Scarborough College
University of Toronto
Fall 1997

CSC A06F: Introduction to Computer Programming
Lecture notes 9:
Graphical User Interfaces

Reading:

- Lewis and Loftus, chapter 10.

Graphical user interfaces

A **GUI** provides a means for a user to interact with a program.

Basic pieces of a GUI include:

- graphical components (buttons, scrollbars, panels, etc.)
- layout managers
- events
- event listeners

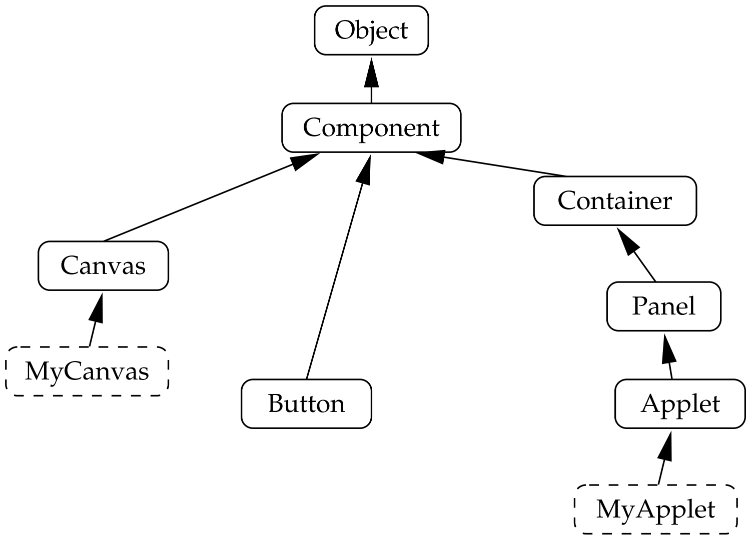
Once the components have been laid out, the system just sits and waits for the user to do things (like pressing a button).

The event listeners wait for these actions, and call the appropriate method(s) when they happen.

This technique is called **event-driven programming** .

Components are classes

Each kind of component is described by a class.



Components

A GUI is composed of **components** .

The `Component` class provides basic services:

- for drawing:
  `setSize(...), paint(...), repaint()`
- for event handling:
  `addActionListener(...)`
- for control of color and fonts:
  `setBackground(...)`

Some components are simple: e.g., `Button`, `Scrollbar`, `TextField`, ...

Some are **containers** , which organize other components on the screen: e.g., `Panel`, `Applet`, `Frame`.

Example: `Button`

A button is a simple push button. It can initiate an action when pushed.

Three steps to setting up a button (or any component) in an applet:

1. declare the button (as an instance variable)

```
Button push_button;
```

2. instantiate and initialize (in the applet's constructor)

```
push_button = new Button ("Press me!");
```

3. add it to the applet (in the applet's `init()` method)

```
add (push_button);
```


Containers

Containers are a special kind of component that contain and arrange components within themselves.

A container can even contain containers.

Every container has two methods:

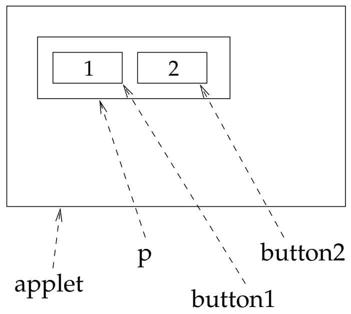
- `add (...)` — add a component to the container
- `setLayout (...)` — tell it what layout manager to use

Steps for setting containers up are the same as for components, except step 3 becomes:

3a. add other components to the container

3b. add the container to the applet

Example




```
// in an applet
Button button1 = new Button("1");
Button button2 = new Button("2");

Panel p = new Panel();

p.add (button1);
p.add (button2);

add (p);
```


Layout Managers

A **layout manager** is an object that decides how to arrange the components within a container.

Every container has a default layout manager. To change it, use the `setLayout(...)` method.

- `FlowLayout`
  - is the default for `Applet` and `Panel`
  - lays out components from left to right, top to bottom.
  e.g., `container.add (button1);`
- `BorderLayout`
  - has 5 places to put components: North, South, East, West, Center
  e.g.,
  ```
  container.setLayout (new BorderLayout());
  container.add (button1, "West");
  ```
- `GridLayout`
  - lays out components in a grid of a specified size
  - each component is placed in a specified cell

Handling events

Basic flow of processing:

1. The user does an action on a component.
2. The component generates an event.
3. The component tells all of its listeners about the event.
4. Each listener handles the event:
    - perhaps by just printing a message;
    - or, it might call a method in the applet;
    - which could call a method in one of the components ....

The system handles 1–3, but you have to write the code for 4, because you have to say exactly what happens.

Event listeners

An **event listener** is an object that 'listens' (i.e., waits) for events on a particular component.

Before events can be handled, you must instantiate an event listener and 'register' it with the component. E.g.,

```
MyListener listener;                       // (step 1)
listener = new MyListener();               // (step 2)
push_button.addActionListener (listener);  // (step 3)
```

Registering the listener is like requesting a 'callback' on your telephone when your friend's phone is busy:

> When an action on `push_button` finally occurs, the listener is called; just like your telephone would be called back automatically when your friend is off the phone.

Implementing a listener

Listeners are defined in the Java API as *interfaces* .

So, you have to implement your own listener(s) and fill in the abstract methods defined in the interface(s).

Why do they make you do this?

Example:

```
class MyListener implements ActionListener {

   // only one method to fill in
   public void actionPerformed (ActionEvent e) {
      System.out.println ("Button pressed!");
   }
}
```

A `Button` will generate an `ActionEvent` and call the `actionPerformed` method in its `ActionListener`.

A more complex example 1

Normally we want user actions to affect the applet or some component of it. That means the listener will have to call a method in the applet.

Let's say we want the following method in the applet to be called when someone presses the button:

```
// change the label on the button
public void buttonPressed () {
   push_button.setLabel ("Thanks for pressing me!");
}
```

Problem: The listener can't call `buttonPressed()` because it doesn't know about the applet. So, …

A more complex example 2

Rewrite the listener:

```
class MyListener implements ActionListener {
   MyApplet gui;    // this is the applet

   // when you construct the listener, pass it the applet
   public MyListener (MyApplet applet) {
      gui = applet;
   }

   public void actionPerformed (ActionEvent e) {
      gui.buttonPressed();
   }
}
```

Of course, when we create an instance of this listener in the applet, we have to use the constructor properly:

```
// ... in the applet's constructor
listener = new MyListener (this);
```


Events

We only looked at `ActionEvent`.

But, there are many kinds of event, and each has its own kind of listener that you would have to implement.

User Action	Event Generated	Event Listener
Pressing a button component	ActionEvent	ActionListener
Moving mouse within a component	MouseEvent	MouseListener
Moving mouse into a component	FocusEvent	FocusListener
Typing on the keyboard	KeyEvent	KeyListener
Filling in a TextField	ActionEvent	ActionListener

Summary

Setting up the GUI

1a. Declare components.

1b. Declare their listeners.

2a. Instantiate and initialize the components.

2b. Instantiate and initialize the listeners.

3a. Add components to their containers.

3b. Add components to the applet.

3c. Register the listeners with their components.

Setting up for event handling

1. Write methods in the applet for each kind of action that the user can take.

2. Write a class (or classes) for each kind of event listener, and fill in the methods.

See `ButtonsGUI.java` for an example of everything here.