Notes on tutorial #4 (for week 7)
(Feb 20, 2004)

26th February 2004

Q1 (RESEARCH): The MVC (Model View Controller) programming pattern (?) is an old software pattern, first described over 25 years ago, and first implemented in the language **`smalltalk-80`**. This language had certain properties that encouraged the development of this sort of software pattern.

(a) what is the specific property of **`smalltalk-80`** that "*encouraged the development of this sort of software pattern*"?

Answer: *It is an actor language, with an operational model that includes the idea of separate concurrent communicating objects. This maps closely to the MVC architecture which proposes three separate concurrent communicating objects.*

(b) Swing uses a *modified* MVC architecture. What is the modification?

Answer: *According to the Java/Swing people "The first Swing prototype followed a traditional MVC separation in which each component had a separate model object and delegated its look-and-feel implementation to separate view and controller objects.*
We quickly discovered that this split didn't work well in practical terms because the view and controller parts of a component required a tight coupling (for example, it was very difficult to write a generic controller that didn't know specifics about the view). So we collapsed these two entities into a single UI (user-interface) object".

Q2: Write Tcl code that creates a new Tcl command structure - the **recurse x y until lessthan z** command. This command should execute the function **x** passing it a single parameter. The return value is tested to see if it is less than **z**. If **yes**, then the command returns the number of recursions done. If **no**, then it recurses again, using the previous return value as the new **y** value. for example - if we had a function **`half`** which returns its argument divided by two, then **recurse half 17 until lessthan 1** should return **5**.

Answer: *Something like the following:*

CODE LISTING **tut4q2** Page 1/1

```
proc recurse {x y until lessthan z} {
    set count 1
    set loop 1
    while {$loop==1} {
       set val [$x $y]
       if {$val<$z} {set loop 0} {set y $val; incr count}
    }
    return $count
}
proc half {x} {
   return [expr round($x/2)]
}

recurse half 17 until lessthan 1
```

Q3: Modify the Tk application **`tkpaint`** (found at **`~cs3283/ftp/tkpaint`**) to *save* and *restore* paintings.

Answer: *Something like the following:*

CODE LISTING **tut4q3** Page 1/1

```
======= Replace this  ====>

button .exitbutton   -bitmap @exit.xbm      -command {exitandsave}

======= And add this  ====>

proc exitandsave {} {
   global nodes
   set f [open dataq3 w]
   foreach x [.net find all] {
      puts $f ".net create [.net type $x] [.net coords $x] \
                           -tag node$x -fill [.net itemcget $x -fill]"
      puts $f ".net bind node$x <Enter> \".net itemconfigure node$x -width 5\""
      puts $f ".net bind node$x <Leave> \".net itemconfigure node$x -width 1\""
      puts $f ".net bind node$x <ButtonPress-3> \"beginmove %x %y\""
      puts $f ".net bind node$x <B3-Motion> \"domove node$x %x %y\""
      puts $f ".net bind node$x <ButtonPress-2> \"docurrentop node$x %x %y\""
   }
   puts $f "set nodes $nodes"
   close $f
   exit
}
```

Q4: Modify the Tk application **`tkpaint`** (found at **`~cs3283/ftp/tkpaint`**) to use a menu system instead of the buttons

Answer: *Something like this:*

CODE LISTING **tut4q4** Page 1/1

```
======== Replace the buttons with this ===>

menubutton .m.f -text File -underline 0 -menu .m.f.menu
menu .m.f.menu -tearoff 0
.m.f.menu add command -label "Load..." -command "source dataq3"
.m.f.menu add command -label "Save..." -command "saveit"
.m.f.menu add command -label "Print..." -command "printit"
.m.f.menu add separator
.m.f.menu add command -label Quit -command exit
pack .m.f  -side left

menubutton .m.e -text Edit -underline 0 -menu .m.e.menu
menu .m.e.menu -tearoff 1
.m.e.menu add command -label "Next Colour..." -command "setanewcolour"
.m.e.menu add separator
.m.e.menu add radiobutton -label Grow... -variable currentop -value grow
.m.e.menu add radiobutton -label Shrink... -variable currentop -value shrink
.m.e.menu add separator
.m.e.menu add radiobutton -label Rect... -variable currenttool -value rectangle
.m.e.menu add radiobutton -label Oval... -variable currenttool -value oval
pack .m.e  -side left

menubutton .m.help -text Help -underline 0 -menu .m.help.menu
menu .m.help.menu -tearoff 0
.m.help.menu add command -label "About ThisApp..." -command "aboutcommand"
pack .m.help  -side right

proc aboutcommand {} {
    tk_dialog .win {About this program} "Simple paint application"  {} 0 OK
}
```