STANDARD WIDGET TOOLKIT

Óscar Darias Plasencia

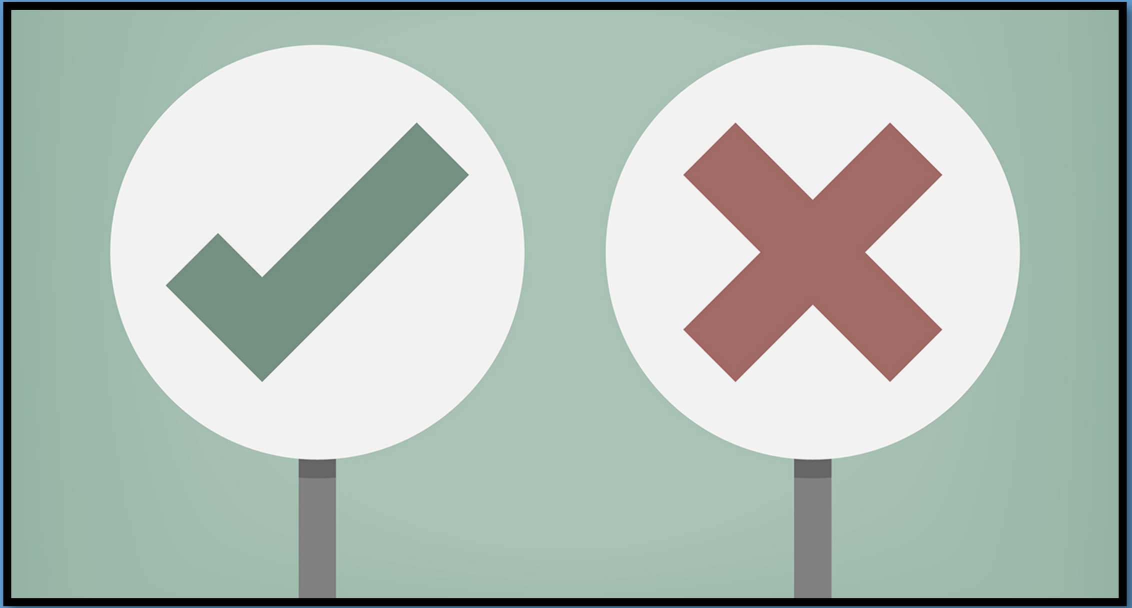
INDEX

INTRODUCTION

1. WHY DOES IT EXIST?

“Retrieves the original idea of the AWT library to use native components –when possible”

2. SWT VS SWING



2. SWT VS SWING

PROS SWING

- No external libraries needed.
- Same results on every platform.
- Graphic editors.
- Supported by official Java extensions.
- Extensive documentation.

PROS SWT

- Native components.
- Strongly supported by Eclipse IDE.

2. SWT VS SWING

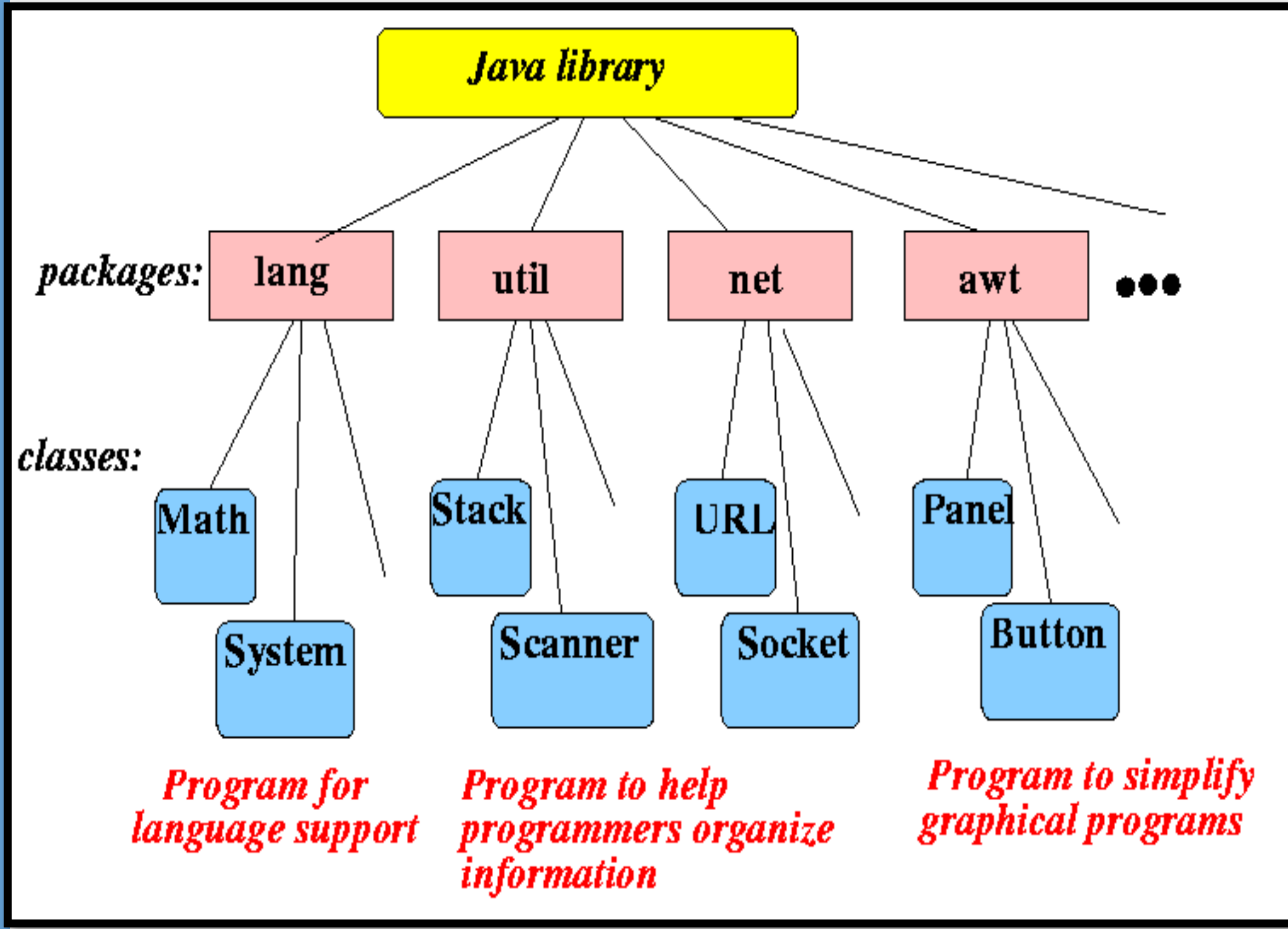
CONS SWING

- Far from a native experience.
- SWING light components are superposed by heavy components.

CONS SWT

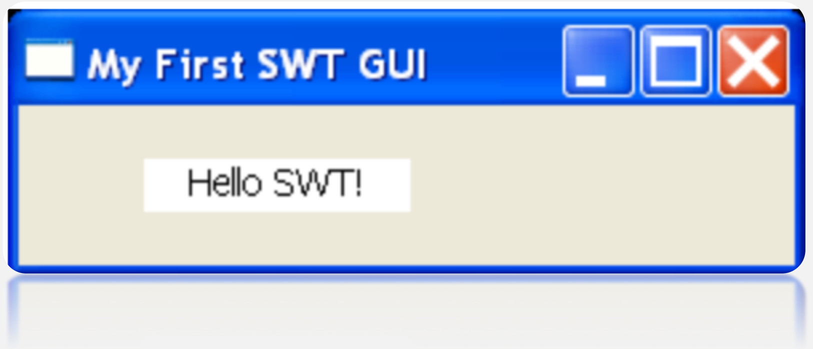
- Requires native libraries.
- It may not be able to emulate every behaviour.

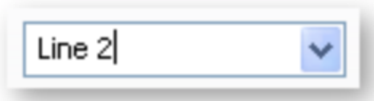
3. ECLIPSE LIBRARIES


Java library
packages:
lang
util
net
awt
•••
classes:
Math
System
Stack
Scanner
URL
Socket
Panel
Button
Program for language support
Program to help programmers organize information
Program to simplify graphical programs

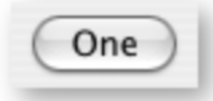
4. HOW TO USE IT

5. BASICS ELEMENTS

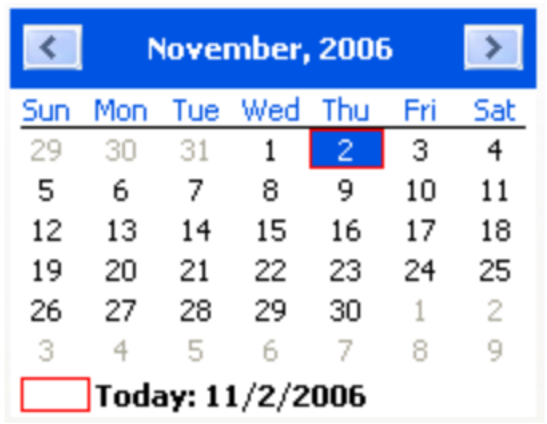




Combo



Button (`SWT.PUSH`)



November, 2006

Sun	Mon	Tue	Wed	Thu	Fri	Sat
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	1	2
3	4	5	6	7	8	9

Today: 11/2/2006

DateTime

Jack and Jill went up the hill to fetch a pail of water, Jack fell down and broke his crown and Jill came tumbling after!

Label

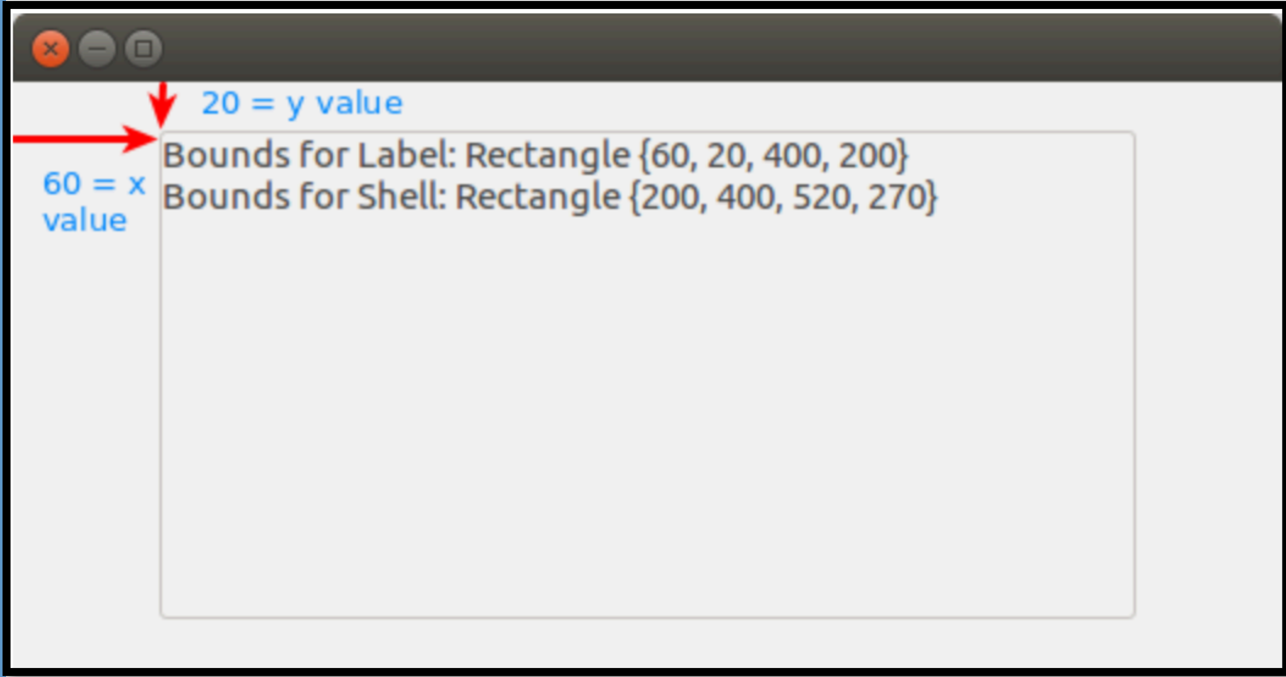
IN DEEP

CREATING A JAVA APPLICATION USING SWT

1. THE STYLEBITS

- Defined constants in SWT class.
- Specify widget properties.
- Each Widget accepts a wide range of *stylebits*.
- They allow to have few classes for the enormous amount of possible widgets.

2. POSITIONING WIDGETS



```
Bounds for Label: Rectangle {60, 20, 400, 200}
Bounds for Shell: Rectangle {200, 400, 520, 270}
```

2.1 LAYOUT DATA

GridData

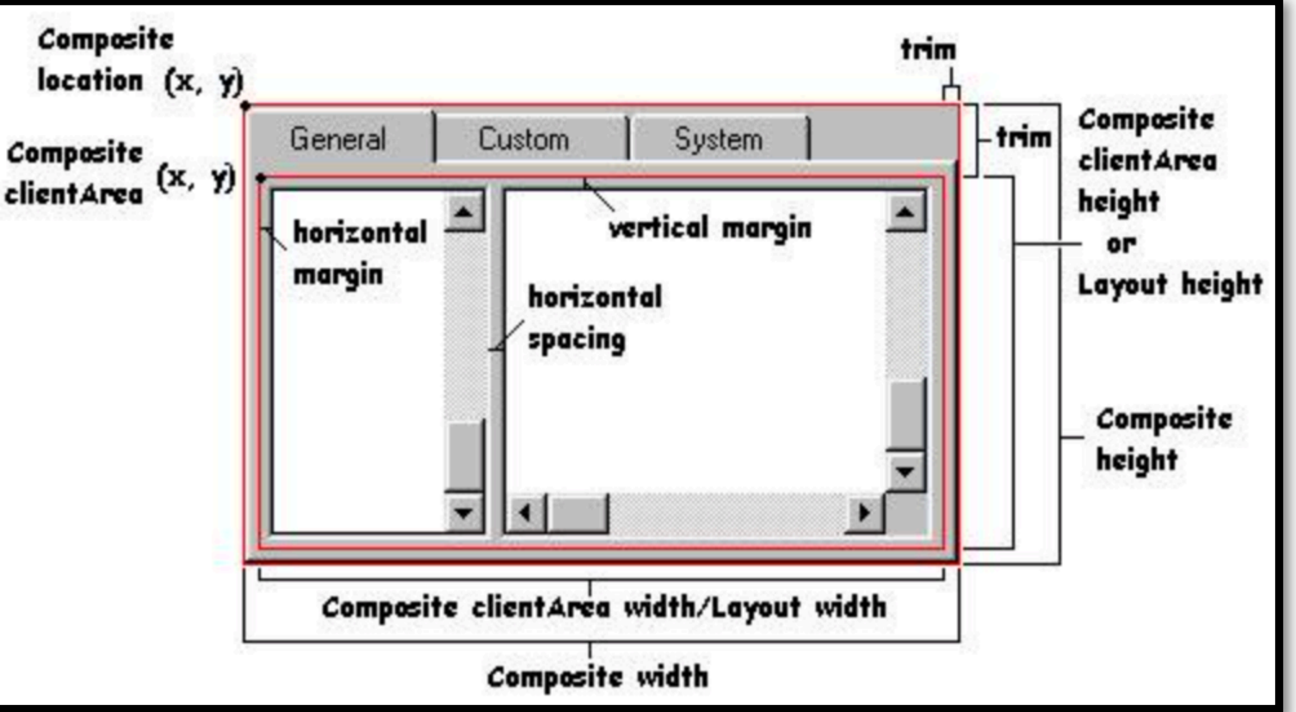
RowData

FormData

- Allow the developer to control the arrangement of the widgets within the layout.

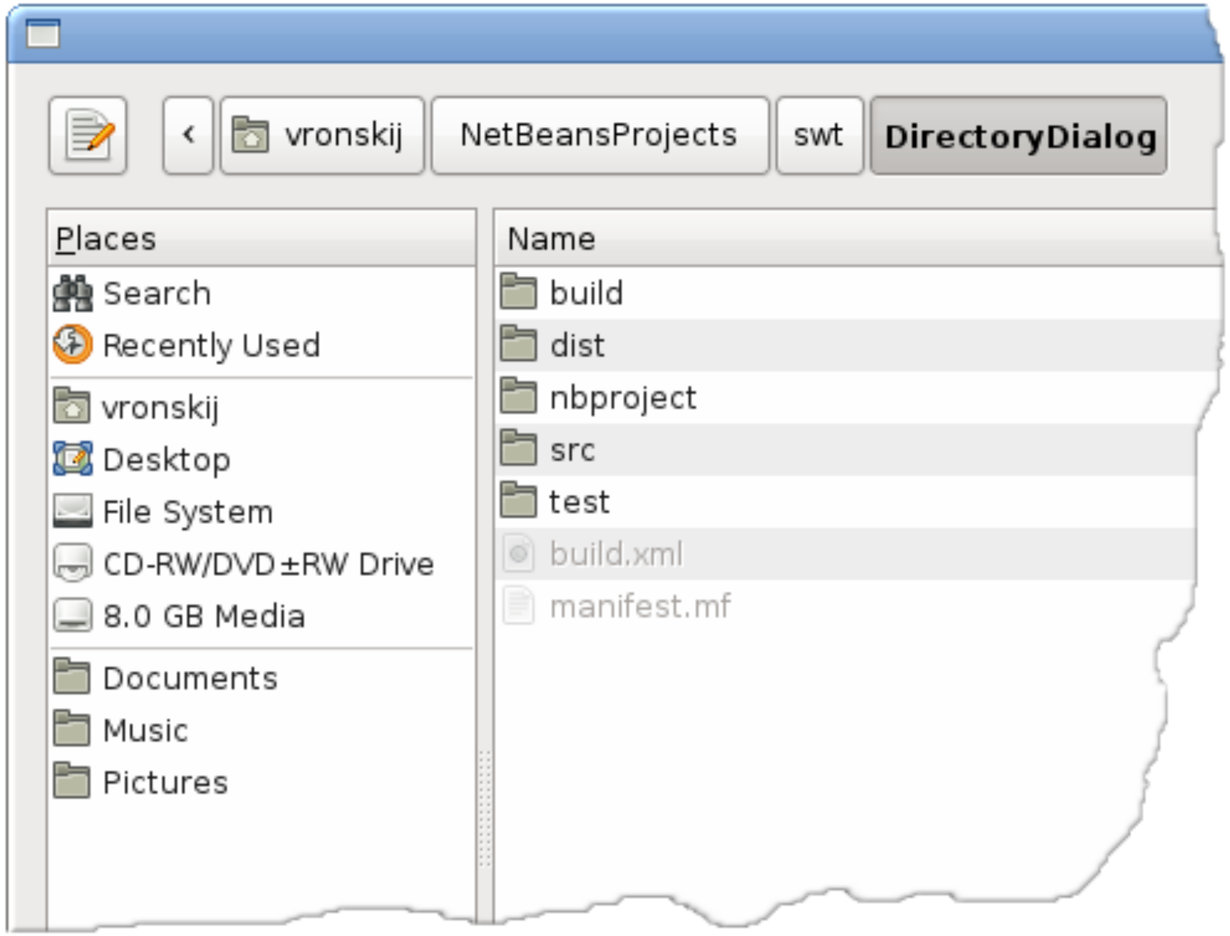
```
button = new Button(parent, SWT.PUSH);
GridData gridData = new GridData();
gridData.horizontalSpan = 2;
button.setLayoutData(gridData);
```

3. LAYOUTS


Composite location (x, y)
trim
Composite clientArea (x, y)
General
Custom
System
trim
Composite clientArea height or Layout height
horizontal margin
vertical margin
horizontal spacing
Composite height
Composite clientArea width/Layout width
Composite width

- FillLayout
- RowLayout
- GridLayout
- FormLayout

4. DIALOGS



- MessageBox
- Directory Dialog
- Color Dialog
- File Dialog

5. COLORS IN SWT

- You can define colors in SWT.
- You can also use system colors, just like in SWING.

```
// Creating colors
Device device = Display.getCurrent();
Color red = new Color (device, 255, 0, 0);

// Using system colors
Display display = Display.getCurrent();
Color blue =
display.getSystemColor(SWT.COLOR_BLUE);

// Free resources
red.dispose();
blue.dispose();
```

6. FONTS IN SWT

```
Label label = new Label(parent, SWT.NONE);
Font font = new Font(label.getDisplay(), new
    FontData("Mono", 10, SWT.ITALIC));
label.setFont(font);
```

```
Label label = new Label(parent, SWT.NONE);
FontData fData= label.getFont().getFontData()[0];
fData.setStyle(SWT.ITALIC);
label.setFont(new Font(label.getDisplay(), fData));
```

```
boolean fontLoaded =
Display.getDefault().loadFont(”path");
if (fontLoaded) {
    Font font = new Font(Display.getDefault(),
        "Custom Font", 12, SWT.NORMAL);
    label.setFont(font);
}
```

7. GRAPHICS IN SWT

- The package `org.eclipse.swt.graphics` contains classes that allow management of graphic resources.
- Every object that implements `Drawable` can be drawn on.
- Graphic Context Class (GC), encapsulates all the drawing API.
- Most of SWT graphics drawing occurs through events.

```
Image image = new Image(display, ”path/to/img”);
GC graphicsContext = new GC(image);
Rectangle bounds = image.getBounds();
graphicsContext.drawLine(0, 0, bounds.width, bounds.height);
graphicsContext.drawLine(0, bounds.height, bounds.width, 0);
graphicsContext.dispose();
image.dispose();
```

```
shell.setLayout(new FillLayout());
final Canvas canvas = new Canvas(shell,SWT.NO_REDRAW_RESIZE);

canvas.addPaintListener(new PaintListener() {
    public void paintControl(PaintEvent e) {
        Rectangle clientArea = canvas.getClientArea();
        e.gc.setBackground(display.getSystemColor(SWT.COLOR_CYAN));
        e.gc.fillOval(0,0,clientArea.width,clientArea.height);
    }
});
```

8. EVENT ORIENTED PROGRAMMING

- **Untyped listeners** can lead to smaller code.
  - offer a generic, low-level mechanism to listen for events.

- **Typed listeners** lead to more modular designs.
  - can be used to listen for only one particular typed event. For example, SelectionListener is a typed listener for event SelectionEvent.

8.2.1 UNTYPED LISTENERS

```
object.addListener(SWT.DISPOSE, new Listener() {
  public void handleEvent(Event e) {
    // Simple managing of an event
  }
});
```

- The untyped listener interface is represented by the Listener interface
- It contains one method: void handleEvent(Event event)
- To add an untyped listener to a widget, call addListener() on it.
- eventType contains one of the event type constants from the SWT class. Once again, a stylebit.

8.2.2 TYPED LISTENERS

```
object.addDisposeListener(new DisposeListener() {
  public void widgetDisposed(DisposeEvent de) {
    // Specific managing of a DisposeEvent
  }
});
```

- Typed listeners use classes and interfaces specific to each possible event.
- All typed events ultimately derive from a common class: TypedEvent.
- Some examples:
  - ControlListener
  - KeyListener
  - MenuListener
  - MouseListener
  - And more…

FINAL EXAMPLES

ANY QUESTION?

REFERENCES

- Eclipse Documentation
  - https://www.eclipse.org/swt/
- Vogella Tutorial
  - http://www.vogella.com/tutorials/SWT/article.html
- ZenCode Tutorial
  - http://zetcode.com/gui/javaswt/
- Wikipedia
  - https://es.wikipedia.org/wiki/SWT

THANK YOU