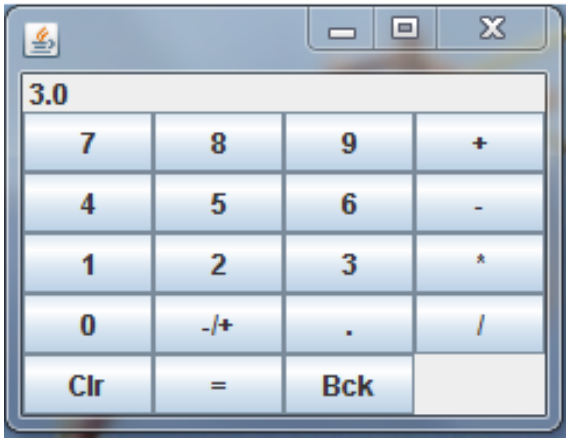
Chapter 9
Designing Graphical User Interfaces (GUIs)

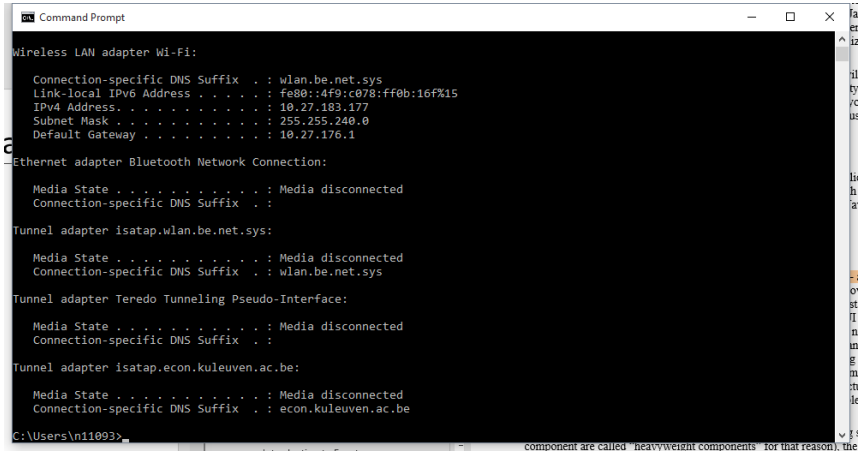
Overview

- The basics of GUIs in Java
- A tour of Java GUI libraries
- Containers and components
- Swing: the full picture
- Layout managers
- Understanding events and event listeners
- Closing topics

The basics of GUIs in Java

- What is a graphical user interface?





Graphical User Interface (GUI) Vs. Command Line Interface (CLI)

The basics of GUIs in Java

- Java has quite a number of GUI frameworks to build graphical
- Mostly due to history
- But also functional:
  - Some frameworks try to mimic the host system as closely as possible
  - Others go for fancy, modern look
  - Focus on robustness, simplicity versus performance (3D graphics etc.)

A tour of Java GUI libraries

Abstract Window Toolkit (AWT)

- Together with first Java release in 1995
- Provides a standard widget toolkit
- Widgets: buttons, checkboxes, etc.
- Thin layer above host operating system (“heavyweight components”): AWT is platform dependent
- Also contains many other aspects: layout managers, event listeners, classes to deal with keyboard and mouse

A tour of Java GUI libraries

Swing

- Originally conceived as more advanced version of AWT
- Not closely related to OS ("lightweight components"), written entirely in Java
  - There are still a few heavyweight components in Swing, more about these later
- Thus: more customization options possible
- Hence also includes drawing code (to draw lines, rectangles, etc.)
- Implemented as an extension of AWT
- But still tries to emulate the "look and feel" of the host OS
- Old but remains in very widespread use today

A tour of Java GUI libraries

Standard Widget Toolkit (SWT)

- Origins also date back to beginnings of Java
- Conceived by IBM as an alternative to AWT, which was considered buggy at the time
- Also heavyweight and closely coupled to OS
- Currently not that common anymore, but still used by Eclipse to construct its user interface

A tour of Java GUI libraries

JavaFX

- Original goal to offer a rich platform for visual applications
- Around since 2008, but only part of Java JRE and JDK recenty
- Oracle has been trying to put more effort behind the project, but uptake still relatively small

A tour of Java GUI libraries

Others

- JGoodies: not a GUI toolkit but extension library for Swing
- SwingX: also a Swing extension library
- Apache Pivot: positioned as alternative to JavaFX
- Qt Jambi
- GTK-Java
- Set of widgets on Android (Android-specific GUI library developed by Google)

A tour of Java GUI libraries

- We'll use Swing in this chapter as our GUI library
- Robust, easy to learn, and still very well known and used in Java community

Containers and components

- Many GUI libraries offer a set of “GUI widgets” or “GUI components” or “GUI controls”
- These are the building blocks you can use to construct a user interface
- Elementary, basic GUI entities
  - A button, a text label, a checkbox, and so on
- In Swing: each of these has its own class

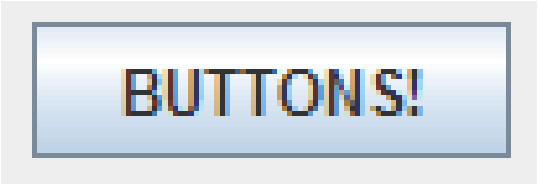
Containers and components

- All Swing components extend `javax.swing.JComponent`
- Class names start with “J” (to differentiate from AWT components)

Containers and components

- Button: `JButton`

```
JComponent component = new JButton("BUTTONS!");
```



Containers and components

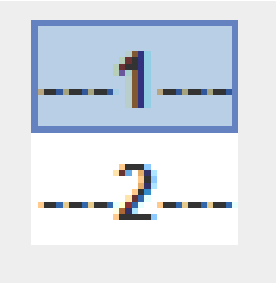
- Text label: `JLabel`

```
JComponent component = new JLabel("A label");
```

Containers and components

- Lists: `JList`

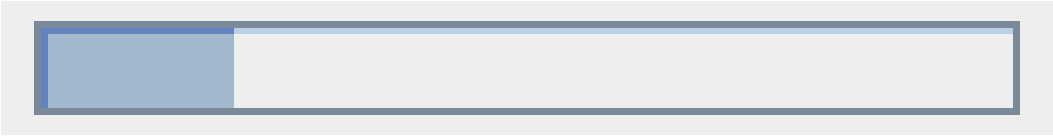
```
JComponent component = new JList<String>(
    new String[]{
      "---1---",
      "---2---"
});
```



Containers and components

- Progress bars: `JProgressBar`

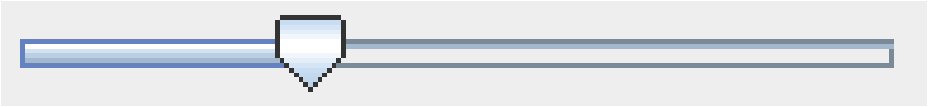
```
JComponent component = new JProgressBar(0, 100);
component.setValue(20);
```



Containers and components

- Sliders: `JSlider`

```
JComponent component = new JSlider(0, 100, 33);
```



Containers and components

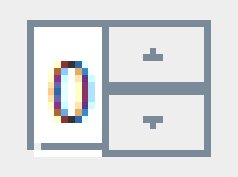
- Text field: `JTextField`

```
JComponent component = new JTextField("Text field");
```

Containers and components

- Spinner: `JSpinner`

```
JComponent component = new JSpinner();
```



Containers and components

- Text area: `JTextArea`

```
JComponent component = new JTextArea("Text area");
```

Containers and components

- Combo box: JComboBox

```
JComponent component = new JComboBox<String>(
    new String[]{
      "---1---",
      "---2---"
});
```

Containers and components

- Check box: `JCheckBox`

```
JComponent component = new JCheckBox("Check boxes");
```

Containers and components

- Radio button: `JRadioButton`

```
JComponent component = new JRadioButton(
    "And radio buttons");
```

Containers and components

- Apart from GUI components, there is also a second GUI element: “containers”
- Containers hold components together in a specific layout
- Can also contain sub-containers
- As such, a container is a special kind of component, with the purpose of grouping and organizing other components
- In Swing: `JApplet`, `JFrame`, `JDialog`, `JWindow` and `JPanel`

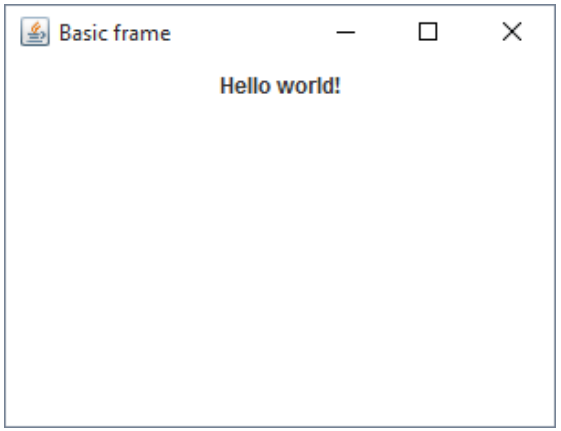
Containers and components

- Swing containers: `JApplet`, `JFrame`, `JDialog`, `JWindow` and `JPanel`
- `JApplet`, `JFrame`, `JDialog`, `JWindow` are drawn by the host operating system – i.e. these are the only "heavyweight" components in Swing

Containers and components

- Basic example

```
import java.awt.Color;
import java.awt.Dimension;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class BasicGUI {

 public static void main(String[] args) {
  JFrame frame = new JFrame();
  frame.setDefaultCloseOperation(
    JFrame.EXIT_ON_CLOSE);
  frame.setTitle("Basic frame");
  JPanel panel = new JPanel();
  panel.setBackground(Color.white);
  panel.setPreferredSize(
    new Dimension(300, 200));

  JLabel label = new JLabel("Hello world!");
  panel.add(label);

  frame.getContentPane().add(panel);

  frame.pack();
  frame.setVisible(true);
 }

}
```



Containers and components

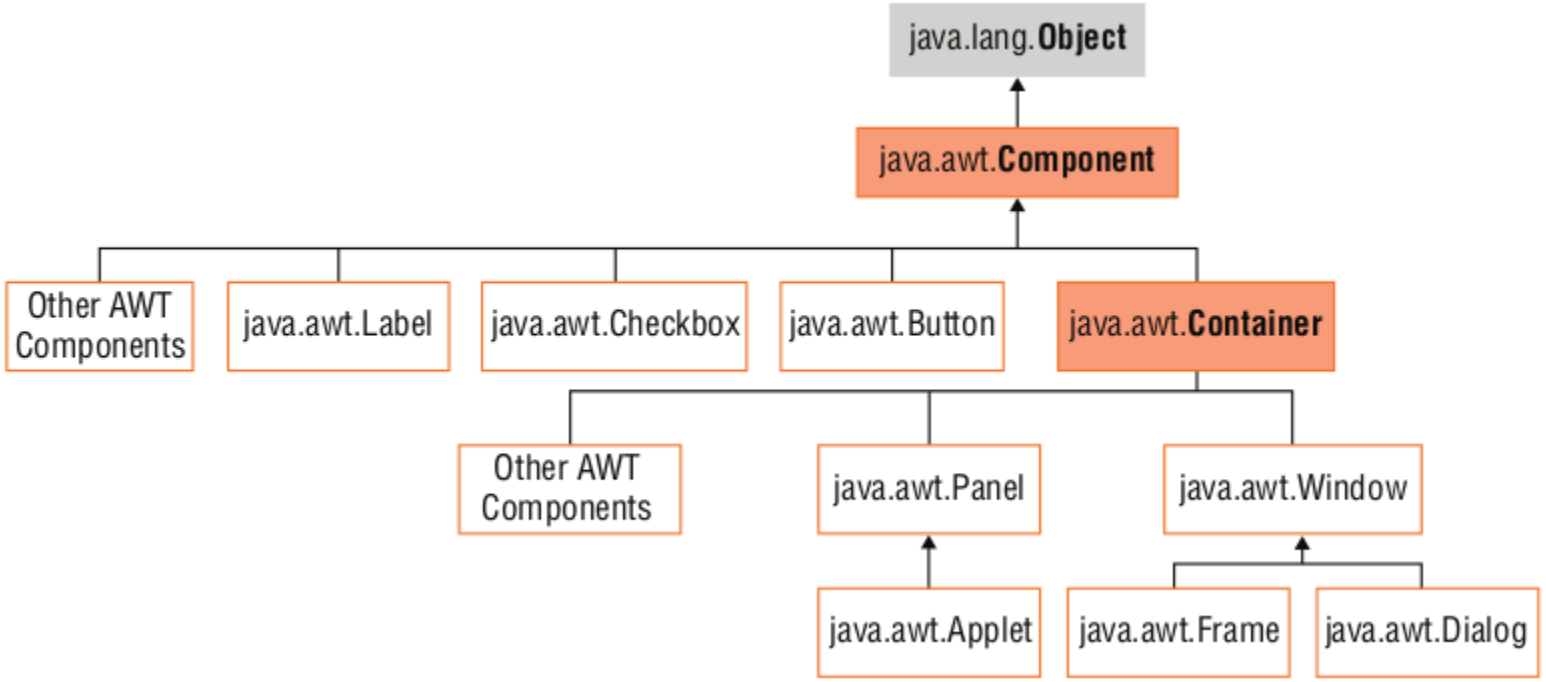
- Adding a component to a container:
  - `container.add(component);`
- Removing a component from a container:
  - `container.remove(component);`
- Removing all components from a container:
  - `container.removeAll();`

Swing: the full picture

- Swing classes extend (older) AWT classes
- As such, GUI class hierarchy in Java is somewhat confusing

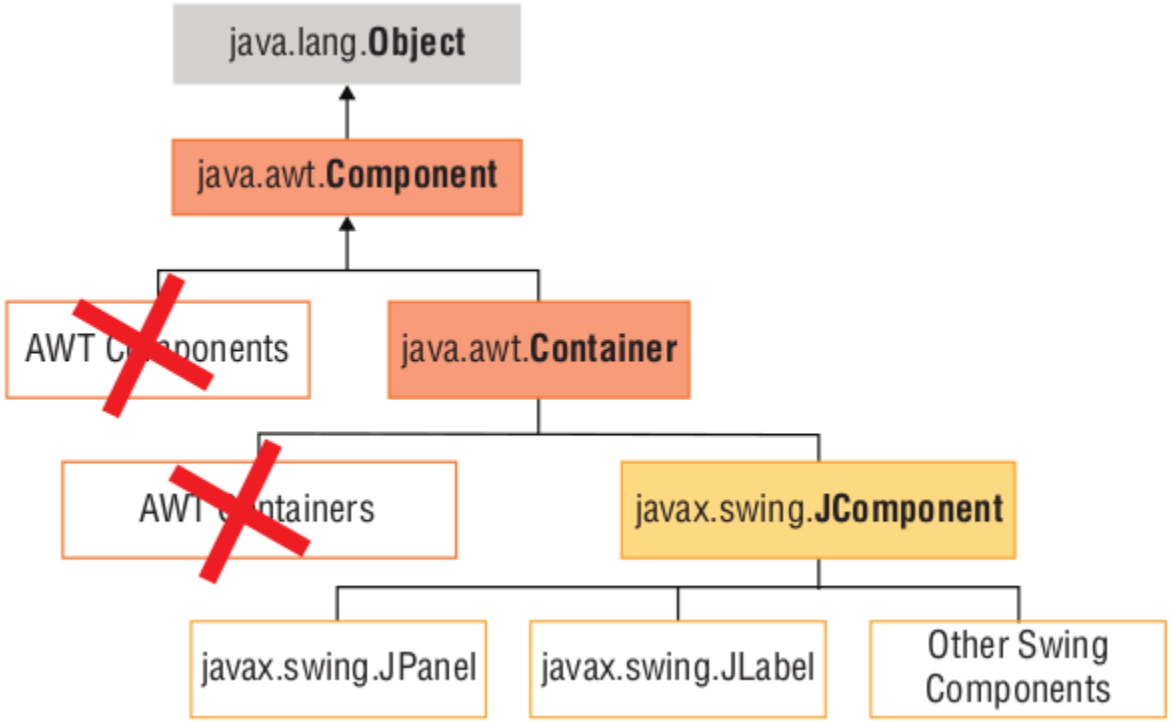
Swing: the full picture

- Let's start with AWT classes:



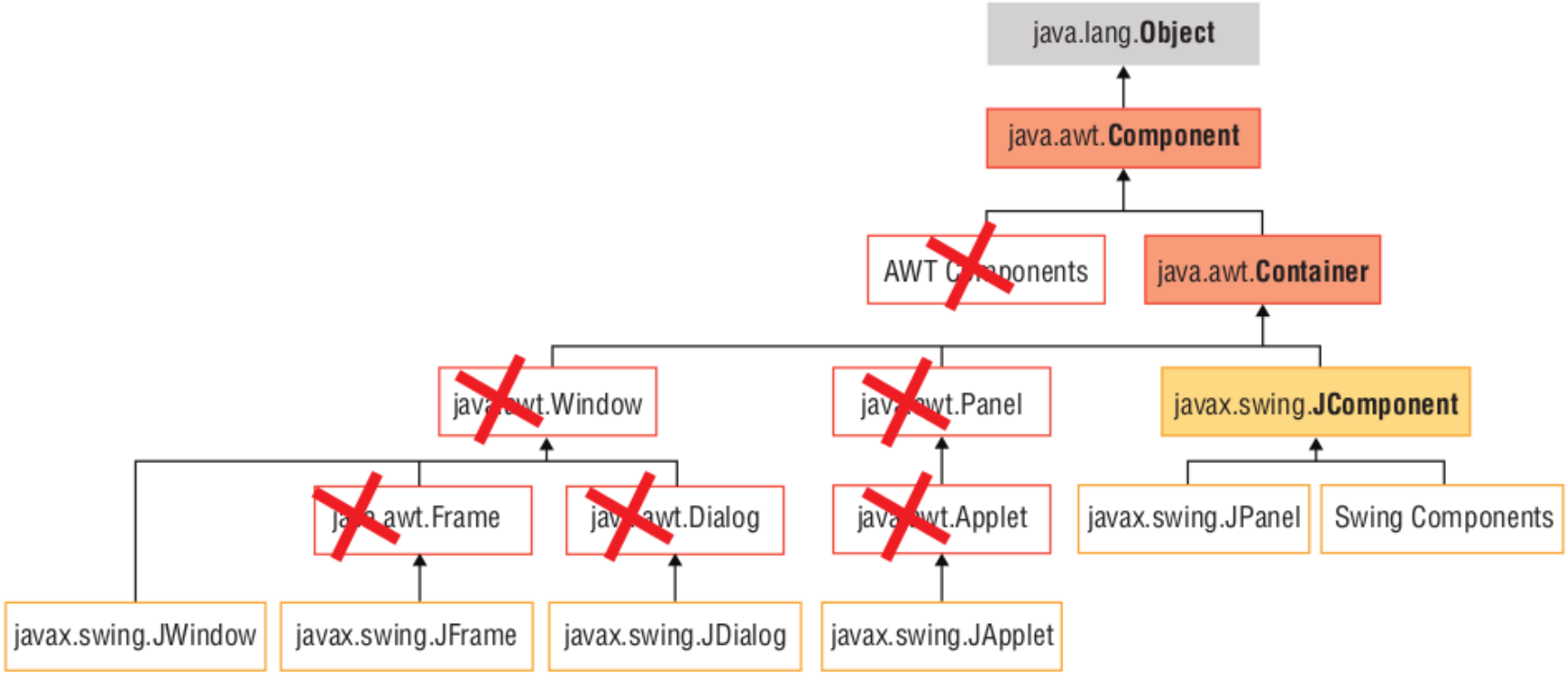
Swing: the full picture

- Swing's `JComponent` extends AWT's Container, and all Swing components extend `JComponent`, so you can ignore AWT's components and containers:



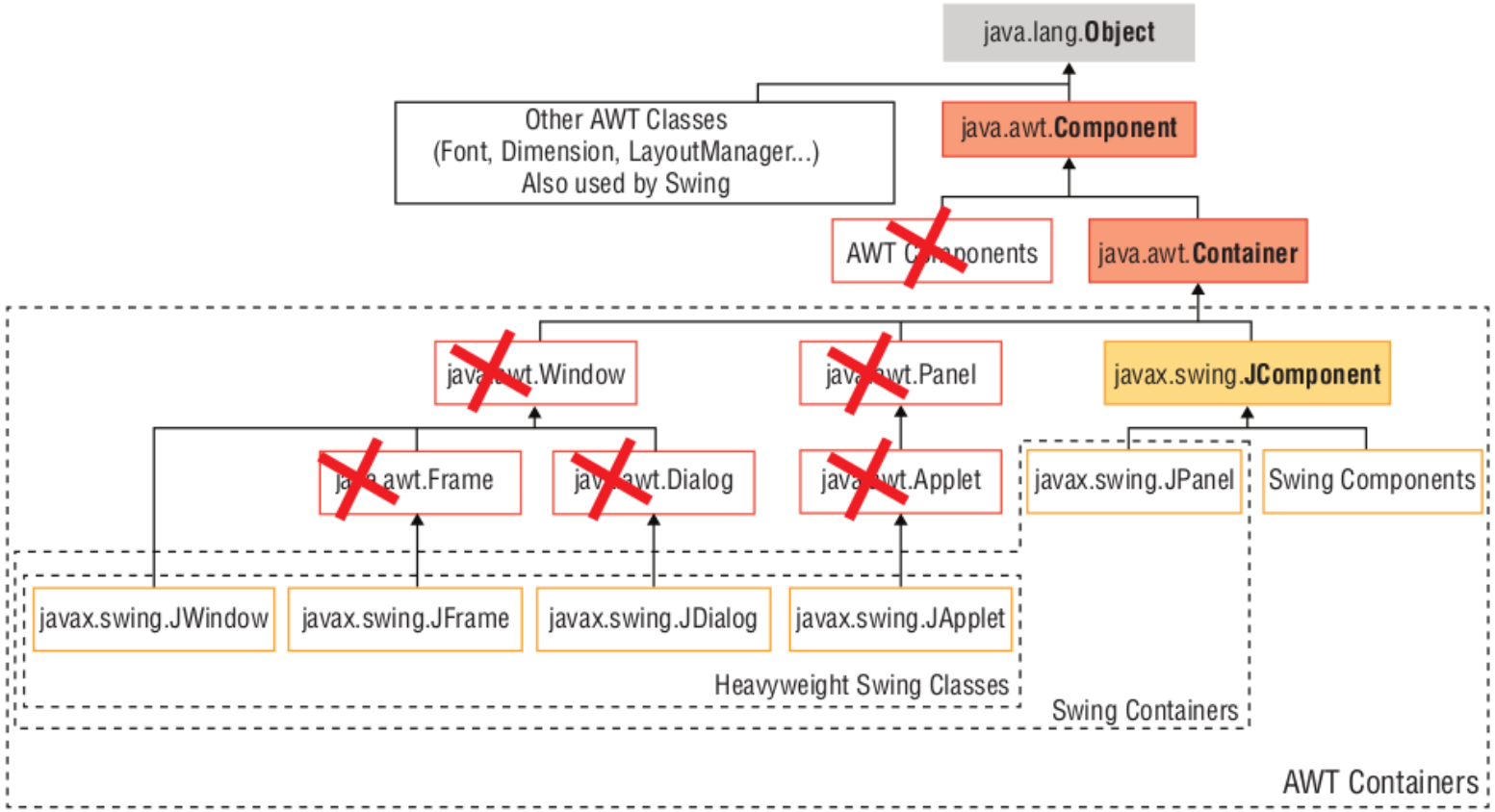
Swing: the full picture

- Except for Swing's heavyweight containers (`JWindow`, `JFrame`, `JDialog` and `JApplet`), they directly extend their AWT counterpart:



Swing: the full picture

- AWT also has some other classes which remain used by Swing as well, giving us the full picture:



Swing: the full picture

- In short: don’t use AWT classes if there is a Swing counterpart available!

Layout managers

- One group of AWT's classes that is reused in Swing are the Layout Managers
- A Layout Manager is an object specifying how components in a container should be laid out
- Java offers a number of them out of the box, but creating custom ones also possible
- By nesting multiple containers with different layout managers, building complex interfaces is possible

Layout managers

- Specifying a layout manager: call the following method on a container object:

```
public void setLayout(LayoutManager manager)
```

- E.g.:

```
JPanel redPanel = new JPanel();
redPanel.setLayout(new BorderLayout());
```

Layout managers

- FlowLayout:

```
panel.setLayout(new FlowLayout());
```

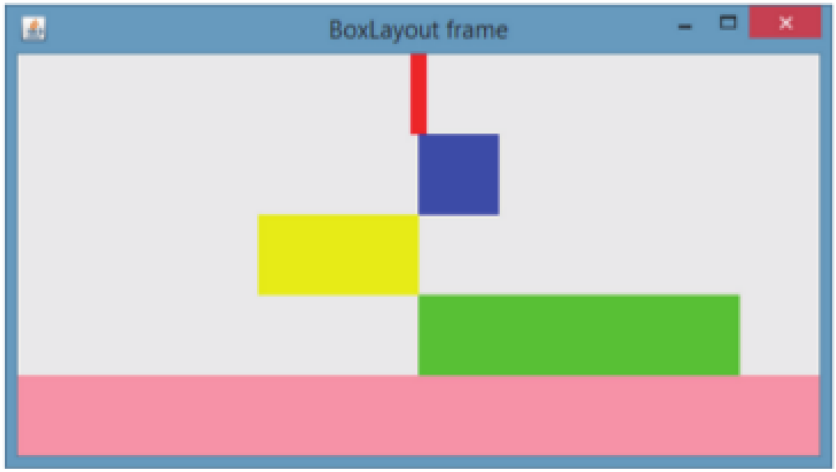
- Components organized from left-to-right, top-to-bottom
- This is the default layout for `JPanel`

Layout managers

- BoxLayout:

```
panel.setLayout(new BoxLayout(panel, BoxLayout.PAGE_AXIS));
```

- BoxLayout stacks components on top of each other, or places them in a row
- Also takes the alignment of components into account

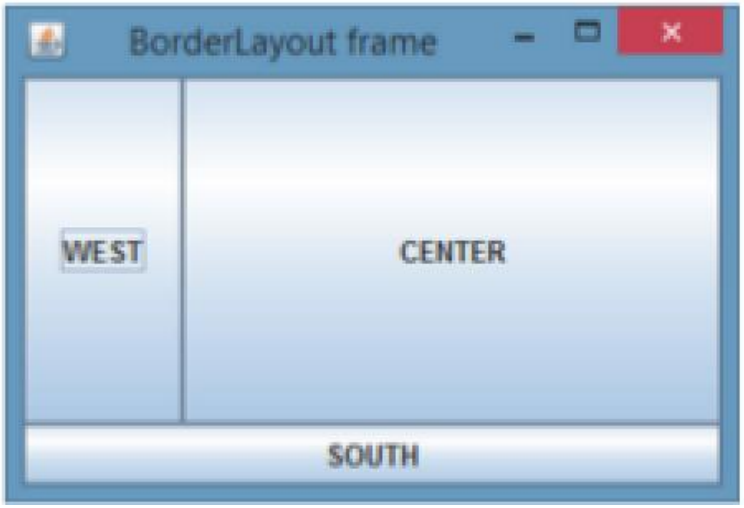


Layout managers

- BorderLayout:

```
panel.setLayout(new BorderLayout());
panel.add(new JButton("NORTH"), BorderLayout.NORTH);
```

- Container is organized into five zones: North, South, West, East and Center (not all need to be filled)
- This is the default layout for windowed containers such as `JFrame`

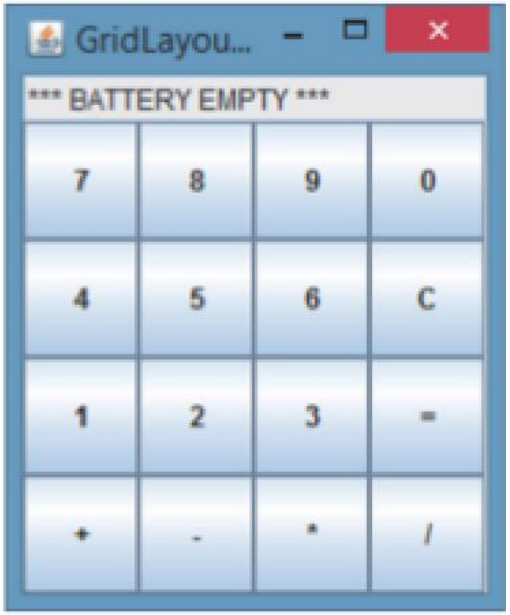


Layout managers

- GridLayout:

```
panel.setLayout(new GridLayout(rows, cols));
```

- Components organized in a grid of cells

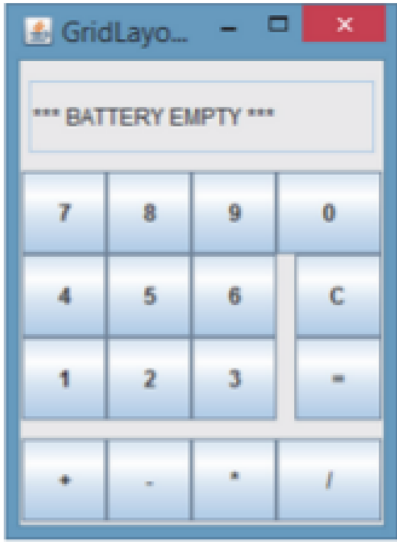


Layout managers

- GridBagLayout:

```
panel.setLayout(new GridbagLayout());
```

- Like GridLayout, but allows to set additional constraints (different widths and heights for the columns and rows, different spacing, spanning components across cells)



Layout managers

- Absolute positioning (no layout manager):

```
panel.setLayout(null);
```

- Leaves the positioning of components entirely up to you

Layout managers

- Others:
  - CardLayout (organize multiple components that share the same display space in tabs)
  - GroupLayout (used for graphical UI designers, complex to construct manually)
  - SpringLayout (even more complex, very complex to construct manually)

Layout managers

- General tips:
  - Sizing methods for components (`setSize` and `setBounds`) only work when a layout manager is not controlling these components
  - Better to resort to `setMinimumSize`, `setPreferredSize` and `setMaximumSize` to give hints to layout manager
  - Good idea to draw nested containers out on paper before making your UI classes
  - Sometimes components which are added at runtime will not appear right away, use `revalidate` and `repaint` methods to force Java to redraw them

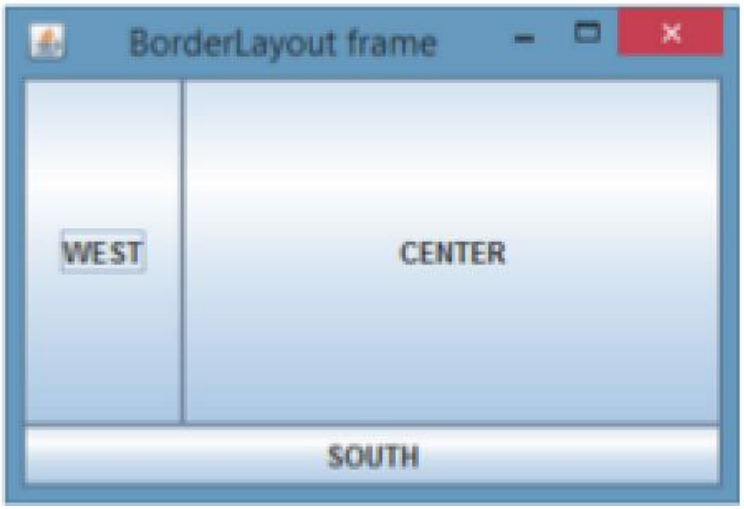
Layout managers: FlowLayout example

```
import java.awt.Color;
import java.awt.Dimension;
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class FlowLayoutFrame {

 public static void main(String[] args) {
  JFrame frame = new JFrame();
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  frame.setTitle("FlowLayout frame");
  frame.getContentPane().setLayout(new FlowLayout());
  frame.getContentPane().add(makePanel(Color.red));
  frame.getContentPane().add(makePanel(Color.orange));
  frame.getContentPane().add(makePanel(Color.green));
  frame.getContentPane().add(makePanel(Color.blue));
  frame.getContentPane().add(makePanel(
    new Color(75, 0, 130)));
  frame.getContentPane().add(makePanel(
    new Color(138, 43, 226)));
  frame.pack();
  frame.setVisible(true);
}
```

```
 private static JPanel makePanel(Color color) {
  JPanel panel = new JPanel();
  panel.setBackground(color);
  panel.setPreferredSize(new Dimension(100, 100));
  return panel;
 }

}
```

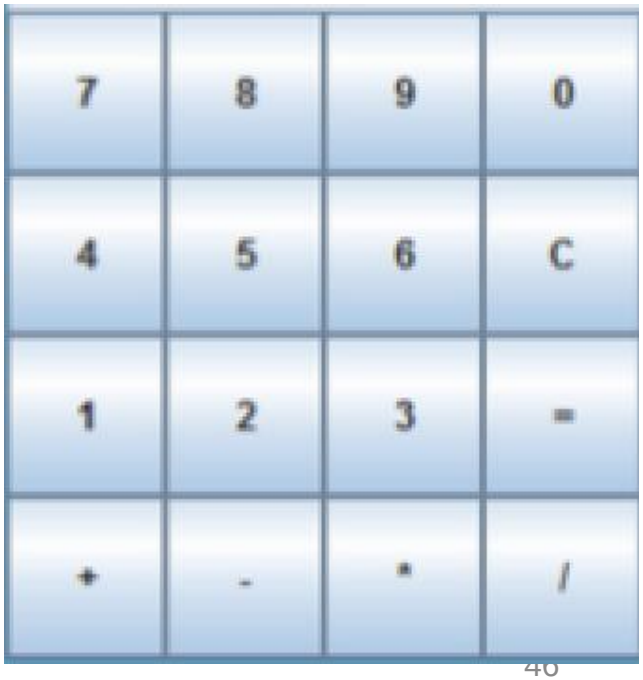
Layout managers: BorderLayout example

```
import java.awt.BorderLayout;
import javax.swing.JButton;
import javax.swing.JFrame;

public class BorderLayoutFrame {
 public static void main(String[] args) {
  JFrame frame = new JFrame();
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  frame.setTitle("BorderLayout frame");
  /* Since the default layout manager for bordered
containers is already a BorderLayout, the following line
is OPTIONAL here. */
  frame.getContentPane().setLayout(new BorderLayout());

  frame.getContentPane().add(new JButton("NORTH"),
    BorderLayout.NORTH);
  // ... or BorderLayout.PAGE_START

  frame.getContentPane().add(new JButton("WEST"),
    BorderLayout.WEST);
  // ... or BorderLayout.LINE_START

  frame.getContentPane().add(new JButton("EAST"),
    BorderLayout.EAST);
  // ... or BorderLayout.LINE_END

  frame.getContentPane().add(new JButton("SOUTH"),
    BorderLayout.SOUTH);
  // ... or BorderLayout.PAGE_END

  frame.getContentPane().add(new JButton("CENTER"),
    BorderLayout.CENTER);

  frame.pack();
  frame.setVisible(true);
 }
}
```



Layout managers: GridLayout example

```
import java.awt.Container;
import java.awt.GridLayout;
import javax.swing.JButton;
import javax.swing.JFrame;

public class GridLayoutFrame {
 public static void main(String[] args) {
  JFrame frame = new JFrame();
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  frame.setTitle("GridLayout frame");
  frame.getContentPane().setLayout(new GridLayout(4, 4));
  addButtons(frame.getContentPane(),
    "7", "8", "9", "0", "4", "5", "6", "C",
    "1", "2", "3", "=", "+", "-", "*", "/"
  );
  frame.pack();
  frame.setVisible(true);
 }

 private static void addButtons(Container contentPane,
   String ... strings) {
  for (String label : strings) {
   contentPane.add(new JButton(label));
  }
 }
}
```



Layout managers: BoxLayout example

```
import java.awt.Color;
import java.awt.Component;
import java.awt.Dimension;

import javax.swing.BorderFactory;
import javax.swing.BoxLayout;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class BoxLayoutFrame {
 public static void main(String[] args) {
  JFrame frame = new JFrame();
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  frame.setTitle("BoxLayout frame");
  frame.getContentPane().setLayout(
    new BoxLayout(frame.getContentPane(),
      BoxLayout.PAGE_AXIS));
  frame.getContentPane().add(makePanel(Color.red, 10,
    Component.CENTER_ALIGNMENT));
  frame.getContentPane().add(makePanel(Color.blue, 50,
    Component.LEFT_ALIGNMENT));
  frame.getContentPane().add(makePanel(Color.yellow, 100,
    Component.RIGHT_ALIGNMENT));
  frame.getContentPane().add(makePanel(Color.green, 200,
    Component.LEFT_ALIGNMENT));
  frame.getContentPane().add(makePanel(Color.pink, 500,
    Component.CENTER_ALIGNMENT));
  frame.pack();
  frame.setVisible(true);
 }

 private static JPanel makePanel(Color col, int w,
    float a) {
  JPanel panel = new JPanel();
  panel.setBackground(col);
  panel.setAlignmentX(a);
  panel.setPreferredSize(new Dimension(w, 50));
  panel.setMaximumSize(panel.getPreferredSize());
  panel.setMinimumSize(panel.getPreferredSize());
  // The following line shows you how to
  // draw borders around components
  panel.setBorder(
    BorderFactory.createLineBorder(Color.black));
  return panel;
 }
}
```

Understanding events and event listeners

- Currently, all our UI components to not have any kind of functionality associated to them
- How can you make a button actually do something
- With events: a happening of something
  - A user clicks a button
  - A user closes a window
  - A user moves the mouse around in a window
- In Java, an “event loop” is continuously running in the background of UI programs, and passes all these events on to any “event listeners” you have defined in your programs

Understanding events and event listeners

- Adding an event listener:

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.SwingUtilities;

public class ActionListenerExample extends JFrame {
 public ActionListenerExample() {
  super();
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  JButton btn = new JButton("Click Me!");
  btn.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent arg0) {
    JOptionPane.showMessageDialog(null,
      "You clicked me, nice!", "Aw yeah!",
       JOptionPane.PLAIN_MESSAGE);
   }
  });
  this.add(btn);
  pack();
  setVisible(true);
 }

 public static void main(String[] args) {
  SwingUtilities.invokeLater(new Runnable() {
   public void run() {
    new ActionListenerExample();
   }
  });
 }
}
```

Understanding events and event listeners

```
SwingUtilities.invokeLater(new Runnable() {
   public void run() {
    // Execute UI code here
   }
  });
```

- Ensures that all UI-related code is executed in the “event loop”
- “All code which does something with UI must run in the event loop”

Understanding events and event listeners

```
btn.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent arg0) {
    JOptionPane.showMessageDialog(null,
      "You clicked me, nice!", "Aw yeah!",
        JOptionPane.PLAIN_MESSAGE);
   }
  });
```

- Adding a new “action listener”
- Very common event listener (i.e. to catch button clicks)
- First way to define: as an anonymous inner class
  - Easy and simple, but can cause code to become messy and does not allow you to re-use the same listener object over multiple components

Understanding events and event listeners

```
public class ActionListenerExample extends JFrame implements ActionListener {
 public ActionListenerExample() {
  JButton btn = new JButton("Click Me!");
  // The JFrame is now also the listener object
  btn.addActionListener(this);
  this.add(btn);
  pack();
  setVisible(true);
 }
 public void actionPerformed(ActionEvent arg0) {
  // React to click here
 }
}
```

- Second way to define: make the container itself the listener
  - Helpful when you have multiple components to listen to and you already have a custom class for your container, but needs some extra code to detect which button was clicked (as every button now calls the same listener method)

Understanding events and event listeners

```
MySimpleButtonListener listener = new MySimpleButtonListener();
btn.addActionListener(listener);

public class MySimpleButtonListener implements ActionListener {
 public void actionPerformed(ActionEvent arg0) {
  // React to click here
 }
}
```

- Third way to define: define a listener in its own class
  - Easy to re-use listeners across different components, keeps code clean and separated, but requires a little more effort to set up

Understanding events and event listeners

- Many types of listeners exist:
  - Action Listener (ActionListener) - addActionListener
  - Change Listener (ChangeListener) - addChangeListener
  - Window Listener (WindowListener) - addWindowListener
  - Mouse Listener (MouseListener) - addMouseListener
  - Mouse Motion Listener (MouseMotionListener) - addMouseMotionListener
  - Mouse Wheel Listener (MouseWheelListener) - addMouseWheelListener
  - See: https://docs.oracle.com/javase/tutorial/uiswing/events/api.html

Understanding events and event listeners

- Same listener for many buttons: how to detect which one was pressed?

```
public void actionPerformed(ActionEvent ev) {
 String message;
 if (((JButton)ev.getSource()).getText().equals("Click Me!"))
  message = "First button was clicked";
 else
  message = "Second button was clicked";
 JOptionPane.showMessageDialog(null, message, "Aw yeah!", JOptionPane.PLAIN_MESSAGE);
}
```

- Unsafe! Are you sure the incoming source is always a button? What if the text changes later on?

Understanding events and event listeners

- Same listener for many buttons: how to detect which one was pressed?

```
public void actionPerformed(ActionEvent ev) {
 String message = "";
 if (ev.getSource() == btn1)
  message = "First button was clicked";
 else if (ev.getSource() == btn2)
  message = "Second button was clicked";
 JOptionPane.showMessageDialog(null, message, "Aw yeah!", JOptionPane.PLAIN_MESSAGE);
}
```

- Better: perform a direct object comparison, make the buttons members of the containing (container) class

Understanding events and event listeners

- Same listener for many buttons: how to detect which one was pressed?

```
String ACTION_SAVE = "Save";
button.setActionCommand(ACTION_SAVE);
public void actionPerformed(ActionEvent ev) {
 String action = ev.getActionCommand();
 if (action.equals(ACTION_SAVE)
  //…
}
```

- Better: using `getActionCommand` and `setActionCommand`

Example: Disco Lights

```
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.SwingUtilities;

public class DiscoLightsExample extends JFrame implements ActionListener {

 private final String ACTION_ON = "LIGHT ON";
 private final String ACTION_OFF = "LIGHT OFF";
 private final String ACTION_CYCLE = "CYCLE COLOR";
 private final Color[] COLORS = new Color[] {
   Color.white, Color.green, Color.red, Color.yellow, Color.orange, Color.pink };
 private int currentColor = 0;
 private boolean isLightOn = false;
```

Example: Disco Lights

```
public DiscoLightsExample() {
 setTitle("Disco Light Party Frame");
 setLayout(new FlowLayout());
 setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 JButton btnOffOn = new JButton("Lights On");
 JButton btnColor = new JButton("Cycle Color");
 btnOffOn.setActionCommand(ACTION_ON);
 btnColor.setActionCommand(ACTION_CYCLE);
 btnOffOn.addActionListener(this);
 btnColor.addActionListener(this);
 getContentPane().add(btnOffOn);
 getContentPane().add(btnColor);
 pack();
 setVisible(true);
}
```

Example: Disco Lights

```
public void actionPerformed(ActionEvent ev) {
 String action = ev.getActionCommand();
 switch (action) {
  case ACTION_ON:
   isLightOn = true;
   getContentPane().setBackground(COLORS[currentColor]);
   ((JButton) ev.getSource()).setText("Lights Off");
   ((JButton) ev.getSource()).setActionCommand(ACTION_OFF);
   break;
  case ACTION_OFF:
   isLightOn = false;
   getContentPane().setBackground(Color.black);
   ((JButton) ev.getSource()).setText("Lights On");
   ((JButton) ev.getSource()).setActionCommand(ACTION_ON);
   break;
  case ACTION_CYCLE:
   if (isLightOn)
    getContentPane().setBackground(COLORS[++currentColor % COLORS.length]);
   break;
  }
 }
```

Example: Disco Lights

```
 public static void main(String[] args) {
  SwingUtilities.invokeLater(new Runnable() {
   public void run() {
    new DiscoLightsExample();
   }
  });
 }
}
```

Closing topics

```
SwingUtilities.invokeLater(new Runnable() {
  public void run() {
    // Execute your UI code here
  }
});
```

- Make sure to run UI code in the “event loop”

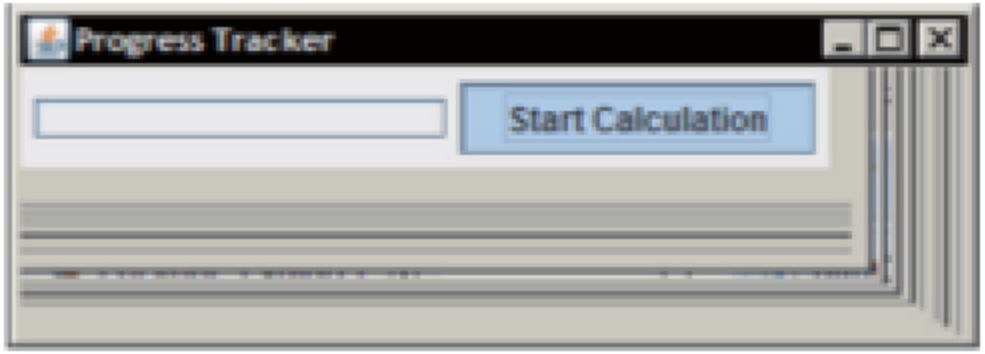
- Let’s say you are executing a heavy task and wish to update the GUI to provide progress…

Closing topics

```
public class ProgressTrackingFrame extends JFrame implements ActionListener {
 private boolean isRunning = false; private final JProgressBar bar = new JProgressBar(0, 100);
 private final JButton btn = new JButton("Start Calculation");

 public ProgressTrackingFrame() {
  setLayout(new FlowLayout());
  btn.addActionListener(this);
  add(bar); add(btn); pack(); setVisible(true);
 }

 public void actionPerformed(ActionEvent arg0) {
  if (isRunning) {
   // How to cancel here?
  } else {
   isRunning = true;
   btn.setText("Stop Calculation");
   long total = 1000000000; for (long i = 0; i < total; i++) {
    int perc = (int) (i * (bar.getMaximum() - bar.getMinimum()) / total);
    bar.setValue(perc);
   }
  }
  isRunning = false;
  btn.setText("Start Calculation");
 }
}
```

Closing topics

- Run this program and notice how it is not updating the progress bar
- UI blocks until program finished



Closing topics

```
public SwingWorker<Boolean, Integer> makeBackgroundTask(final long total) {
 SwingWorker<Boolean, Integer> task = new SwingWorker<Boolean, Integer>(){
  protected Boolean doInBackground() throws Exception {
   btn.setText("Stop Calculation");
   for (long i = 0; i < total; i++) {
    if (isCancelled()) {
     bar.setValue(0);
     return false;
    }
    int perc = (int) (i * (bar.getMaximum() - bar.getMinimum()) / total);
    publish(perc);
   }
   return true;
  }
  protected void process(List<Integer> percs) {
   for (int perc : percs)
    if (bar.getValue() < perc)  bar.setValue(perc);
   }
  public void done() {
   btn.setText("Start Calculation");
   backgroundTask = null;
  }
 };
 return task;
}
```

Closing topics

```
public void actionPerformed(ActionEvent arg0) {
 if (backgroundTask == null) {
  backgroundTask = makeBackgroundTask(1000000000);
  backgroundTask.execute();
 } else {
  backgroundTask.cancel(true);
 }
}
```

Closing topics

```
SwingWorker<TypeOfFinalResult, TypeOfIntermediateResult>
```

- Use this class whenever you want to perform heavy, long-running work in the background whilst updating your UI and continuing to listen to events

Closing topics

- Helpful AWT classes still used in Swing:
  - `java.awt.Color;`
  - `java.awt.Dimension;`
  - `java.awt.Graphics;`
  - `java.awt.Point;`
  - `java.awt.Font;`

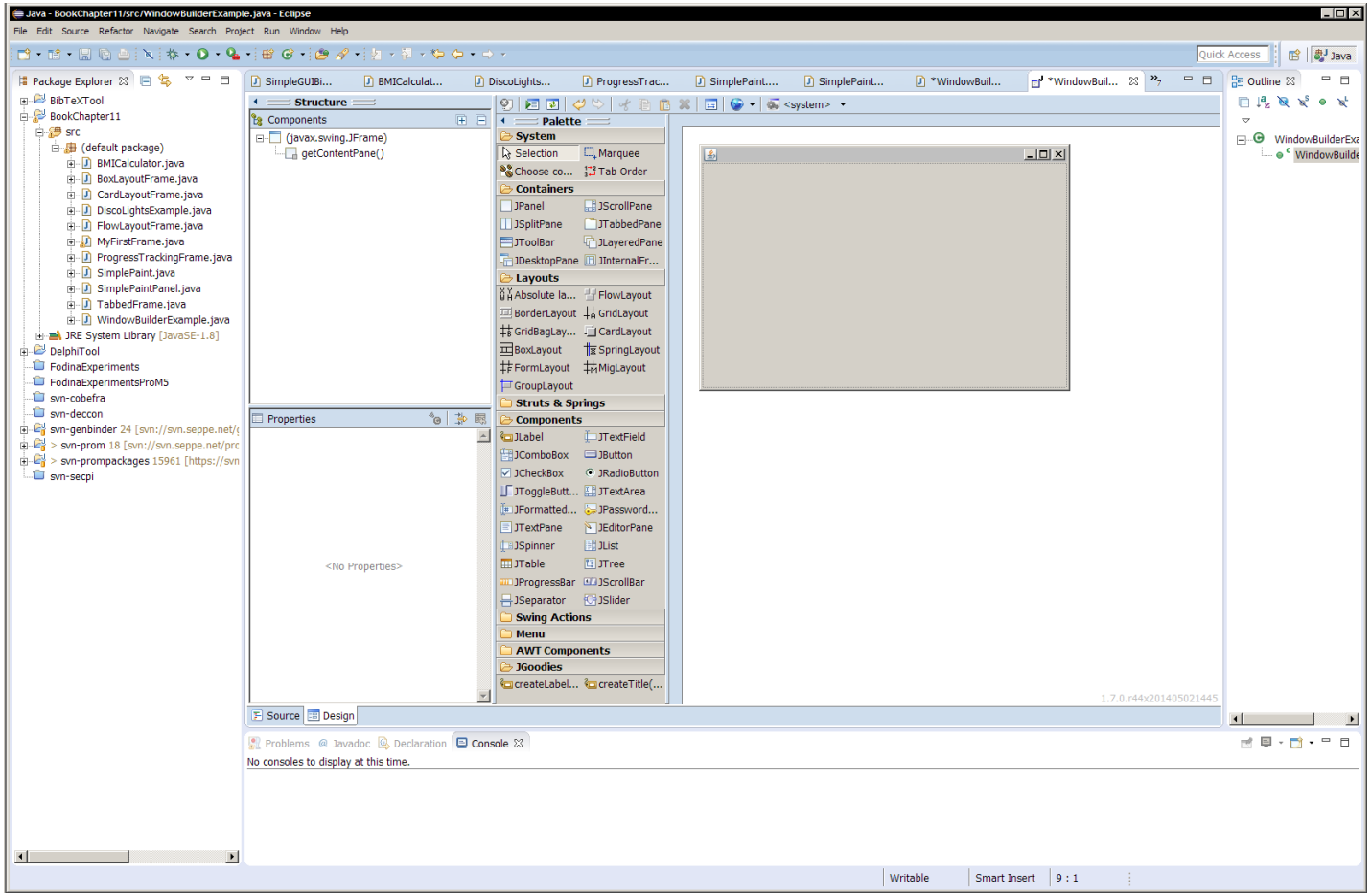
Closing topics

- Defining custom paint behavior by overriding `paint` method of a component:

```
public void paint(java.awt.Graphics g) {
 int w = this.getWidth();
 int h = this.getHeight();
 g.setColor(Color.white);
 g.fillRect(0, 0, w, h);
 g.setColor(Color.black);
 for (Point point : blackPixels)
  g.drawRect(point.x, point.y, 1, 1);
 }
}
```

Closing topics

- Visual UI designers, e.g. WindowBuilder:



Closing topics

- Best practice: keep logic and looks separated
  - Ideally, you should be able to perform all your application logic using pure Java objects and without using any graphical user interface at all
  - Always a good idea to construct your core domain concepts first
  - Then build UI around them
  - Don’t stick everything in a main method
  - Don’t create huge event listeners in an inner class
  - Add another (nested) container when you’re fighting to layout your components correctly

Conclusions

- The basics of GUIs in Java
- A tour of Java GUI libraries
- Containers and components
- Swing: the full picture
- Layout managers
- Understanding events and event listeners
- Closing topics