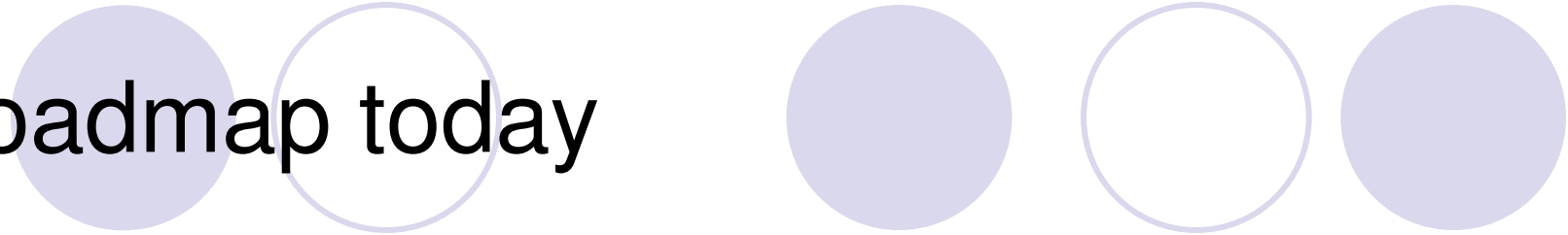
CS3101-3 Programming Language - JAVA

Fall 2004

Oct. 20th

Roadmap today



- More GUI
- JAR file
- Javadoc
- More advanced topic
  - XML parser
  - SQL access

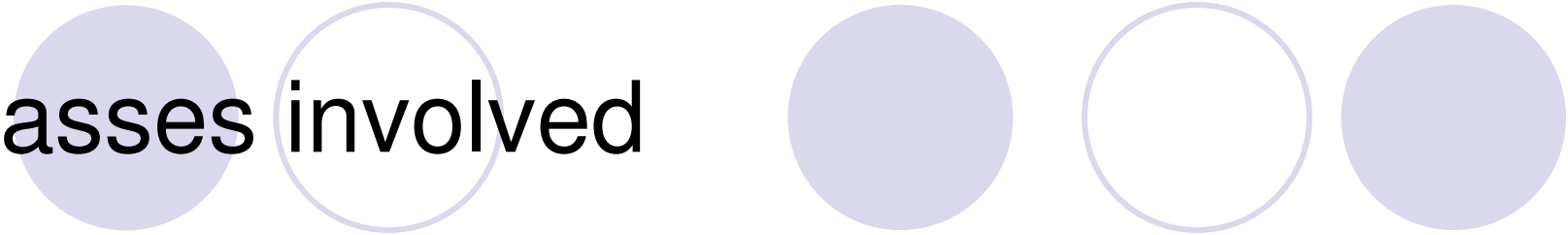
GUI

- To become familiar with common UI components such as menus, combo boxes, buttons, text
- To understand the use of layout manager to arrange components
- To build programs that handles events from UI components
- How to use Java document

Event Handling

- Event: user input, click button, mouse move, hit keyboard…etc.
- Java window manager sends a notification to the program that an ***Event*** occurred.
  - Huge number of events
  - Program has no interest in most of them
  - Should not be flooded by the boring events
- Program must indicate which events it likes to receive: ***event listener***

Classes involved

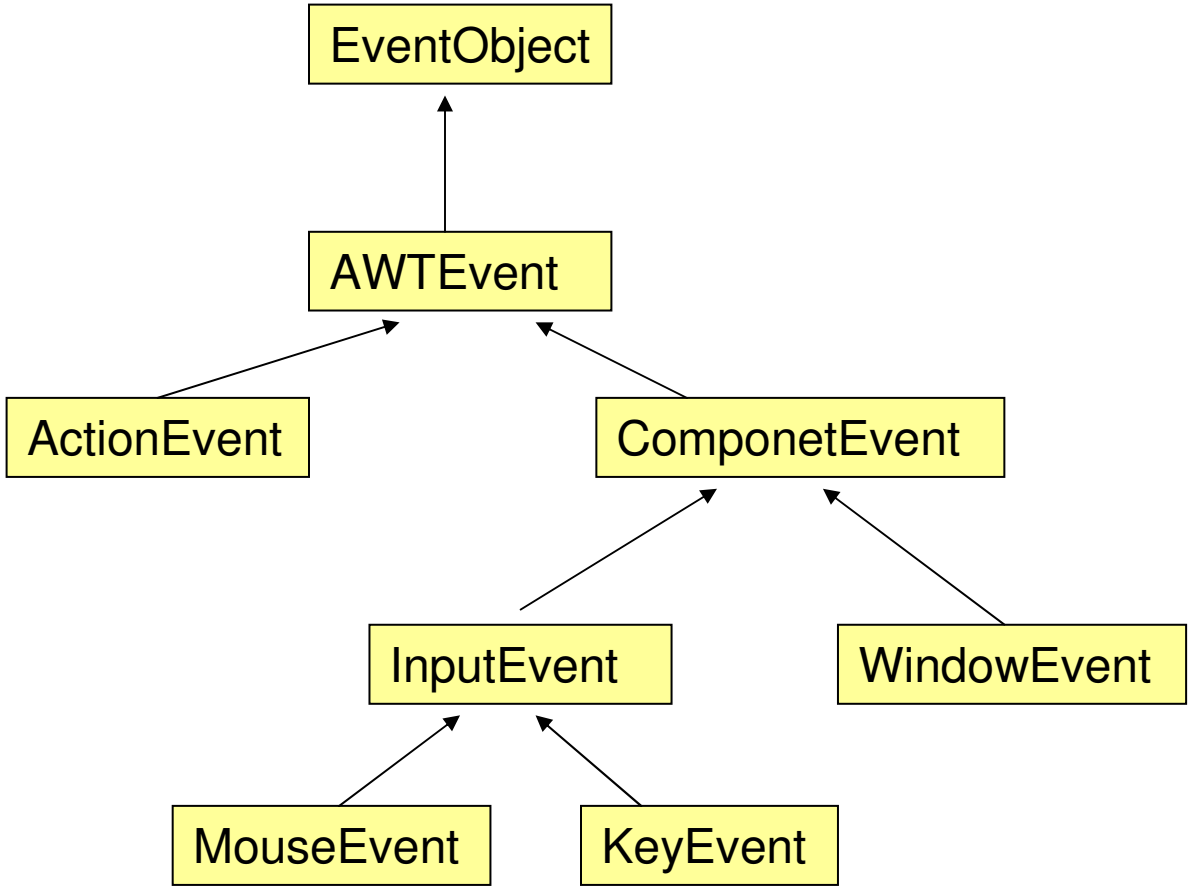


- Event class
  - Mouse move: MouseEvent (tell you x, y position of the mouse or which button clicked)
- Listener class
  - Implements the MouseListener interface, each method has a MouseEvent parameter
- Event source
  - The component that generates the event

Event Classes

EventObject has **getSource()** method, which returns the object that generated this event.

The subclasses have other methods that describe the event further

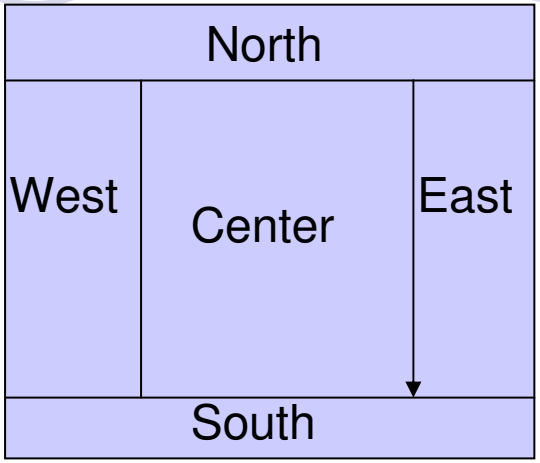


Event adapters

- A listener must implement all the methods in the corresponding listener interface
  - Sometimes too tedious
- Provide Adapter class so that user don't need to define every method, only the one interested in

```
public interface MouseListener{
        void mouseClicked(MouseEvent e);
        void mouseEntered(MouseEvent e);
        void mouseExited(MouseEvent e);
        void mousePressed(MouseEvent e);
        void mouseReleased(MouseEvent e);
}
public class MouseAdapter implements MouseListener{
        //define all the five methods above, but all do nothing
}
Class MyMouseListener extends MouseAdapter { … }
```

JFrame: border layout




```
class MyFrame extends JFrame{
    public MyFrame(){
            MyPanel panel = new JPanel();
            Container contentPane = getContentPane();
            contentPane.add(panel, “Center”);
    }
}
```

Layout

- Border layout
  - Content pane
- Flow layout: simply arranges its components in a row and starts a new row when there is no more room
  - JPanel by default is flow layout
  - panel.setLayout(new BorderLayout());
- Grid layout: arrange components in a grid with fixed number of rows and columns, each has same size
  - panel.setLayout(new GridLayout(4, 3));
- More: BoxLayout, GridBagLayout, SpringLayout

Example

JButton

- Can be constructed with a label, an icon or both
  - new JButton(“Next”);
  - new JButton(new ImageIcon(“next.gif”);
  - new JButton(“Next”, new ImageIcon(“next.gif”);
- Listener needs to implements ActionListener interface, actionPerformed() method
- If multiple buttons use the same listener, need to use getSource() method to use which one been clicked
  - When multiple buttons do similar task

Text component

- JTextField: a single line of text
  - Specify number of characters in constructor
  - Hitting Enter key generates ActionEvent
  - setEditable(false) makes it only for display
  - setText() to set the content
  - Usually putting a JLabel next to it
- JTextArea: multiple lines of text
  - Specify number of rows and columns in constructor
  - Hitting Enter key just start a new line. Need to add a button to generate the event

Choices

- Radio Buttons
  - Choices are mutually exclusive
  - Only one button out of a set can be selected. When one set on, the others set off
- Check Boxes
  - Choices are not exclusive, can have multiple choices
- Combo Boxes
  - When the choice candidates are too many to display as buttons
  - Exclusive choice

Radio Buttons

- To create a set of radio buttons, first create each button individually, then add to the set
- The location of each button is not necessarily be close to each other
- setSelected(true) to turn on a button
- isSelected() to find out whether a button is on or off

```
JRadioButton smallButton = new JRadioButton(“Small”);
JRadioButton mediumButton = new JRadioButton(“Medium”);

ButtonGroup sizeGroup = new ButtonGroup();
sizeGroup.add(smallButton);
sizeGroup.add(mediumButton);
```

Check Boxes

- Check boxes are separate from each other
- JCheckBox check = new JCheckBox(“bold”);
- Do not place check boxes inside a button group

Combo Boxes

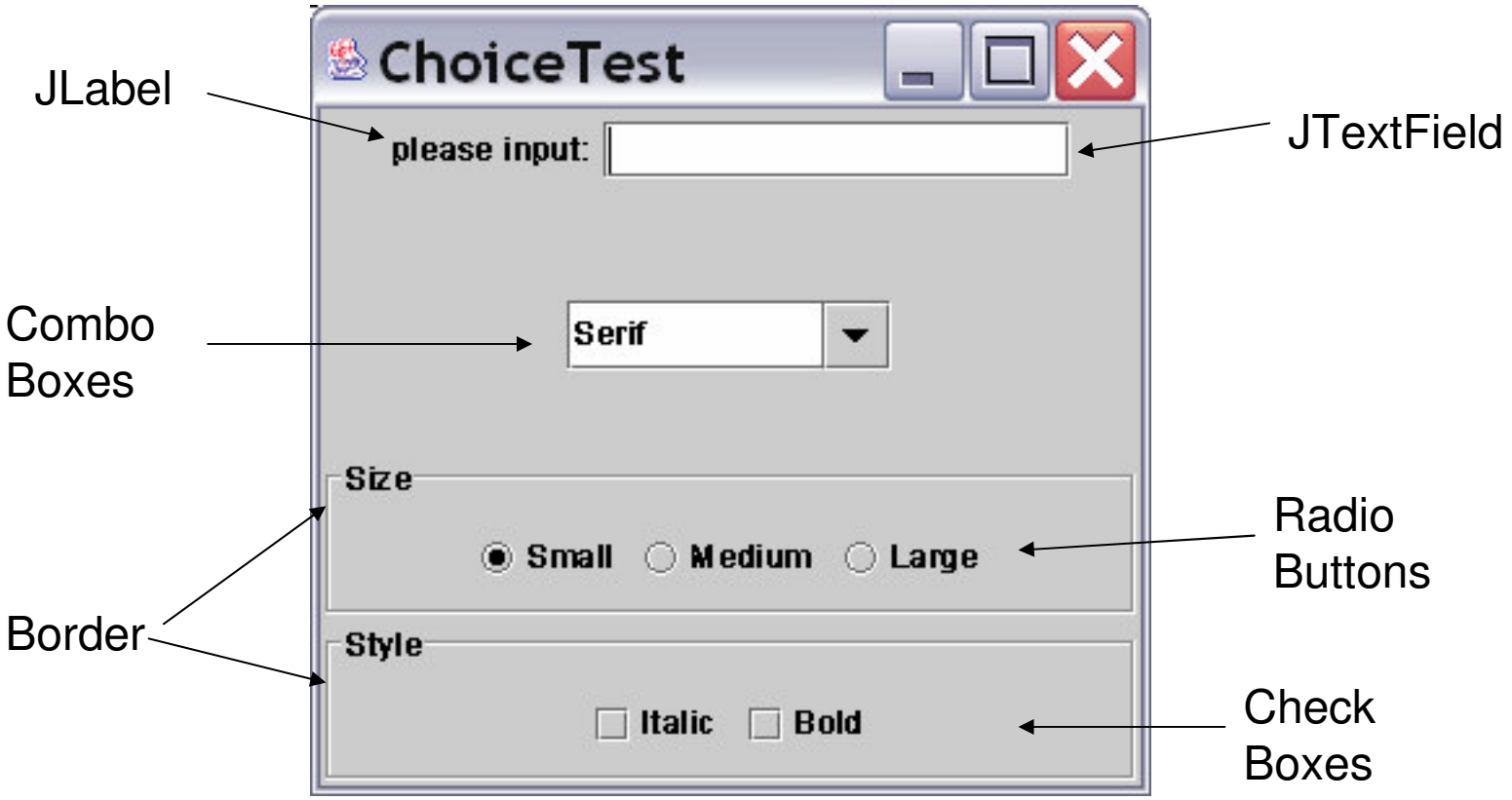
- A list of selections displayed when click on the arrow of it
- Choose one of them
- Can make the box editable by setEditable(true)
- getSelectedItem() to return the selected one, need to cast type
- isSelected(), setSelectedItem();

```
JComboBox nameCombo = new JComboBox();
nameCombo.addItem(“Micheal”);
nameCombo.addItem(“John”);

String s = (String)nameCombo.getSelectedItem();
```

Add border

- You can add a border to a panel to make it visible
- Many times of border
- Can add border to any component, but usually to panel
- Borders can be layered also.

```
JPanel myPanel = new JPanel();
myPanel.setBorder(new EtchedBorder());

myPanel.setBorder(new TitledBorder(new EtchedBorder(), “size”);
```

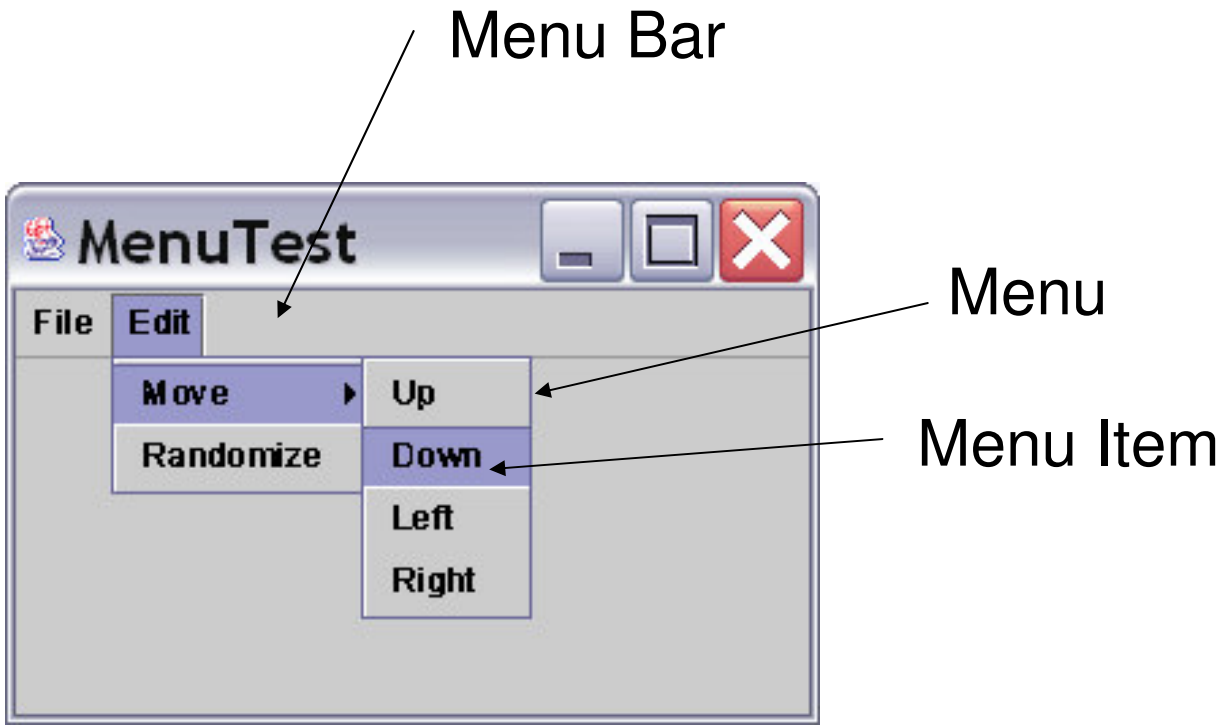

ChoiceTest
JLabel
please input:
JTextField
Combo
Boxes
Serif
Size
Small
Medium
Large
Radio
Buttons
Border
Style
Italic
Bold
Check
Boxes

Example

Menus

- Menu bar is the top-level container, and attach to frame
- Then add menus to the menu bar
- A menu is a collection of menu items, and more menus

```
public class myFrame extends JFrame{
    public myFrame(){
            JMenuBar menuBar = new JMenuBar();
            setJMenuBar(menuBar);
            JMenu menu = new JMenu("File");
            menuBar.add(menu);

            JMenuItem item = new JMenuItem("New");
            menu.add(item);
    }
}
```


Menu Bar
MenuTest
File
Edit
Move
Randomize
Up
Down
Left
Right
Menu
Menu Item

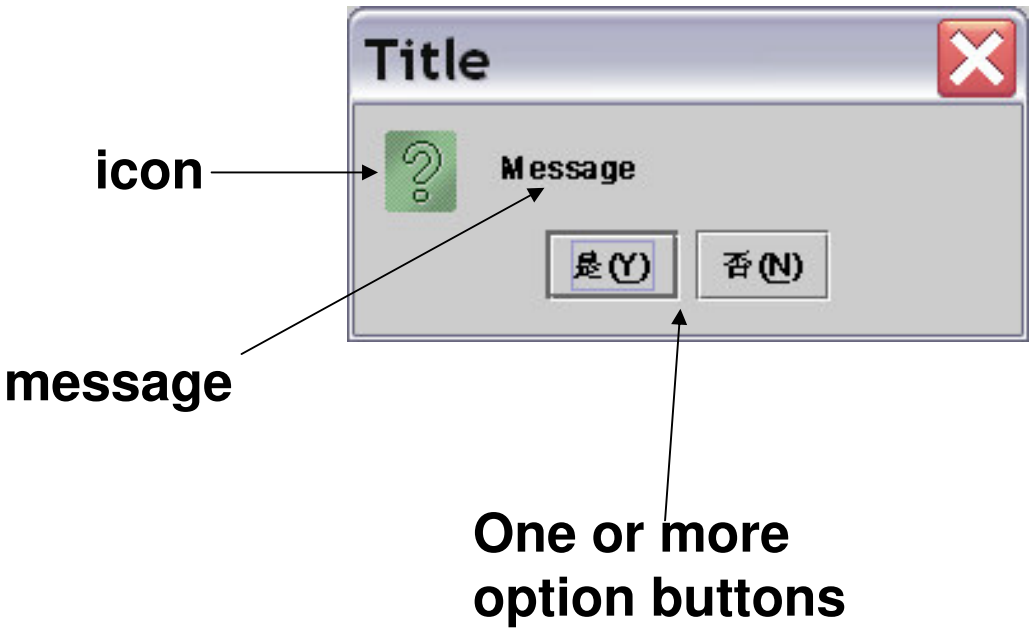
Example

Dialog Boxes

- Modal dialog box
  - User cannot interact with the remaining window until he/she deals with the dialog box
    - Yes/no answer
    - Choose a file
- Modeless dialog box
  - User can proceed with the remaining window with the dialog box poped up
  - Typical example: toolbar

JOptionPane

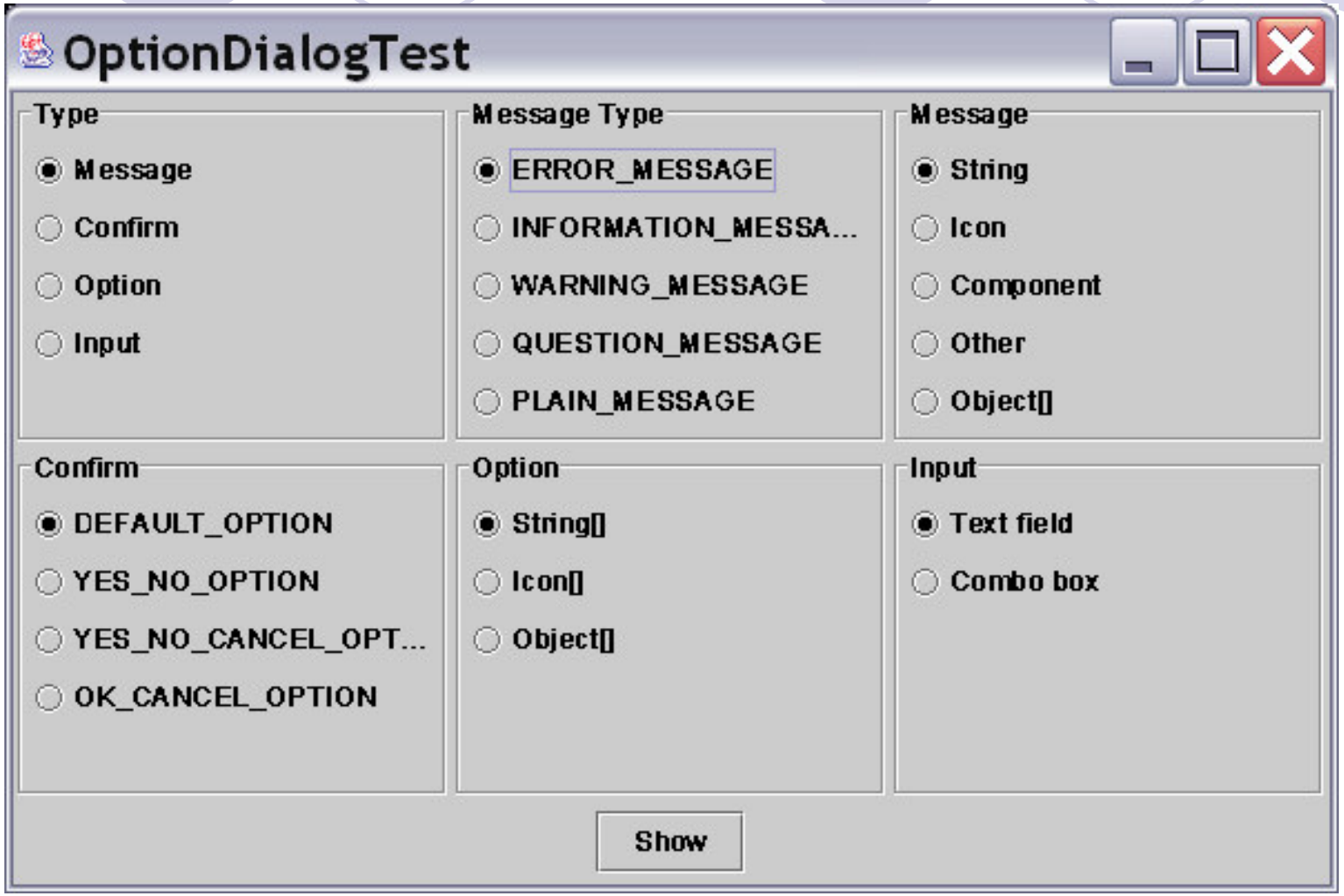
- Simples
- Modal dialog with a single message
- But still have a lot of choices



The icon depends on the message types.

The option button depends on the option types (confirm type)

The choices for JOptionDialog

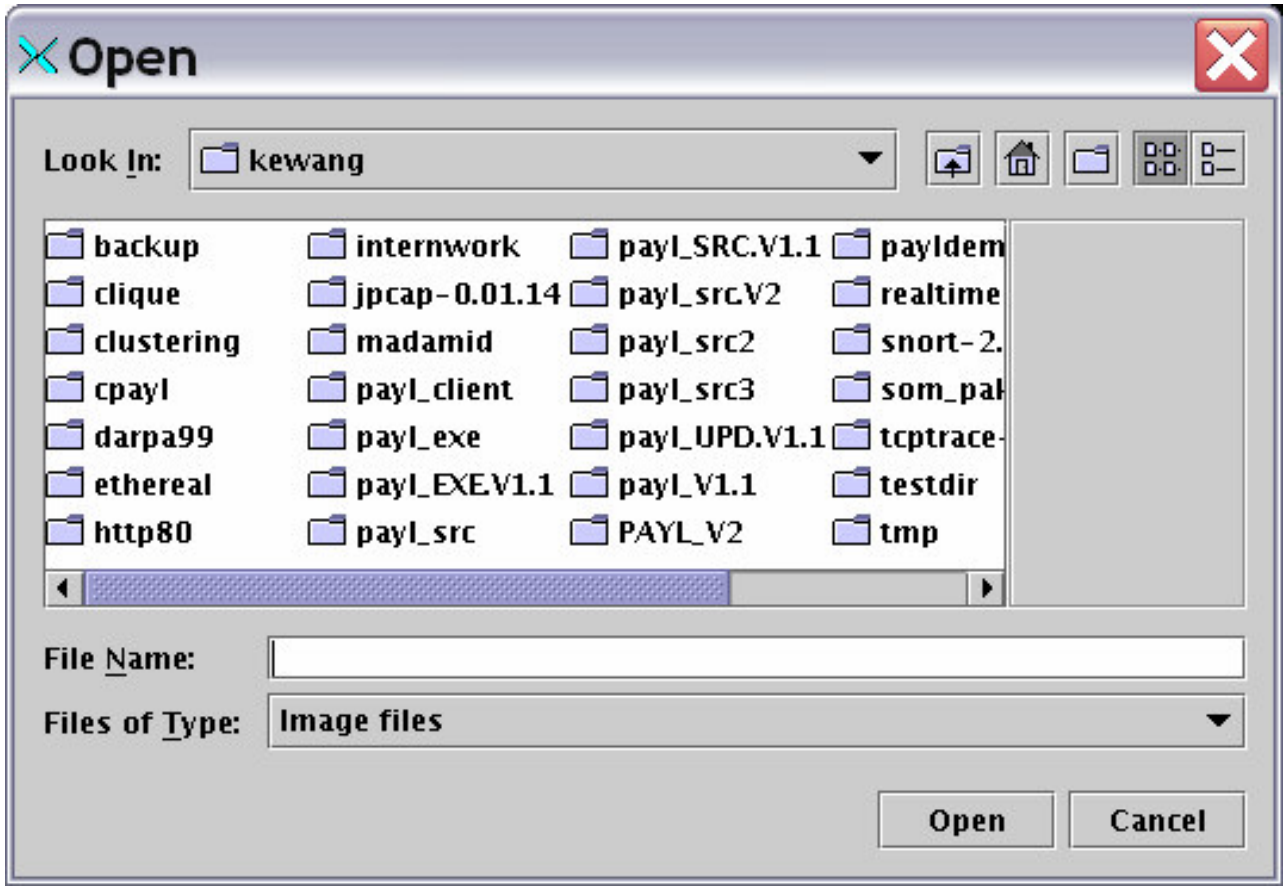


Sample code piece

```
Int selection = JOptionPane.showConfirmDialog(
        parent,   //parent component, can be null
        “Message”,  // message to show on the dialog
        “Title”,  // the string in the title bar of dialog
        JOptionPane.OK_CANCEL_OPTION, // confirm type
        JOptionPane.WARNING_MESSAGE); //message type

If(selection == JOptionPane.OK_OPTION) …
```

File Dialogs

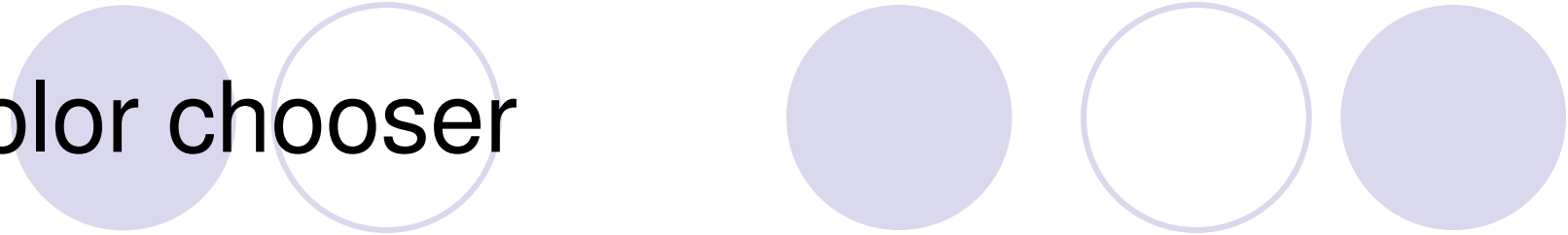
- Shows files and directories and lets user navigate the file system
- JFileChooser class
  - Not a subclass of JDialog!!
- Always modal
- showOpenDialog() opens a dialog for opening a file
- showSaveDialog() for save a file


Open
Look In:
kewang
backup
clique
clustering
cpayl
darpa99
ethereal
http80
internwork
jpcap-0.01.14
madamid
payl_client
payl_exe
payl_EXE.V1.1
payl_src
payl_SRC.V1.1
payl_src.V2
payl_src2
payl_src3
payl_UPD.V1.1
payl_V1.1
PAYL_V2
payldem
realtime
snort-2.
som_pal
tcptrace-
testdir
tmp
File Name:
Files of Type:
Image files
Open
Cancel

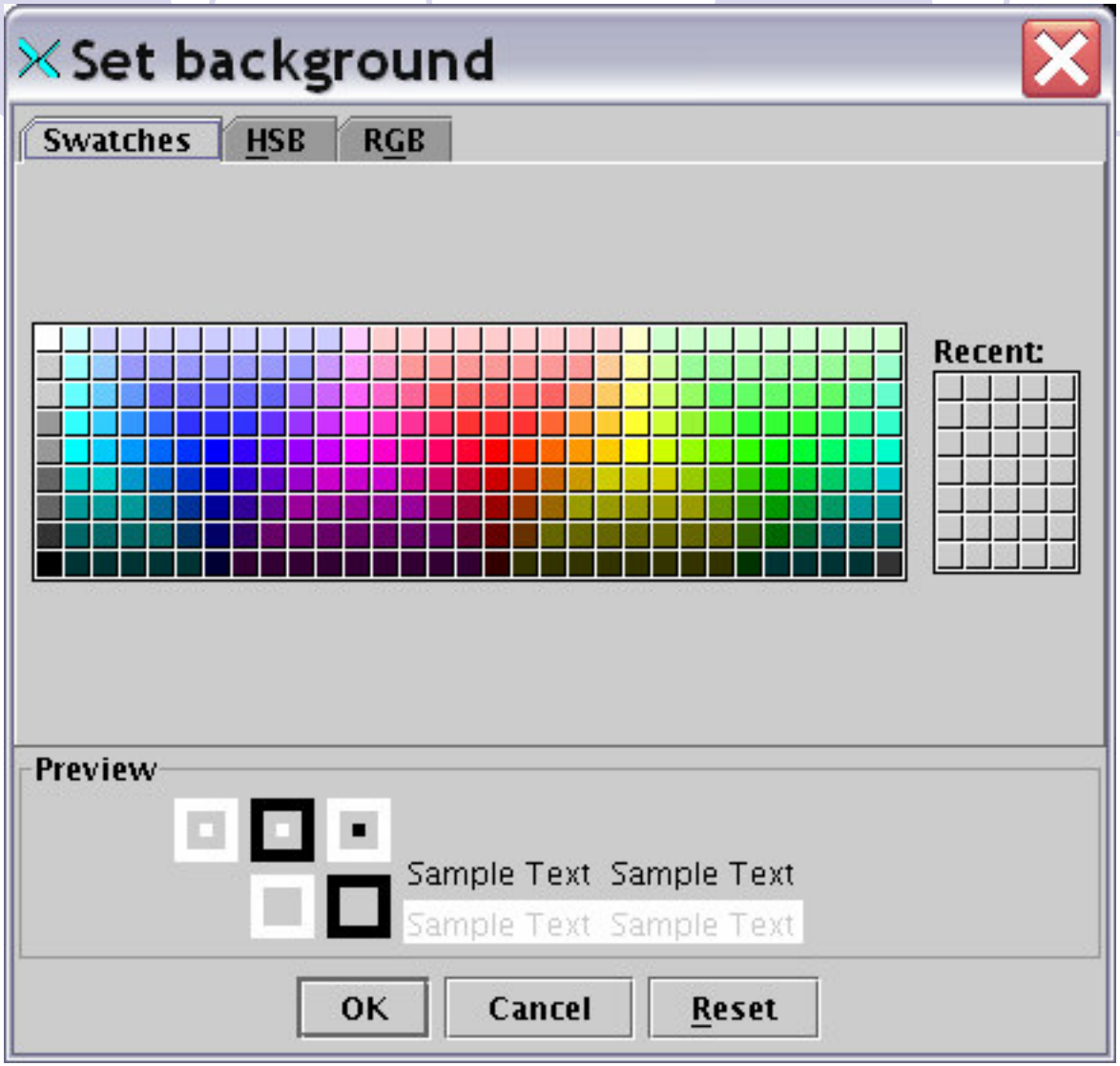
Common Usage

```
JFileChooser chooser = new JFileChooser();

chooser.setCurrentDirectory(new File(“.”));

chooser.setMultiSelectionEnabled(true);

chooser.showOpenDialog(parent);

String filename = chooser.getSelectedFile().getPath();
```

Color chooser



- JColorChooser class
- Similar to JFileChooser class
- Go check API


Set background
Swatches
HSB
RGB
Recent:
Preview
Sample Text Sample Text
Sample Text Sample Text
OK
Cancel
Reset

How to explore Swing documents

- Too many UI components to be covered in class, and each component has too many options
- You have to find out the one you need by yourself
- Go to Java API document, look at the names of all classes starts with J
- Or run samples with JDK
- Or google it!

What to know before use it

- How to constructor it?
- How can I get notified when the user has choose it, or modify it?
  - What kind of event does it generate?
- How can I tell to which value the user has set it?

Classpath

- List of location where JVM should look for classes
- An environment variable
- Can also be specified at run time
  - java –classpath .;./lib/xx.jar myProgram
- Often includes “.” in the list, to look in current directory
- “;” seperator in windows, “:” in unix

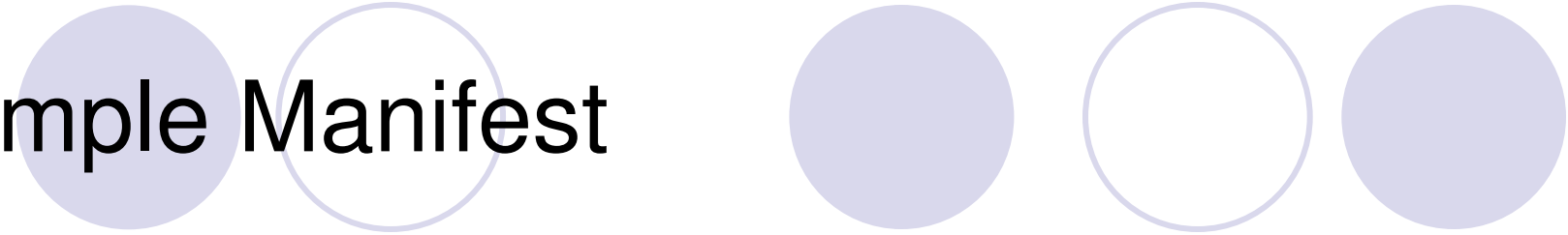
JAR Files

- JAR: Java Archive files
- Package all needed files into a single file
  - Applications, code libraries
  - JRE is contained in a large file rt.jar
- Compressed files, using the ZIP compression
- Can contain both class files and other file types such as image and sound
- Use jar tool to make it
  - jar cvf JARFilename file1 file2 …

Self-running JAR files

- Need a Manifest file to specify the *main class* of the application
  - Manifest file describes the special features of the archive
- Put it together with .class files into the .jar file
  - jar –cvfm myProgram.jar Manifest *files to add*
- Then run it
  - java –jar myProgram.jar

Simple Manifest



```
Manifest-Version: 1.0
Main-Class: MyMainClass
```

Double click to launch

- On Windows, Java runtime installer creates file association with “.jar” extension, and launch the file with “javaw –jar” command
  - Javaw doesn’t open a shell window
- Solaris, OS also recognize .jar file and launch with java –jar command

Javadoc utility

- Javadoc is used to generate documentation that can be inspected by a web browser for your program
- Must use the exact format for method comments
- Then run “javadoc MyProg.java”, and generate MyProg.html.
  - The html looks similar to the JDK API

```
/**
    purpose
    @param name description
    @param name description
    @return description
*/
public void method(){ …}
```

JDBC

- Need to install driver for your DB
  - http://servlet.java.sun.com/products/jdbc/drivers
- Establish a connection with a database or access any tabular data source
- Send SQL statements
- Process the results

Connection

- A *Connection* object represents a connection with a database
- A connection session includes the SQL statements that are executed and the results that are returned over that connection.
- A single application can have one or more connections with a single database, or it can have connections with many different databases.
- url has format: jdbc:<subprotocol>:<subname>
  - subprotocol: the name of the driver or the name of a database connectivity mechanism
  - Subname: a way to identify the data source
    - //hostname:port/subsubname

```
String url = = "jdbc:mysql://winwood/met1";
connection = DriverManager.getConnection(url, user, passwd);
```

Query

```
Statement statement = connection.createStatement();
String query = "SELECT count(*) FROM " + table;
ResultSet resultSet = statement.executeQuery(query);

metaData = resultSet.getMetaData();
numberOfColumns =  metaData.getColumnCount();

while(resultSet.next()) {
            for(int i=0;i<numberOfColumns;i++)
                Object obj = resultSet.getObject(i);
}
```

What is XML ?

- XML is a text-based markup language for data interchange on the Web.
- It identifies data using “tags”

```
<message>
    <to>you@yourAddress.com</to>
    <from>me@myAddress.com</from>
    <subject>Hello my friend!</subject>
    <text>
        blah blah blah blah …..
    </text>
</message>
```

More XML

- The XML Prolog
  - The declaration that identifies the document as an XML document

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
```

- Tags can also contain attributes

```
<message to="you@yourAddress.com" from="me@myAddress.com"
                subject="Hello my friend!">
    <!-- This is a comment -->
    <text>
                blah blah blah blah …..
    </text>
</message>
```

XML Sample Code

```
<?xml version="1.0" encoding="utf-8" ?>
<!--  A SAMPLE set of slides  -->
- <slideshow title="Sample Slide Show" date="Date of publication" author="Yours Truly">
    <!-- TITLE SLIDE  -->
  - <slide type="all">
      <title>Wake up to WonderWidgets!</title>
    </slide>
    <!-- OVERVIEW  -->
  - <slide type="all">
      <title>Overview</title>
    - <item>
        Why
        <em>WonderWidgets</em>
        are great
      </item>
      <item />
    - <item>
        Who
        <em>buys</em>
        WonderWidgets
      </item>
    </slide>
  </slideshow>
```

JAVA APIs for XML

- SAX: Simple API for XML
  - "serial access" protocol for XML, fast-to-execute
  - Event-driven protocol: register your handler with a SAX parser, which invokes your callback methods whenever it sees a new XML tag
- DOM: Document Object Model
  - Converts an XML document into a collection of objects in your program. You can then manipulate the object model in any way that makes sense.
  - Flexible, but slow and need a lot of memory to store the tree
- http://java.sun.com/xml/jaxp/dist/1.1/docs/tutorial/overview/1_xml.html

Import the Classes

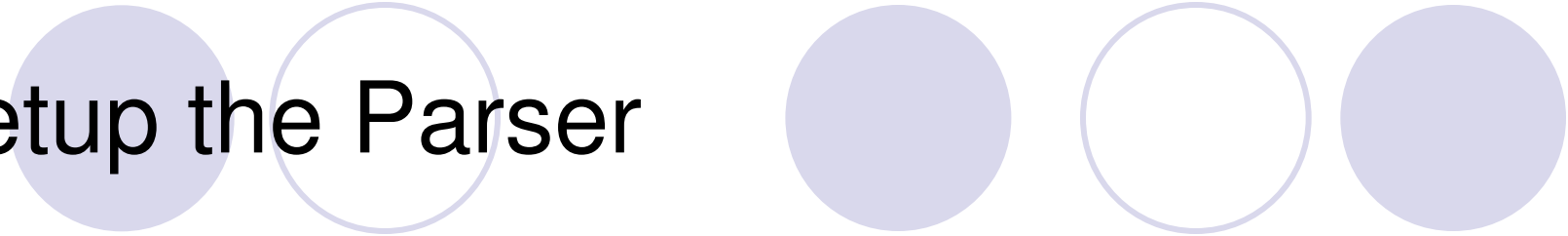
```
import java.io.*;
import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
```

Implement the DefaultHandler Interface

```
public class Echo extends DefaultHandler
{
    ...
}
```

The handler's method get called when the Parser meet the tags

Setup the Parser



```
// Use an instance of ourselves as the SAX event handler
DefaultHandler handler = new Echo();

// Use the default (non-validating) parser
SAXParserFactory factory = SAXParserFactory.newInstance();

// Parse the input
SAXParser saxParser = factory.newSAXParser();
saxParser.parse( new File(argv[0]), handler );
```

Write the Output

```
private void emit(String s) throws SAXException
{
    try {
        out.write(s);
        out.flush();
    } catch (IOException e) {
        throw new SAXException("I/O error", e);
    }
}
```

Process the start of document

```
public void startDocument() throws SAXException
{
    emit("<?xml version='1.0' encoding='UTF-8'?>");
    nl();
}

public void endDocument() throws SAXException
{
    try {
        nl();
        out.flush();
    } catch (IOException e) {
        throw new SAXException("I/O error", e);
    }
}
```

Process the *start-element* Event

```
public void startElement(String namespaceURI,
    String sName, // simple name (localName)
    String qName, // qualified name
    Attributes attrs) throws SAXException
{
    String eName = sName; // element name
    if ("".equals(eName)) eName = qName; // namespaceAware = false
    emit("<"+eName);
    if (attrs != null) {
        for (int i = 0; i < attrs.getLength(); i++) {
            String aName = attrs.getLocalName(i); // Attr name
            if ("".equals(aName)) aName = attrs.getQName(i);
            emit(" ");
            emit(aName+"=\""+attrs.getValue(i)+"\"");
        }
    }
    emit(">");
}
```

Process the *end-element* Event

```
public void endElement(String namespaceURI,
                       String sName, // simple name
                       String qName // qualified name )
    throws SAXException
{
    emit("</"+sName+">");
}
```

Echo the Characters the Parser Sees

```
public void characters(char buf[], int offset, int len)
    throws SAXException
{
    String s = new String(buf, offset, len);
    emit(s);
}
```