Introduction to Object Oriented / Functional Programming 5

Object-Oriented Software Development

COMP4001
CSE UNSW Sydney

Lecturer: John Potter
GUI Programming with Swing

- Scala wrappers for standard Swing library
- Hides much of the complexity of the Java libraries
- Simple GUI applications can be easy to develop

- Examples from Programming in Scala Chpt 32 (Chpt 34 in 2nd edition)
- Slightly modified downloadable .jar via COMP4001 lecture notes directory
Example FirstSwingApp

Imports scala.swing._ via implicit import of scala._

```scala
import swing._

object FirstSwingApp extends SimpleGUIApplication {
  def top = new MainFrame {
    title = "First Swing App"
    contents = new Button {
      text = "Click me"
    }
  }
}
```

Provides main method with Swing setup, then calls top

Top-level GUI component

Setter methods for title and contents

GUI container property contents: Seq[Component]. Frame contents have just one component with setter contents_=(Component c)
Example SecondSwingApp

import swing._

object SecondSwingApp extends SimpleGUIApplication {
  def top = new MainFrame {
    title = "Second Swing App"
    val button = new Button {
      text = "Click me"
    }
    val label = new Label {
      text = "No button clicks registered"
    }
    contents = new BoxPanel(Orientation.Vertical) {
      contents += button
      contents += label
      border = Swing.EmptyBorder(30, 30, 10, 30)
    }
  }
}

Panel for stacking of contents
Panel contents can be added to
Event Handling

- Associate events with actions
- Components may publish events
  - And subscribe to publishers of those event
  - Publisher = event source
  - Subscriber = event listener
- Subscribe to an event source via
  - `listenTo(source)`

- Handling events is neater in Scala than in Java
  - Events are real objects in Scala
    - `case class ButtonPressed(button : Button)`
  - Handlers have a reactions property
    - Event cases with associated actions are added to the reactions
    - Similar to the react method for actors
Example ReactiveSwingApp

import swing._
import event._

object ReactiveSwingApp extends SimpleGUIApplication {
  def top = new MainFrame {
    title = "Reactive Swing App"
    val button = new Button {
      text = "Click me"
    }
    val label = new Label {
      text = "No button clicks registered"
    }
    contents = new BoxPanel(Orientation.Vertical) {
      contents += button
      contents += label
      border = Swing.EmptyBorder(30, 30, 10, 30)
    }
  }
}
Example ReactiveSwingApp ...

```javascript
listenTo(button)
var nClicks = 0
reactions += {
    case ButtonClicked(b) =>
        nClicks += 1
        label.text = "Number of button clicks: " + nClicks
}
```
Example TempConverter

```scala
import swing._
import event._

object TempConverter extends SimpleGUIApplication {
  def top = new MainFrame {
    title = "Celsius/Fahrenheit Converter"
    object celsius extends TextField { columns = 5 }
    object fahrenheit extends TextField { columns = 5 }
    contents = new FlowPanel {
      contents += celsius
      contents += new Label(" Celsius = ")
      contents += fahrenheit
      contents += new Label(" Fahrenheit")
      border = Swing.EmptyBorder(15, 10, 10)
    }
  }
}
```

- Import scala by default. Subsequent imports are nested package
- Editable fields
- Panel with Flow Layout
Example TempConverter ...

```scala
listenTo(celsius, fahrenheit)
reactions += {
  case EditDone(`fahrenheit`) =>
    val f = fahrenheit.text.toInt
    val c = (f - 32) * 5 / 9
    celsius.text = c.toString
  case EditDone(`celsius`) =>
    val c = celsius.text.toInt
    val f = c * 9 / 5 + 32
    fahrenheit.text = f.toString
}
```

- Top frame subscribes to TextFields
- Event when TextField edit is done
- Pattern matching on celsius object. Backticks are used to escape usual introduction of a pattern variable