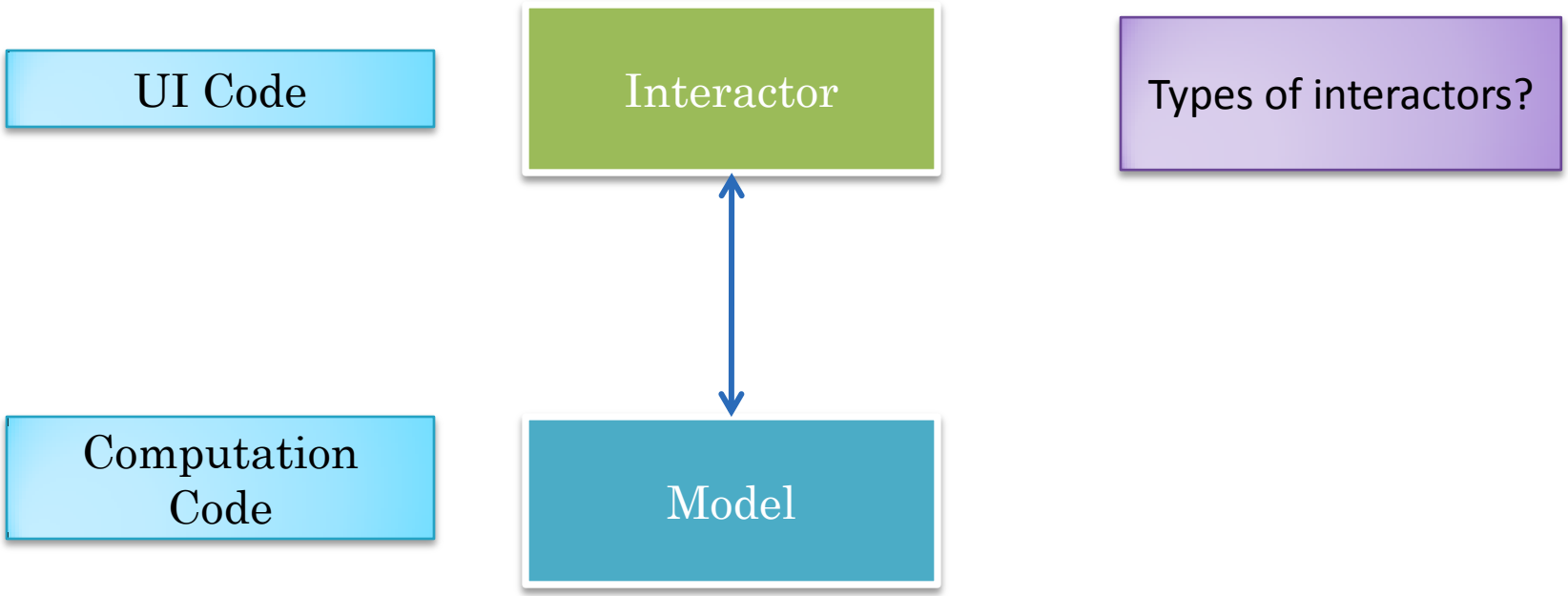
Types of Interactors

Prasun Dewan
Department of Computer Science
University of North Carolina at Chapel Hill
dewan@cs.unc.edu
Code available at: https://github.com/pdewan/ColabTeaching

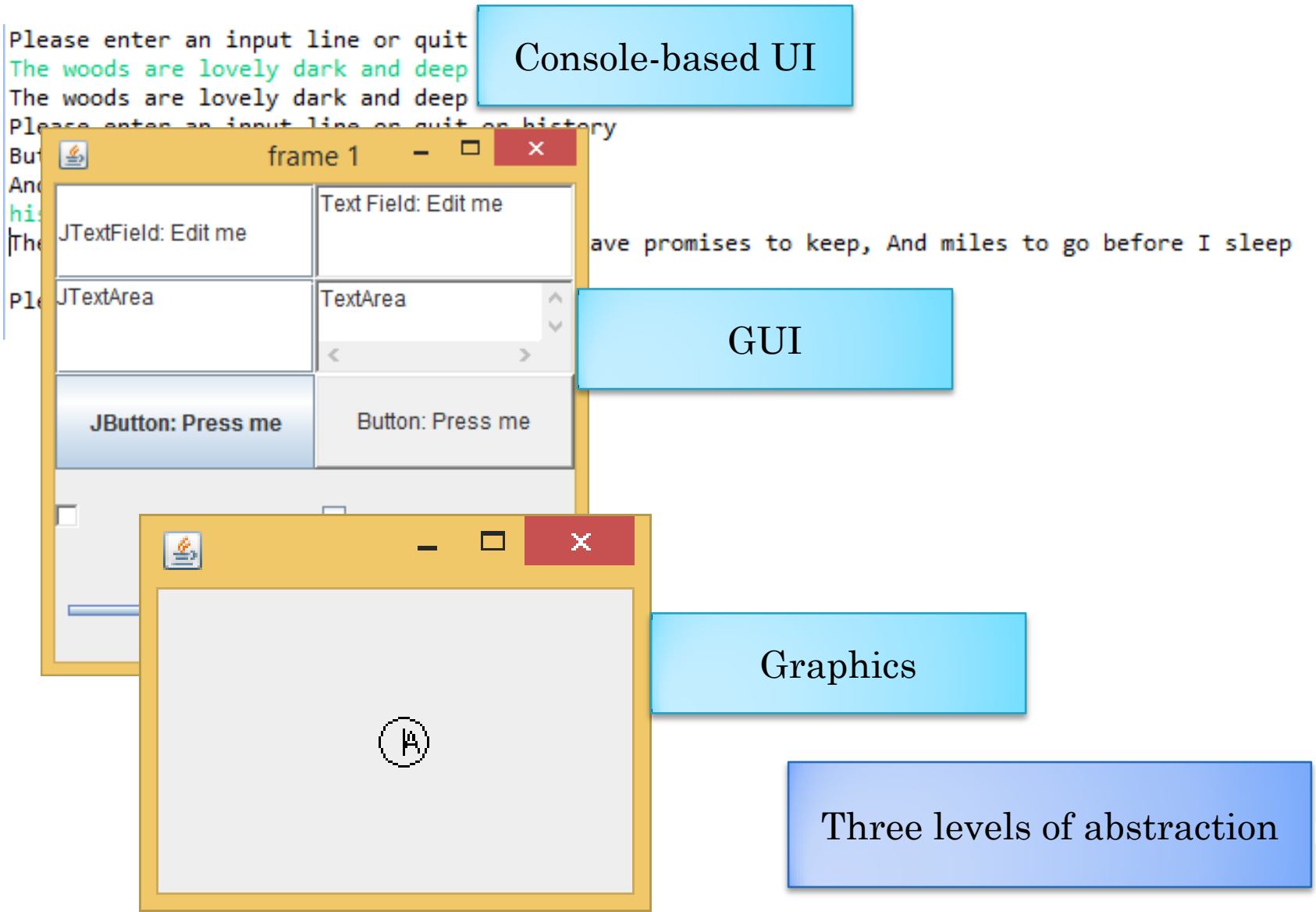
Pre-Requisites

- Model-Interactor Separation

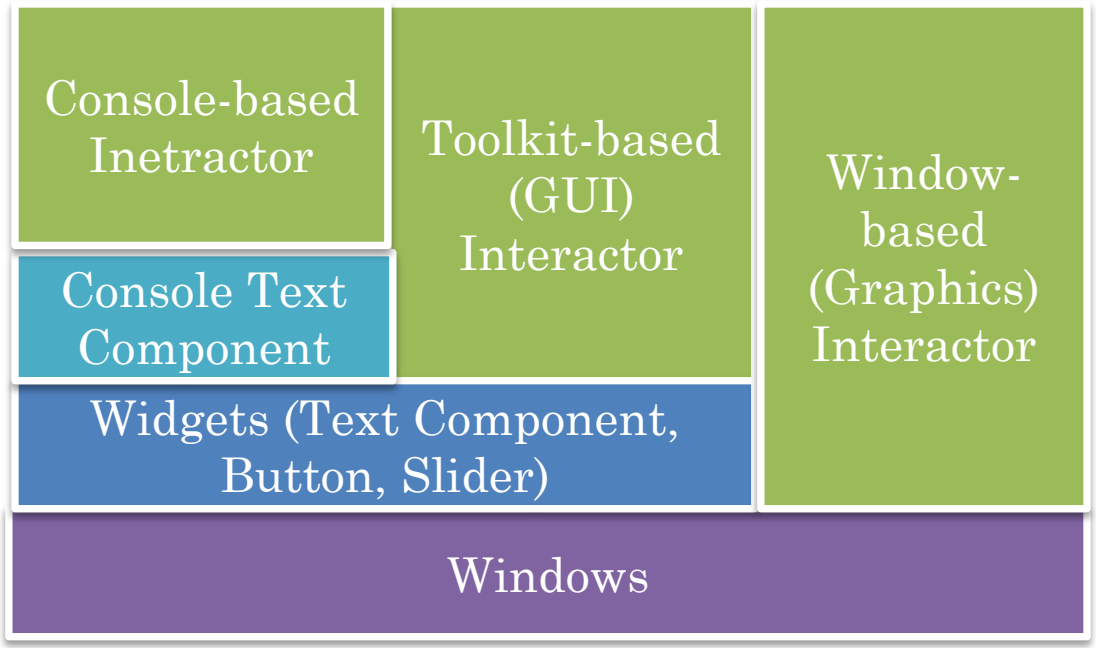
Interactor Types



Types of Interactors

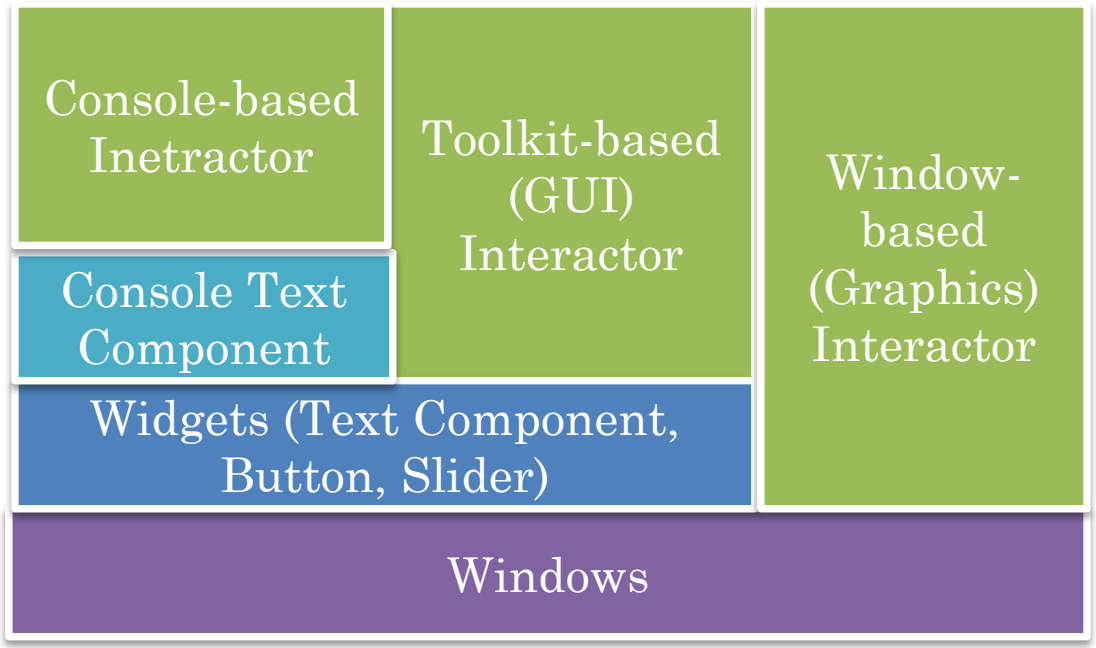


Abstraction Layers



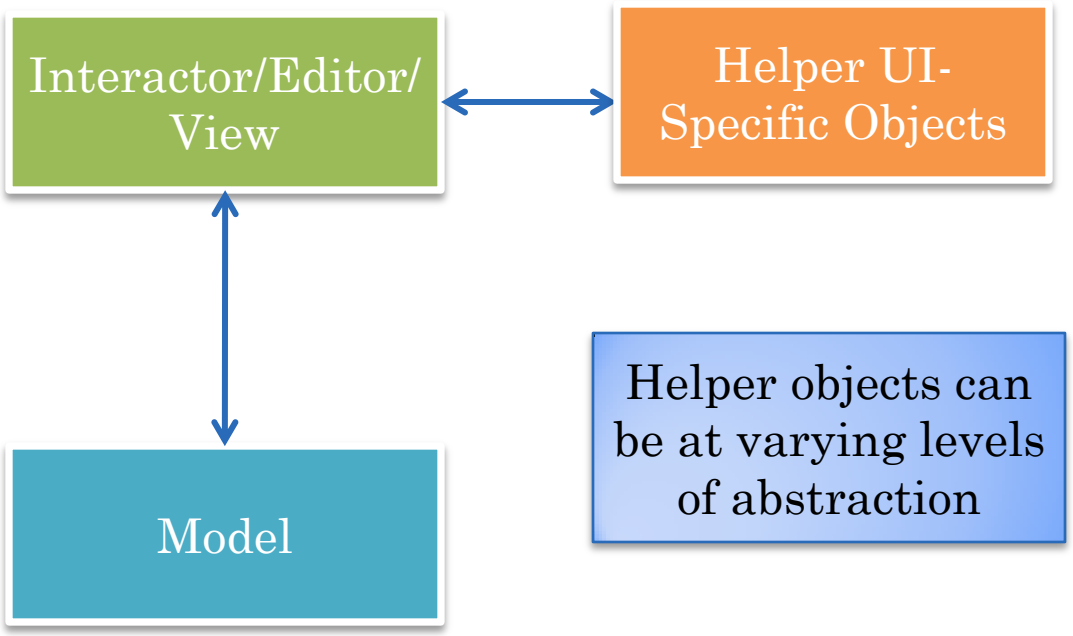
Flexibility vs. Automation Tradeoff in Abstraction Design

Abstraction Layers (Review)

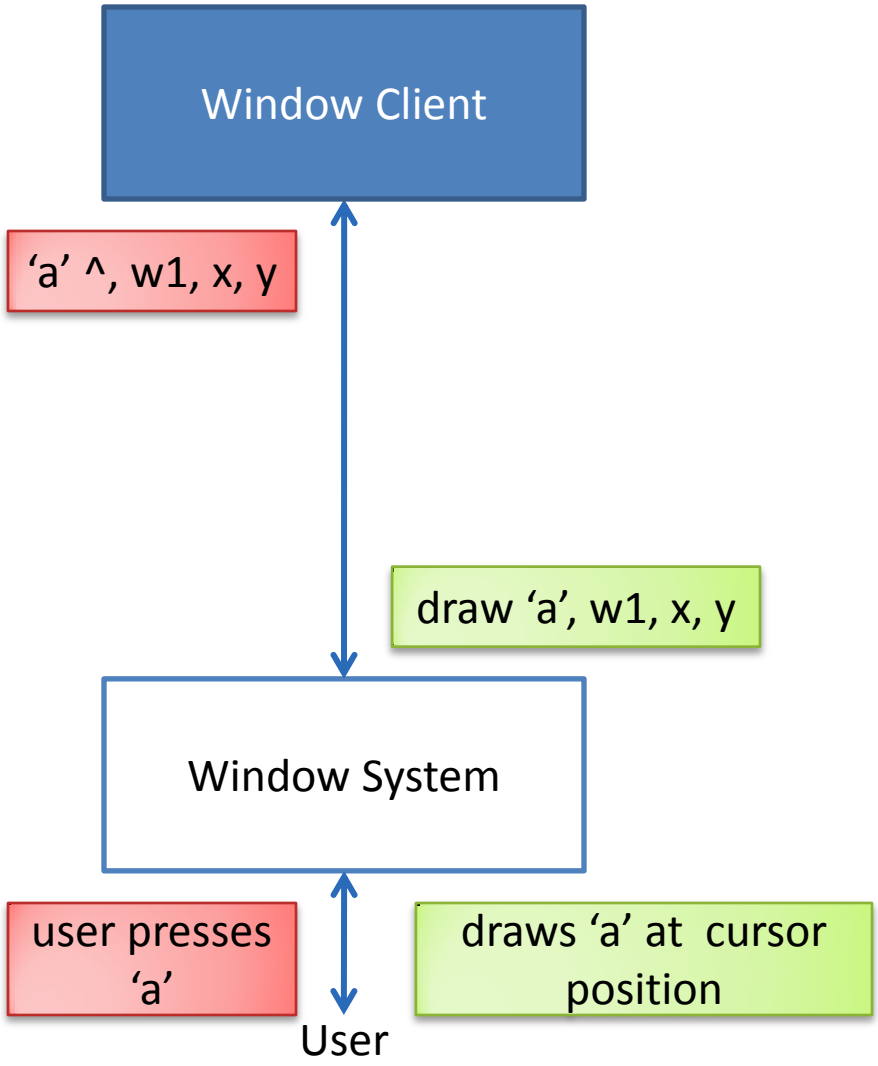


Flexibility vs. Automation Tradeoff in Abstraction Design

Relationship between Interactor and Different kinds of Objects



Windows

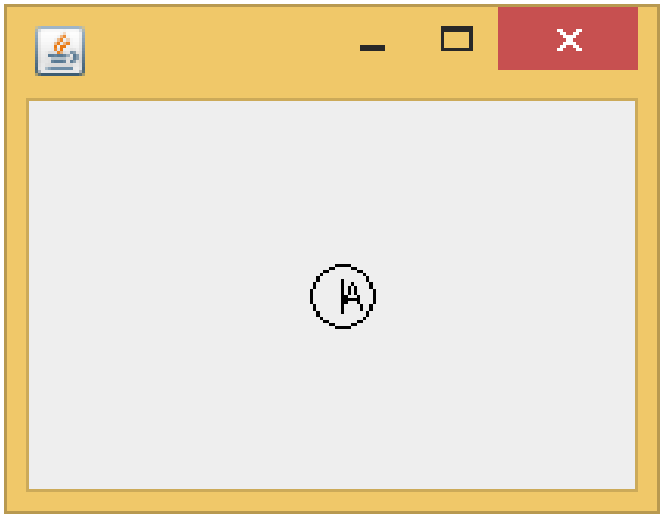


Window are untyped rectangular screen areas in which each point is a pixel

Input indicates keyboard and/or mouse operations

Output draws text, shapes

Example



Draws the last character entered at the last position at which the mouse was clicked

Draws a carat next to the character and a circle around it

Using Java Window to Define a Widget

```
public class ACircledCharacterDrawer extends JFrame implements
MouseListener,
…
// called when                    event for this component is dequeued
public void paint (Graphics g) {
  super.paint(g); // clears the window
  // better to use FontMetrics to center circle
  g.drawOval(charX - X_OFFSET, charY - Y_OFFSET, DIAMETER, DIAMETER);
  g.drawLine(charX, charY, charX, charY - CARAT_LENGTH);
  g.drawString("" + lastChar, charX, charY);
}
public void keyTyped(KeyEvent event) {
  setChar(event.getKeyChar());
}
public void setChar(char newValue) {
  lastChar = newValue;
  repaint();// enqueues a paint event
}
public void mousePressed(MouseEvent
  charX = event.getX();
```

Widget = Window + reusable functionality

Window

Output painter

Input notification method

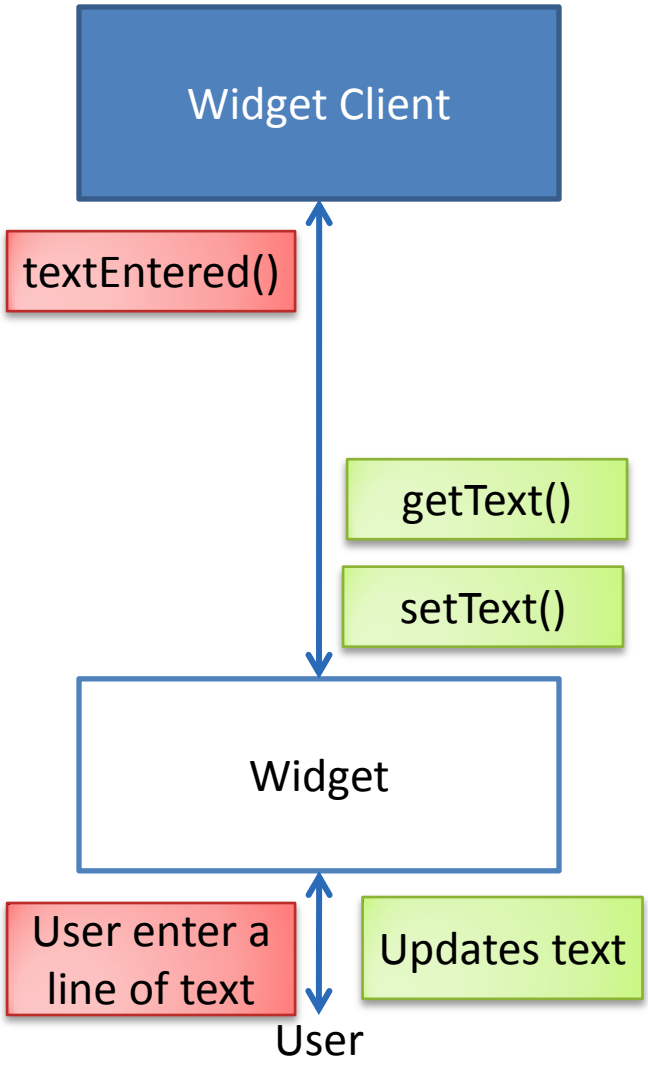
Reusable functionality

In Java >= 1.1 input is provided through the observer pattern

This subclass of a window is listening to its own window events (registration methods in constructor not shown)

Notification method called by a lower-level abstraction == callback

Widgets and Toolkit



Widgets are typed windows

Input callbacks are widget specific (e.g. slider moved, text changed, text inserted)

Output and other calls are widget specific (change text or slider position)

Toolkit = Set of all Widgets

e.g. AWT, Swing

Widget Use Example

```
JTextField jTextField = new JTextField("JTextField: Edit me");
AJTextFieldListener jTextFieldListener = new
AJTextFieldListener(jTextField);
jTextField.addActionListener(jTextFieldListener);
jTextField.getDocument().addDocumentListener(jTextFieldListener);
```

Widget creation and observer registration

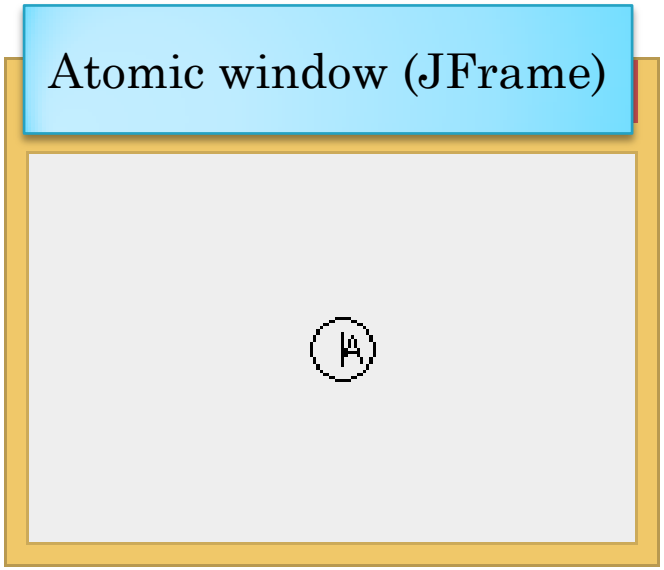
```
public class AJTextFieldListener implements ActionListener,
DocumentListener{
  JTextField jTextField;
  public AJTextFieldListener(JTextField aJTextField) {
    jTextField = aJTextField;
  }
  public void actionPerformed(ActionEvent e) {
    System.out.println("New text entered:"  + jTextField.getText());
  }
  public void insertUpdate(DocumentEvent e) {
    int newPos = e.getOffset();
    char newChar = jTextField.getText().charAt(newPos);
    System.out.println("Character " + newChar + "  inserted at " +
newPos);
  }
   …
```

Notification of new line entered

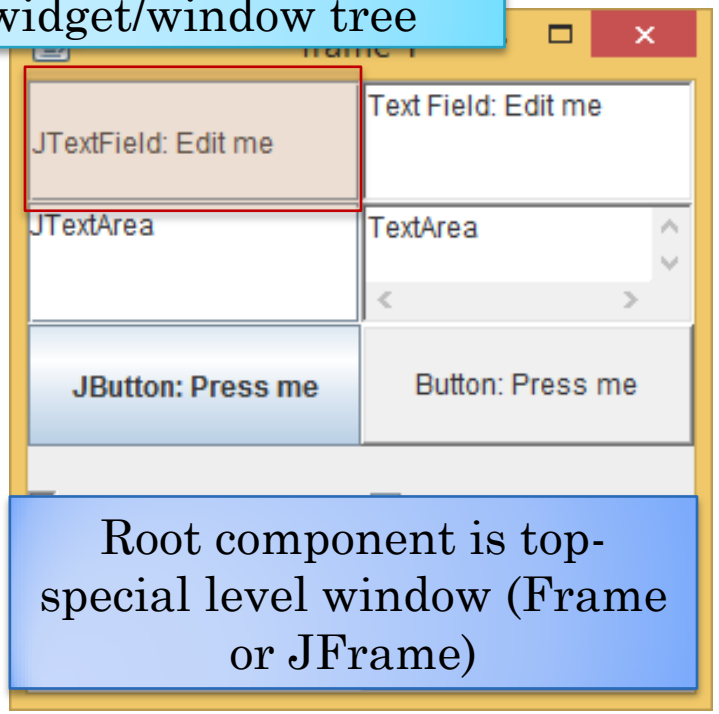
Notification of new character insertion

Atomic vs Composite Widgets/Windows

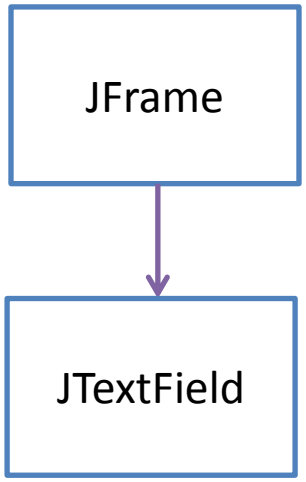
Atomic window (JFrame)



Atomic component of a widget/window tree



Root component is top-special level window (Frame or JFrame)



Top-level window manipulated by (customizable) window manager which puts border and provides operations to move, resize, iconify it

Creating, Laying-out and Displaying a Hierarchy

```
JFrame frame = new JFrame(theTitle);
frame.setLayout(new GridLayout(5, 2));
JTextField jTextField = new JTextField("JTextField: Edit me");
…
frame.add(jTextField);
…
frame.setSize(300, 300);
frame.setVisible(true);
…
```

Parent node

(Sub)tree layout

Child node

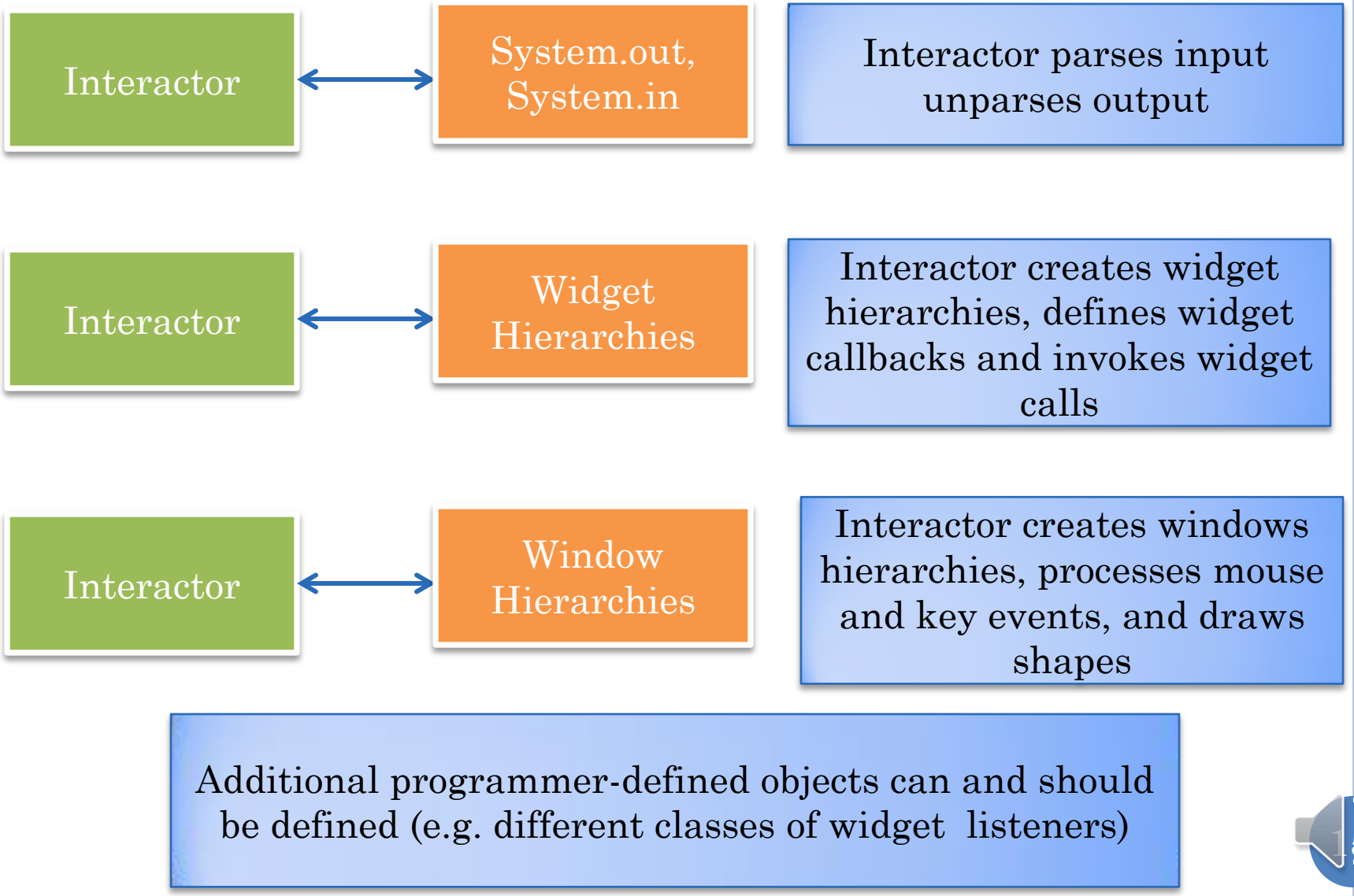
Link creation

Display Tree

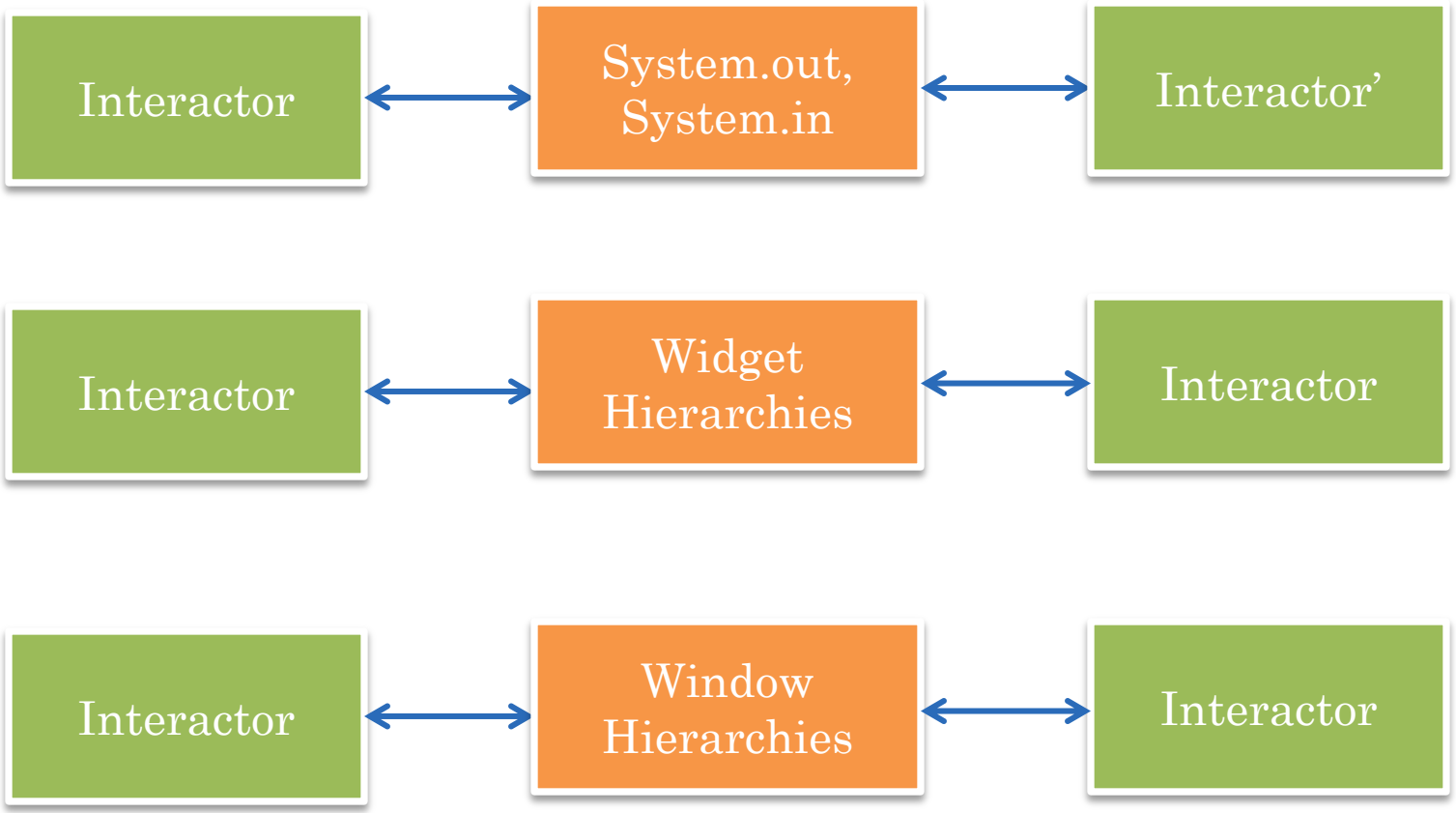
Here child created independent of parent and can be re-parented

In some systems a child is created as part of a parent: parent specified when child created

Different kinds of (Predefined) Helper UI Abstractions



Interactor-UI Abstraction Decoupling



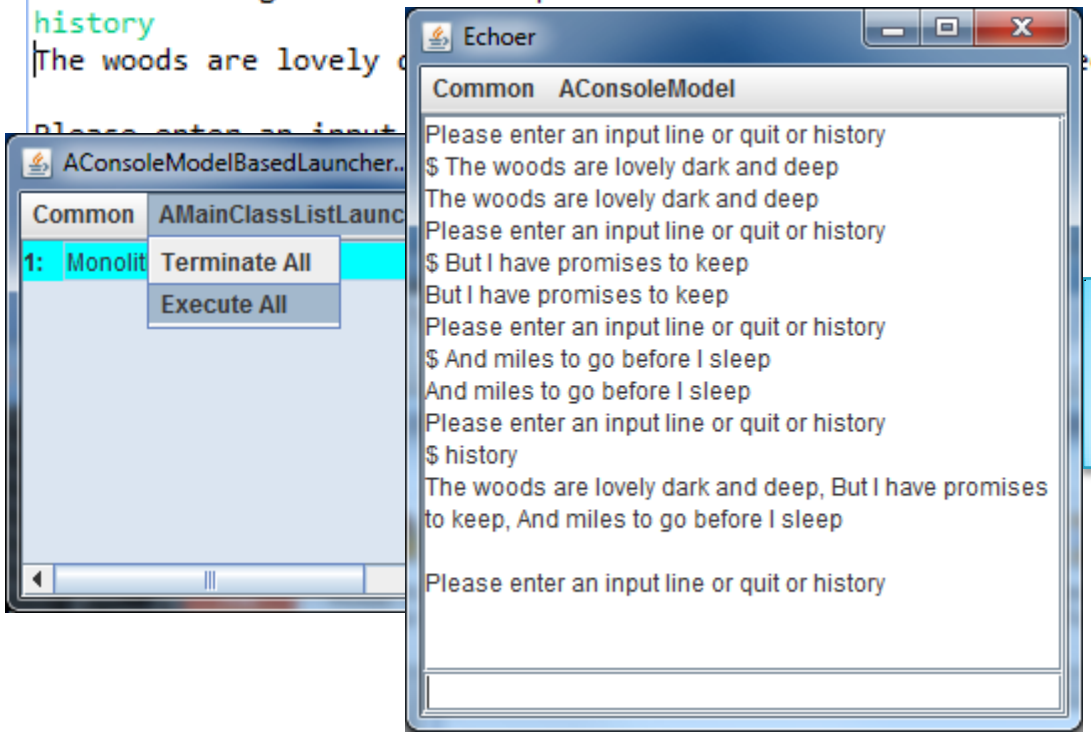
An interactor can be bound to different kinds of UI abstractions

A UI abstraction can be bound to different kinds of interactors

Separation in Console-based UIs

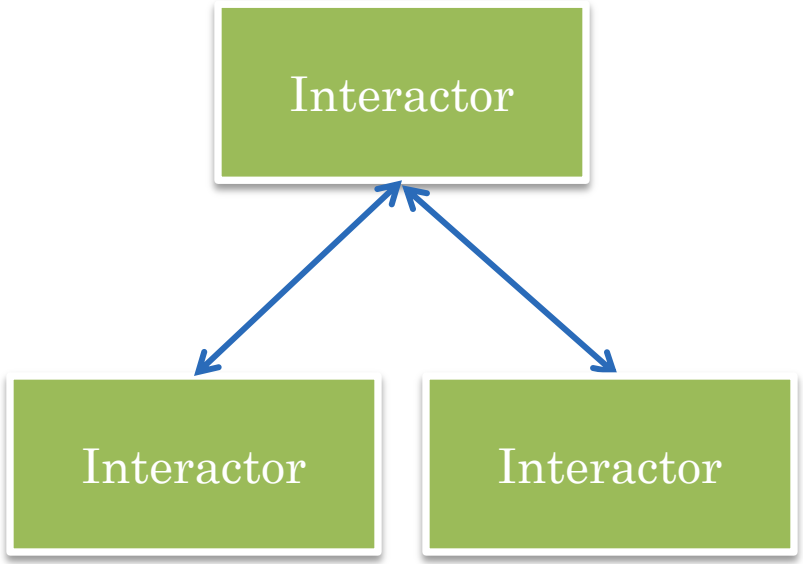

```
Please enter an input line or quit or history
The woods are lovely dark and deep
The woods are lovely dark and deep
Please enter an input line or quit or history
But I have promises to keep
And miles to go before I sleep
history
The woods are lovely d                    eep, And miles to go before I sleep
```

Eclipse Interactor



Custom interactor for launching multiple processes

Interactors can be Structured



When user interfaces are composed

Subinteractors can intract directly with models or through parent interactoes

Summary of Concepts in Interactors

- Window, Widget, and Console Layers
- Calls (callbacks) invoked by higher (lower) layer on lower (higher) layer
- Window: Rectangular Area
  - Input (callbacks): Key, Mouse Events
  - Output (calls) : shape draw calls (drawLine, …)
- Widget: Window embellished with higher-level behavior
  - Input (calls): arbitrary (e.g. new text changed)
  - Output (callbacks): arbitrary (e.g. change text)
- Console: a text widget used to enter and display text lines
- Window/Widget Hierarchies:
  - Trees associated with layouts
  - Usually made visible after they have been created
- An interactor uses one of more of the UI abstractions above as helper objects
- Interactor and the UI abstraction objects are decoupled
  - System.in, System.out used in different kinds of interactors

Model/Interactor Pattern