E Fundamentals… with Donuts

Marc Stiegler
Visiting Scholar, HP

Introduction

- Questions?
- Installing E?
- Tragedy At the Coffee Pot
- Near-Tragedy with Gray Goo
- Promises, ToDo Lists, Immediate Calls, Eventual Sends
- NonBlocking Dialogs
- Overwrite Answer Example

Tell Me If I Go
Too Fast Or
Too Slow!

Eventual Sends, Promises, When-Catches


```
? println <- run("hello")
# value: <Promise>

hello
?

? def print2() {
>     println <- ("hello1")
>     println("hello2")
> }
# value: <print2>

? print2()
hello2
hello1
?
```

```
? def showWhenCatch() {
>     def printVow := println <-("Hello")
>     when (printVow) -> done(printed) {
>         println("Beyond Hello")
>     } catch prob {println ("dead: " + prob)}
> }
# value: <showWhenCatch>

? showWhenCatch()
Hello
Beyond Hello
?
? def num
# value: <Resolver>

? num
# value: <Promise>

? bind num := 3
# value: 3
```

Promise Resolvers

```
? def vowDouble(numVow) {
>     def [double,res] := Ref.promise()
>     when (numVow) -> done(num) {
>         res.resolve(2*num)
>     } catch prob {res.smash(prob)}
>     return double
> }
# value: <vowDouble>

? def [val, resolver] := Ref.promise()
# value: [<Promise>, <Resolver>]

? def doubleVal := vowDouble(val)
# value: <Promise>

? resolver.resolve(3)
? val
# value: 3

? doubleVal
# value: 6
```

```
? def vowDouble(numVow) {
>     return numVow <- multiply(2)
> }
# value: <vowDouble>

? def val
# value: <Resolver>

? def doubleVal := vowDouble(val)
# value: <Promise>

? bind val := 3
# value: 3

? doubleVal
# value: 6

val <- multiply(2) <- multiply(3)
```

makeDialogVow

```
def makeDialogVowAuthor (disposablesKit, makeFrame, traceline) :near  {
    #…window manipulation setup code
    def makeDialogVow (title, labelTree, optDefaultValue, buttonNames) :any {
        def [finishedDialogVow, resolver] := Ref.promise()
        def dialog
        #…. lots of window manipulation code
        #make the window
        def myWin := makeFrame()
        def disposeListener {
            to widgetDisposed(event) :void {
                resolver.resolve(dialog)
            }
        }
        #….more manipulation

        #create the fulfilled dialog object
            bind dialog {
                to optEnteredText() :any {
                    var answer := null
                    if (myClickedButton != null &&
                         optDefaultValue != null) {
                        answer := userEnteredText
                    }
                    answer
                }
                to optClickedButton() :any  {myClickedButton}
            }
            finishedDialogVow
    }
}
```

Overwrite Dialog Part 1

```
def alwaysOverwrite(fileName) :any {true}

def neverOverwrite(fileName) :any {false}

def askOverwrite(fileName) :any {
    def result
    def answerDialog := makeDialogVow("Overwrite?", "Overwrite file “
          + fileName + "?", null, ["Yes","No"])
    when (answerDialog) -> done(answer) {
        bind result := answer.getClickedButton() == "Yes"
    } catch err {}
    return result
}
```

Overwrite Answerer Part 2

```
def makeOverwriteAnswerer() {
    var answerFunctionSelectionAlreadyStarted := false
    def answerFunction
    def determineOverwritePlan() :void {
        answerFunctionSelectionAlreadyStarted := true
        def planDialogVow := makeDialogVow ("OverwritePolicy", "When should
                                    files be overwritten?", null, ["Always","Never","After Confirmation"])
        when (planDialogVow) -> done(planDialog) :void {
            def button := planDialog.getClickedButton()
            if (button == "Always") {
                bind answerFunction := alwaysOverwrite
            }else if (button == "Never") {
                bind answerFunction := neverOverwrite
            }else {bind answerFunction := askOverwrite}
        } catch err {}
    }
    def overwriteAnswerer {
        to vowOverwriteAnswer(fileName) :any {
            if (! answerFunctionSelectionAlreadyStarted) {determineOverwritePlan()}
            return answerFunction <- run(fileName)
        }
    }
    return overwriteAnswerer
}
```

OverWrite Answerer Part 3

```
def overwriteAnswerer := makeOverwriteAnswerer()

For each file in files {
    def newFile := newDir[file.getName()]
    if (newFile.exists()) {
        when (overwriteAnswerer <- vowOverwriteAnswer(file.getName()) ->
                                        done(shouldOverwrite) {
            if (shouldOverwrite) {newFile.setBytes(file.getBytes())}
        } catch prob {throw (prob)}
    } else {newFile.setBytes(file.getBytes())}
}
```

Why not immediate call?

Why not eventual send?

Simple guards

```
def a :int := 3
var b :0..4 := 2
def adder(x :int, y:int) :int {return x + y}
adder(a :int, b:Data)
def c :vow[int]
boolean
float64
String
rcvr
notNull
nullOk
near
any
void
pbc
Data
<import:java.IO.File>.asType()
```

```
interface Point {
    to getX()
    to getY() :int
}
def makePoint(x,y) :Point {
    def point implements Point {
        to getX() {return x}
        to getY() {return y}
    }
    return point
}
def origin :Point := makePoint(0,0)


interface Point guards PointStamp {
    # method list
}
# now def point implements PointStamp
```