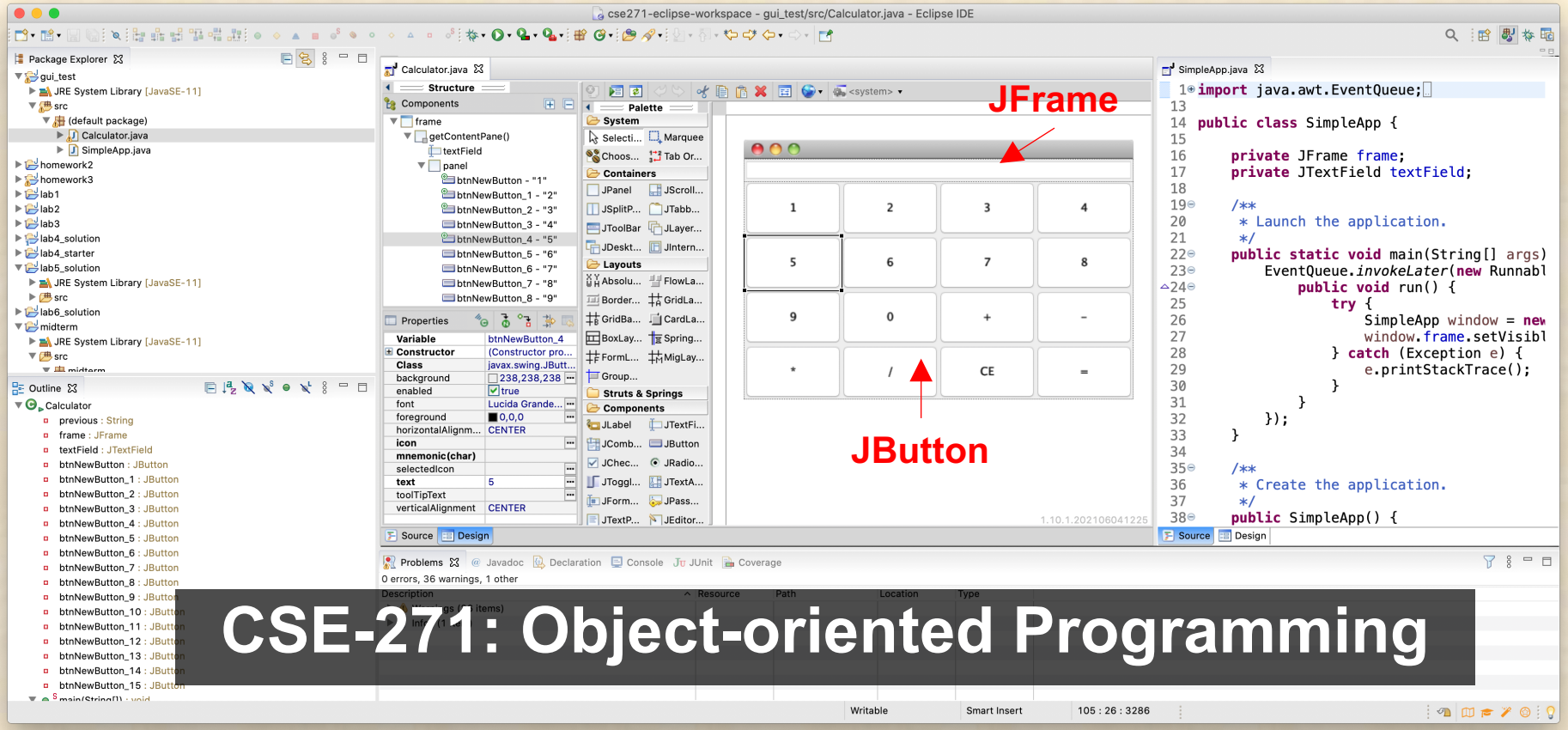
Object-oriented Design & Event-driven Programming for GUI



CSE-271: Object-oriented Programming

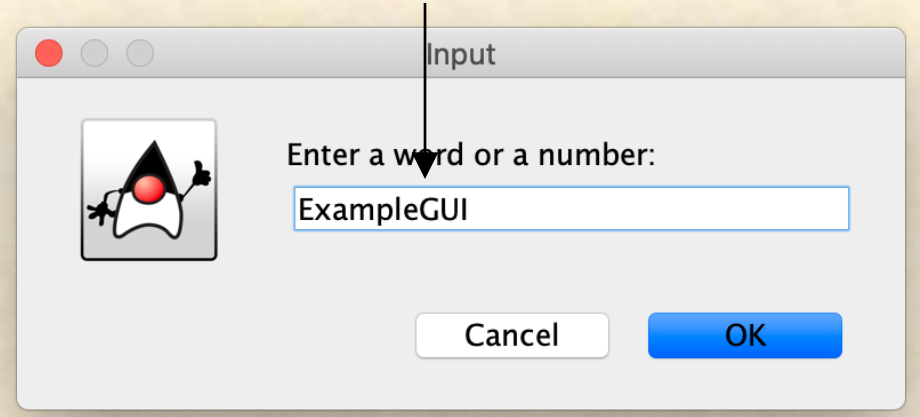
Text-based Input-Output

- So far we have used only text-based I/O
  - Read/Write data to devices (keyboard/screen) using `Scanner` or `PrintWriter` (*e.g.*, `System.out`)
  - Read one input at a time (line-by-line) and process it
- Advantages of text-based I/O
  - Simple and works on all machines in a uniform manner
    - **Requires very little energy and compute power**
  - Operating system provides mechanisms to redirect I/O and multiple programs can be chained together to achieve more complex functionality
- Drawbacks of text-based I/O
  - Can be cumbersome when several related data needs to be entered and it is inconvenient to backtrack and fix incorrect inputs.
  - Certain types of inputs (using pictographic techniques) cannot be effectively achieved using text-based I/O
  - It is hard to manipulate digital photographs and images in text mode
  - Aesthetic appeal is harder to achieve

Graphical User Interface (GUI)

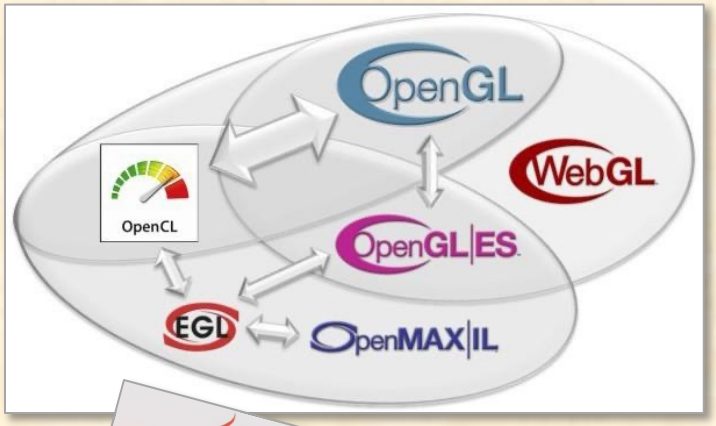
- Alternative for text-based I/O
  - Uses graphics features of modern computer
- Essentially uses images to display information
  - Uses different input devices, such as: mouse, touch, keyboard, etc.
- Modern GUI use:
  1. Hardware capabilities of computers
  2. Uses software libraries
     - Typically object-oriented
  3. Uses event-driven programming

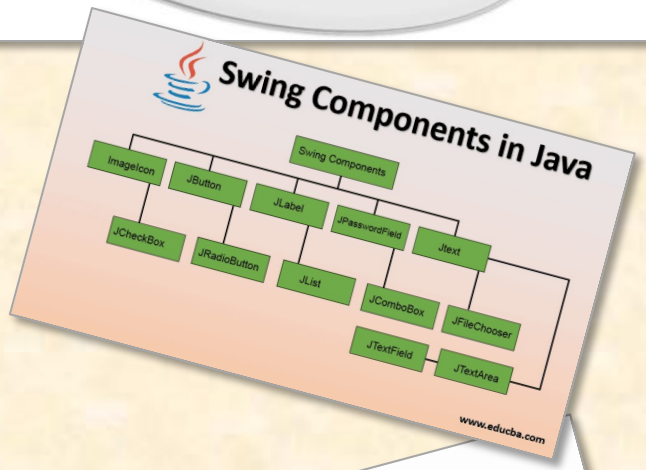
Event inputs and characters are essentially images



Graphics libraries for developing GUIs

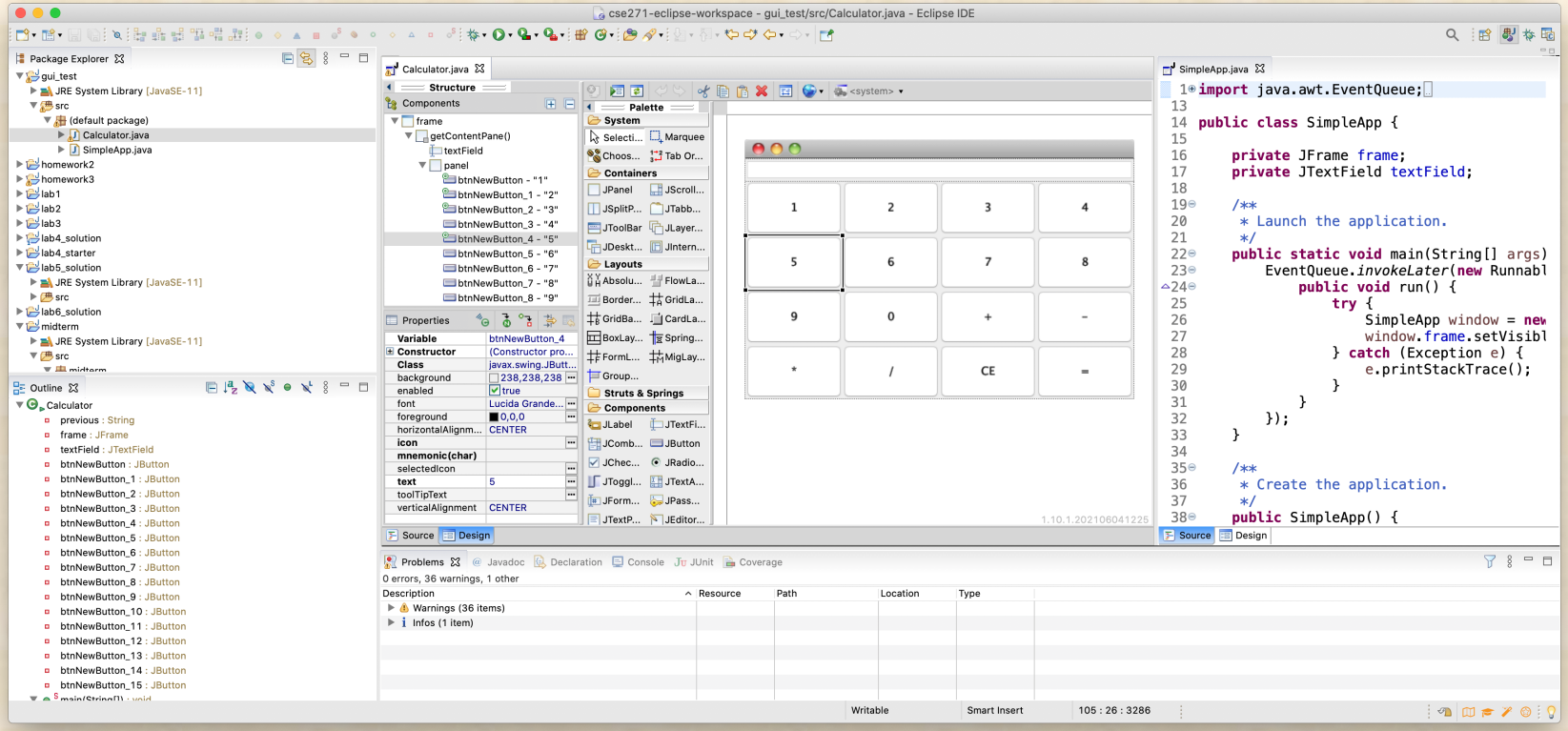
- Using graphics mode to display text, images, and objects is a complex task
  - A number of methods need to be developed even to display simple text
- Consequently, several libraries have been developed to ease effective use of graphical displays
  - Some are a part of the OS
  - Some are a part of the Java language
    - Example: AWT (*older*), Swing, and JavaFX (*newer*)
- In some cases you have to develop your own libraries for performing specialized tasks.





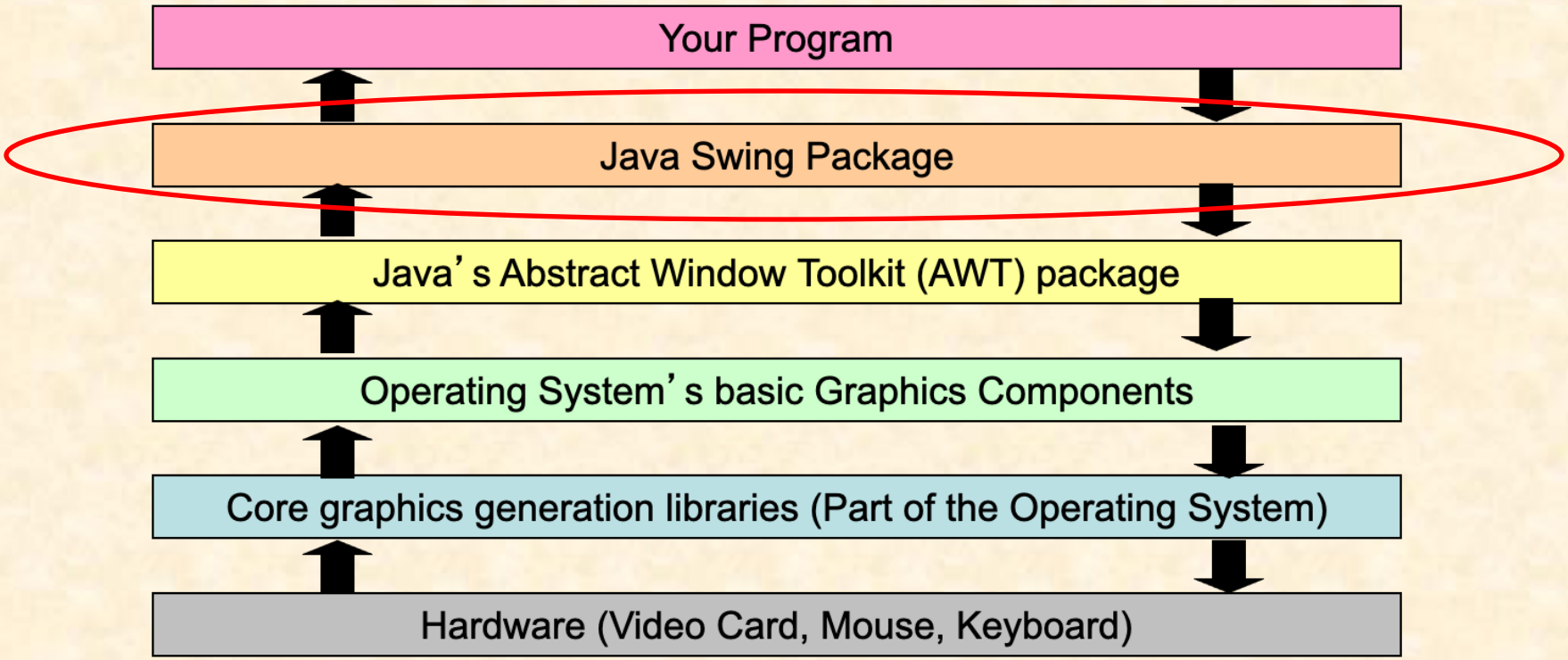
Java Swing: Library

- A versatile high-level GUI library developed in Java
  - Provides graphical input & output classes aka components
    - Inputs: Text entry boxes, Buttons, Checkboxes, etc.
    - Outputs: such as text and images, free-hand drawing



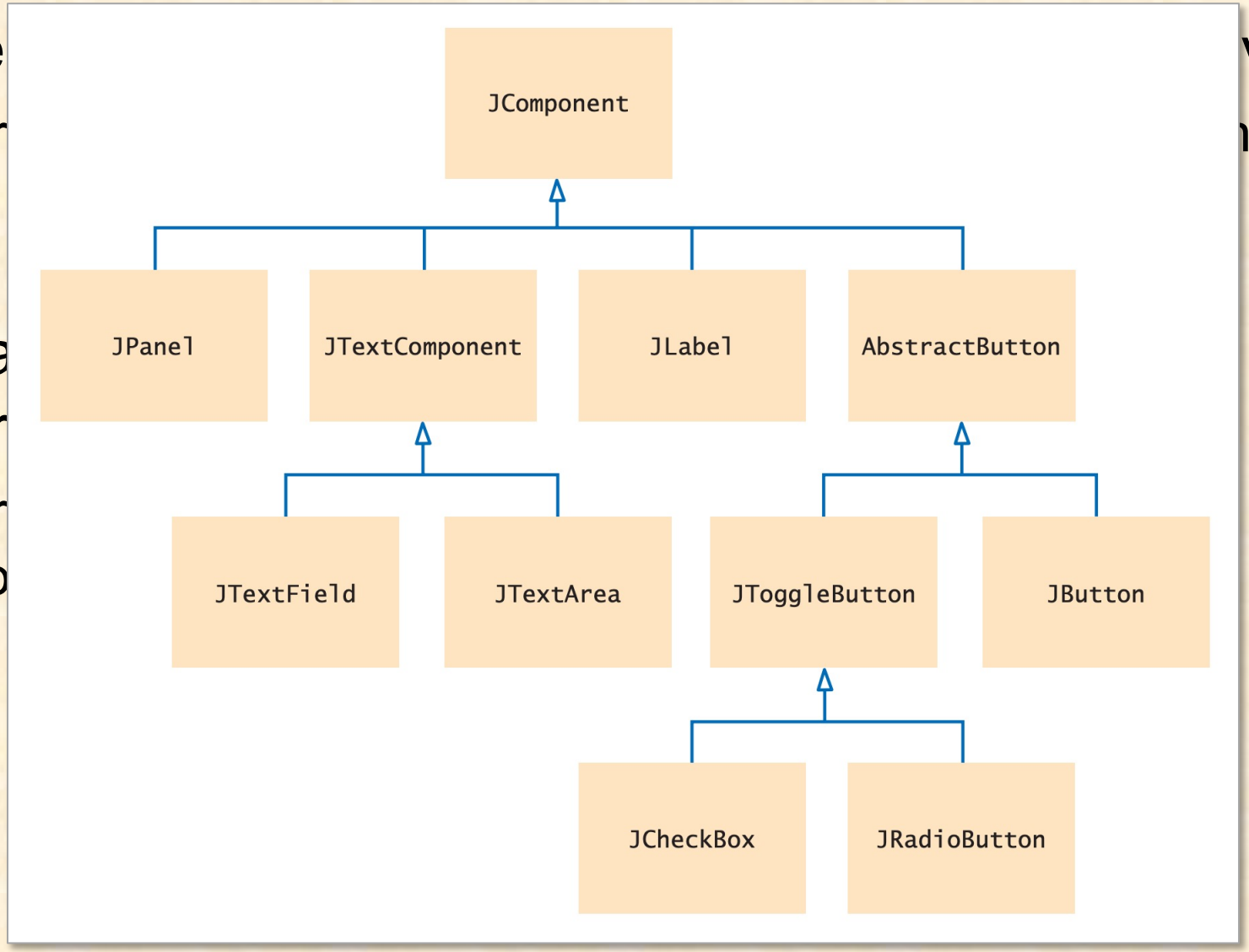
Java Swing: Library

- A versatile high-level GUI library developed in Java
  - Provides graphical input & output classes aka components
    - Inputs: Text entry boxes, Buttons, Checkboxes, etc.
    - Outputs: such as text and images, free-hand drawing



Java Swing: OOP

- A ve va
  - Pr ents
    -
    -
- Java
  appr
  - Pr
    co



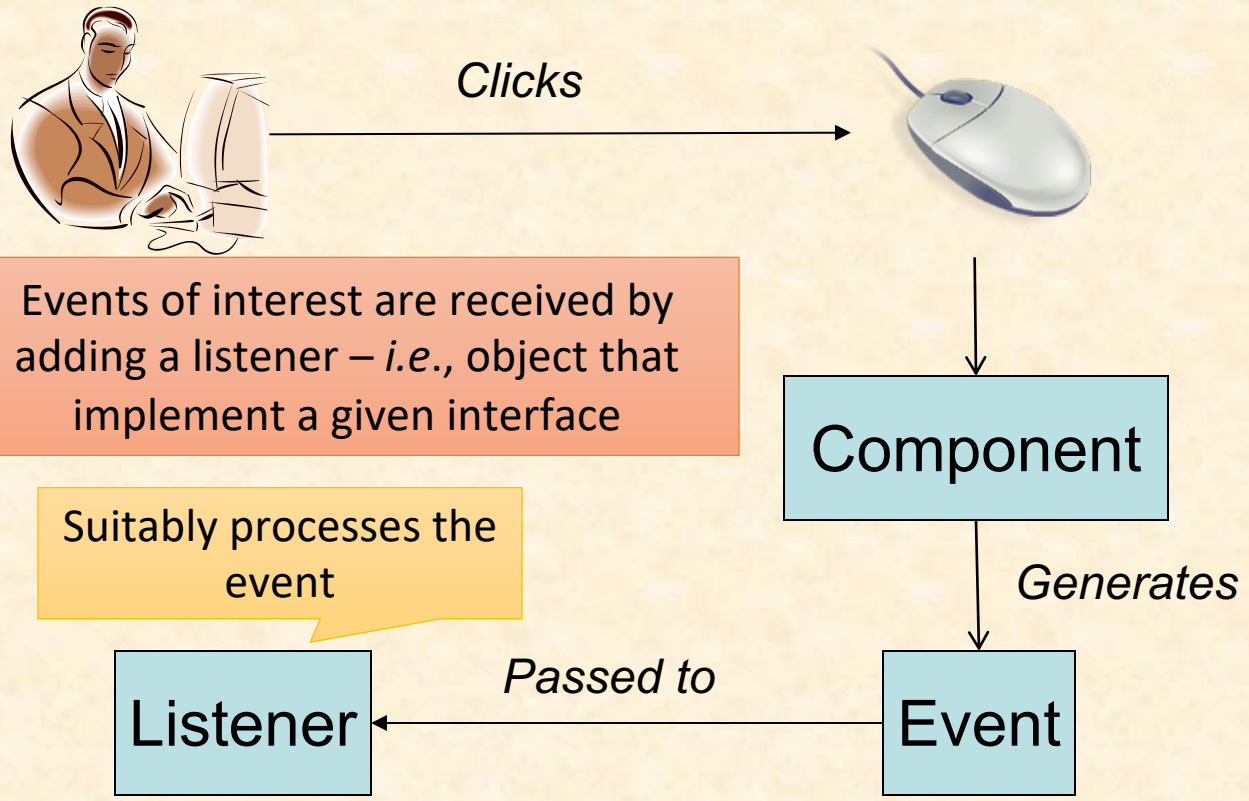
Java Swing: Event-driven

- A versatile high-level GUI library developed in Java
  - Provides graphical input & output classes aka components
    - Inputs: Text entry boxes, Buttons, Checkboxes, etc.
    - Outputs: such as text and images, free-hand drawing
- Java Swing heavily uses object-oriented design approaches
  - Provides an hierarchy of classes for different GUI components
- It uses an event-driven approach for interactions
  - Handles mouse motions and other default operations
  - Your program must indicate the specific types of events to which it wishes to respond
    - By registering action listeners (implement `ActionListener` interface)

Java Swing: Event-driven

- A
  - va
  - ents
- Ja
  - ap
- It
  - ts to

Overview of event-driven approach for GUI events



Event-driven Programming

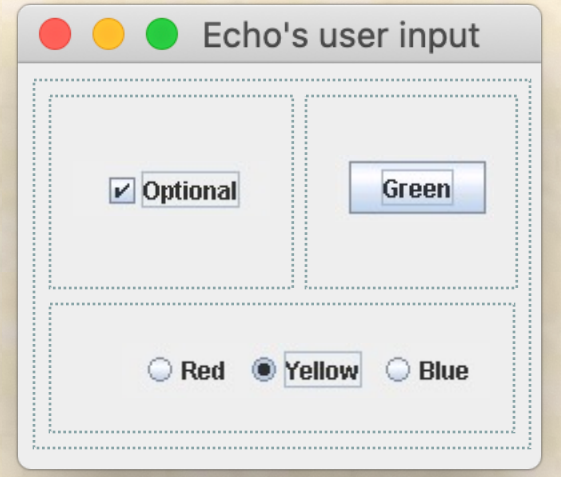
- All modern graphical interactions are performed using a concept called Event-driven programming
  - This includes Swing and AWT
- Event-driven programming
  - All actions (drawing, mouse movements, key press on keyboard) are represented as events occurring in the system
    - Process of generating an event is called *firing an event*
- Events are stored in an event queue and processed one after another
  - Classes or objects that are interested in receiving certain types of events are called *listeners*
  - Not all listeners may process (or consume) an event. They may decide they are not interested in the event and pass it on to some other listener
- Classes or objects that actually process events are called *event handlers*

Simple Swing GUI methods

Manually creating a Swing GUI

Overview of the process

- Creating a Swing-GUI involves several steps
  1. First have a clear mental model of GUI
     - Best to hand-draw the GUI
     - Consider CSE-211: UI/UX course
  2. Create a subclass of `JFrame`
  3. Use `JPanel`(s) to organize components
     - Set appropriate layout for each panel
     - Nesting panels with different layouts is tricky
  4. Add different UI components
     - Example: `JButton`, `JTextField`, etc.
  5. Add event listeners (*i.e.*, objects) to components
     - Event listeners perform associated logic
     - These are standard methods and can perform any operation
  6. Finally pack and show your GUI
- Like every program, develop the GUI incrementally
  – Test each feature as you add them

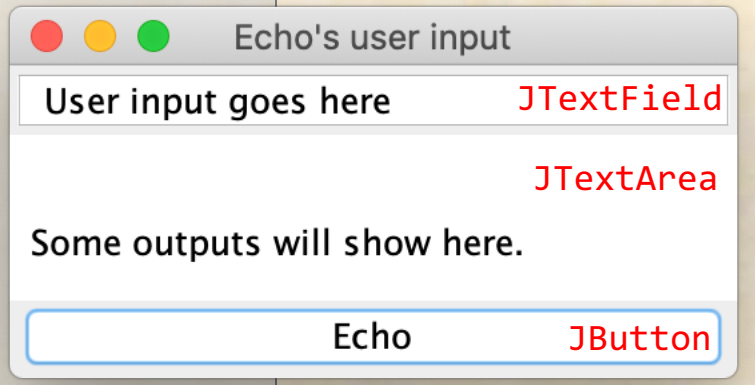


Example GUI
(No event processing, yet)

```
public class EchoGUI extends JFrame {
  private JTextArea output = new JTextArea();
  private JTextField input = new JTextField();
  private JButton   doIt   = new JButton("Echo");

  public EchoGUI() {
    super("Echo's user input");
    setPreferredSize(new Dimension(600, 400));
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    JPanel bigBox = new JPanel(new BorderLayout());
    bigBox.add(input, BorderLayout.NORTH);
    bigBox.add(output, BorderLayout.CENTER);
    bigBox.add(doIt, BorderLayout.SOUTH);
    // addBtnAction();
    this.add(bigBox);  // Add Panel to JFrame
  }

  public static void main(String[] args) {
    EchoGUI eg = new EchoGUI();
    eg.pack();  // Important
    eg.setVisible(true);
  }
}
```

Stops the program when user clicks on the close (☒) button

An unseen component to layout the GUI



Action Listener

- In the previous example, clicking on the button does absolutely nothing
  - Even though the button does fire an event when it is clicked
- Reason: No action listeners have been specified for the button
- Solution: Add an action listener to the button
  - Action listeners have to implement the `ActionListener` interface.
  - `ActionListener` interface has only 1 method namely `actionPerformed(ActionEvent e)`
    - The `ActionEvent` contains information about which component generated the event and details about the event.
    - The action listener method typically inspects the event and performs some action.

Simplifying Syntax for Anonymous Classes using Lambdas

- Lambdas are anonymous methods with terse syntax
- Java streamlines the use of Lambdas for ***functional interfaces***
  - Functional interfaces have exactly 1 method
  - Hence their parameters & returns values are easily deduced
- The `ActionListener` is a functional interface
  - It has only one method, *i.e.*, `actionPerformed`
- Hence, lambda syntax can be used for `ActionListeners`

```
doIt.addActionListener(new
  ActionListener() {
    @Override
    public void
    actionPerformed(ActionEvent e) {
      System.out.println("Click!");
    }
  });
```

Lambda:

```
doIt.addActionListener((e) ->
   { System.out.println("Click!"); });
```

Lambdas with 1 parameter and 1 line can be further simplified to:

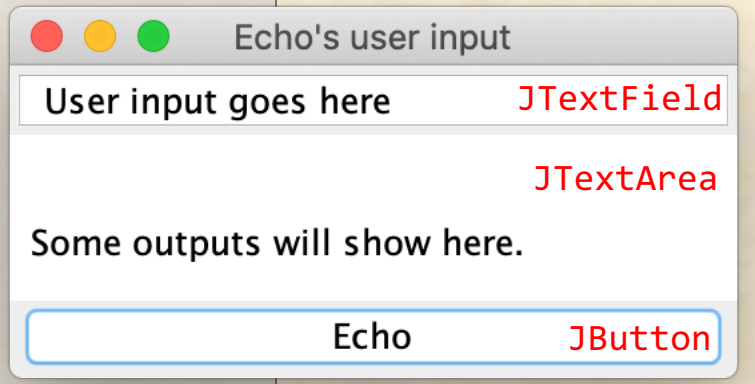
```
doIt.addActionListener(e ->
  System.out.println("Click!"));
```

Adding event processing

```
public class EchoGUI extends JFrame {
  private JTextArea output = new JTextArea();
  private JTextField input = new JTextField();
  private JButton   doIt   = new JButton("Echo");

  public EchoGUI() {
    super("Echo's user input");
    setPreferredSize(new Dimension(600, 400));
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    JPanel bigBox = new JPanel(new BorderLayout());
    bigBox.add(input, BorderLayout.NORTH);
    bigBox.add(output, BorderLayout.CENTER);
    bigBox.add(doIt, BorderLayout.SOUTH);
    doIt.addActionListener(e -> output.append(
                input.getText()));
    this.add(bigBox);
  }

  public static void main(String[] args) {
    EchoGUI eg = new EchoGUI();
    eg.pack();  // Important
    eg.setVisible(true);
  }
}
```

Stops the program when user clicks on the close (☒) button

An unseen component to layout the GUI



Layout managers to organize panels & components

Need for Layout Managers

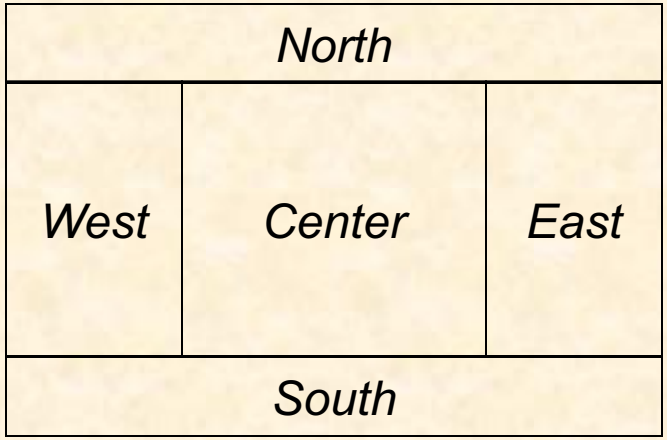
- Panels are containers that can have
  - Several components
  - And (sub) panels in them
- When multiple objects are placed in a panel
  - They need to be organized in some way
    - Particularly when window is resized
  - Organization includes:
    - Size and placement of objects within the panel
- Organization of components is performed using Layout Managers
  - Layout managers decide size and placement of each component in a container.

Layout Managers

- There are numerous layout managers
  - Only one layout manager can be set for each container
    - Choosing an appropriate layout manager can be tricky
    - Sometimes you may have to create sub-panels with different layout managers to achieve desired effects
- Some of the commonly used layout managers:
  1. Border Layout
  2. Flow Layout
  3. Grid Layout

Border Layout

- Places no more than 5 components into five regions:
  - *North*, *South*, *East*, *West*, and *Center*
    - Constants are defined in `BorderLayout` class.
  - Each region can have only 1 component or panel.
  - Use polymorphic `add()` method (defined for all containers) with suitable arguments to add components to the appropriate area.



- Height of North and South regions is determined based on required height of the component (or panel) added to that region
- Width of West and East regions are determined based on required width of component (or panel) added to that region
- Component (or panel) added to the Center region takes remainder of the space

Border Layout Example

```
public JPanel createSubPanel()  {
  JLabel l1 = new JLabel("Top");
  JLabel l2 = new JLabel("Bottom");
  JLabel l3 = new JLabel("Left");
  JLabel l4 = new JLabel("Right");
  JLabel l5 = new JLabel("Middle");

  JPanel sp = new JPanel(new BorderLayout());
  sp.add(l1, BorderLayout.NORTH);
  sp.add(l2, BorderLayout.SOUTH);
  sp.add(l3, BorderLayout.WEST);
  sp.add(l4, BorderLayout.EAST);
  sp.add(l5, BorderLayout.CENTER);

  return sp;
}
```

`JLabel` is a Swing Component that can be used to display non-editable information on a GUI.

Create a panel with suitable layout manager. You can also use `setLayout()` method to set/change the layout manager at any time.

Add labels to appropriate regions.

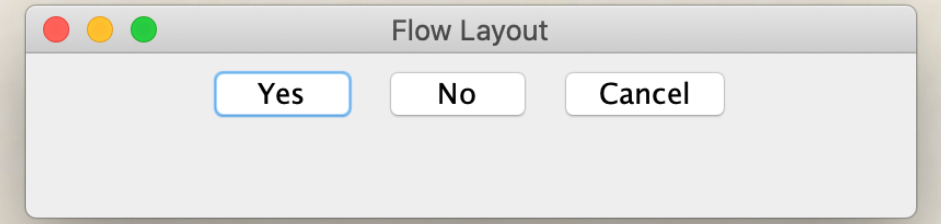
Flow Layout

- Organizes components one after another
  - Default layout for all `JPanel` objects
  - Can do: Left-to-right, right-to-left, or center
    - Left-to-right is the default behavior
    - In the order they were added to the container
    - Preferred width of the components are preserved
    - Components can have different widths but have the same height
    - Spacing between components can be adjusted

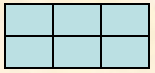
Flow Layout Example

```
public static JPanel createButtonPanel()  {
   JButton b1 = new JButton("Yes");
   JButton b2 = new JButton("No");
   JButton b3 = new JButton("Cancel");

   JPanel sp;
   sp = new JPanel(new FlowLayout(FlowLayout.CENTER));
   sp.add(b1);
   sp.add(b2);
   sp.add(b3);

   return sp;
}
```

Use a FlowLayout organization where buttons will be centered.



Grid Layout

- Organize components into a grid (*row* x *col*)
  - 2x3 grid: 
  - 1x3 grid: 
  - 2x1 grid: 
- Grid characteristics
  - Grids are specified when layout manager is created
    - Can be changed later on.
  - Each grid spot has the same size
  - Components are stretched to fill the grid space

A 3x3 Grid of buttons

```
public class Grid extends JFrame {
  public Grid() {
    setPreferredSize(new Dimension(150, 150));
    setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    // Create a panel with 9 buttons
    JPanel subPanel = new JPanel(new GridLayout(3, 3));
    for(int i = 1; (i < 10); i++) {
      JButton button = new JButton("" + i);
      subPanel.add(button);
    }
    add(subPanel);
  }

  public static void main(String[] args) {
   Grid grid = new Grid();
   grid.pack();
   grid.setVisible(true);
  }
}
```

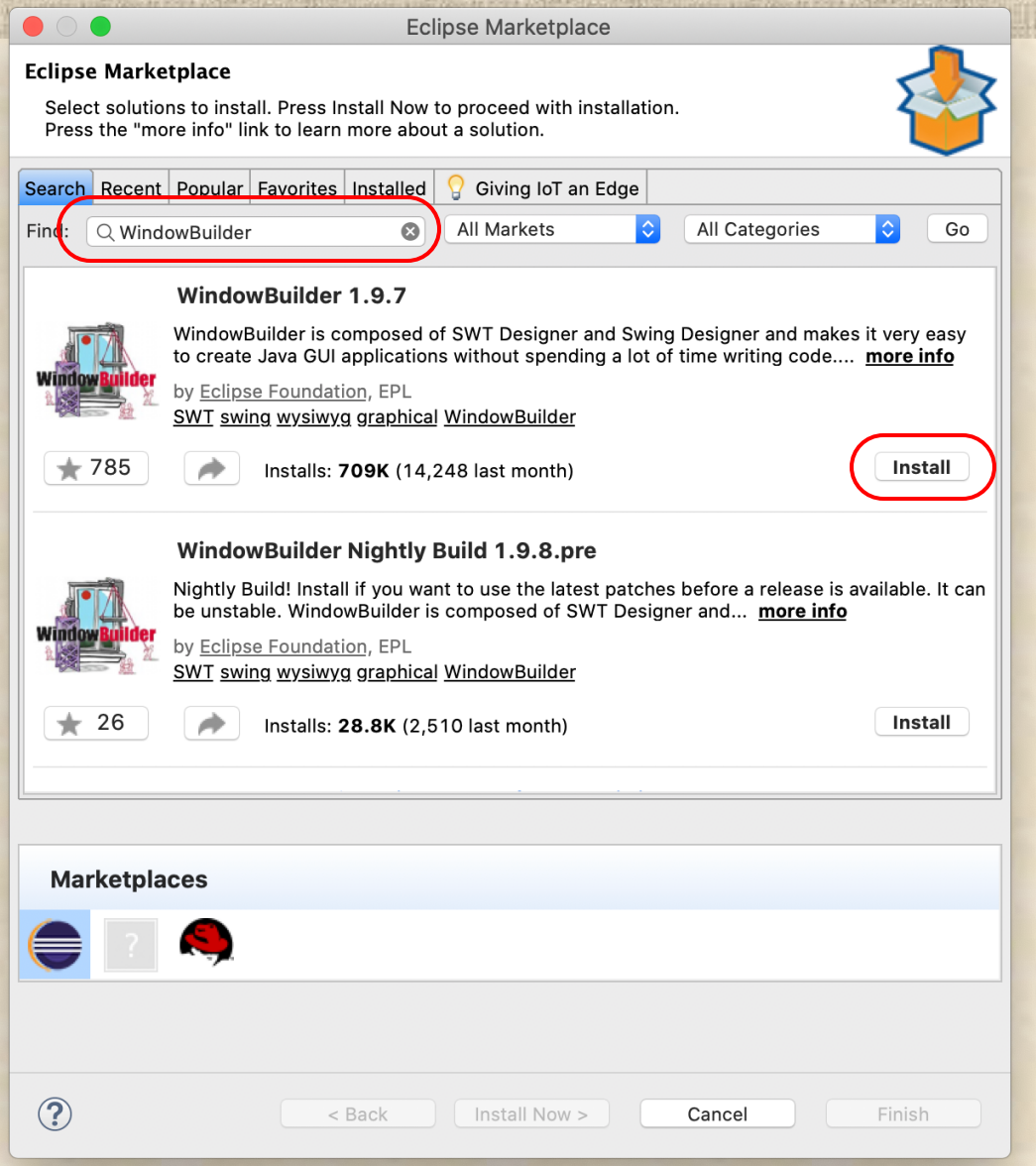
GUI Builders: Automated tools for developing GUIs

GUI Builders

- Developing GUIs can be tedious work
  - Requires a lot of imagination and understanding of components
  - Can be hard for customers to visualize end product
- GUI builders are software that help build GUIs
  - They let users drag-n-drop components and generate source code
  - They are popular in many domains, including web-development
- Advantages:
  - They enable rapid prototyping and have a lower learning curve
  - Good for GUIs that are of low to medium complexity
- Disadvantages:
  - They cannot handle complex GUIs and interactions
  - The GUIs can be hard to maintain and generated code is very clunky

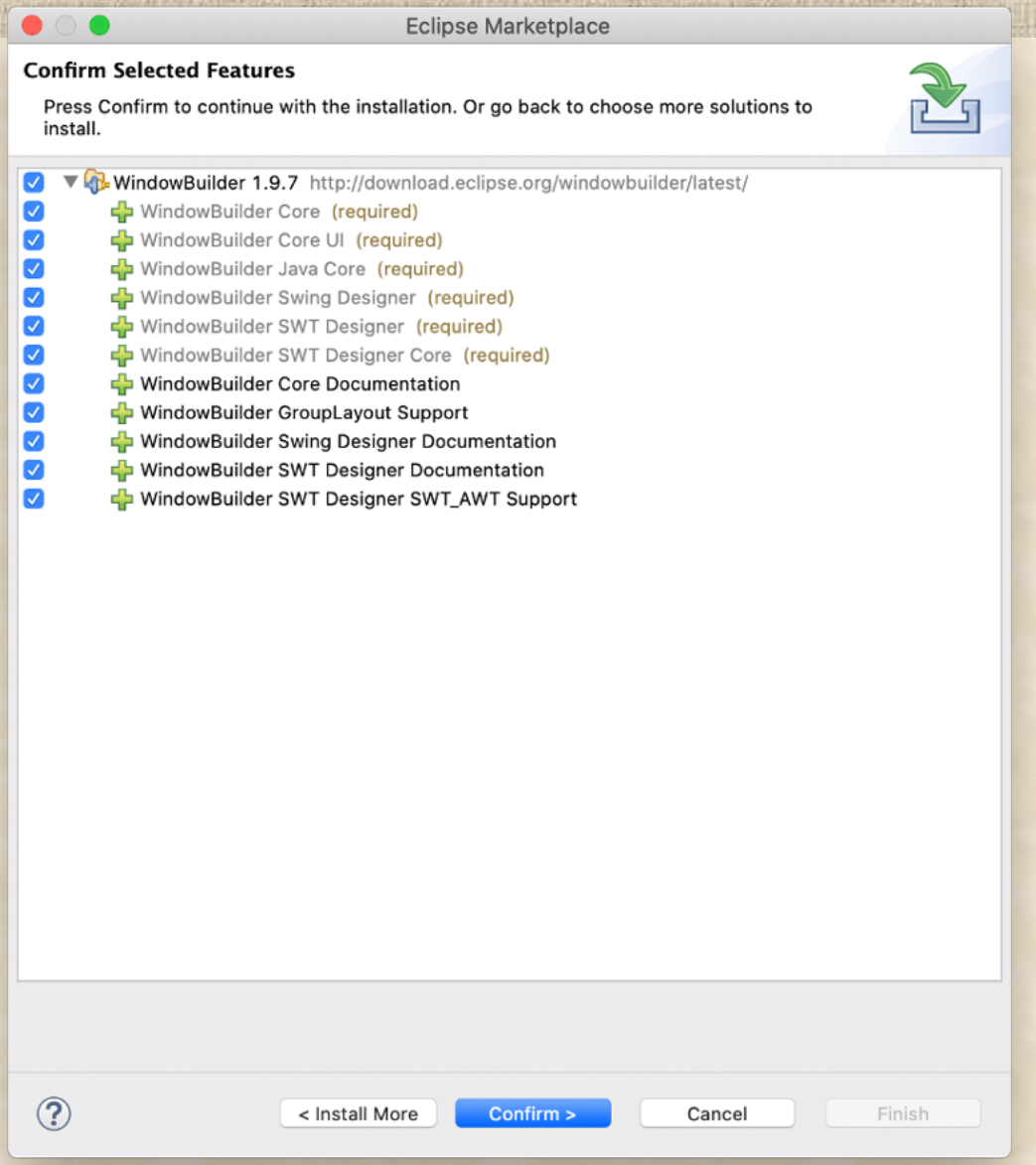
Installing Eclipse Window Builder

1. From Eclipse's main menu select `Help` → `Eclipse Marketplace...`
   - Search for `WindowBuilder` and click `Install`



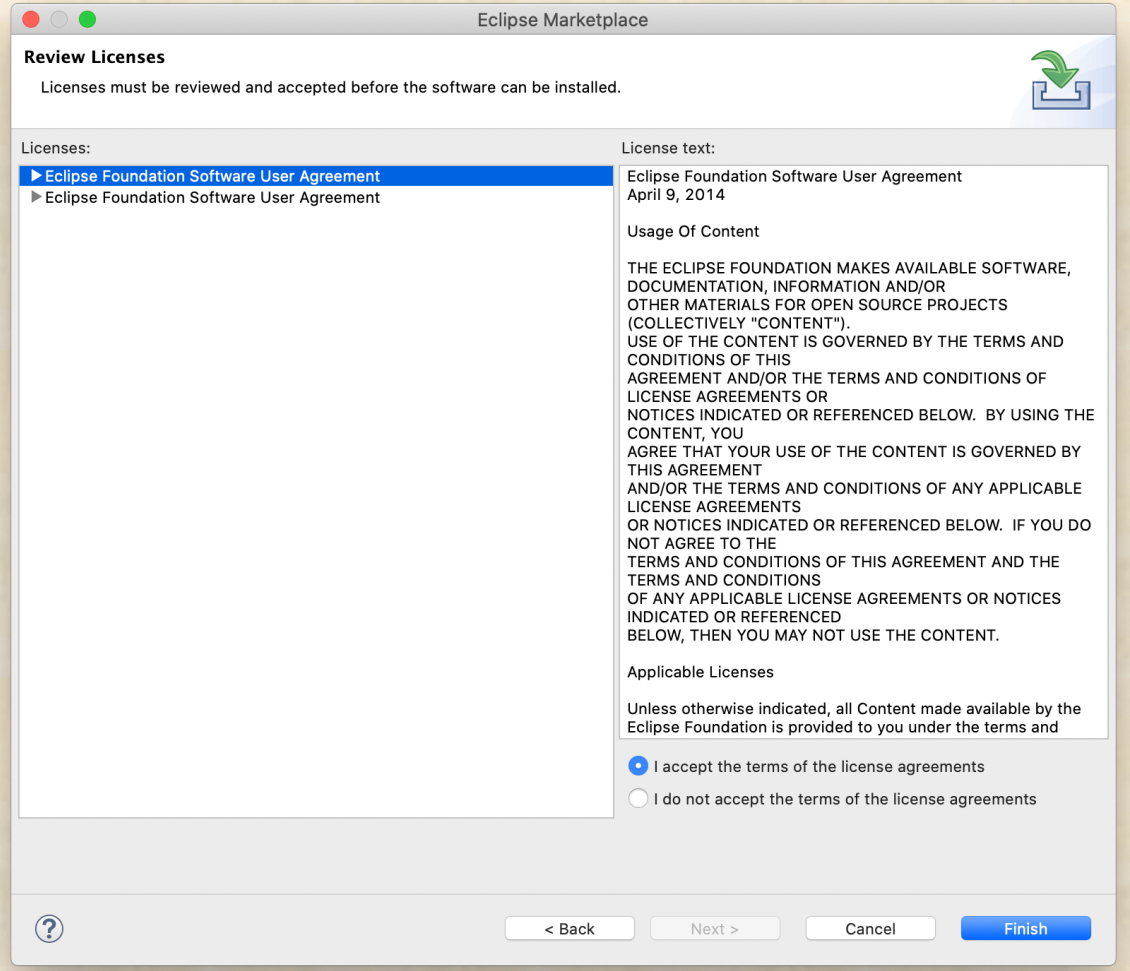
Installing Eclipse Window Builder

1. From Eclipse's main menu select `Help` → `Eclipse Marketplace...`
   - Search for `WindowBuilder` and select it
2. Select all the components



Installing Eclipse Window Builder

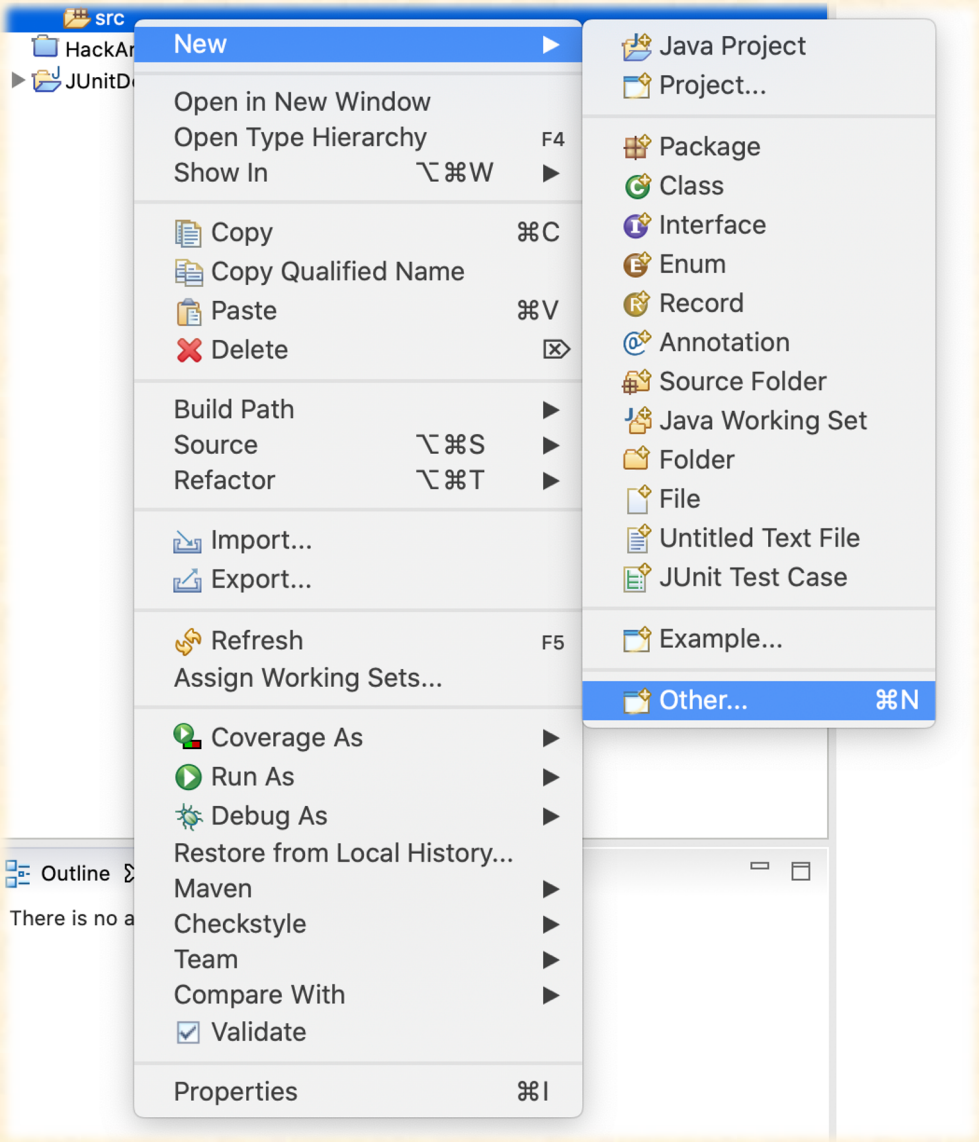
1. From Eclipse's main menu select `Help` → `Eclipse Marketplace...`
   - Search for `WindowBuilder` and select it
2. Select all the components
3. Accept license and the software will be installed



Using Window Builder

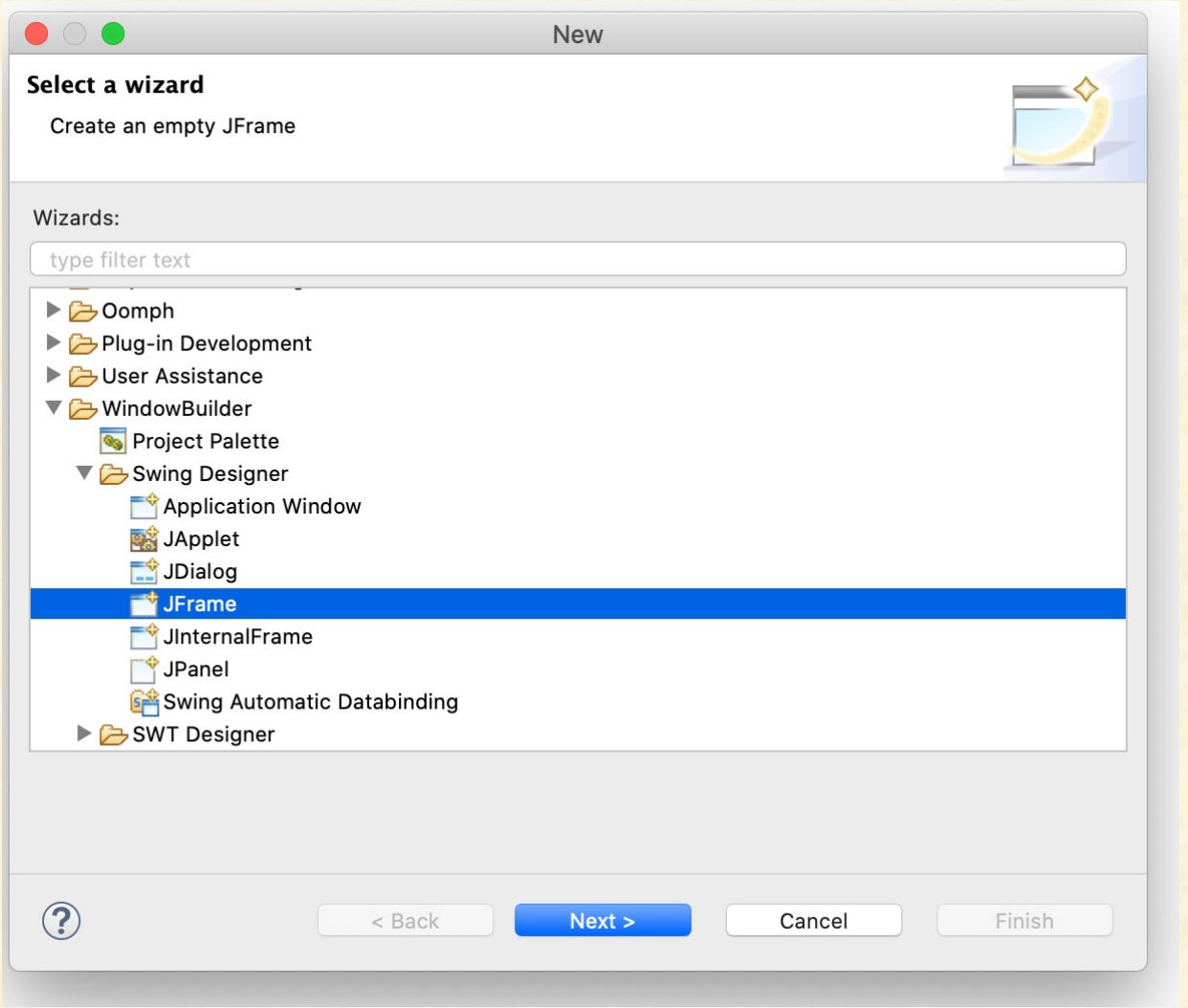
- To create a GUI using Window Builder

1. Create a Java project as usual
2. Next, right click on src folder and select `New` ➢ `Other...` option



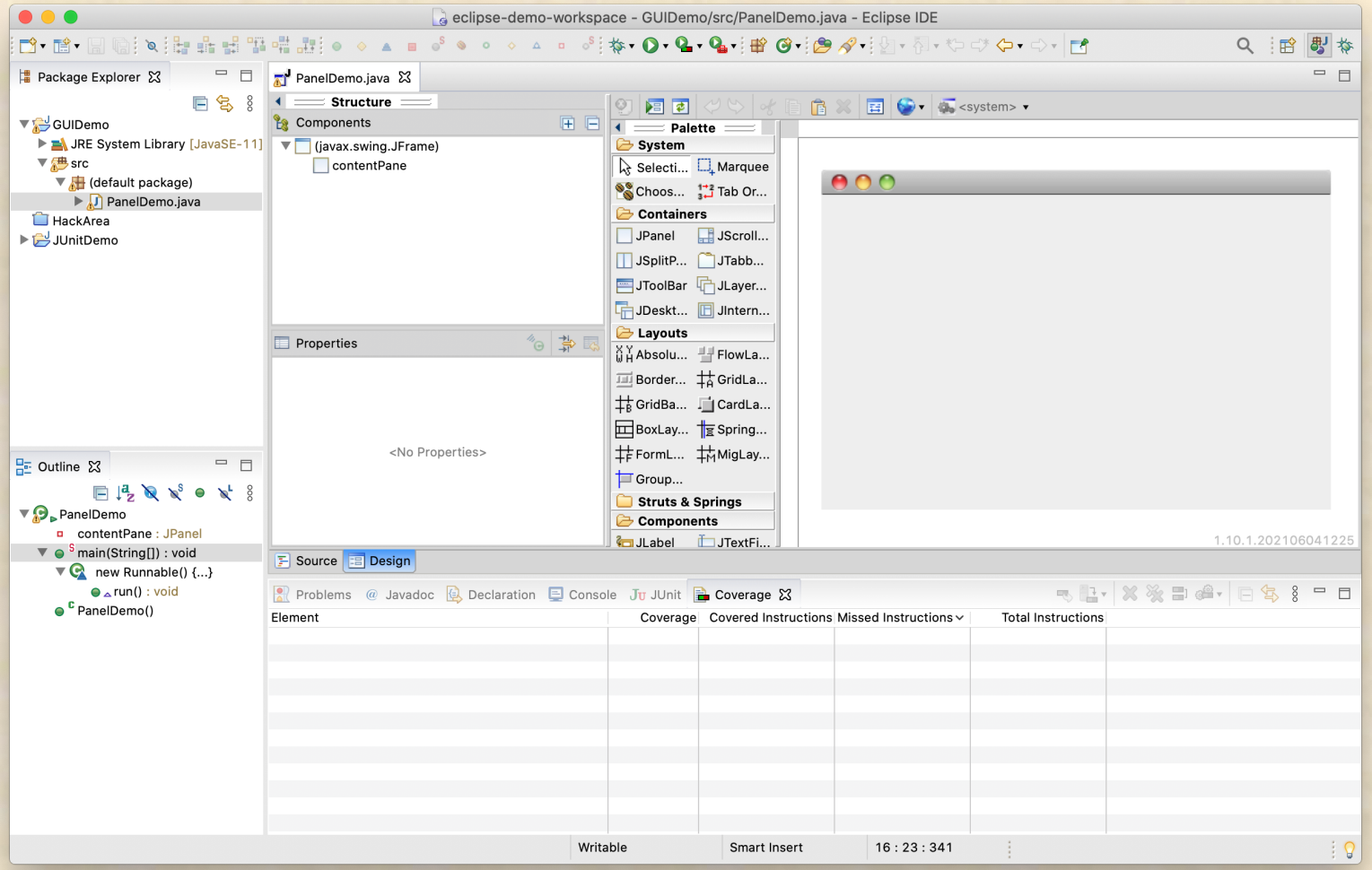
Select type of GUI to design

- In the dialog box, select `WindowBuilder` ▻ `Swing Designer`
- Select `JFrame` create a GUI with a default main method



Use design view to create GUI

- Click “Design” tab to use various graphical tools to create the GUI
  - Source code is automatically updated



See video on Canvas for demo of Eclipse Window Builder