# CSE 3461

Focus Subsystem

## The Focus Subsystem

- Many components — even those primarily operated with the mouse, such as buttons — can be operated with the keyboard.
- For a key press to affect a component, the component must have the keyboard focus.
- In the case of multiple windows,
  - the focused window is the window that contains the focus owner.

## The Focus Subsystem

- From the user's point of view, the component with the keyboard focus is generally prominent
  - E.g. a dotted or black border (can be altered programmatically)
- The window containing the component is also more prominent than other windows onscreen.
- These visual cues let the user know to which component any typing will relate.
- Only one component at a time in the window system can have the keyboard focus.

## How does a Window gain focus?

- Depends on the windowing system
  - when if becomes the front window? (Microsoft Windows)
  - when the cursor is on it? (Solaris)
  - The behavior of the `Window.toFront` method can be different across windowing systems.
- There is no foolproof way, across all platforms, to ensure that a particular window always gains the focus.

## How does a Component gain Focus?

- A component generally gains the focus when the user clicks it, or when the user tabs between components, or otherwise interacts with a component.
- A component can also be given the focus programmatically

## Focus Traversal Cycles

- A focus cycle (or focus traversal cycle) is a set of components that share a common ancestor in the containment hierarchy.
- The focus cycle root is the container that is the root for a particular focus traversal cycle.
- A focus cycle root can itself contain one or more focus cycle roots.
- In a hierarchy of focus cycle roots, upwards traversal takes the focus out of the current cycle into the parent cycle.

## Focus Traversal Policy

- A focus traversal policy determines the order in which a group of components are navigated.
- Within a focus cycle, components can be navigated in a forward or backward direction.
- Swing provides the LayoutFocusTraversalPolicy class
  - it determines the order of navigation
  - Default: the order in which the components were added to their container determines order in the focus traversal cycle
  - FTP can be based on layout manager-dependent factors, such as size, location, and orientation of components.

## Focus Traversal Cycles

- By default, every `JWindow` and `JInternalFrame` component can be a focus cycle root.
- The following Swing objects can be focus cycle roots:
  - `JApplet`
  - `JDesktopPane`,
  - `JDialog`,
  - `JEditorPane`,
  - `JFrame`,
  - `JInternalFrame`, and
  - `JWindow`.
- While it might appear that `JTable` and `JTree` objects are focus cycle roots, they are not.

## `JWindow` and Focus

- A `JWindow` component's owning frame must be visible in order for any components in the window to get the focus.
- By default, an invisible owning frame is created for `JWindow` components
  - Thus, components in the `JWindow` component might not be able to get the focus.
- The solution:
  - specify a visible owning frame when creating the `JWindow` component
  - use an undecorated `JFrame` component instead.

## Default Focus Traversal Keys

- In most Look and Feel models, components are navigated using the Tab and Shift-Tab keys.
- These keys are the default focus traversal keys
  - Tab shifts the focus in the forward direction.
  - Shift-Tab moves the focus in the backward direction.
  - The Control key is used by convention to move the focus out of any component that treats Tab in a special way, such as `JTable`.

## Default Behaviours

- Example: `FocusConceptsDemo`
  - The first button has the initial focus.
  - Tabbing moves the focus through the buttons into the text area.
  - Additional tabbing moves the cursor within the text area but not out of the text area because, inside a text area, Tab is not a focus traversal key.
  - However, Control-Tab moves the focus out of the text area and into the first text field.
  - Likewise, Control-Shift-Tab moves the focus out of the text area and into the previous component.

## KeyboardFocusManager

- The `KeyboardFocusManager` is a critical element of the focus subsystem.
- It manages state and initiates changes.
- The keyboard manager tracks the focus owner — the component that receives typing from the keyboard.
- The `KeyboardFocusManager` dispatches key events:
  - recall that input components (e.g., `JTextField`) have UI delegates
  - one of the responsibilities of the UI delegate is to install any required listeners

## Programmatic Control of The Focus Subsystem

- The focus subsystem is designed to do the right thing as invisibly as possible.
- In most cases it behaves in a reasonable manner.
- If the focus subsystem doesn't behave as desired, its behavior can be tweaked in various ways programmatically.

## Programmatic Interventions

1. The ordering is as desired, but the first component doesn't gain focus as desired.
2. The focus traversal ordering is wrong.
3. A component must to be prevented from losing focus, or you need to check a value in a component before it loses focus.
4. A custom component is not getting the focus.

## Initial Component Focus

Example

```
//Make textField get the focus whenever frame is activated.

frame.addWindowListener(new WindowAdapter() {
public void windowGainedFocus(WindowEvent e) {
  textField.requestFocusInWindow();
  }
});
```

- This will give a particular component the focus every time the window gains the focus
- Other triggers: window opened, window de-iconified, etc

## Initial Component Focus

- How to ensure that a particular component gains the focus the first time a window is activated?
  - Invoke the `requestFocusInWindow` method on the component after the component has been realized, but before the frame is displayed.
  - Alternatively, you can apply a custom `FocusTraversalPolicy` to the frame and call the `getDefaultComponent` method to determine which component will gain the focus.

```
//...Where initialization occurs...
JFrame frame = new JFrame("Test");
JPanel panel = new JPanel(new BorderLayout());

//...Create a variety of components here...

//Create the component that will have the initial focus.
JButton button = new JButton("I am first");
panel.add(button);
frame.getContentPane().add(panel);  //Add it to the panel

frame.pack();  //Realize the components.
//This button will have the initial focus.
button.requestFocusInWindow();
frame.setVisible(true); //Display the window.
```

## Altering Focus Traversal Order

If the focus traversal ordering is wrong:

- change the containment hierarchy
  - e.g., change the order that the components are added to their containers
- create a custom focus traversal policy

## Prevent Focus Loss

- A component must to be prevented from losing focus
- you need to check a value in a component before it loses focus.
- Input verification is a solution to this problem.

`InputVerificationDemo.java`
`InputVerificationDialogDemo.java`

## Failure to Gain Focus

- A custom component is not getting the focus.
  - ensure all requirements are satisfied

## Focus Traversal Keys

- The default focus traversal keys can be changed programmatically:

```
Set forwardKeys = getFocusTraversalKeys(
       KeyboardFocusManager.FORWARD_TRAVERSAL_KEYS);
Set newForwardKeys = new HashSet(forwardKeys);
newForwardKeys.add(
       KeyStroke.getKeyStroke(KeyEvent.VK_ENTER, 0));
setFocusTraversalKeys(
       KeyboardFocusManager.FORWARD_TRAVERSAL_KEYS,
       newForwardKeys);
```

*This adds Enter as a forward focus traversal key*

## Making a Custom Component Focusable

- For a component to gain the focus, it must satisfy three requirements:
  - it must be visible, enabled, and focusable.
- A call to the setFocusable(true) method makes the component focusable.
- An alternative approach: explicitly provide an input map of key bindings (to be discussed later)

## Customizing Focus Traversal

- You can set a focus traversal policy on any Container:
  - using the `setFocusCycleRoot` method, or
  - pass focus traversal policy providers to the `FocusTraversalPolicy` methods

## Customizing Focus Traversal

- Implementing customized focus traversal only makes sense if the particular Container is a focus traversal policy provider.
  - if the container is not a focus cycle root, it may have no apparent effect.
  - Use the `isFocusTraversalPolicyProvider()` method to determine whether a Container is a focus traversal policy provider.
  - Use the `setFocusTraversalPolicyProvider()` method to set a container for providing focus traversal policy.

`FocusTraversalDemo.java`

## Tracking Which Component has Focus

- An application may need to track which component has the focus.
  - to dynamically update menus, status bars, etc.
- If you need to track the focus only on specific components
  - implement a `FocusEventListener`
  - register a `PropertyChangeListener` on the `KeyboardFocusManager`.
- The property change listener is notified of every change involving the focus, including changes to the focus owner, the focused window, and the default focus traversal policy.

`TrackFocusDemo.java`

## Timing Focus Transfers

- Focus transfers are asynchronous.
- This quality can lead to some odd timing-related problems and assumptions, especially during automatic transfers of the focus.

## Timing Focus Transfers

- For example, imagine an application with a window containing:
  - a Start button,
  - a text field, and
  - a Cancel button.

  (added in that order)
- In this example, the desired behavior is that:
  - the Start button has the initial focus
  - when the Start button is clicked, it is disabled, and then the Cancel button receives the focus.

## Timing Focus Transfers

- How to implement this behavior?
  - add the components to the container in the desired order
  - create a custom focus traversal policy.
  - programmatically (if, for some reason, the above two options are not possible)

## Timing Focus Transfers

- Programmatically:

```
public void actionPerformed(ActionEvent e) {
  //v1: This works.
   start.setEnabled(false);
   cancel.requestFocusInWindow();
}
```

cf

```
public void actionPerformed(ActionEvent e) {
  //v2: This does not work.
  cancel.requestFocusInWindow();
  start.setEnabled(false);
}
```

29

## Timing Focus Transfers

- V1:
  - the focus goes from the Start button to the Cancel button, rather than to the text field.
- V2:
  - The call to the `requestFocusInWindow` method initiates the focus transfer, but it does not immediately move the focus to the Cancel button.
  - When the Start button is disabled, the focus is transferred to the next component (so there is always a component with the focus)
  - In this case, it would then move the focus to the text field, not to the Cancel button.
  - End result: focus is requested on the Cancel button before it has left the Start button

30

## Timing Focus Transfers

Morale:
- Make focus requests (in some situations) only once all other changes that might affect the focus have been completed.
- For instance:
  - Hiding the focus owner.
  - Making the focus owner non-focusable.
  - Calling the removeNotify method on the focus owner.
  - Doing any of the above operations to the container of the focus owner, or causing changes to the focus policy so that the container no longer accepts the component as the focus owner.
  - Disposing of the top-level window that contains the focus owner.

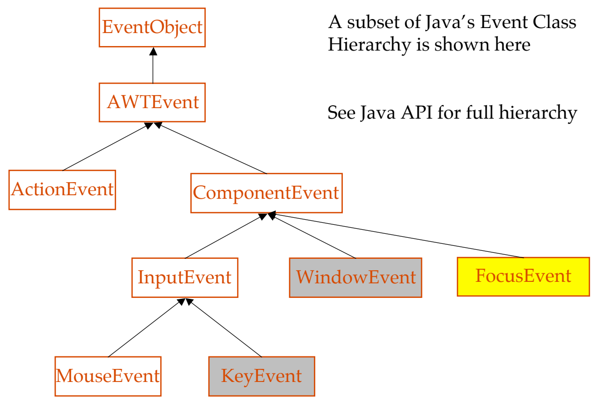
31

## Types of Events

| **User Action** | **Event that Occurs** |
|---|---|
| click a button | `ActionEvent` |
| press Enter while in a text field | `ActionEvent` |
| choose a menu item | `ActionEvent` |
| close a frame (main window) | `WindowEvent` |
| press a mouse button (while the cursor is over a component) | `MouseEvent` |
| move the mouse over a component | `MouseMotionEvent` |
| component becomes visible | `ComponentEvent` |
| component gets the keyboard focus | `FocusEvent` |
| Table or list selection changes | `ListSelectionEvent` |

32

## Java's Event Class Hierarchy



A subset of Java's Event Class Hierarchy is shown here

See Java API for full hierarchy

## Listeners and Corresponding Adapters

| Listener interface (# methods) | Adapter class |
|---|---|
| `WindowListener` (7) | `WindowAdapter` |
| `ActionListener` (1) | **not defined** |
| *Later we'll discuss…* | |
| `KeyListener` (3) | `KeyAdapter` |
| `MouseListener` (5) | `MouseAdapter` |
| `MouseInputListener` (7)* | `MouseInputAdapter` |
| `ItemListener` (1) | **not defined** |
| `FocusListener` (2) | `FocusAdapter` |

* `MouseInputListener` combines `MouseListener` and `MouseMotionListener`