CS 349
Scripting

Byron Weber Becker
Spring 2009

Slides mostly by Michael Terry

Examples added by Byron Weber Becker

University of Waterloo

Application Value

- Applications specialize in producing, manipulating specific data types
- An application's value can thus be seen in two lights
  - The data it produces/manages
  - The functionality it provides

Application Value

- But…
  - No one application do everything
  - No one application can include support for every possible use
  - Our data may not always be in the application's native format
  - The application's data may not be in the final format we require
- How can we increase the chance the application can bend to unforeseen, real-world needs?

Planning for Flexibility

- An application gains value if it can:
  - Be internally scripted
  - Be externally controlled/scripted
  - Import/export data
- Gains further value if the source code is freely available…
- These facilities extend the range of possibilities beyond the delivered capabilities

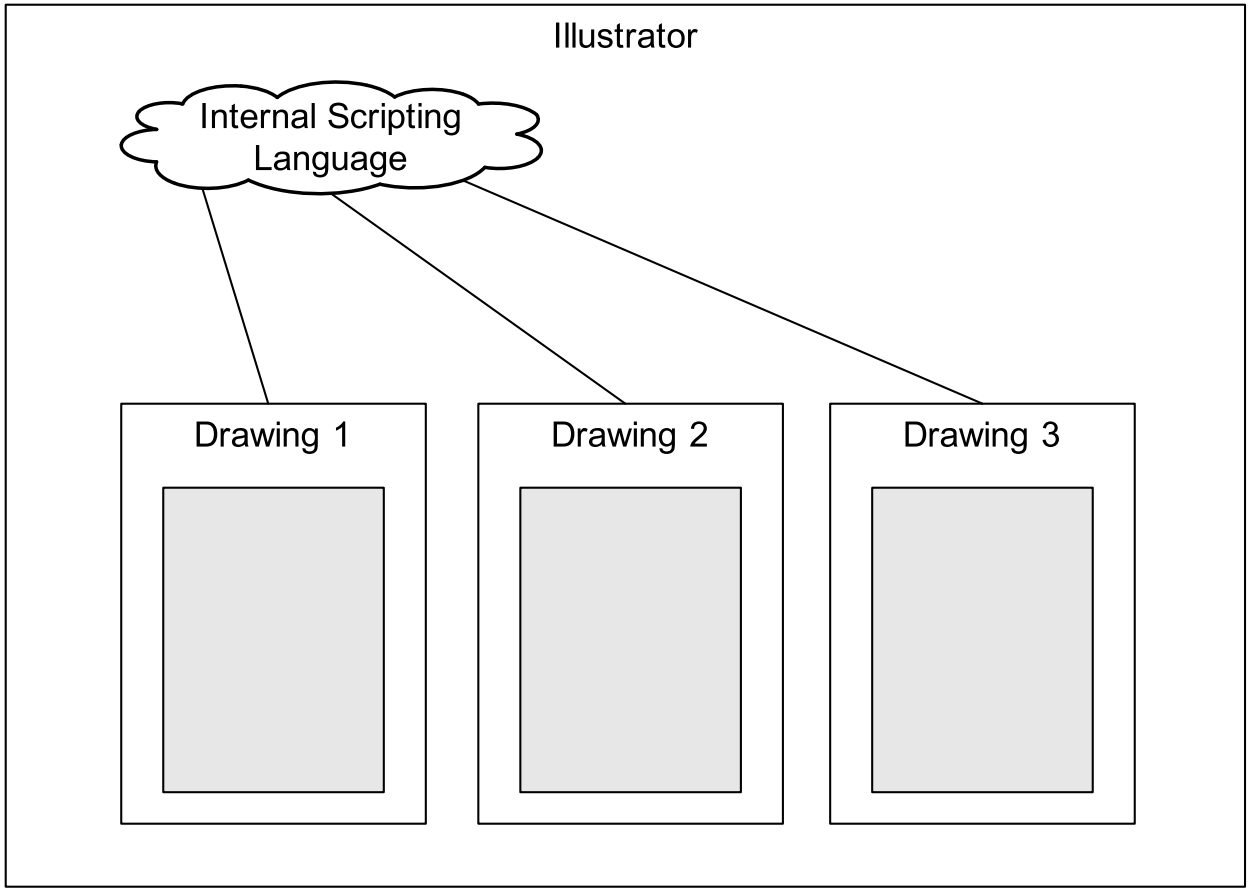
University of Waterloo

Scripting

- Internal scripting
- External scripting / control

University of Waterloo

Internal Scripting



University of Waterloo

Code Review: Scripting Triangles

- Demo1:
  model.setBase(30)
  model.setHeight(40)
- Demo2:
  import time

  for i in range(0, 101, 5):
      model.setValues(i, i)
      time.sleep(0.25)

University of Waterloo

```
// Import the python interpreter
import org.python.util.PythonInterpreter;
…
public class ScriptingView extends JPanel {
…
   private JTextArea script;
   private JButton executeButton;
   private PythonInterpreter pyInterp = null;
…
   private void registerListeners() {
       this.executeButton.addActionListener(new ActionListener() {
           public void actionPerformed(ActionEvent e) {
               if (pyInterp == null) {
                   pyInterp = new PythonInterpreter();

                   // Make these objects available to scripts
                   pyInterp.set("model", ScriptingView.this.model);
                   pyInterp.set("app", Application.getInstance());
                   pyInterp.set("frame", Application.getInstance()
                           .getActiveFrame());
               }
```

```
            // Get the script we are to run.
            final String s =
                        ScriptingView.this.script.getText();
            // Start undo/redo support
            TriangleUndoableEdit cmd =
                        TriangleUndoableEdit.begin(model);

            // Execute the script
            pyInterp.exec(s);

            // Finish undo/redo support
            cmd.end(model.getBase(), model.getHeight());
            ScriptingView.this.undo.addEdit(cmd);
        }
    });
}
```

Internal Scripting

- Internal objects accessible to scripting engine
- Jython provides dead-simple scripting engine for Java
  - Doesn't get any easier than that
- IronPython does same for .NET languages (by same guy who did Jython)
- Other Java scripting possibilities
  - BeanShell (http://www.beanshell.org/)
    - Very Java-like syntax
  - Groovy (http://groovy.codehaus.org/)
    - Python, Ruby, Smalltalk influences
  - Rhino (http://www.mozilla.org/rhino/)
    - Java-based JavaScript
  - JRuby (http://jruby.sourceforge.net/)
    - Ruby port running in the JVM

Initialization Scripting

```
    public void newDocument() {
        final model.TriangleModel model = new …
...
        if (new File("tri.ini").canRead()) {
            PythonInterpreter pyInterp = new
                                    PythonInterpreter();
            pyInterp.set("app", Application.getInstance());
            pyInterp.set("model", model);

            pyInterp.set("frame", frame);
            pyInterp.execfile("tri.ini");
        }
    }
...
```

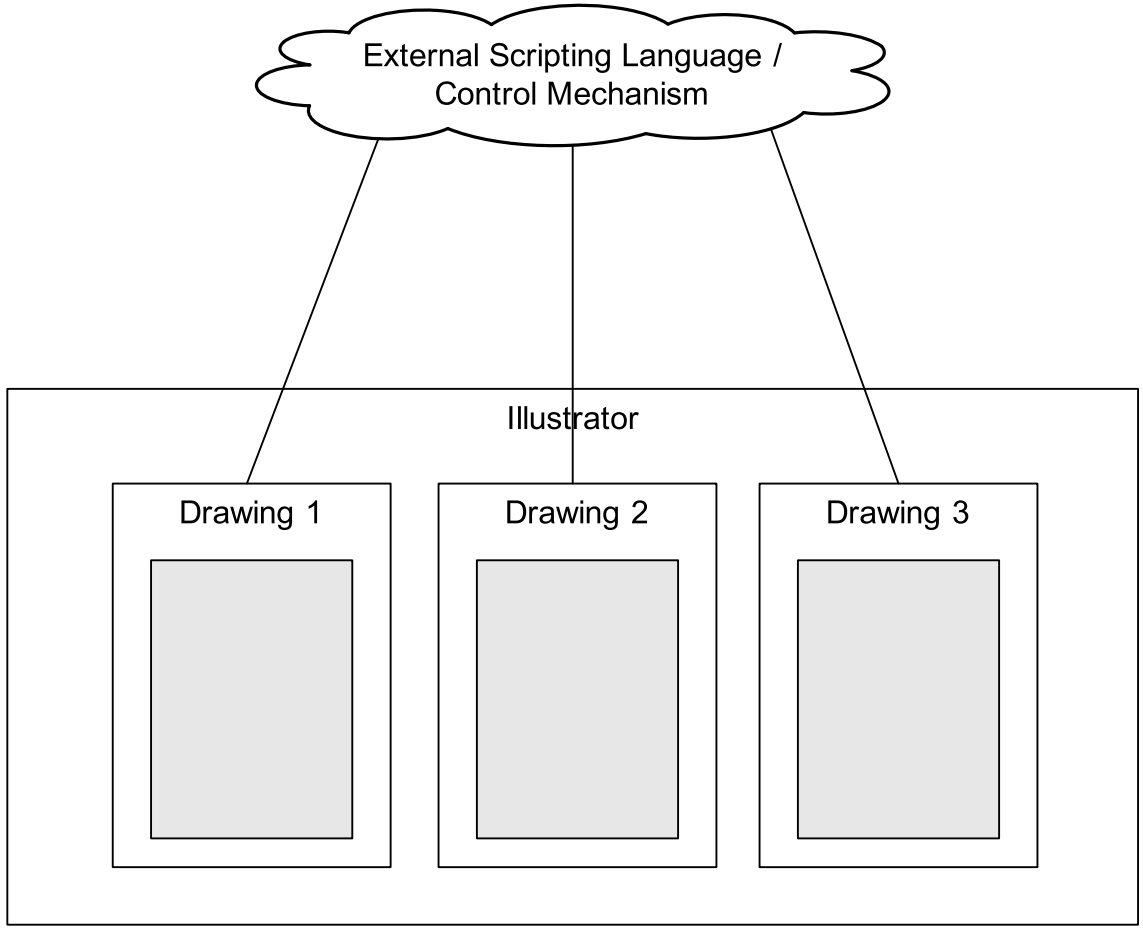
University of Waterloo

22-June-09 Announcements

- Cutting Edge UI
- invokeLater came up last day; discuss in the next lecture unit.

University of Waterloo

External Scripting / Control



University of Waterloo

Code Review: Scripting Triangles

- Demo:
- code:

```
public static void main(String[] args) {

    if (args.length == 1) {
        // Running a script -- no GUI needed.
        TriangleModel model = new TriangleModel();
        PythonInterpreter pyInterp = new
                                        PythonInterpreter();
        pyInterp.set("model", model);
        pyInterp.execfile(args[0]);
    } else {
        // Create a new document and GUI.
        Application.getInstance().newDocument();
    }
}
```

University of Waterloo

External Scripting / Control

- More complex than internal scripting
- Why?

University of Waterloo

External Scripting / Control

- Must expose scriptable portions of application in a standardized way to “outside world”
- How can such functionality be exposed?

University of Waterloo

Approaches to Exposing Functionality

- Command-line switches, pipes
  - Run once
  - Not suitable for scripting a running, interactive application
- Stream-based protocol
  - Develop a protocol for communicating to application
  - Develop data formats
  - Create a network server/client paradigm
  - Named pipes
- Other possibilities
  - Shared memory, blackboards

University of Waterloo

AppleScript

- Define dictionary to application
  - Suites
    - Collection of classes, commands, events
  - Classes
    - Elements
      - If contains elements, it's a container
    - Properties
      - One thing
  - Commands
    - The functions that manipulate objects
    - Not part of the class
    - One command can work with multiple, different object types
  - Events
    - Callbacks from your app
  - Specifier
    - Locates object within object hierarchy or element within a container
- App sends itself commands to support recordability
- See
  - http://www.cocoadev.com/index.pl?HowToSupportAppleScript

University of Waterloo

Links

- http://developer.apple.com/technotes/tn2002/tn2106.html (Guidelines for making applications scriptable with AppleScript)
- Demo

University of Waterloo

Jython Language Tutorial

- Most of what follows is useful for learning/using Jython. It's here as a reference.
- You are *not* responsible for Python syntax details on exams.
- Examples of what you are responsible for:
  - "duck typing"
  - integrating a Jython interpreter with a Java program
  - making Java objects available to a script
  - executing a script
- Examples of what you are not responsible for:
  - syntax of python lists, tuples, dictionaries
  - how to define a python class or convert a python class to a Java class

University of Waterloo

Jython Language Tutorial

- Strongly typed language
  - Dynamic (run-time) type checking
  - No declaration of variable types
  - Employs “duck typing”
- Examples
  - `anInt = 3`
  - `aFloat = 3.0`
  - `aString = "String"`
  - `anotherString = 'Single quote with " inside'`
  - `moreStrings = """Triple quoted`
    `can span multiple lines"""`

University of Waterloo

Duck Typing

- From Python Tutorial Glossary

> **Duck typing**: Pythonic programming style that determines an object's type by inspection of its method or attribute signature rather than by explicit relationship to some type object ("If it looks like a duck and quacks like a duck, it must be a duck.") By emphasizing interfaces rather than specific types, well-designed code improves its flexibility by allowing polymorphic substitution. Duck-typing avoids tests using type() or isinstance(). Instead, it typically employs hasattr() tests or *EAFP* programming.

University of Waterloo

Duck Typing Example in Jython

```
def printComponentText(c):
      print c.getText()

import javax.swing as swing

button = swing.JButton("button")
label = swing.JLabel("label")

for component in [button, label]:
      printComponentText(component)

# Note that JButton and JLabel do not share a
# common parent class defining the getText() method
```

University of Waterloo

Lists

- Mutable sequences (lists are actually objects)
- Hold heterogeneous sets of objects
- Can retrieve “slices”
- *Can’t* be used as keys in dictionaries because they are mutable
- Examples

```
myList = [3, "String", 3.14]
myList.append(10) # add another item to the list
len(myList) # returns 4
print myList[0:2] # [3, "String"]
print myList[:2]  # [3, "String"]
print myList[2:]  # [3.14, 10]
print myList[-3:] # ["String", 3.14, 10]
myList[:] # Creates a copy of myList
```

University of Waterloo

Tuples

- *Immutable* sequences (tuples also are objects)
- *Can* be used as keys in dictionaries because immutable
- Similar properties, semantics to lists
  - Just can't modify tuple once made
- Examples

```
myTuple = (3, "String", 3.14)
len(myTuple) # returns 3
print myTuple[0:2] # [3, "String"]
oneElementTuple = (3,)
```

Dictionaries

- Key-value mapping container (also an object)
- Examples

```
myMap = {3 : "Three", 4 : "Four"}
print myMap[3] # prints "Three"
myMap[5] = "Five"
print myMap[5] # prints "Five"
myMap["Six"] = 6
print myMap["Six"] # prints 6
```

University of Waterloo

Membership Testing

- Can use `in` to test membership in lists, tuples, dictionaries
- Examples
  - `3 in [3, 4, 5] # Returns True`
  - `3 in (2, 4) # Returns False`
  - `3 in {'Three' : 3} # Returns False, 3 is not a key`
  - `'Three' in {'Three' : 3} # Returns True`

University of Waterloo

Blocks

- Whitespace (indentation) used to delineate blocks
  - No braces
  - No “begin/end” keywords
- Whitespace must be of same type/length within a block
  - One tab, one space, two tabs, two spaces, etc.
  - *Strongly recommended* to use spaces to avoid problems

Block Examples

```
if x < 2:
     print "x is less than 2"

for num in [1, 3, 5]:
     if num < 3:
          print "Less than 3"
     else:
          print "Greater than 3"
```

University of Waterloo

Conditionals

```
if expression:
    block
elif expression:
    block
else:
    block
```

University of Waterloo

Conditionals

- Boolean tests are very C-like
  - Non-zero values are “true”
  - 0, None, empty sequences (including empty strings) are “false”
  - None is similar to “null” in Java
- `not`, `and`, `or` for Boolean logic
- Parentheses not needed for tests
- Examples
  - `if not []: # True: Empty list negated is true`
  - `if 13:     # True: (non-zero value)`
  - `if 'A string': # True: A “non-None” object`

University of Waterloo

Looping

```
for var in sequence:
     block
else: # optional
     block

while expression:
     block
else: # optional
     block
```

University of Waterloo

Looping

- *Sequence* in `for` loops is any list or tuple
  - `range()` function can be used to iterate a set number of times
  - `for x in range(10): # Produces a list from 0-9`
  - `for x in range(5, 10): # [5, 6, 7, 8, 9]`
  - `for x in range(5, 10, 2): # [5, 7, 9]`
- Can use `break` and `continue` as in Java
- `pass` can be used to signify no-ops in blocks
- (optional) `else` block executed for normal loop termination (i.e., not as a result of `break` being called)

University of Waterloo

Functions

```
def functionName(args):
    block
```

- Functions can have optional arguments
  - Can use argument names when calling function
- Examples
  - ```
    def foo():
        print "Foo"
    ```
  - ```
    def addIt(a, b, c=5):
        return a + b + c
    ```
  - `addIt(3,5) # returns 13`
  - `addIt(1, 2, 3) # returns 6`

University of Waterloo

Classes

```
class MyClass:
  def __init__(self, args):
     self.myField = 5
     block
  def foo(self):
     block

class SubClass(MyClass):
  def __init__(self, args):
     MyClass.__init__(self, args)
     block
```

Classes

- `self` parameter is equivalent to "`this`" in Java
  - Must be included for each method definition
  - Is not a keyword, but `self` is used by convention
- `__init__` method is the constructor for a Python/ Jython class
- Jython class can inherit from Java classes as if they were Jython classes
  - Can call constructor using `ParentClass.__init__(self, [args])` calling convention

University of Waterloo

Modules

- Any python file can be a module
  - Just needs to be imported
- Modules are also objects
- Java packages also modules in Jython
- To use module, use `import`
  - `import javax.swing # b = javax.swing.JButton()`
  - `import javax.swing as swing`
    `# b = swing.JButton()`
  - `from javax.swing import *`
    `# b = JButton()`

University of Waterloo

Jython/Java Integration

- Jython offers a number of conveniences for tight integration with Java
  - get/setFoo() -> attributes
  - Assignment shortcuts
  - Attributes settable in constructors
  - Listeners -> attributes
    - Listeners are changed into attributes in Jython, become *much more* like delegates in C#

Get/Set Shortcuts

- Any getFoo() methods in *Java classes* become readable attributes of object
  - `b = JButton()`
  - `print b.getText()`
  - `print b.text`
- Any setFoo() methods in *Java classes* become writable attributes of object
  - `b.text = "OK"`
- Jython automatically inspects objects at runtime to achieve this behavior
  - Works on any class with getFoo()/setFoo() conventions
- More "Pythonic" to access values directly

University of Waterloo

Assignment Shortcuts

- Java class attributes that take an object with a constructor of multiple arguments can be reduced to a tuple
- Example

```
f = swing.JFrame()

# Long way
newSize = awt.Dimension(400, 400)
f.size = newSize

# Short way
f.size = (400, 400)
```

University of Waterloo

Setting Attributes in Constructors

- A *Java object's* attributes can be set in constructor even if the class does not define arguments for this
- To do this, must use the named argument convention
- Example

```
f = swing.JFrame(size=(400,400),
  layout=awt.FlowLayout(), visible=1)
```

University of Waterloo

Listeners as Attributes

- Jython looks for `addFooListener()` methods to turn into settable attributes
- Example
  - JButton defines `addActionListener(ActionListener listener)` method
  - `ActionListener` defines a method, `actionPerformed`
  - Any JButton object will have an `actionPerformed` attribute that can be assigned a function

University of Waterloo

Listener Example

```
def foo(event):
      print “Foo”
def bar(event):
      print “Bar”
button = swing.JButton() # Assuming we’ve done
                         # import javax.swing as swing
button.actionPerformed = foo # set one listener
button.actionPerformed.append(bar) # add a second
button.actionPerformed = foo # resets it
                             # to be only foo
```

University of Waterloo

Embedding Jython

- Basic steps
  - Get jython installation
  - Add jython.jar to Java's class path so it can find it when compiling and executing your code
  - Create a new PythonInterpreter object
  - Set any Java variables in interpreter using the interpreter's set(String, Object) method
  - Use interpreter's exec(String) or eval(String) methods to execute / evaluate any Jython code
  - Can also retrieve values using interpreter's get method
  - Call PyObject's __tojava__(String, Class) method to cast returned Jython objects to Java equivalents (if possible)

University of Waterloo

Embedding Jython: Example

```
import org.python.core.PyObject;
import org.python.util.PythonInterpreter;
/* ... Some class here */
public static main(String[] args) {
 PythonInterpreter interpreter = new
PythonInterpreter();
 interpreter.set("a", new Integer(1));
 interpreter.set("b", new Integer(2));
 PyObject result = interpreter.eval("a+b");
 Integer javaResult =
         (Integer)result.__tojava__(Integer.class);
 System.out.println("Result: " + javaResult);
}
```

University of Waterloo

Embedding Jython

- execfile(String) method of PythonInterpreter also extremely handy
  - Reads in, then executes a script file

Compiling to .class Files

- Use `jythonc`
- Will need to have jython.jar on classpath when executing resultant class files