GUI Programming

CS102A Lecture 11

James YU
`yujq3@sustech.edu.cn`

Department of Computer Science and Engineering
Southern University of Science and Technology

Nov. 23, 2020

Objectives

- GUI and its brief history
- Build simple GUIs with containers and components
- Event handling
- Layout management

What is GUI?

- The Graphical User Interface (GUI), is a type of user interface that allows users to interact with electronic devices through graphical icons and visual indicators.

GUI vs. CLI

- Before GUI became popular, text-based Command-Line Interface (CLI) was widely-used (mainly in 1970s and 1980s).
- Because CLIs consume little resources, they are still available in modern computers with GUIs and are widely-used by professionals.

A bit history about GUI

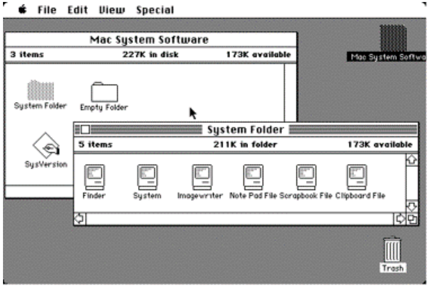
In 1973, Xerox PARC developed Alto, the first personal computer with GUI (not commercialized).

In 1981, Xerox Star workstation introduced the first commercial GUI OS (did not achieve market success).

A bit history about GUI

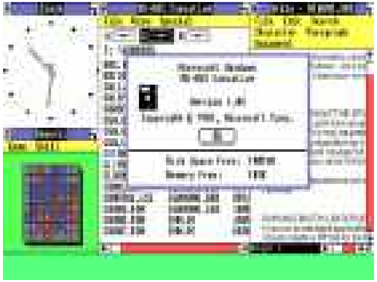
南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Apple Lisa (1983) and Macintosh (1984) (Steve Jobs visited Xerox PARC and was amazed by Alto).



Macintosh GUI (1984).

A bit history about GUI



Windows 1.0, a GUI for the MS-DOS operating system was released in 1985. The market's response was not so good.

The Windows OS becomes popular with the 1990 launch of Windows 3.0.

Java GUI programming APIs

- AWT (Abstract Windowing Toolkit): introduced in JDK 1.0.
- AWT components are **platform-dependent**. Their creation relies on the operating system's high-level user interface module.
    - For example, creating an AWT check box would cause AWT directly to call the underlying native subroutine that created a check box.
    - This makes GUI programs written in AWT look like native applications
- AWT contains 12 packages of 370 classes (Swing and FX are more complex, 650+ classes).
    - They are developed by expert programmers with advanced design patterns.
    - Writing your own graphics classes (re-inventing the wheels) is mission impossible!

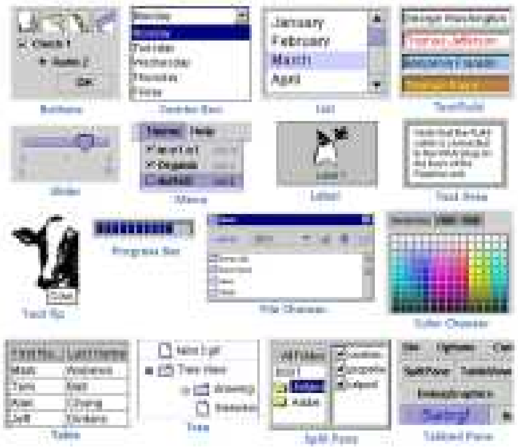
Java GUI programming APIs

- Swing, introduced in 1997 after the release of JDK 1.1, provides a much more comprehensive set of UI widgets than AWT.
- Unlike AWT's UI widgets, Swing's are not implemented by platform-specific code. They are written entirely in Java and platform-independent.
- Swing draws its widgets by calling low-level subroutines in the local graphics subsystem instead of relying on the OS's high-level UI module (thus becomes light-weight).
- Pluggable look and feel: Swing component can have the native platform's "look and feel" or a cross-platform look and feel (the "Java Look and Feel").

Java GUI programming APIs

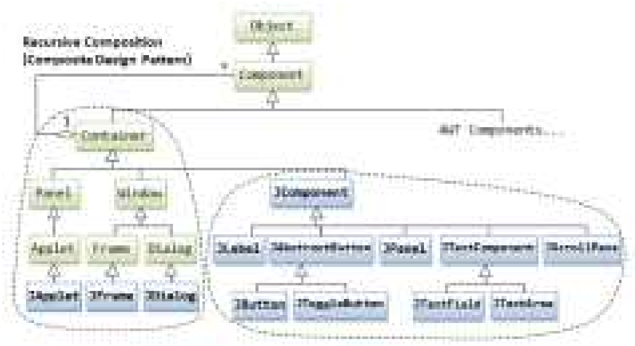
- Java FX, released in 2008, is Java's new GUI library for creating and delivering desktop applications.
- The latest JavaFX 8, which was integrated into JDK 8, was meant to replace Swing, but for some reasons didn't get popular:
  - Emergence of mobile devices and applications.
  - The popularity of the browser/server architecture (no client programs need to be installed).
- Both Swing and FX will be included in JDK for the foreseeable future. Swing is more mature and more widely-used than FX.

Swing GUI components



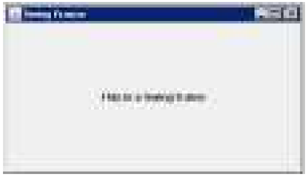
Swing classes

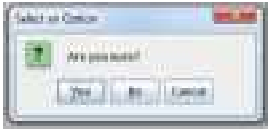
- There are two groups of classes (in package `javax.swing`): *container*s and *component*s. A container is used to hold components. A container can also hold containers because it is a (subclass of) component.

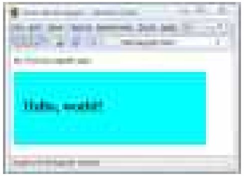


Containers

- A Swing application requires a *top-level container*. There are three top-level containers in Swing:
  - `JFrame`: used for the application's main window (with an icon, a title, minimize/maximize/close buttons, an optional menu-bar, and a content-pane).
  - `JDialog`: used for secondary pop-up window (with a title, a close button, and a content-pane).
  - `JApplet`: used for the applet's display-area (content-pane) inside a browser's window.







Containers

- A Swing application requires a *top-level container*. There are three top-level containers in Swing:
    - `JFrame`: used for the application's main window (with an icon, a title, minimize/maximize/close buttons, an optional menu-bar, and a content-pane).
    - `JDialog`: used for secondary pop-up window (with a title, a close button, and a content-pane).
    - `JApplet`: used for the applet's display-area (content-pane) inside a browser's window.
- There are secondary containers (such as `JPanel`) which can be used to group and layout relevant components.

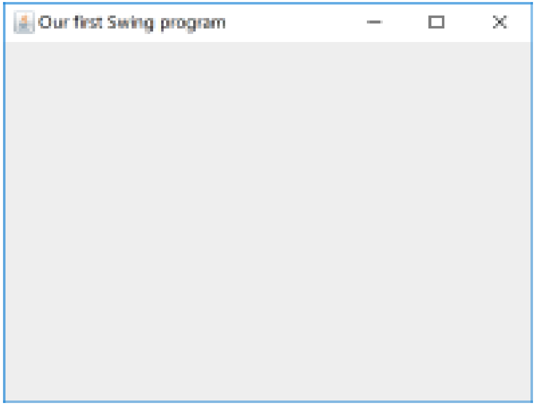
Build our first Swing program

```java
import javax.swing.JFrame;

public class HelloWorld extends JFrame {
  public HelloWorld() {
    super("Our first Swing program");
  }

  public static void main(String[] args) {
    HelloWorld gui = new HelloWorld();
    gui.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    gui.setSize(800, 600);
    gui.setVisible(true);
  }
}
```

Build our first Swing program



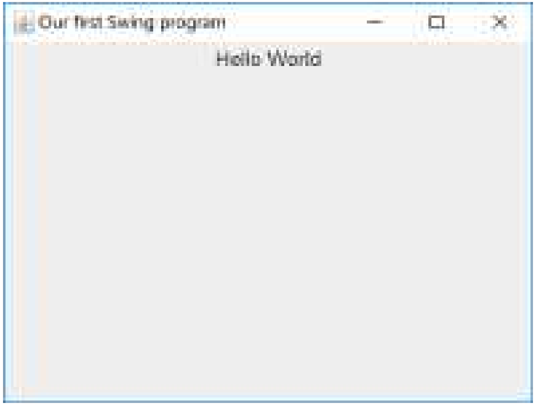
Build our first Swing program

```
public class HelloWorld extends JFrame {
  private JLabel label;

  public HelloWorld() {
    super("Our first Swing program");
    setLayout(new FlowLayout());
    label = new JLabel("Hello World");
    label.setFont(new Font("San Serif", Font.PLAIN, 30));
    add(label);
  }

  public static void main(String[] args) { // same as earlier }
}
```

Build our first Swing program



Hello World

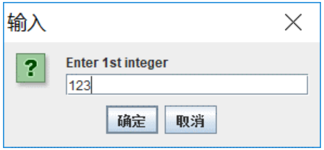
Simple GUI-based input/output

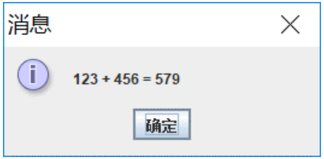
- JOptionPane is a widely-used Swing class for popping up a dialog box that prompts users for a value or informs them of something.

```java
public static void main(String[] args) {
  String str1 = JOptionPane.showInputDialog("Enter 1st integer");
  String str2 = JOptionPane.showInputDialog("Enter 2nd integer");
  int num1 = Integer.parseInt(str1);
  int num2 = Integer.parseInt(str2);
  int sum = num1 + num2;
  JOptionPane.showMessageDialog(null,
    num1 + " + " + num2 + " = " + sum);
}
```

Build our first Swing program

```java
public static void main(String[] args) {
  String str1 = JOptionPane.showInputDialog("Enter 1st integer");
  String str2 = JOptionPane.showInputDialog("Enter 2nd integer");
  int num1 = Integer.parseInt(str1);
  int num2 = Integer.parseInt(str2);
  int sum = num1 + num2;
  JOptionPane.showMessageDialog(null,
    num1 + " + " + num2 + " = " + sum);
}
```





```
123 + 456 = 579
```

Events (in GUI programming)

- In GUI programming, events describe the change in the state of a GUI component when users interact with it.
- For example, events will occur when
  - A button is clicked;
  - The mouse is moved;
  - A character is entered through keyboard;
  - An item from a list is selected;
  - ...

Event handling

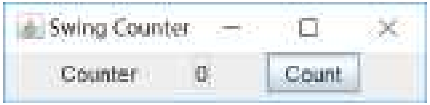
- Event handling is the mechanism that controls the event and decides what should happen if an event occurs. Three key concepts:
  - *Event source*: the GUI component with which the user interacts.
  - *Event object*: encapsulate the information about the event that occurred.
  - *Event listener*: an object that is notified by the event source when an event occurs.
    - A method of the event listener receives an event object when the event listener is notified of the event.
    - The listener then uses the event object to respond to the event.
- Such a handling model is known as *delegation event model*, because an event’s processing is delegated to an event listener object

How to handle events in Swing?

- We use a counter program to illustrate the steps.

```
public class SwingCounter extends JFrame {
  private JTextField tfCount;
  private JButton btnCount;
  private int count = 0;
  public SwingCounter() {
    setLayout(new FlowLayout(FlowLayout.LEFT, 50, 0));
    add(new JLabel("Counter"));
    tfCount = new JTextField("0");
    tfCount.setEditable(false); add(tfCount);
    btnCount = new JButton("Count"); add(btnCount);
  }
  public static void main(String[] args) { SwingCounter sc = new
      SwingCounter(); ... }
}
```

How to handle events in Swing?



- Nothing will happen when we click the button (we have not handled the event yet).

How to handle events in Swing?

- **Step 1**: check what event will occur when `JButton` is clicked.
- An `ActionEvent` (in `java.awt.event` package) will occur whenever the user performs a component-specific action on a GUI component.
  - When user clicks a button.
  - When user chooses a menu item.
  - When user presses Enter after typing something in a text field...
- **Step 2**: define the event listener class by implementing the corresponding listener interface.

```java
public class ButtonClickListener implements ActionListener {
  @Override
  public void actionPerformed(ActionEvent arg0) {
      // code to react to the event
  }
}
```

How to handle events in Swing?

- The event listener class is often declared as an inner class.
  - An inner class is a member of the outer class. Therefore, it can access the private members of the outer class (this is very useful).

```java
public class SwingCounter extends JFrame {
  private JTextField tfCount;
  private JButton btnCount;
  private int count = 0;

  public class ButtonClickListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent arg0) {
      ++count;  tfCount.setText(count + "");
    }
  }
}
```

How to handle events in Swing?

- **Step 3**: register an instance of the event listener class as a listener on the corresponding GUI component.

```java
btnCount.addActionListener(new ButtonClickListener());
```

How to handle events in Swing?

```java
public class SwingCounter extends JFrame {
  private JTextField tfCount;
  private JButton btnCount;
  private int count = 0;
  public SwingCounter() {
    setLayout(new FlowLayout(FlowLayout.LEFT, 50, 0));
    add(new JLabel("Counter"););
    tfCount = new JTextField("0");
    tfCount.setEditable(false); add(tfCount);
    btnCount = new JButton("Count"); add(btnCount);
    btnCount.addActionListener(new ButtonClickListener());
  }
  public class ButtonClickListener implements ActionListener {
    public void actionPerformed(ActionEvent arg0) { ... }
  }
  public static void main(String[] args) { ... }
}
```

Layout management

- Layout managers control how to place the GUI components (containers can also be treated as components) in a container for presentation purposes.
- You can use the layout manager for basic layout capabilities instead of determine every GUI component's exact position and size (which is non-trivial and error-prone).
- All layout managers in Java implement the interface `LayoutManger` (in the package `java.awt`).
- Commonly-used layout managers: `FlowLayout`, `BorderLayout`, `GridLayout`.

FlowLayout

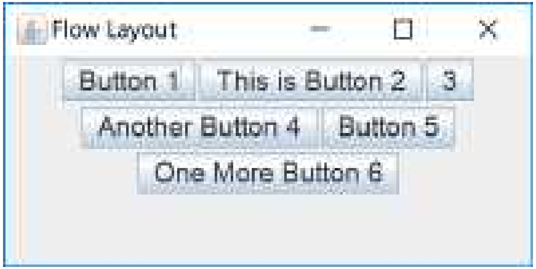
```java
public class FlowLayoutDemo extends JFrame {
  private JButton btn1, btn2, btn3, btn4, btn5, btn6;

  public FlowLayoutDemo() {
    super("Flow Layout");
    setLayout(new FlowLayout());
    btn1 = new JButton("Button 1"); add(btn1);
    btn2 = new JButton("This is Button 2"); add(btn2);
    btn3 = new JButton("3"); add(btn3);
    btn4 = new JButton("Another Button 4"); add(btn4);
    btn5 = new JButton("Button 5"); add(btn5);
    btn6 = new JButton("One More Button 6"); add(btn6);
  }

  public static void main(String[] args) { ... }
}
```

FlowLayout

- Default layout manager for the secondary container `javax.swing.JPanel`.
- Places components in a straight horizontal line. If there is no enough space to fit all component into one line, simply move the next line.



GridLayout

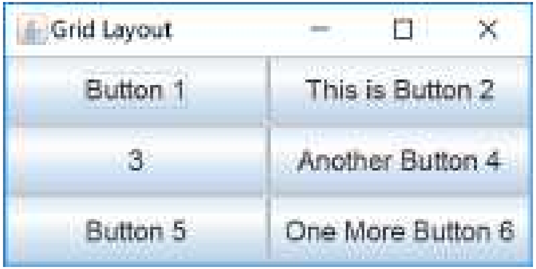
```java
public class GridLayoutDemo extends JFrame {
  private JButton btn1, btn2, btn3, btn4, btn5, btn6;

  public GridLayoutDemo() {
    super("Grid Layout");
    setLayout(new GridLayout(3, 2, 3, 3));
    btn1 = new JButton("Button 1"); add(btn1);
    btn2 = new JButton("This is Button 2"); add(btn2);
    btn3 = new JButton("3"); add(btn3);
    btn4 = new JButton("Another Button 4"); add(btn4);
    btn5 = new JButton("Button 5"); add(btn5);
    btn6 = new JButton("One More Button 6"); add(btn6);
  }

  public static void main(String[] args) { ... }
}
```

GridLayout

- Places components into rows and columns.



BorderLayout

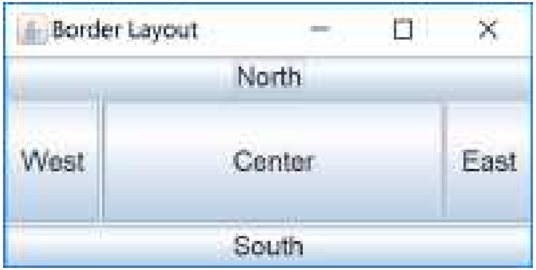
```java
public class BorderLayoutDemo extends JFrame {
  private JButton btnNorth, btnSouth, btnCenter, btnEast, btnWest;

  public BorderLayoutDemo() {
    super("Border Layout");
    setLayout(new BorderLayout(3, 3));
    btnNorth = new JButton("North"); add(btnNorth, BorderLayout.NORTH);
    btnSouth = new JButton("South"); add(btnSouth, BorderLayout.SOUTH);
    btnCenter = new JButton("Center"); add(btnCenter, BorderLayout.CENTER
        );
    btnEast = new JButton("East"); add(btnEast, BorderLayout.EAST);
    btnWest = new JButton("West"); add(btnWest, BorderLayout.WEST);
  }

  public static void main(String[] args) { ... }
}
```

BorderLayout

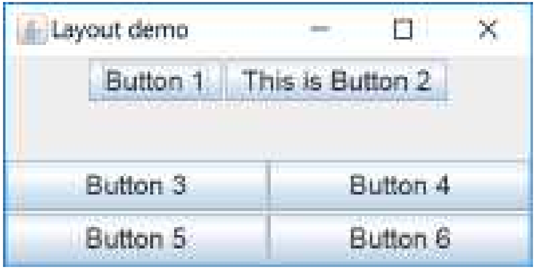
- Default layout manager for the top level container `javax.swing.JFrame`.
- Arranges the GUI components into five pre-defined areas: `NORTH`, `SOUTH`, `EAST`, `WEST`, `CENTER`.



Using secondary containers

```java
public class LayoutDemo extends JFrame {
  private JButton btn1, btn2, btn3, btn4, btn5, btn6;
  public LayoutDemo() {
    super("Layout demo");
    setLayout(new GridLayout(2, 1));
    JPanel panel1 = new JPanel(new FlowLayout());
    JPanel panel2 = new JPanel(new GridLayout(2, 2, 3, 3));
    add(panel1); add(panel2);
    btn1 = new JButton("Button 1"); panel1.add(btn1);
    btn2 = new JButton("This is Button 2"); panel1.add(btn2);
    btn3 = new JButton("Button 3"); panel2.add(btn3);
    btn4 = new JButton("Button 4"); panel2.add(btn4);
    btn5 = new JButton("Button 5"); panel2.add(btn5);
    btn6 = new JButton("Button 6"); panel2.add(btn6);
  }
  public static void main(String[] args) {...}
}
```

Using secondary containers


Layout demo
Button 1
This is Button 2
Button 3
Button 4
Button 5
Button 6