example:

- `Button btnColor = new Button("Red");`
- `this.add(btnColor);`
- `...`
- `btnColor.setLabel("green");`
- `btnColor.getLabel();`



18

2.4. AWT Component Classes

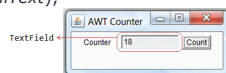
- java.awt.TextField**: single-line text box to enter texts. (**TextArea**: multiple-line text box)

- Constructor:

- `public TextField(String strInitialText, int columns);`
- `public TextField(String strInitialText);`
- `public TextField(int columns);`

- Public methods:

- `public String getText();`
- `public void setText(String strText);`
- `public void setEditable(boolean editable);`



19

2.5. Layout Managers

- Layout manager: arranges a container's components
- Layout managers from AWT: (in package `java.awt`)
 - `FlowLayout`
 - `GridLayout`
 - `BorderLayout`
 - `GridBagLayout`
 - `BoxLayout`
 - `CardLayout`

20

Set a layout manager

- A container has a `setLayout()` method to set its layout manager:
 - `public void setLayout(LayoutManager mgr)`
- To set up the layout of a Container:
 - Construct an instance of the chosen layout object, e.g., `new FlowLayout()`
 - Invoke the `setLayout()` method, with the layout object created as the argument;
 - Place the GUI components into the Container using the `add()` method in the correct order; or into the correct zones.
- Example:


```
Panel p = new Panel();
p.setLayout(new FlowLayout());
p.add(new JLabel("One"));
p.add(new JLabel("Two"));
p.add(new JLabel("Three"));
```

21

Layout managers

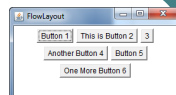
- Construct a Panel with a layout
 - `// Construct a Panel in the given layout`
 - `// By default, Panel (and JPanel) has FlowLayout`
 - `public void Panel (LayoutManager layout)`
 - Example: create a Panel in BorderLayout


```
Panel mainPanel = new Panel(new BorderLayout());
```
- To get layout of a Container: use `getLayout()`

```
Panel awtPanel = new Panel();
System.out.println(awtPanel.getLayout());
//java.awt.FlowLayout [hgap=5, vgap=5, align=center]
```

22

a. FlowLayout



- Inside a Container with FlowLayout:
 - components are arranged from left-to-right (in the added order)
 - when one row is filled, new row will be started
- Constructors:
 - `public FlowLayout();`
 - `public FlowLayout(int align);`
 - `public FlowLayout(int align, int hgap, int vgap);`
- Align:
 - `FlowLayout.LEFT` (or `LEADING`)
 - `FlowLayout.RIGHT` (or `TRAILING`)
 - `FlowLayout.CENTER`
- `hgap`, `vgap`: horizontal/vertical gap between the components.
- By default: `hgap=5`, `vgap=5`, `align=CENTER`

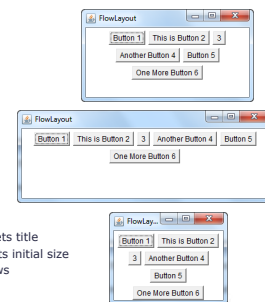
23

FlowLayout example

```
import java.awt.*;
import java.awt.event.*;
public class AWTFlowLayout extends Frame {
    public AWTFlowLayout () {
        setLayout(new FlowLayout());
        add(new Button("Button 1"));
        add(new Button("This is Button 2"));
        add(new Button("3"));
        add(new Button("Another Button 4"));
        add(new Button("Button 5"));
        add(new Button("One More Button 6"));
    }

    setTitle("FlowLayout"); // "this" Frame sets title
    setSize(280, 150); // "this" Frame sets initial size
    setVisible(true); // "this" Frame shows
}

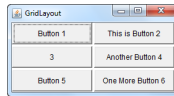
public static void main(String[] args) {
    new AWTFlowLayout(); // Let the constructor do the job
}
```



24

b. GridLayout

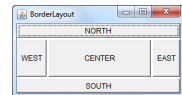
- Inside a Container with FlowLayout:
 - components are arranged in a grid of rows and columns
 - components are added in a left-to-right, top-to-bottom manner in the added order
- Constructor:
 - `public GridLayout(int rows, int columns);`
 - `public GridLayout(int rows, int columns, int hgap, int vgap);`
- By default: rows=1, cols=0, hgap=0, vgap=0



25

c. BorderLayout

- With BorderLayout, container is divided into 5 zones: EAST, WEST, SOUTH, NORTH, and CENTER
- To add a components:
 - `aContainer.add(aComponent, aZone)`
 - `aZone`: can be
 - BorderLayout.NORTH (or PAGE_START)
 - BorderLayout.SOUTH (or PAGE_END)
 - BorderLayout.WEST (or LINE_START)
 - BorderLayout.EAST (or LINE_END)
 - BorderLayout.CENTER
 - `aContainer.add(aComponent)`: adds the component to the CENTER
- No need to add components to all the 5 zones
- Constructors:
 - `public BorderLayout();`
 - `public BorderLayout(int hgap, int vgap);`
 - By default hgap=0, vgap=0



26

Content

- I. Introduction
- II. Programming GUI with AWT
- III. AWT Event-Handling**

27

III. AWT Event-Handling

- 3.1. Introduction
- 3.2. Event-Handling Steps
- 3.3. Available pairs of Event and Listener
- 3.4. Adapters

28

3.1. Introduction

- Event-handling model: "Event-driven"
 - When event has been fired (by user input): a piece of event-handling codes is executed
- Package java.awt.event: contains AWT's event-handling classes
- 3 objects involved in the event-handling: **source**, **listener**, **event**
 - **source** object interacts with the user to create an **event** object
 - **event** object will be messaged to all the *registered listener* objects
 - appropriate event-handler method of the **listener(s)** is called-back to provide the response

29

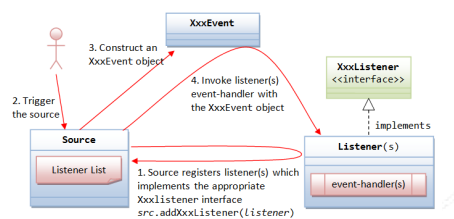
3.1. Introduction

- Use **subscribe-publish** or **observable-observer** design pattern:
 - The **listener(s)** must be registered with the **source** to express interest for a certain **event** triggered on a **source**
 - The **listener(s)** "subscribes" to an **event** of a **source**, and the **source** "publishes" the **event** to all its subscribers upon activation

30

3.2. Event-Handling Steps

- Use **subscribe-publish** or **observable-observer** design pattern:



31

a. Source object registers for a certain type of event

- The **source** & **listener** understand each other via an agreed-upon interface
- 3 steps: (to support XxxEvent event type for a Source)
 - Declare an interface called XxxListener, container the names of the handler methods
 - Listeners interested in the XxxEvent must implement the XxxListener interface
 - Source has to maintain the list of listener object(s).
 - public void addXxxListener(XxxListener l);
 - public void removeXxxListener(XxxListener l);

32

b. Example to handle MouseEvent

- Step 1: Declare MouseListener interface (by awt)

```
interface MouseListener {
    // Called back upon mouse-button pressed
    public void mousePressed(MouseEvent evt);
    // Called back upon mouse-button released
    public void mouseReleased(MouseEvent evt);
    // Called back upon mouse-button clicked (pressed and released)
    public void mouseClicked(MouseEvent evt);
    // Called back when mouse pointer entered the component
    public void mouseEntered(MouseEvent evt);
    // Called back when mouse pointer exited the component
    public void mouseExited(MouseEvent evt);
}
```

33

b. Example to handle MouseEvent

- Step 2: Create a Listener class implement MouseListener interface

```
class MyMouseListener implements MouseListener {
    @Override
    public void mousePressed(MouseEvent e) {
        System.out.println("Mouse-button pressed!");
    }
    @Override
    public void mouseReleased(MouseEvent e) {
        System.out.println("Mouse-button released!");
    }
    @Override
    public void mouseClicked(MouseEvent e) {
        System.out.println("Mouse-button clicked (pressed and released)!");
    }
    @Override
    public void mouseEntered(MouseEvent e) {
        System.out.println("Mouse-pointer entered the source component!");
    }
    @Override
    public void mouseExited(MouseEvent e) {
        System.out.println("Mouse exited-pointer the source component!");
    }
}
```

34

b. Example to handle MouseEvent

- Step 3: Create a Listener class implement MouseListener interface

```
import java.awt.*;
public class ButtonEventExample extends Frame {
    public ButtonEventExample () {
        setLayout(new FlowLayout());
        Button b = new Button("Button");
        add(b);
        b.addMouseListener(new MyMouseListener());

        setTitle("Button Event Example"); // "this" Frame sets title
        setSize(280, 150); // "this" Frame sets initial size
        setVisible(true); // "this" Frame shows
    }
    public static void main(String[] args) {
        new ButtonEventExample(); // Let the constructor do the job
    }
}
```

35

3.3. Available pairs of Event and Listener

- a. ActionEvent and ActionListener Interface
- b. WindowEvent and WindowListener Interface
- c. MouseEvent and MouseListener Interface
- d. MouseEvent and MouseMotionListener Interface
- e. KeyEvent and KeyListener Interface
- and more:
 - <http://docs.oracle.com/javase/1.4.2/docs/api/java/awt/event/package-summary.html>

36

a. ActionEvent and ActionListener Interface

- To fire an ActionEvent
 - Click a Button
 - Pushing the "enter" key on a TextField
- The ActionEvent will be sent to all listeners
 - Listener for ActionEvent must implement ActionListener interface.

```
interface ActionListener {
    // Called back upon button clicked, enter key pressed
    public void actionPerformed(ActionEvent e);
}
```

37

a. ActionEvent and ActionListener Interface-Example

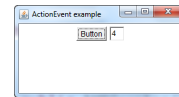
```
public class AWTCounter extends Frame implements ActionListener {
    public int count = 0;
    private TextField txt;

    public AWTCounter(){
        setLayout(new FlowLayout());
        Button b = new Button("Button");
        add(b);
        b.addActionListener(this);
        txt = new TextField();
        add(txt);

        setTitle("ActionEvent example");
        setSize(280, 150);
        setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent evt) {
        count++;
        txt.setText(count + "");
    }

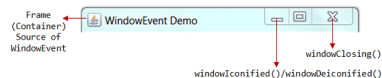
    public static void main(String args[]){
        new AWTCounter();
    }
}
```



38

b. WindowEvent and WindowListener Interface

- A WindowEvent is fired when a window (e.g., Frame) has been:
 - opened/closed
 - activated/deactivated
 - iconified/deiconified
 via the 3 buttons at the top-right corner or other means.
- The source of a WindowEvent shall be a **top-level** window-container such as Frame.



39

b. WindowEvent and WindowListener Interface

- A WindowEvent listener must implement WindowListener interface.
- ```
/* Called-back when the user attempts to close the window by clicking the window close button. This is the most-frequently used handler*/
public void windowClosing(WindowEvent e);
/* Called-back the first time a window is made visible. */
public void windowOpened(WindowEvent e);
/* Called-back when a window has been closed as the result of calling dispose on the window.*/
public void windowClosed(WindowEvent e);
/* Called-back when the Window is set to be the active Window.*/
public void windowActivated(WindowEvent e);
/* Called-back when a Window is no longer the active Window*/
public void windowDeactivated(WindowEvent e);
/* Called-back when a window is changed from a normal to a minimized state.*/
public void windowIconified(WindowEvent e);
/* Called-back when a window is changed from a minimized to a normal state*/
public void windowDeiconified(WindowEvent e);
```

40

### c. MouseEvent and MouseListener Interface

- A MouseEvent is fired when you
  - press, release, or click (press followed by release) a mouse-button (left or right button) at the source object;
  - or position the mouse-pointer at (enter) and away (exit) from the source object.
- A MouseEvent listener must implement the MouseListener interface
 

```
public void mouseClicked(MouseEvent e);
public void mousePressed(MouseEvent e);
public void mouseReleased(MouseEvent e);
public void mouseEntered(MouseEvent e);
public void mouseExited(MouseEvent e);
```
- Example already presented

41

### d. MouseEvent and MouseMotionListener Interface

- A MouseEvent is also fired when we moved and dragged the mouse pointer at the source object.
  - But we need to use MouseMotionListener to handle the mouse-move and mouse-drag.
- The MouseMotionListener interface:

```
interface MouseMotionListener{
 /* Called-back when a mouse-button is pressed on the
 source component and then dragged.*/
 public void mouseDragged(MouseEvent e)
 /* Called-back when the mouse-pointer has been moved onto
 the source component but no buttons have been pushed.*/
 public void mouseMoved(MouseEvent e)
}
```

42

### Example

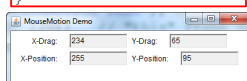
```
public class MouseMotionDemo extends Frame
 implements MouseMotionListener {
```

```
 private TextField tfMouseDragX;
 private TextField tfMouseDragY;
 private TextField tfMousePositionX;
 private TextField tfMousePositionY;
```

```
 @Override
 public void mouseMoved(MouseEvent e) {
 tfMousePositionX.setText(e.getX() + "");
 tfMousePositionY.setText(e.getY() + "");
 }
```

```
 @Override
 public void mouseDragged(MouseEvent e) {
 tfMouseDragX.setText(e.getX() + "");
 tfMouseDragY.setText(e.getY() + "");
 }
```

```
 public static void main(String[] args) {
 new MouseMotionDemo();
 }
```



```
public MouseMotionDemo() {
 setLayout(new FlowLayout());

 add(new Label("X-Drag: "));
 tfMouseDragX = new TextField(10);
 tfMouseDragX.setEditable(false);
 add(tfMouseDragX);
 add(new Label("Y-Drag: "));
 tfMouseDragY = new TextField(10);
 tfMouseDragY.setEditable(false);
 add(tfMouseDragY);
```

```
 add(new Label("X-Position: "));
 tfMousePositionX = new TextField(10);
 tfMousePositionX.setEditable(false);
 add(tfMousePositionX);
 add(new Label("Y-Position: "));
 tfMousePositionY = new TextField(10);
 tfMousePositionY.setEditable(false);
 add(tfMousePositionY);
```

```
 addMouseMotionListener(this);
```

```
 setTitle("MouseMotion Demo"); // "this" Frame sets title
 setSize(400, 120); // "this" Frame sets initial size
 setVisible(true); // "this" Frame shows
}
```

43

### e. KeyEvent and KeyListener Interface

- A KeyEvent is fired when we pressed, released, and typed a key on the source object.
- A KeyEvent listener must implement KeyListener interface:

```
interface KeyListener{
 /* Called-back when a key has been typed (pressed and
 released) */
 public void keyTyped(KeyEvent e)
 /* Called-back when a key has been pressed */
 public void keyPressed(KeyEvent e)
 /* Called-back when a key has been released */
 public void keyReleased(KeyEvent e)
}
```

44

## Example of handling KeyEvent

```
public class KeyEventDemo extends Frame
 implements KeyListener {
 private TextField tfInput;
 private TextArea taDisplay;

 public KeyEventDemo() {
 setLayout(new FlowLayout());

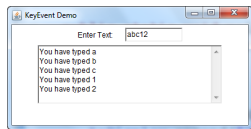
 add(new Label("Enter Text: "));
 tfInput = new TextField(10);
 add(tfInput);
 taDisplay = new TextArea(5, 40);
 add(taDisplay);

 tfInput.addKeyListener(this);

 setTitle("KeyEvent Demo");
 setSize(400, 200);
 setVisible(true);
 }

 public static void main(String[] args) {
 new KeyEventDemo();
 }

 @Override
 public void keyTyped(KeyEvent e) {
 taDisplay.append("You have typed " +
 e.getKeyChar() + "\n");
 }
 @Override
 public void keyPressed(KeyEvent e) {}
 @Override
 public void keyReleased(KeyEvent e) {}
}
```



45

## 3.4. Adapter

- Disadvantages of using XxxListener interfaces:
  - Each contains more than 1 method. If we care about only 1, we have to implement all (see previous KeyEvent example)
    - many have empty body → harder to read & maintain
- To avoid: AWT provides an **adapter** class for each listener interface with more than one method
  - An adapter class **implements** empty versions of all its interface's methods (e.g., **MouseAdapter** implements **MouseListener**)
- To use an adapter, we create a subclass of it, instead of directly implementing a listener interface

46

## Example of using Adapter

Exit the application if we press close button

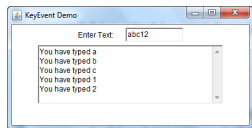
```
public KeyEventDemo() {
 setLayout(new FlowLayout());
 add(new Label("Enter Text: "));
 tfInput = new TextField(10);
 add(tfInput);
 taDisplay = new TextArea(5, 40);
 add(taDisplay);

 tfInput.addKeyListener(new KeyAdapter() {
 @Override
 public void keyPressed(KeyEvent e) {
 taDisplay.append("You have typed " +
 e.getKeyChar() + "\n");
 }
 });

 addWindowListener(new WindowAdapter() {
 public void windowClosing(WindowEvent e) {
 setVisible(false);
 dispose();
 System.exit(0);
 }
 });

 setTitle("KeyEvent Demo"); // "this" Frame sets title
 setSize(400, 200); // "this" Frame sets initial size
 setVisible(true); // "this" Frame shows
} //End of Constructor
```

Anonymous inner Class



7

## Quick quiz (1/2)

- How many are there top-level containers in AWT? What are they?
- How many are there secondary containers in AWT? What are they?
- Which utilities should be used to organize components inside a container? Which one can arrange components from left-to-right in the added order?
- Which model AWT uses to handle event? How many objects involved in the event-handling? What are they?

48

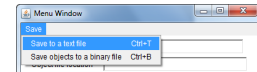
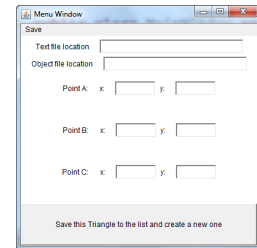
## Quick quiz (2/2)

- 5. When we click onto a Button, which event will be fired?
  - a. ButtonClickedEvent
  - b. ButtonPressedEvent
  - c. MouseEvent
  - d. ButtonEvent
  - e. ActionEvent
  - f. WindowEvent
- 6. Which ways should be used, implementing a XxxListener or extending a XxxAdapter to handle AWT events?

49

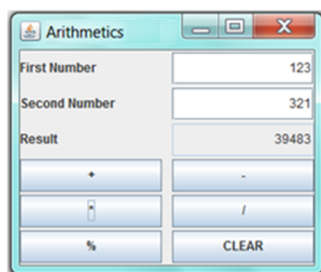
## Quiz

- Create an AWT application for what you have done in the previous lesson
  - The interface will look like the pictures
  - Users can enter add one by one Triangle to a list
  - Users can save the list in form of a text file or binary file, using a menu with shortcuts.
  - Remember to alert appropriate messages to users



50

## Exercises




51

## Review


- GUI is a type of user interface that allows users to interact with electronic devices using images rather than text commands.
- Two core sets of Java APIs for graphics programming are AWT (Abstract Windowing Toolkit) and Swing
- AWT is huge with 12 packages
- There are two types of GUI elements: Component and Container
- Top-level AWT containers are Frame, Dialog and Applet.
- Secondary AWT Containers are Panel, ScrollPane

52



## Review

- Layout manager can be used to arrange a container's components: FlowLayout, GridLayout, BorderLayout, ...
- A container has a setLayout() method to set its layout manager
- Java adopts the so-called "Event-Driven" (or "Event-Delegation") programming model for event-handling
- 3 objects involved in the event-handling: **source**, **listener**, **event**
- AWT supports many kinds of XxxEvent & XxxListener
- Use XxxAdapter to overcome disadvantages of XxxListener



53