JavaOne℠

Sun's Worldwide Java Developer Conference

JavaOne℠

Sun's Worldwide Java Developer Conference

Creating Rich GUIs in Java™

Amy Fowler
Staff Engineer
JavaSoft

Tutorial Overview

- Toolkit principles & architecture
- GUI layout management
- Building custom components
- AWT futures

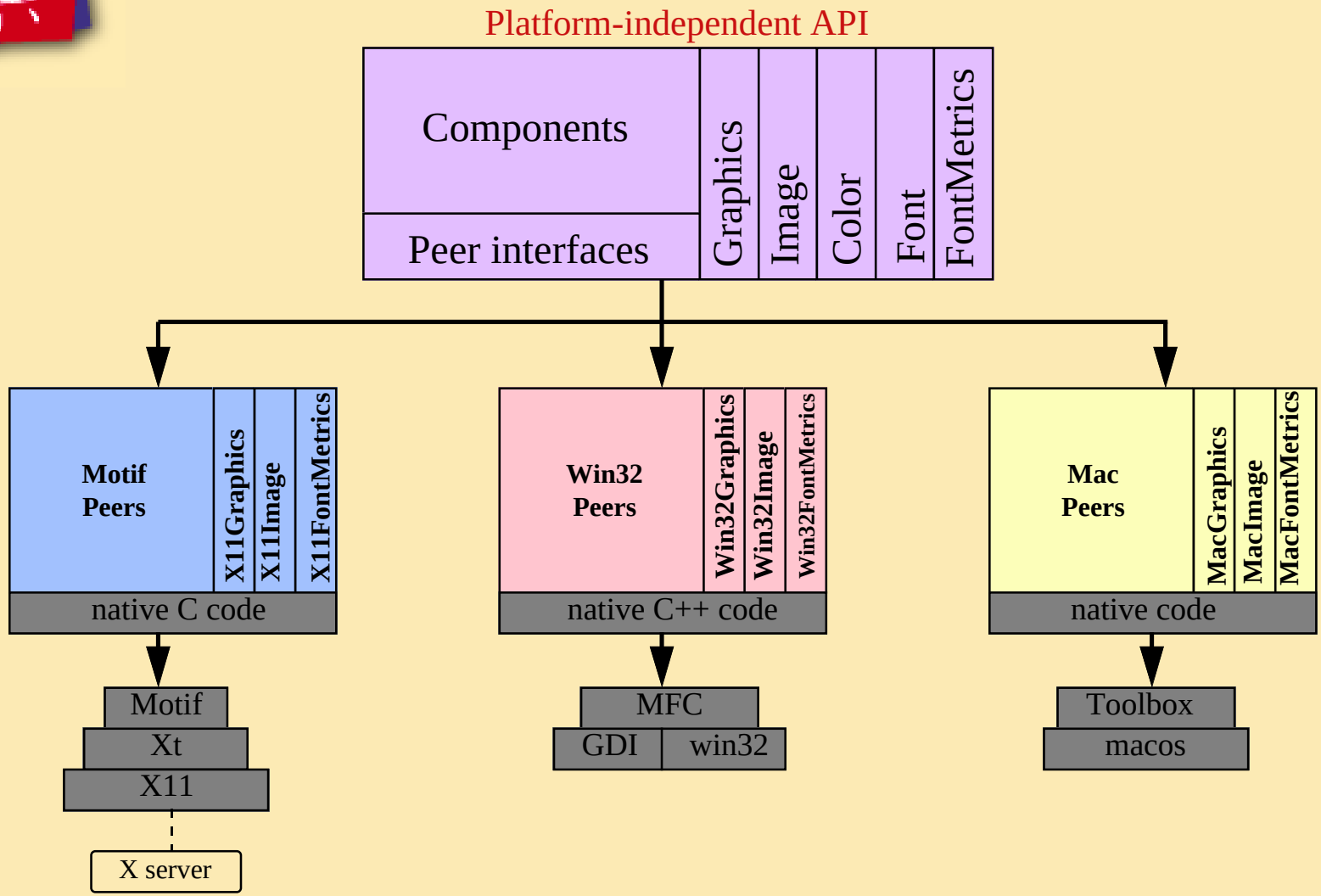
Toolkit Principles & Architecture: Overview

- Toolkit design principles
- AWT architecture
- The peer model

Toolkit Design Principles

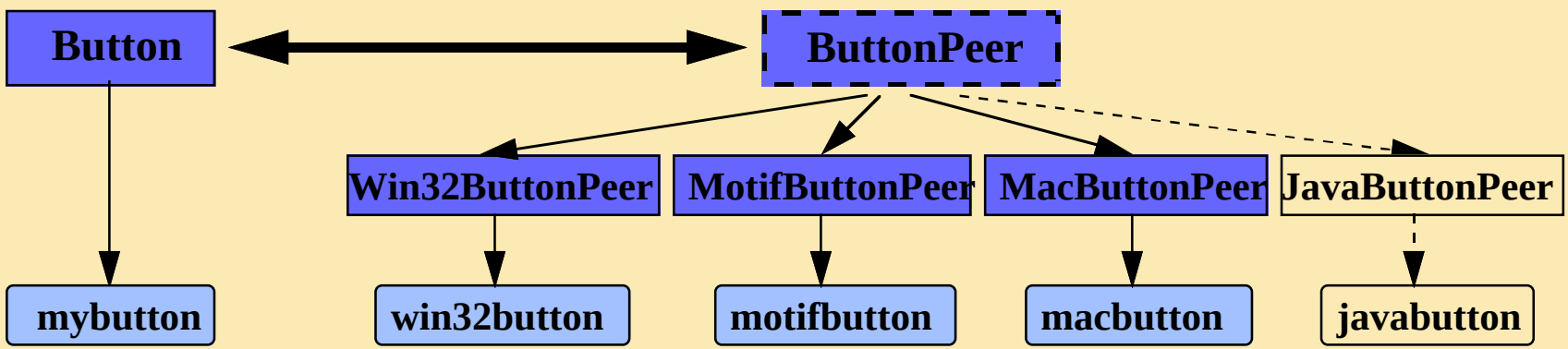
- Platform independent API
- Native look and feel
- Flexibility & extensibility
- Well defined porting interface

AWT Architecture



The Peer Model

- Means for supporting multiple coexisting implementations
- Peer encapsulates native widget in platform-independent class



Why Peers?

- Allows flexibility in implementation
- Pure “factory” approach precludes subclassing
- Enforces API consistency
- Eases source management

What You Need to Know About Peers

- Components maintain state regardless of peer existence
- Limitations when subclassing
- Some operations depend on peers

Operations Depending on Peers

- Getting component's natural size
- Laying-out containers
- Rendering components
  - `getGraphics()`
- Requesting, migrating focus
  - `requestFocus()`
  - `nextFocus()`

Peer Life Cycle

- Peer created when… (`addNotify()`)
  - Component added to container AND container's peer already created
  - Encompassing window packed or shown
- Peer destroyed when… (`removeNotify()`)
  - Component removed from container
  - Component reparented
  - Component destroyed

Peer Life Cycle Example

```
class myFrame extends Frame {
    public myFrame() {
        super("My Frame");
        Button b1 = new Button("boo");
        add(b1);
        pack();
        show();
    }
    public boolean action(Event e, Object arg) {
        if (arg.equals("boo")) {
            Button b2 = new Button("hoo");
            add(b2);
            return true;
        }
        return false;
    }
```

creates peers for both frame and button

creates peer for button

GUI Layout Management: Overview

- Nature of dynamic layout
- Importance of dynamic layout for Java
- AWT Layout Model
- Insets
- The GridBag de-mystified

Nature of Dynamic Layout

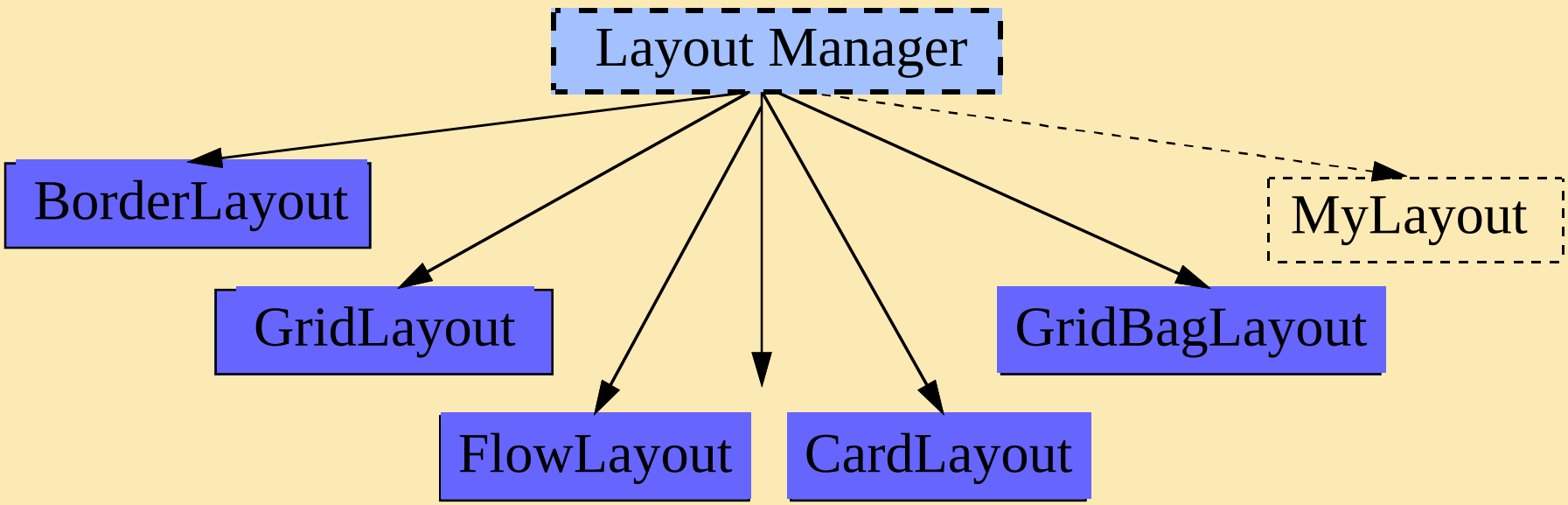
- Component positioning not fixed point locations
- Component sizing determined at runtime
- GUI layout responds well to dynamic changes
  - Internal: component's geometry changes
  - External: user resizes window

Importance of Dynamic Layout for Java™

- Cross-platform differences
  - Native components
  - Font metrics
- Future, unanticipated Java™-based platforms
- Localization

AWT Layout Model

- Container layout delegated to LayoutManager



Layout Validation Model

- Components marked “invalid” when:
  - State change affects geometry
  - Container has child added or removed
- Validation automatic when window packed or shown
- Visible components marked invalid are *not* automatically validated

Validation Example

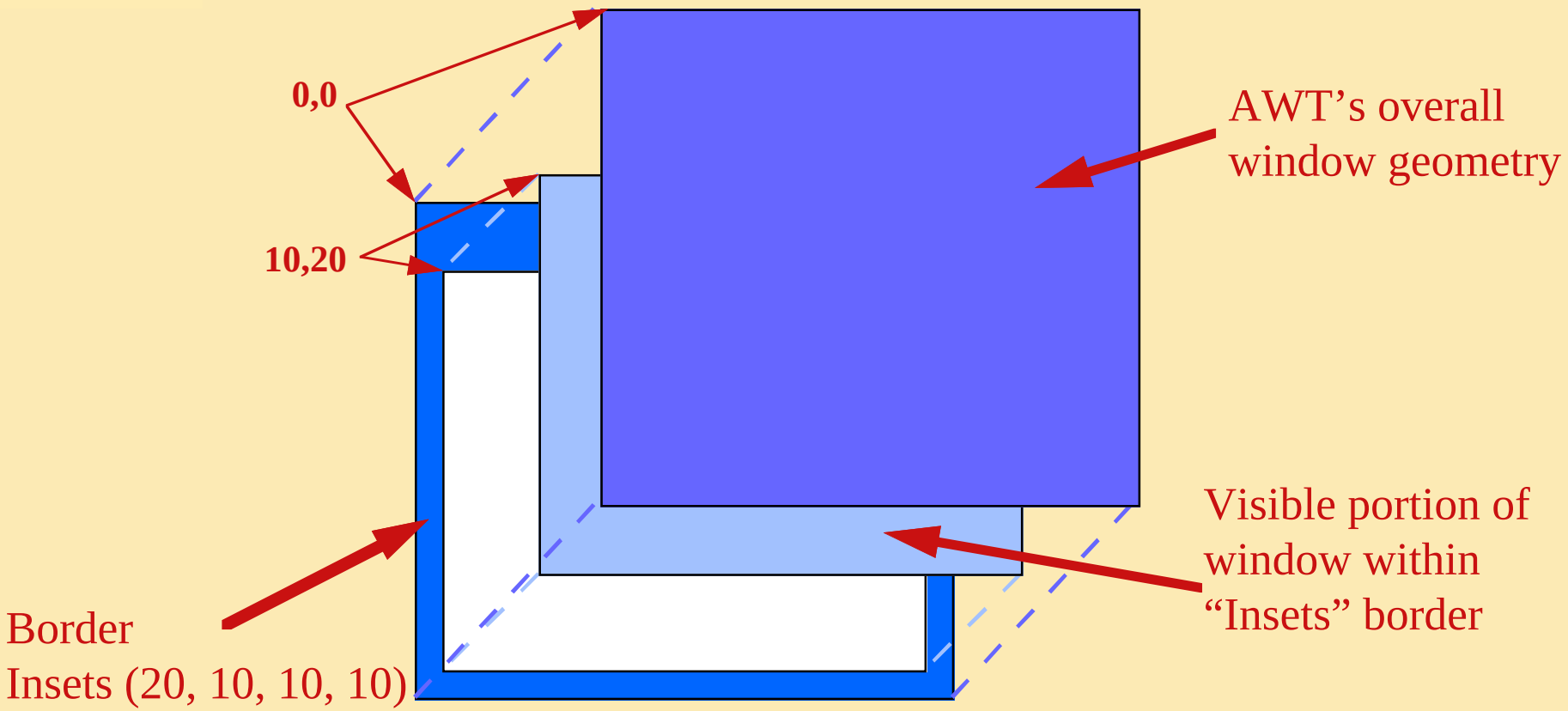
```
public boolean action(Event e, Object arg) {
    if (arg.equals("Change Font")) {
        button1.setFont(newfont);
        button2.setFont(newfont);
        label.setFont(newfont);
        field.setFont(newfont);

        validate();
        return true;
    }
    ...
```

Batches all layout calculations to a single pass

Insets

- Attribute which defines border geometry
  - (top, left, bottom, right)
- Defined for containers
  - Windows: defines window decoration geometry
  - Panels: can be used to render borders

Insets for Windows



The GridBag De-Mystified!

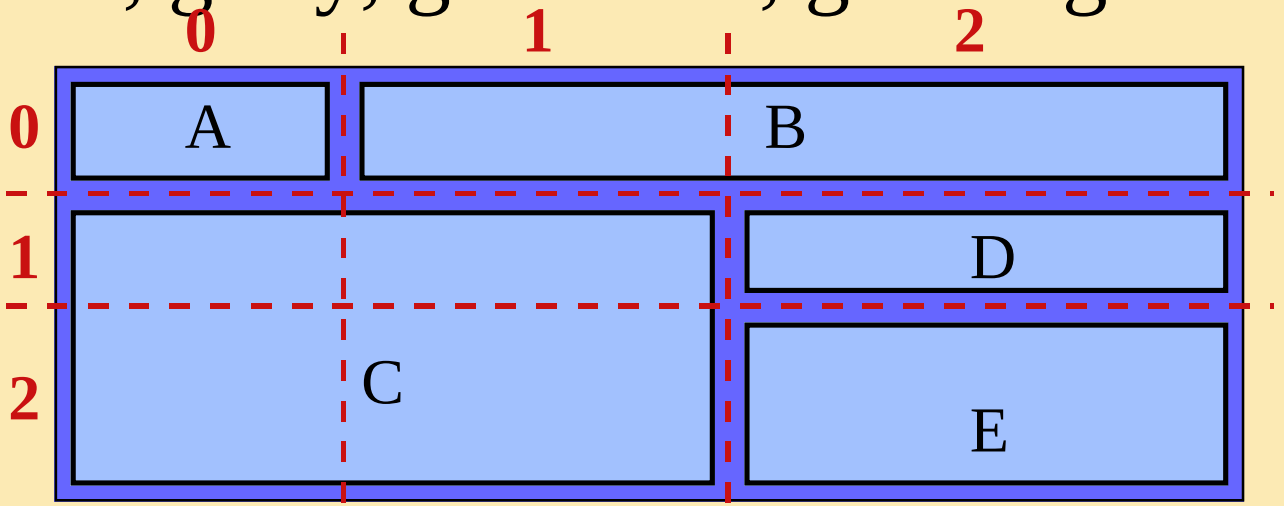
- Extremely flexible and powerful
- Supports constraint-based model
- API is more complex
  - Learn *model* before API

GridBag Model

- Provides dynamic grid of cells
- Component constraints define dynamic positioning and resizing within grid
  - Upper-Left position
  - Number of cells to span (=display area)
  - Gravity/fill within display area
  - Percentage of resize absorption
  - Internal/external padding within display area

Constraints: Display Area

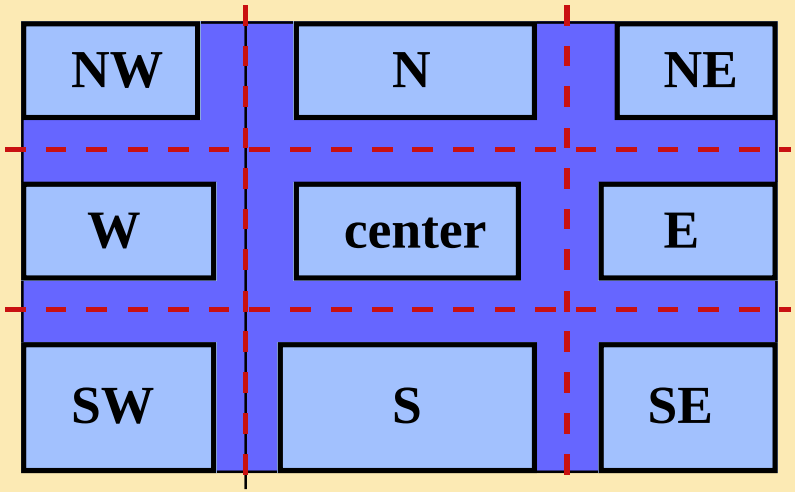
- gridx, gridy, gridwidth, gridheight



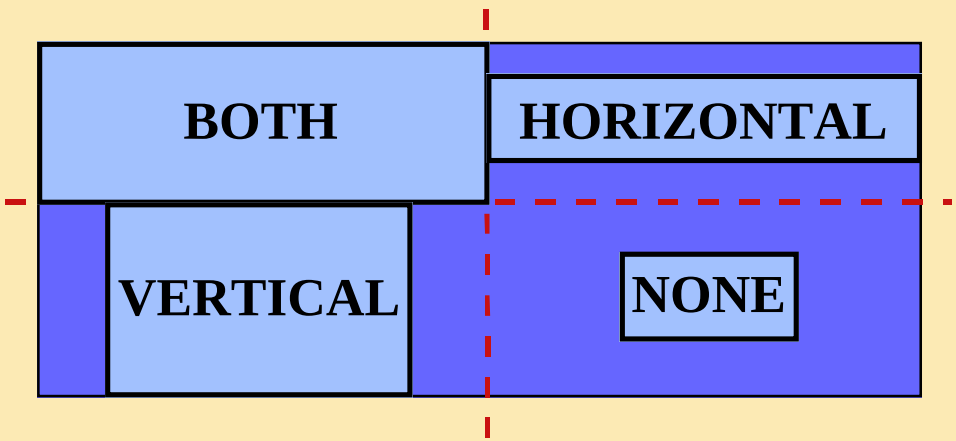
	gridx	gridy	gridwidth	gridheight
A	**0**	**0**	**1**	**1**
B	**1**	**0**	**2**	**1**
C	**0**	**1**	**2**	**2**
D	**2**	**1**	**1**	**1**
E	**2**	**2**	**1**	**1**

Constraints: Gravity/Fill

- anchor

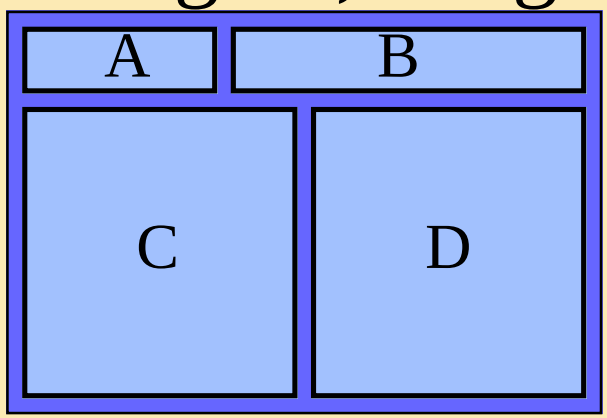


- fill

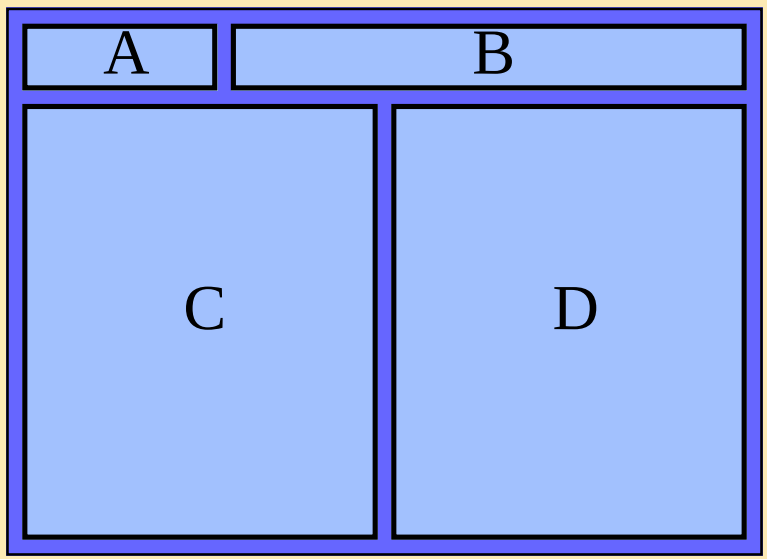


Constraints: Resize Absorption

- weightx, weighty



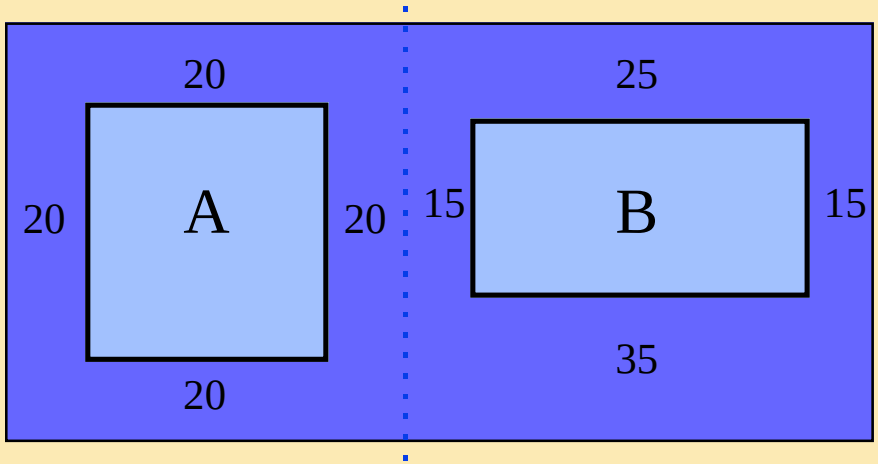
after resize...



	weightx	weighty	xΔ	yΔ
A	**0.0**	**0.0**	**0%**	**0%**
B	**1.0**	**0.0**	**100%**	**0%**
C	**1.0**	**1.0**	**50%**	**100%**
D	**1.0**	**1.0**	**50%**	**100%**

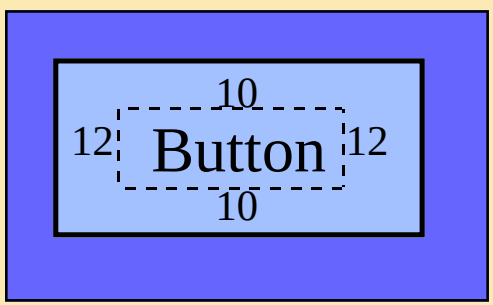
Constraints: Internal/External Padding

- insets



Insets	
A	(20, 20, 20, 20)
B	(25, 15, 35, 15)

- ipadx, ipady



ipadx	ipady
12	10

GridBag Tips

- Use paper/pencil first ➡ draw grid!
- Use gridx, gridy, gridwidth, gridheight
  - Ignore GridBagConstraints.RELATIVE
- Create convenience method to make constraint-setting easier

GridBag Example

```
GridBagLayout gridbag = new GridBagLayout();

void addComponent(Component comp,int gridx,int gridy,
                  int gridw,int gridh){
    GridBagConstraints c = new GridBagConstraints();
    c.gridx = gridx;
    c.gridy = gridy;
    c.gridwidth = gridw;
    c.gridheight = gridh;
    gridbag.setConstraints(comp, c);
    add(comp);
}
```

Constraint-setting convenience method

GridBag Example (cont.)

```
public GBPanel extends Panel() {
    setLayout(gridbag);

    addComponent(new Label(“name:”), 0, 0, 1, 1);
    addComponent(new TextField(12), 1, 0, 2, 1);
    addComponent(new TextArea(32,10), 0, 1, 2, 2);
    addComponent(new Checkbox(“Yes?”), 2, 1, 1, 1);
    addComponent(new List(), 2, 2, 1, 1);
}
```

Building Custom Components: Overview

- The Formula
- Simple example
- Issues to ponder

The Formula

- Subclass from Canvas or Panel
- For Look, override:
  - `paint(), update()`
  - `minimumSize(), preferredSize()`
- For Feel, override:
  - `handleEvent()`

Design For Re-use

- Doc comments (Javadoc!)
- Implement get/set methods
- Exception handling up front
- Design for extensibility
  - Field access

Simple Example: Separator

```
public class Separator extends Canvas {
    public final static int HORIZONTAL = 0;
    public final static int VERTICAL = 1;

    int orientation;
    Dimension sepSize;

    public Separator(int len,int thickness,int orient)
        orientation = orient;
        if (orient == HORIZONTAL) {
            sepSize = new Dimension(len, thickness);
        } else { // VERTICAL
            sepSize = new Dimension(thickness, len);
        }
    }
```

Example: Get/Set Methods

```
public int getOrientation() {
    return orientation;
}
public void setOrientation(int orient) {
    if (orient > VERTICAL || orient < HORIZONTAL) {
        throw new IllegalArgumentException(
              “illegal orientation”);
    }
    if (orientation != orient) {
        orientation = orient;
        sepDim = new Dimension(
              sepDim.height, sepDim.width);
        invalidate();
        // no validate or repaint here!
    }
}
```

Example: Sizing

```
public Dimension preferredSize() {
    return sepDim;
}

public Dimension minimumSize() {
    return sepDim;
}
```

Example: Painting

```
public void paint(Graphics g) {
    int x1, y1, x2, y2;
    Rectangle bbox = bounds();
    Color c = getBackground();
    Color brighter = c.brighter();
    Color darker = c.darker();

    if (orientation == HORIZONTAL) {
        x1 = 0;
        x2 = bbox.width - 1;
        y1 = y2 = bbox.height/2 - 1;
    } else { // VERTICAL
        x1 = x2 = bbox.width/2 - 1;
        y1 = 0;
        y2 = bbox.height - 1;
    }
```

Query for state dynamically

Example: Painting (cont.)

```
        // draw the separator

        g.setColor(darker);
        g.drawLine(x1, y1, x2, y2);

        g.setColor(brighter);
        if (orientation == HORIZONTAL) {
            g.drawLine(x1, y1+1, x2, y2+1);
        } else { // VERTICAL
            g.drawLine(x1+1, y1, x2+1, y2);
        }
    }

} // END Separator
```

Issues with Custom Components

- Look & feel trade-off
  - Portable vs. truly native
- Difficult to override look of native components
- Reinventing the wheel

AWT Futures: Overview

- Quality/performance
- Filling in the gaps
- Building infrastructure
- Enabling visual tools

Futures: Quality & Performance

- Win32 re-write
- Repaint/layout algorithm improvements
- Hooks for performance-monitoring and testing
- More bug fixing

Futures: Filling in the Gaps

- More components
  - Pop-up menu, image button, …
- Rich 2D rendering model
- Bunch of little stuff
  - Menu accelerators
  - Cursors per component
  - Desktop color model
  - …

Futures: Building Infrastructure

- Internationalization
- Printing
- Data Transfer
  - Clipboard
  - Drag & Drop
- Light-weight component framework

Futures: Enabling Visual Tools

- Delegation-based event model
- Normalization of get/set methods
- Reflection
- Persistence

Tutorial Summary

- Understand basic AWT principles
  - Peers aren't always your enemy
- Learn, exploit, extend AWT's powerful layout mechanisms
- Be creative – AWT makes it easy to build new things!
- AWT evolving to enable richer graphical applications

Questions ?