

Programming Using Tcl / Tk
Seree Chinodom
seree@buu.ac.th
<http://lecture.compsci.buu.ac.th/TclRk>

Hi everyone once again ,welcome to this PS course, will be continuing our lecture on T k ,we finished talking about Tcl , I hope you have all the things they will understood now, now we will talk about TK actually ,we are actually discussing TK for the last two lectures , today we will be continuing that, but before we go into like today s , topics itself I am going to give you a brief overview of what we talked about .

Some things may not be very clear in the last couple of lectures ,but I am going to see how I can clarify those points .

(Refer Slide Time: 00:49)

Tk Recap

- ▲ **Tk fundamentals**
- ▲ **GUI fundamentals**
 - Event – Any operation or movement using mouse or keypad.
 - Move – Moving mouse from one location to another
 - Key press – Depressing mouse keys or a key in the keyboard
- ▲ **Window**
 - Any specific area in the terminal (usually rectangle) bounded by a frame.
- ▲ **Widget – a part within a window that does a specific task (or command).**

So let us do a recap ,so mainly like we talked about the TK fundamentals ,so the GUI essentially, GUI fundamentals actually basically , this is a concept of event ,the event is the any operation or

movement using mouse or keys or a keypad, nowadays even like the finger moments on touch screen, things like that are also like you can call it, as an event but traditionally .

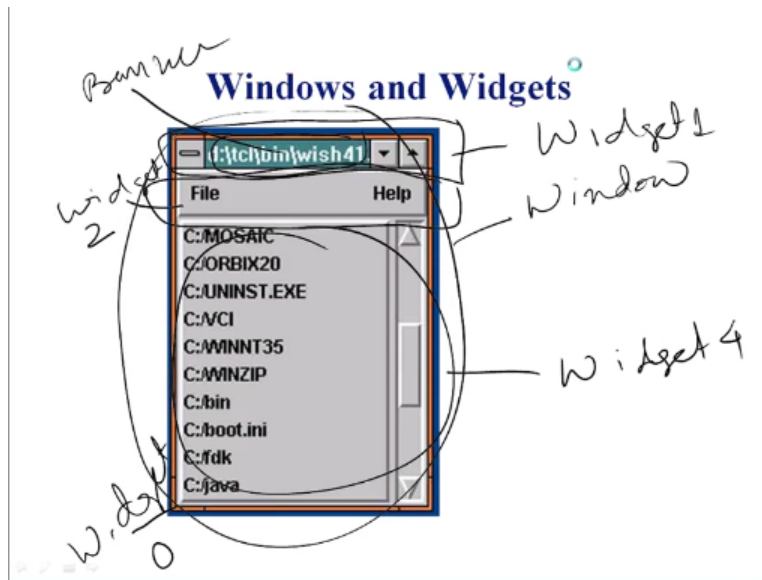
What we meant was one of all the mouse moves, like I do not know you can see that , there is a rock ,moving forward or some kind of keypad press, is an event on a window and then the move, itself ,you can say that is moving the mouse, from one point to another point, that it constitutes a move and then the key press is basically, the press any key either the mouse keys or the particular one like the right click, left click or middle click or a key on the keyboard .

You can also add additional stuff, like the role , rolling of the mouse, things like that, so the main thing that we will be talking about , in case ,how do we develop applications, that can capture these GUI interface and work with them to achieve ,whatever that you want , actually want or execute a particular command ,so the other concept is also regarding the windows, the window is any specific area of the terminal .

Usually it is a rectangle and it is bounded by a frame, so we will talk about it small thing and then widget is the other concept which is part of the window, that does a specific task basically it has a shape and it does a specific task ,so these are the two points that you want to understand basically regarding the disk, in most of the text books and things, like that we use window and widget as synonyms. But in this class I want to separate it out.

And basically window is a collection of widget and how we will define ,we will see this again and then most of the things ,that we will be bringing about ,this is regarding the widget ,how do we enable the widgets and how do we work with widgets ,how do we assign commands to widgets ,how do we create with these things, like that is all covered under this TKR toolkit.

(Refer Slide Time: 03:39)



So here is a brief look at the concept ,so you can think of this entire region, being a window okay so the window includes actually, if you look at it ,there are four widgets and there is a frame which is probably ,the widget number 1, now actually the widget number 0 and then you have the widget number 3 and then you have this portion of this window, which is itself is a widget ,that we call it as widget number 1,we will see like I mean .

We have already seen like all the information, but anyway and then there is another piece ,here which is also embedded here , this is widget number 2 and then the third piece ,which is basically although directly means this is widget number 3, in fact you can also think of this structure here ,as another widget, we get widget 4 ,so all together ,now you get 5 widgets, so remember this concept ,we will come back to this essentially I mean , so and each one has a different function as I mentioned .

So for example this one, is what is called the banner widget ,we just split the banner, the frame will get essentially like outline popping and then this is the menu widget ,which contains like what are the menus, is different venues is file and help and then the main display widget ,which is this 1+ there is a menu bar widget ,which is essentially make sure that this scroll bar ,can move up and down ,okay.

(Refer Slide Time: 05:30)

Widgets and Class

- ▲ Widgets are objects, instances of classes that represent buttons, frames, and so on. So the first thing you'll need to do is identify the specific class of the widget you'd like to instantiate.
- ▲ Classes are important to determine precedence of multiple bindings for a single event.
- ▲ BTW...
 - Multiple bindings are legal and will result in multiple outputs.
 - Example – Cursor move can cause a window to become active as well as some command starts running.
- ▲ **Widget command is deleted when widget is destroyed.**
- ▲ **Principle: all state should be readable, modifiable, anytime.**



So that is the basic concept of windows and widgets, so in the sense ,widgets are objects instances of classes , so now we are introducing a new term of classes ,that represent buttons ,frames and so on, so if you look at here actually this could be ,we can make this as a button and we know already that, there is the frame okay ,so the first thing you will need to identify, in the need to do, is to identify the specific class of widgets, widgets that you would like to instantiate .

So for every class of widget ,we have an equivalent command ,which we will talk about it a little later in fact ,we covered this in lecture 2 or lecture number 1 run rather, so classes are important ,because they determine the precedence, if there are multiple bindings to a single image ,so what are this binding essentially, the bindings are essentially like I mean, once you have an event, you can track that event and then you can make the widgets perform certain functions based on the ones even could be like a key press, moving .

Whatever it is basically like ,you can make it perform certain things and to determine like I mean how these go , there are multiple widgets involved, the class actually can be one of them, so we will see like whether ,we want do and by the way the multiple bindings are legal and will result in multiple outputs ,which is also pretty much okay, for example here there is a cursor move, from one place to another place and cause the windows to become active .

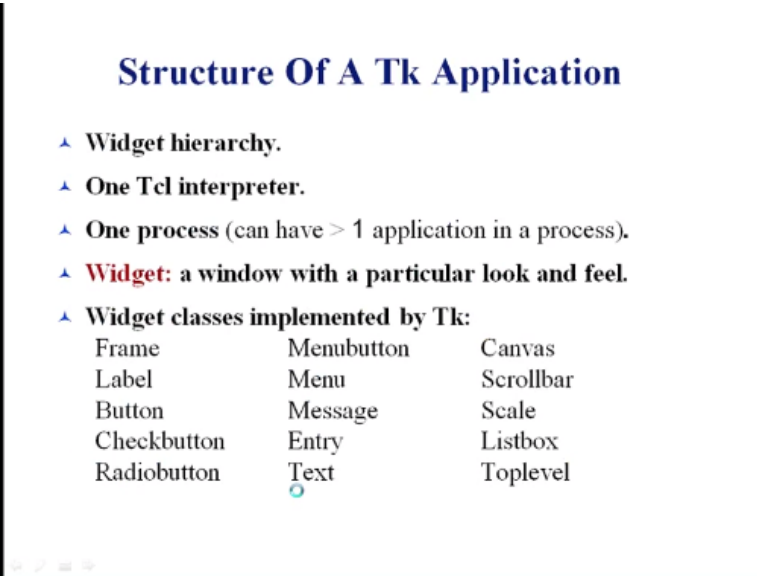
As well as some command ,that may start running within that window, so both of them are simultaneously and having both of them activated and are running simultaneously ,which

perfectly , okay sometimes ,we want to be specific ,so this is again something that we covered in the last lecture and then couple of other points to note, is the widget command itself is deleted when the widget is destroyed .

So any command associated with the widgets ,are like completely gone, so where is this command ,what is this command, that will also be covered ,but I will talk about it in the next class, and then one thing to note is , the state of the element of the widget ,or the state of particular window, it should be readable ,modifiable and this is the principle, that we operate with within this framework.

So again like what is escape state ,it is basically once you get an event, how do you respond to it and that is the state and then that state is essentially, what we will be talking about.

(Refer Slide Time: 08:34)



Structure Of A Tk Application

- ▲ **Widget hierarchy.**
- ▲ **One Tcl interpreter.**
- ▲ **One process** (can have > 1 application in a process).
- ▲ **Widget: a window with a particular look and feel.**
- ▲ **Widget classes implemented by Tk:**

Frame	Menubutton	Canvas
Label	Menu	Scrollbar
Button	Message	Scale
Checkbutton	Entry	Listbox
Radiobutton	Text	Toplevel

So like the structure of the TK application, we talked about this basically, widget hierarchy which I will cover once again and then there is one tcl interpreter, which basically interprets all the techie coding underneath ,these widgets and this is just one process ,you can have more than one application in that process ,but it is only one process, in per ticket and then we already talked about the widget basically ,again widget and windows are synonyms kind of most of the cases .

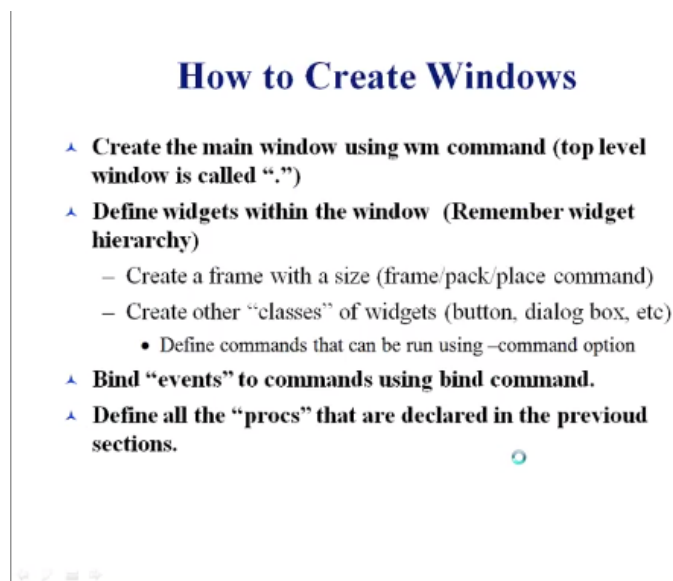
We will say like I mean basically it is even though, it says particular look and feel, it is basically defined very strictly as ,it has the widget has a physical shape and it has a single purpose okay I

mean this is the thing, that I talked about here shape and a task okay, I just add this in physical shape and then a singular purpose ,so widget classes implemented are brief , once I will also like cover some of the next slides .

But then think of this frame ,label , button ,check button, radio button ,menu button ,even in button has like ,so many different classes and we have menu as another one ,message, entry, text ,canvas, scroll bar ,scale ,list box and top level, so we will be talking about some of these things basically in the text ,we will talk about canvas ,and also this we already talked about frame and buttons, other things like menu button and things like that specialized items of these buttons themselves.

So you can actually look at it, up on those things and then we will also probably cover the top level, so let us see how do we create a Windows or create widget or basically an application.

(Refer Slide Time: 11:02)



How to Create Windows

- ▲ **Create the main window using `wm` command (top level window is called “.”)**
- ▲ **Define widgets within the window (Remember widget hierarchy)**
 - Create a frame with a size (frame/pack/place command)
 - Create other “classes” of widgets (button, dialog box, etc)
 - Define commands that can be run using `–command` option
- ▲ **Bind “events” to commands using `bind` command.**
- ▲ **Define all the “procs” that are declared in the previous sections.**

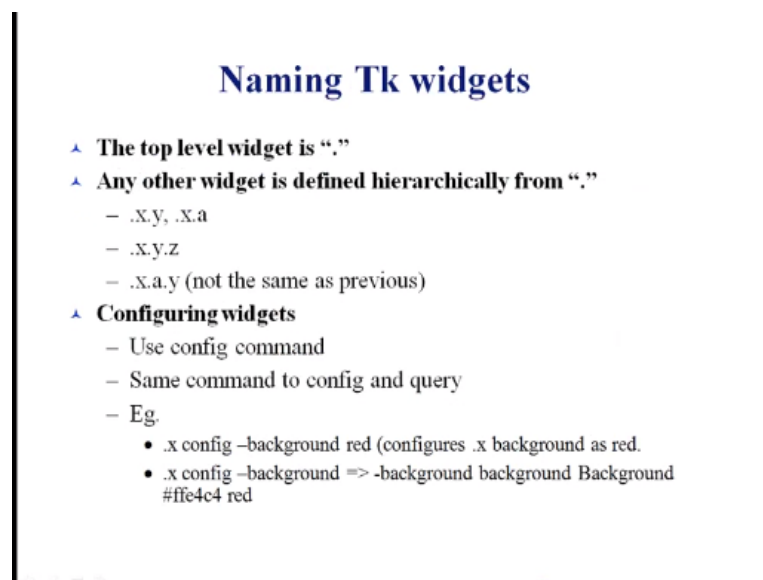
So we need to create the main window using the `wm` command and then the top level actually ,like there is also a top level ,the command called top level to create pop , then we define the widgets between that window, again for this I want you to remember the widget pattern , where we started with the dot and then multiple .txt ,your frame things like that basically ,this is a whole bunch of these hierarchy that we talked about.

So remember that basically and then that is how ,we have defined and the way that we define is ,we start defining widgets one at a time ,so here saying that frame is another one basically with the size and then we can use pack ,place these are all like command ,so we can do that and then we can also create other classes of widgets ,whether within the hierarchy or at the top level whichever one based on the application, needs.

We create these other widgets with a button dialog box and then we define the commands that can be run, on these widgets using the - command option, so this is something, w talked about and then we also bind events, to commands using the bind command ,so the events are again like those key press and things like that, so we can call this button run , and then say run gets activated, only if the key is pressed on top of the run .

So that is the even that we track and then run particular command and then that is denoted by this command and then , once we do this and we create this the graphical picture ,as to how this application looks like and then we go ahead and do the depend all the box within, whatever we are coming up with inside this graphical picture, so all of them we have to define which are declared in all the previous sections .

(Refer Slide Time: 13:25)



Naming Tk widgets

- ▲ The top level widget is “.”
- ▲ Any other widget is defined hierarchically from “.”
 - .X.y, .X.a
 - .x.y.z
 - .x.a.y (not the same as previous)
- ▲ Configuring widgets
 - Use config command
 - Same command to config and query
 - Eg.
 - .x config -background red (configures .x background as red.
 - .x config -background => -background background Background #ffe4c4 red

So with that actually like, we saw some more examples but let us, see like I mean ,so the top level widget it is again the “.” and any other widget is defined as hierarchically from the“.”, so . and then x, a under x you have y and then y you have actually under x you have V and then other

a you have y which is not the same as example x.a.y here x.a.z, so particular widget and then how do we configure widgets, this is another thing, that we saw.

This is using the config command or configure command ,so the beauty of this command is it is the same command ,that also you can use to query the particular configuration ,for example here as you know like I am in window .x and we give this config command and then say like background make it red ,so now the command goes and actually like configures the particular window with the background color as red.

So now if we omit this red and then just give x config - background that ,returns this string and you can say that basically like this is the background and then that is it - background and then actually the background color is determined by #ffe4c4 x or red in and then the actual color red, so gives you like all these information ,one thing ,we notice you know I can query these but you know these things, are this noise I do not want to do any of these .

I only want to know, what is the background color and I want to know this red ,so how do we do it.

(Refer Slide Time: 15:38)

Querying config (cont.)

- ⚡ Using config command to query is cumbersome.
 - Lots of unwanted data
- ⚡ New function for query
 - Cget
- ⚡ `.x cget -background =>red`
- ⚡ Change existing configuration of widgets using config command.



So here there are ,there is one additional function that got defined which is actually a synonym for config get or C get and C get basically ,when you when you give that command with the .x and what is option which is background ,you did it again, just red ,so it is a very handy command policy get, so keep that in mind and then we can actually in fact, if the window is already configured ,we can actually change the existing configuration of the widgets, using config command .

(Refer Slide Time: 16:25)

Popular “Classes”

- ▲ These are created by commands. But they are the defined “class” of objects
 - Button
 - Radiobutton
 - Menu
 - Canvas
 - Frame
 - Text
 - Listbox

So again some of the popular classes of commands ,also the objects essentially in the windows button, radio button, menu ,canvas, frame, text, list box .

(Refer Slide Time: 16:38)

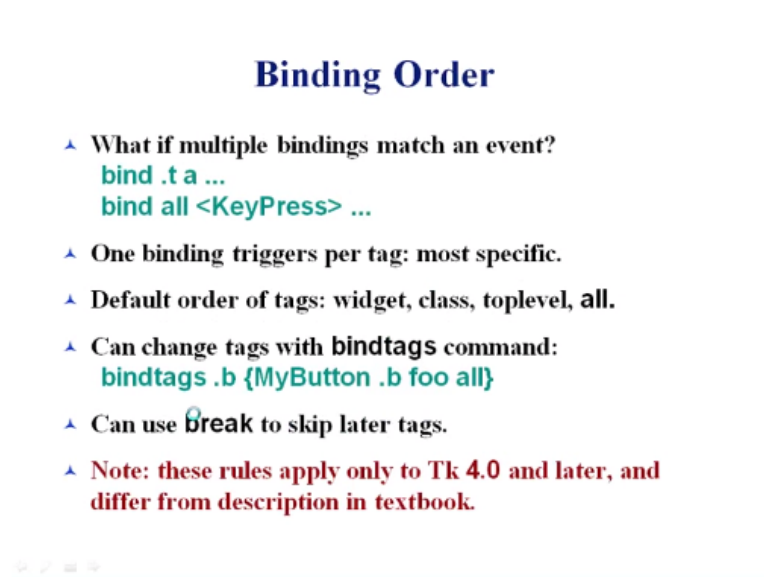
Tags

- ▲ Tags are objects associated with the text widget. Each tag is referred to via a name chosen by the programmer. Each tag can have a number of different configuration options; these are things like fonts, colors, etc. that will be used to format text. Though tags are objects having state, they don't need to be explicitly created; they'll be automatically created the first time the tag name is used.
- ▲ Examples
 - .text tag configure highlightline -background yellow -font "helvetica 14 bold" -relief raised

Now I also talked about tags essentially , the tags are objects ,associated with the text widget essentially and then the each tag is referred to, via name chosen by the program ,each tag can have a number of different configuration options ,the things like fonts, colors, etc that are that will be used to format text can be put in as a tag , even though like I mean the tags ,themselves are objects having a state , they do not need to be explicitly created, they will be automatically created the first time when the tag name is used.

Okay, so here .text for example and the tag, configure highlight line -background yellow- font and really raised basically does that to the text window , as defined in this comment .

(Refer Slide Time: 16:38)



Binding Order

- ▲ What if multiple bindings match an event?
`bind .t a ...`
`bind all <KeyPress> ...`
- ▲ One binding triggers per tag: most specific.
- ▲ Default order of tags: widget, class, toplevel, all.
- ▲ Can change tags with **bindtags** command:
`bindtags .b {MyButton .b foo all}`
- ▲ Can use **break** to skip later tags.
- ▲ **Note: these rules apply only to Tk 4.0 and later, and differ from description in textbook.**

Then we also talked about this binding order essentially, this is what if multiple bindings, naturally as I mentioned everything goes in parallel , but we can make it most specific which is one binding figures ,per tag and it follows the default order of the tag, this is class, top-level, all , then but we can change the tags ,with the bind tags command, so again you can also like bind a particular tag to a particular window .

So one thing ,we notice here is just the general tag is opposite with text, focusing this highlights and combo but in general any tag ,can be bound to a particular window and reclassify that window introducing okay, so here like I mean we can say like, bind tags .b my button and then we can also say that it is bound to this particular actually like I mean, so this is who all ,so that is what ,we have doing the bind tags for and we can you can also like I mean.

If you want to skip a particular tag ,we can just say break and then that can go to skip some later tags as well, so I am going to refer to the textbook ,for the exact definition of the bind tags essentially ,so widget class name basically like a button or whatever it is and then, so that is where it is bound to okay .

(Refer Slide Time: 20:32)

TkCal Example

```
#####
# initialize global variables
set months {January February March April May June July \
            August September October November December}

# figure out the current month and year
# date returns a string like Sun Sep 22 22:26:10 PDT 1996
set d [exec date]
set month [lindex $d 1]
set year [lindex $d 5]

# expand the abbreviation that date gave us
foreach m $months {
    if {[string match ${month}* $m] == 1}{
        set month $m
        break
    }
}
```



So that is what we covered in the last lecture today ,we will talk about some new topics in Tk class basically and then, we will use some examples to illustrate, how TK works for these ,so here is an example for calendar it is called Tk Cal okay, so we first initialize the global variables before this set months, once then we can execute this command, all the date, the date actually returns ,so whatever this particular string which is like the day, the month, the date and then the time, the time zone and then the year .

So we take the month as the index 1 of the string ,can anyone tell me why this is 1, actually this string starts from 0, so this is 1 and then the index for index number 5,denote the year, so again we count here ,from here this the exact date is actually 2, this entire thing is 3 ,this is 4 and this is 5, so now we get like month and the year, now we can expand the abbreviation that date gave us basically 4.

So now we can say that in this loop, we can set ,we can find out like which month , this so basically like run it through this and then essentially like seeing ,whether there is a match for any particular month that will come here, so the way that we are doing it is essentially like a string match ,where we do like \$ month ,because we can go with.

Whatever like this particular month will be read, with all these things and we just do a string match and when we do this multiplication actually, move over this can result in a 0 or 1 based on whether it is matching or not and it is matching .We capture that and then you know, we break the loop.

(Refer Slide Time: 23:44)

TkCal (continued)

```
#####
# Menu action
proc doMenu {m} {
    global month
    set month $m
    .f.mb configure -text $m
}
#####
# run cal and collect output
proc showCalendar {} {
    global month year months
    set m [expr [lsearch $months $month] + 1]
    if {[catch {set cal [exec cal $m $year]} err] == 1}{
        set cal $err
    }
    .t delete 1.0 end
    .t insert 1.0 $cal
}
#####
```

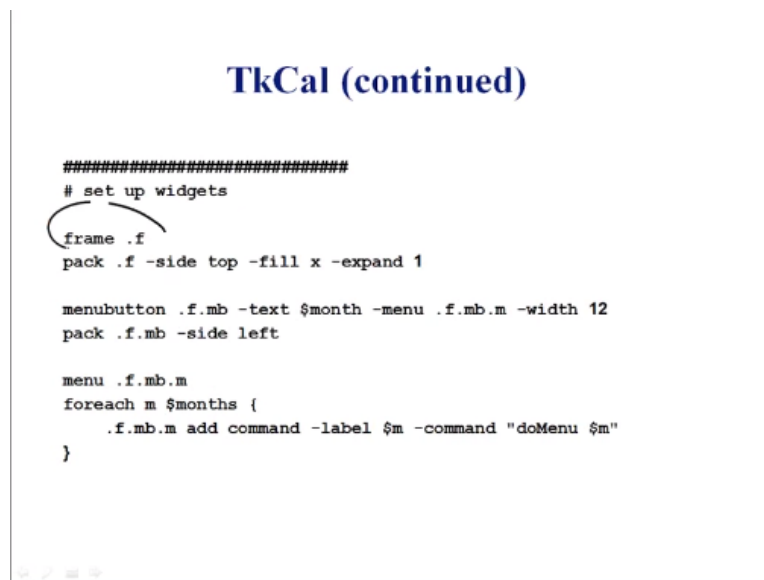
So here ,we define some menus, the menu action basically, it is called a do menu , that is what we are defining ,which is we just set the month to \$ m and this is the global month and then we basically configure these windows ,since we got apparently and then we configure with that particular month , so now this will generate here ,this number and possibly this year, you now we need to generate, the calendar output , so we can run this program per cal and then collect its output .

So ,to show the calendar that is the next , that we are going to write ,which is having the global month ,year and month which is passed from the top level into this, as you know it is already set all , these things ,so now we do a list search of the month with the month itself and then we make it +1 essentially, so this will give you the index +1 ,which is essentially like the actual one representation, so as you know here, this index actually is 0 1 2 3 .

But in reality ,generally it is 1 2 3 4 , so we need to add that from the index, so that we get the actual and then we use the Cal program there we pass the cal this month, the number of the month and then the year and then we basically get that whatever the result is and then here we will just define error conditions also just to make sure that it does not error.

And now for the same window essentially like , it is a text box window, we will talk about that in the next one, we delete and then insert the output of the Cal which is right here, until cal \$ err is here only ,where its index and when is the error ,but otherwise it was set to the calendar from the Cal function.

(Refer Slide Time: 26:49)



Now we set up the top widgets basically, we set up the frame and then we pack ,how to define that and then we have a menu button and then that is the .m b , so here the same thing so that it is configured as 1, but we have another menu item which is defined as a window under the m b, it is called the m and then that has a width of 12 and packed that towards the left ,now how do we populate this one ,that is for essentially like I mean, we say like for each month in m , basically, then we add a command to get the label and put it in .

(Refer Slide Time: 27:46)

TkCal (continued)

```
entry .f.e -textvariable year -width 5
pack .f.e -side left

button .b -text "See Calendar" -command showCalendar
pack .b -side top -pady 3 -fill x

text .t -width 21 -height 8
pack .t -side bottom -fill x

#####
# initial display
showCalendar
```

So now we have look at this picture, we have this solid set ,now we need to get to the date and then we can see the calendar , so now for the year again, we essentially like I mean ,we have a variable per year and we take width 5 and then basically like and then we can enter the year and then we do similar kind of things basically ,we set up a button ,call see calendar, that is this button here , and then that also has a text based and various parameter, so this is a text command ,which is a text window .

And then the pack and then we say the initial display is show calendar okay .so this is one full example of T k Cal calendar or T k calendar, I think like you can try this out first and in your free time and then see how it works and then pretty much then it will become self-explanatory.

(Refer Slide Time: 29:36)

Tcl/Tk Addendum

- ▲ `cget` widget command to query configuration options

```
.w cget -foreground  
=> black
```

Much easier than .w configure -foreground!

- ▲ `env` array containing environment variables

```
cd $env(HOME)  
exec $env(EDITOR)
```



So a few more thing basically, this is something that, we already talked about the `c get` widget essentially that we did they say like `c get- foreground` ,it just gives the black again, as I said earlier it is a very easy and much more easier than doing a `.w configure and – foreground`, this gives all kinds of junk right, you do not need this, now as similar to Perl, the `env` array contains the environment variables, so `env HOME` in the editor segment .

(Refer Slide Time: 30:20)

The Text Widget

▲ Multiline text area widget with 1001 features:

- Editing
 - (Emacs key bindings: ^A, ^E, ^K, ^Y, ^F, ^B, etc)
- Scrolling (manual and from code; X and Y axes)
- Flexible text formatting: multiple fonts and colors
- Text tags: content-sensitive formats and behaviors
- Selection manipulation

▲ Arranges text and embedded widget in lines; lines can wrap but not flow

So you have like multi line area ,text area widget essentially like I mean that the text widget is essentially a multi-line text area widget with number of features ,other than one features, you can do editing and you can use Emacs key bindings ,you can do scrolling from manual and from code ,X and Y-axis and it has a flexible text formatting, multiple points and once the colors are supported, text tags that we talked about earlier and then also selection manipulation.

The selection manipulation is a command ,something what we talked ,about in the last class I will explain some more ,either in this class or later on and it also arranges text and embedded widgets in lines ,the lines can wrap but they cannot flow.

(Refer Slide Time: 31:32)

The Text Widget

▲ Special widget commands:

```
set      get  delete  select  mark
tag      search  see  window  yview
xview  bind
```

▲ Special widget configuration options

```
-state -height  -width
-spacing1  -spacing2 -spacing3
-state -tabs
-insertbackground -selectbackground
```

So what are some special widget commands set, get, delete, select, mark, tag, search, see window, y view, x view, bind these are all special widget commands or text widget and then the configuration options, we have a state, that we can define height and weight, the -spacing 1, -spacing2 and -spacing 3 and then what are the state -tabs and then, we can say like insert background and or select type.

(Refer Slide Time: 32:25)

The Text Widget

▲ Referring to text within a text widget

- index: `line.char` (line starts at 1, char starts at 0)
- position: `@x,y` (character closest to pixel x,y)
- mark: Basically a name for an index.
`.t mark set myplace 6.23`
- tag: Names for sets of ranges of text
`.t tag add mywords 6.23 6.30`
`.t tag add mywords 7.41 7.46`

So the text that is there, inside the text widget has this following things, one is called index which is essentially like `line.char`, so if you have like 5 lines and then say like I mean line starts at 1 and character starts at 0, so this is not that 12345 and then this is 0 through and then now you can actually go exactly to say 53 and then you can read out this, and then we can also do mark, basically a name for an index.

So you can say like I mean `t mark set my place 6.23`, so it will exactly do that and then a tag is also like I mean you can use tag, to find the ranges of text, for example you can say like `t tag add my words 6.23 6.30`, so if you go to the specific location and then perform this function, and then same thing, so this the tag will be very useful.

(Refer Slide Time: 34:16)

The Text Widget

- ▲ Tagged text can be configured

```
.t tag configure mywords -foreground red
```

- ▲ Tagged text can have bindings!

```
.t tag bind mywords <Double-Button-1> {  
    exec TkEdit mywordlist.txt  
}
```

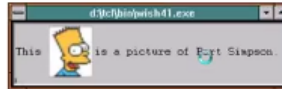
And as I said basically the tag is the one thing, like I mean it is not a permanent command basically the state is save, so a tagged text can also be configured and we can use configured command to configure the tagged text and then the tag text and also have bindings ,we can bind certain key strokes to that particular command and then we can call it to execute ,certain things.

(Refer Slide Time: 35:00)

The Text Widget

- Text widgets can serve as geometry managers for other widgets:

```
image create photo -file bart.gif
label .t.1 -image image1
.t window create 1.5 .t.1
```

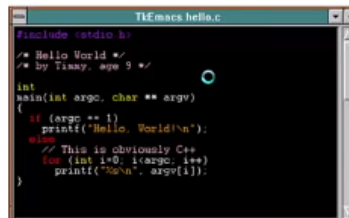


And then usually the text widgets can serve as the geometry managers for other widgets, so here is one quick example ,essentially just using also this new command, so you can image ,create a photo - file called bart.gif , which is this one ,now we say basically like t .l- image image 1 and then t window create 1.5 . t.1 and then that brings up this window .

(Refer Slide Time: 35:46)

A Syntax-Coloring Mini-Editor

- "TkEmacs" does customizable syntax coloring using regular expressions, like Emacs' hilit-19 package
- One text widget
- One scroll widget
- Can read and save files
- About 100 lines of well-commented code



So then now we come to this syntax coloring mini editor , so the TK Emacs does not customize syntax coloring, using the regular expressions like Emacs hilit19 package , if still has one text

widget ,one scroll widget and can read and save files ,there were 100 lines of well commented code available through this program.

(Refer Slide Time: 36:36)

```
TkEmacs (continued)
forAllMatches Routine

proc forAllMatches {w pattern script} {
    # determine number of lines in widget
    scan [$w index end] %d numLines ;# returns e.g. 9.32
    for {set i 1} {$i < $numLines} {incr i} {
        $w mark set last $i.0
        while {[regexp -indices $pattern \
            [$w get last "last lineend" context indices]]} {
            $w mark set first "last + [lindex $indices 0] chars"
            $w mark set last "last + 1 chars + \
                [lindex $indices 1] chars"
            uplevel $script
        }
    }
}
```

So we come to Tk Emacs basically ,like for all machines routine ,so here also like I mean these see basically for all matches ,we give a pattern and the script with the w option, so first of all we determine the number of lines in the widget, which is scan in w to end and if we count the number of lines, it has some value basically and then for every line ,we increment the line and then basically you can do , using regular expression .

We can find what is there and then basically like we generate ,that as a mark set command and then we pass it down.

(Refer Slide Time: 37:56)

TkEmacs (continued)

doColor Routine

```
proc doColor {w} {  
    global syntax_patterns  
    global syntax_colors  
  
    foreach name [array names syntax_patterns] {  
        $w tag delete $name  
        forAllMatches $w $syntax_patterns($name) \  
            "$w tag add $name first last"  
  
        .t tag configure $name \  
        uplevel $script  
    }  
}  
}
```

And here is the do color routine essentially, which is for all the array names and for all the syntax pattern arrays , all name ,we go ahead and delete the tag and then basically ,through the tag first, last okay .

(Refer Slide Time: 38:35)

TkEmacs (continued)

File Routines

```
proc loadFile {w file} {
    $w delete 1.0 end
    set f [open $file]
    while {[eof $f]} {
        \ $w insert end [read $f 1000]
    }
    close $f
}

proc writeFile {w file} {
    set f [open $file "w"]
    puts $f [$w get 1.0 end]
    close $f
}
```

And now we load that file ,uploading the file essentially, we call the file and then we set the file as file open and then we just want to insert, that particular variable and then the right file is essentially like , you open the file and write it.

(Refer Slide Time: 39:12)

TkEmacs (continued)

```
set syntax_patterns(types) \
    {[^A-Za-z0-9]} (char|short|int|long|unsigned|signed|float|double)
set syntax_colors(types) yellow

set syntax_patterns(directives) \
    {(#include|#ifdef|#ifndef|#if|#elseif|#else|#pragma|#endif).*}
set syntax_colors(directives) purple
set syntax_patterns(newline_types) \
    {(char|short|int|long|unsigned|signed|float|double)}
set syntax_colors(newline_types) yellow
set syntax_patterns(keywords) \
    {[^A-Za-z0-9]+(if|then|else|for|goto|while|struct) ([^A-Za-z0-9])}
set syntax_colors(keywords) red
set syntax_patterns(comment1) {(/*.*\n)}
set syntax_colors(comment1) pink
set syntax_patterns(comment2) {(//.*)}
set syntax_colors(comment2) pink
set syntax_patterns(strings) {("[^"]+")}
set syntax_colors(strings) orange
```

So here we define all the colors and basically all the form, that we wanted okay ,so continuing with the text widget, the tag text can be configured, so that is using this .t tag configure my words- foreground red and then the tag text can have bindings essentially, so if you have a text like this it can actually bound to the my word list , list.

So it need not you do not ,have to do it explicitly and then the text widgets can serve as geometry managers and then the syntax coloring Tk Emacs does customized ,that is customizable syntax using coloring of using regular expressions like Emacs highlight 19 package it is a one text widget ,one scroll widget , one read and write, save files and then about 100 lines of well commented code .

What is the procedure behind the Tk Emacs ,so here is another one for all matches routine, so here we define, that all matches w pattern script and then we do a scan basically like to see whether they take this ,here we just calculate the number of lines ,since you and then in this loop essentially ,we get the end and then ,we add the quantum to it and in the do color routine basically does the coloring, here is specified like for each of the arrays, the syntax pattern and then we basically , we can go ahead and delete the tag itself .

And then we configure with some configuration values, so the do color routine essentially the same it ,works the same way basically like, we keep the particular file and then we they will accept a file, open to the \$ file and then until the end of line in the file or the \$ w ,insert N and then so that will be the read of other lines and then the write file same thing basically we do the opposite .

We may write file w file and then we do f open log \$ file , with w command, so here you see this again it is only this open file and then we basically, whatever we collect in the window we just say puts and then the dollar w and then we just close the file .

So we take a look at some of the Emacs , how it behaves in terms of other things ,so one is the set syntax pattern or types ,which gets all the alphanumeric characters and then say syntax colors ,type is yellow check for that ,syntax pattern directives again, syntax colors or directives and purple come that is another thing , then the syntax patterns, for new line types essentially and then there for that .

We cut here ,essentially like ,we can see once we have the syntax patterns, new line, so syntax for new line types essentially has all these various types, whether it is character is short or integer, long ,unsigned ,signed float and bubble, then we declare the color essentially as hello and then the syntax pattern is the regular alphanumeric and then if all these are commands also

and then following the commands again ,and then we also specifically look for certain syntax pattern , *.* here, so this is one thing.

(Refer Slide Time: 45:59)

```

TkEmacs (continued)

# Set up the widgets
text .t \
  -yscrollcommand ".s set" \
  -insertbackground white \
  -width 50 -height 18
pack .t -side left -fill x \
  -expand 1
.t configure -tabs {32 64 96}
.t configure -background black
.t configure -foreground white

scrollbar .s -command \
  ".t yview"
pack .s -side right -fill y

# Key bindings
bind .t <Control-l> {doColor .t}
bind .t <Control-S> \
  {writeFile .t $FILENAME}
bind .t <Control-X> {exit}

# Run a little demo

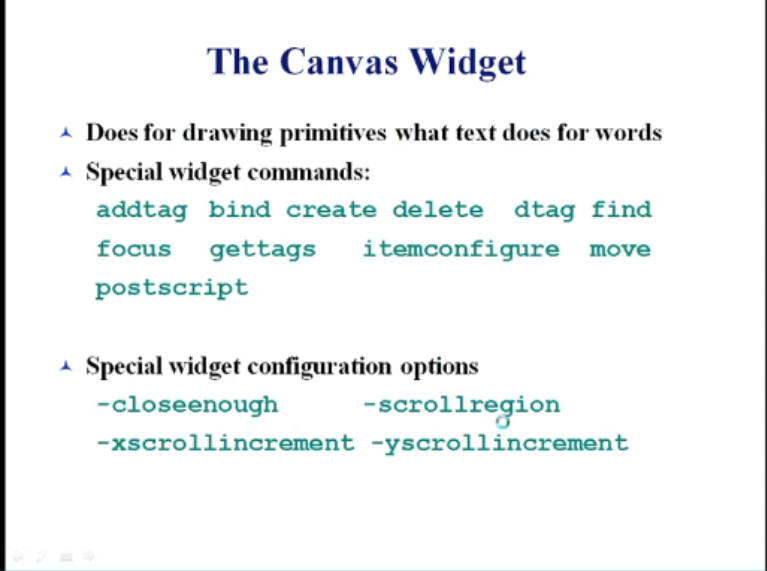
set FILENAME [lindex $argv 0]
loadFile .t $FILENAME
wm title .TkEmacs $FILENAME
doColor .t
```

So now ,we actually comes to setting up , so in this one actually we are reversing the process, little bit, so in the previous example I mentioned that actually we do all the frame and build all the GUI and then go back and fill up the procs, in this one, we actually like get all the processing and then now we are going to do the window system ,so here we set up the widgets basically ,we call it text widget .t .

And then basically we give several other things ,y scroll command, insert background and width and height, so specify all these things and then we pack it and then we adjust from the left side and then we just expand it to fill essentially and then we also give some configuration items, we configure as tabs {32 64 96} and then configure -background is black and then foreground is white, we will come to see like I mean by being sorting and then we also generate a scroll bar .

And then that is on the y side and then, we essentially start doing the key bindings , so we do the control l for color, of this particular window and then control s is to, write the file and save the file essentially and then control x, we bind it to exit ,so if you want to exit this particular window ,we just type control x and it , so the exit corresponds to this .t and now if you run this the demo based essentially and load file to top level wm title .T k Emacs file name and do Color .t and then now we can bring up the Emacs.

(Refer Slide Time: 48:09)



The Canvas Widget

- ▲ Does for drawing primitives what text does for words
- ▲ Special widget commands:
`addtag bind create delete dtag find`
`focus gettags itemconfigure move`
`postscript`
- ▲ Special widget configuration options
`-closeenough` `-scrollregion`
`-xscrollincrement` `-yscrollincrement`

So I will talk about the canvas widget , but we will probably like see, the actual example in the next lecture okay ,so canvas widget is another type of widget which use, it is used for drawing primitives and essentially ,what this text is doing for words, so it is basically it is just an open-ended rectangle ,where whatever you can draw and then that drawing is getting captured, so here again for this particular widget , there are specific commands, basically add tag, bind create, delete ,d tag ,find , focus then talk about this also like in greater detail.

The get tags, item configure and move and postscript ,we also have like the other widget ,configuration option which is close enough scroll region ,x scroll increment and y scrolling.

(Refer Slide Time: 49:41)

The Canvas Widget

- Canvases can be "drawn on"

```
.c create oval 3 3 200 200 \
  -fill yellow -outline black
.c create oval 50 50 60 70 \
  -fill black -outline black
.c create oval 140 50 150 70
  -fill black -outline black

.c create arc 30 60 170 170 -outline black \
  -extent -190 -style arc -width 4 -start 5
.c create arc 20 90 40 110 -outline black \
  -extent -100 -style arc -width 4 -start 320
.c create arc 160 90 180 110 -outline black \
  -extent -100 -style arc -width 4 -start 320
```



So essentially the canvas move, can be drawn on, so here we can draw smiley face with various things and basically coloring inside, so in order to create this ,we create an oval between 3 3 200 200 fill yellow that is, this one, essentially and then outline is black, which is the color of the outline, and then we also have another one 50 50 60 70 eyes ,each one we create exactly the location.

And then we create an arc ,which is for the smiley faces for mouth ,that is also done and then we create this arcs, the tiny arcs, which is the next step ,so this way we create this smiley face.

(Refer Slide Time: 50:48)

The Canvas Widget

- Each drawing primitive or embedded window is an 'item' with a numeric index

```
.c create oval 3 3 200 200 \
  -fill yellow -outline black
=> 1
```

- Items can be tagged, as in text widget

```
.c itemconfigure 2 -tag eyes
.c itemconfigure 3 -tag eyes
```

Now each of these the drawing primitive, is an item in the numerical box ,so when we create that oval ,it returns the number for it and so the items can be tagged as, in the text widget so you can just say ,I can configure 2 and then tag eyes and then same thing eyes .

(Refer Slide Time: 51:17)

The Canvas Widget

- ▲ Each drawing primitive or embedded window is an 'item' with a numeric index

`=> 2` `;# left eye`

- ▲ Items and tags can have bindings

`.c bind eyes <Enter> {Wink %W %x %y}`

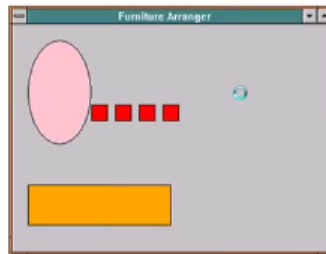
`.c bind eyes <Leave> {Unwink %W %x %y}`

So we tagged these two ovals as eyes and then we can also search items, within this canvas widget and using this fine closest ,so if you give a fine closest and gives it give a coordinate and then basically it returns, to which is left eye and right eye and then items and tags can also have some bindings ,you can say like I mean .c bind eyes enter ,and then you can say wink and then outliving or if you feel like if you press leave then it will unlink.

(Refer Slide Time: 52:15)

Furniture Arranger

- ▲ A simple app to help you decide how to arrange your furniture
- ▲ Uses drawing primitives to represent pieces of furniture
- ▲ Uses multiple tags per item
- ▲ Implements behaviors based on tags



So I want to end at this point essentially, I think and then next lecture, we will take upon the furniture arranger program and how we can design it and how we can execute it using TK, so I think that is pretty much, it we will summarize as we go along and then thank you once again for listening, thanks.