Chapter 9

Graphics and User Interfaces

In the early days of personal computers Macintosh, Windows, and Unix systems had completely different sets of tools for creating graphical user interfaces. Porting a program from one system to another was very difficult because there was no simple way to translate the interface design from one platform to another. This was a difficult situation for both programmers and users. In 1991 John Ousterhout, who was a professor at UC Berkeley, announced the tk toolkit for building user interfaces. This was a system that could be implemented on all of the major platforms and programs would run the same way on any of them. tk is now built into the operating systems of almost all computers. Python provides a module called tkInter to interface with the tk library. In this chapter we will see ways to make use of this module.

This chapter makes lots of use of classes and subclasses, so if you are not on top of that material you might want to review Chapter /refChapter8 before proceeding. In this chapter you will see some of the great advantages of programming with classes: we can use the class structure to hide most of the ugly implementation details and only keep in front of us the portions of the code that change for our particular program. This makes user-interface programming, which used to be the realm of advanced professionals, accessible to all programmers.

9.1 Tk Concepts and Terminology

To get started wth tk we need a little terminology. A *widget* is any kind of graphical element: a window, a button, a checkbox, or anything else that holds information that the user can interact with. A *frame* is any kind of window that can hold other widgets. A *canvas* is a specific type of window that allows us to draw into it. *Buttons* are widgets you click on, *text boxes* are widgets you can write it, *scales* are widgets that allow you to select one value from a range of values, and *labels* are widgets that hold a string they are like text boxes only not interactive. tk treats a menu as a button when you click on it the menu opens up and displays a list of items to choose from; you select an option by clicking on one of these items. There are a few more widgets in tk, but we will see those later. Each of the widgets is represented in tk by a class; we make a widget by constructing an object of the class.

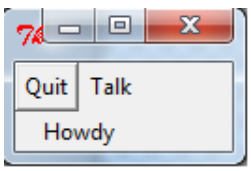
A typical interface for a program will have a main window that holds the entire interface. At the top of this window is a frame that I call a "menu bar"; this holds the menu items, buttons, scales and so forth that control the program. Below the menu bar is a canvas on which we can draw. The program that creates this sets up the widgets and then calls the mainloop() method of the Frame class. This method waits for user interactions and responds to them. Its actual code is more complex, but pseudo-code for the mainloop( ) method is

```
def mainloop(self):
    while True:
        if there-is-user-interactio():
            respond-to-user-interaction()
```

You will notice that there is no way out of this loop. The Frame class has a method quit( ) that exits you from the loop and from the program. If you want to have an animation running on the canvas, you can call an Animate( ) function for the program. This is similar to mainloop() only it does one step of the animation each time around the loop:

```
def Animate(self):
    while True:
        if there-is-user-interactio():
            respond-to-user-interaction()
        <do one step of the animation>
```

Program 9.1.1, which follows, shows a complete graphical program. This is a relatively simple program, with one button and one menu. The menu has options that allow for different strings to appear in a label widget below the control items. Here is a picture of the window this program creates:



The program is simple, but it has all of the elements of more complex GUIs. You can use it as a template for future programs. We will discuss the program in general terms here, then give details for each of the widgets it contains.

Program 9.1.1 creates a class GUI that is a subclass of tkInters standard Frame class. In the main() function it constructs and instance of this class and calls the mainloop() method of the Frame **class**. That is all it does; the rest is handled by the Python system.

Since class GUI is a subclass of Frame, the first thing the constructor for class GUI does is to call the Frame constructor: Frame. __init__ ( ). In addition to self , this has one required argument: the parent object of the Frame being constructed. All widgets need this argument for their constructors. In this case the Frame being constructed is a top-level window; it has no parent, so we pass the "unconstructed object" None. The call to the grid( ) method of the Frame class makes this window visible. All widgets have a grid( ) method. Until you call it the objects will exist but not be visible. There are several kinds of arguments you can give this method. If you call the method with no arguments it makes the widget visible and places it wherever the Python system thinks appropriate. If you call it with row and column arguments, as we do with some of the other widgets, the widget will be made visible and placed at in this location within its parents window.

Next, the constructor calls a function to create a menu system, then a function to create a quit-button; both of these are placed in row 0 of the windows grid. Finally, it creates a Label widget and places it in row 1 of the grid. A string variable called outputString is created to supply the text of this widget. We change labels by modifying this string variable.

Here is the full text of the program. We will follow this with more details about the various widgets created in the program.

```
from tkinter import *

class GUI(Frame):
   def __init__(self):
      Frame.__init__(self, None)
      self.grid()

      MenuBar = Frame(self)
      MenuBar.grid()

      self.makeTalkMenu(MenuBar)

      QuitButton=Button(MenuBar,text="Quit",command=self.quit)
      QuitButton.grid(row = 0, column = 0)

      self.outputString = StringVar(value = "")
      PrintBox = Label(MenuBar, textvar=self.outputString)
      PrintBox.grid(row = 1, column = 0, columnspan=2);

   def makeTalkMenu(self, MB):
      Talk_button = Menubutton(MB, text='Talk')
      Talk_button.menu = Menu(Talk_button)
      Talk_button['menu'] = Talk_button.menu
      Talk_button.grid(row = 0, column = 1)
      Talk_button.menu.add_command(label='Say Hi',\
                        command=self.hi)
      Talk_button.menu.add_command(label='Say Bye',\
                        command=self.bye)

   def hi(self):
      self.outputString.set("Howdy")

   def bye(self):
       self.outputString.set("Bye bye")

def main():
   window = GUI()
   window.mainloop()

main()
```

Program 9.1.1: A first GUI program

Menus

In TkInter, a "menu" consists of

1. A menu button, which is the top-level string the user clicks on to display the menu options.
2. A menu of items, one of which will be selected when the user clicks on it.

We create this in three steps:

a. Calling the Menubutton( ) constructor to make the menu button. This needs to arguments: the parent window where this button lives, and a text variable that holds the string that represents the menu.

b. Setting up the menu that holds the items. This takes two statements: one call to the Menu( ) constructor,which needs only the name of the menu button, and one assignment that links the Menu( ) constructor to the menu button

c. A series of statements that create the individual menu items. Each of these needs a string for its label,and a command function that will be called when this item is selected. This command function should take no arguments.

For our program these statements are

```
Talk_button=Menubutton(MB, text='Talk')
Talk_button.menu=Menu(Talk_button)
Talk_button['menu']=Talk_button.menu
Talk_button.grid(row = 0,column = 1)
Talk_button.menu.add_command(label='Say Hi',command=self.hi)
Talk_button.menu.add_command(label='Say Bye',command=self.bye)
```

We could have as many Talk_button.menu.add_command( ) lines as we wish; each adds another item to the menu.

Buttons

These are created with one call to the Button( ) constructor. This needs 3 arguments:

a. The parent widget that gives the window where the button lives

b. The text to be printed on the button

c. The "command" or *callback* function to be called when the user clicks on the button. Again, this should be a function with no arguments.

In our program we create a quit-button with

```
QuitButton = Button(MenuBar, text="Quit", command = self.quit)
```

The callback function we use, self.quit( ), is a standard method of the Frame class. We create the button in our GUI subclass of Frame, so this method is inherited by the subclass.

Labels

These are generally static holders of strings To make a simple label we use the Label( ) constructor which needs 2 arguments:

a. The parent widget that gives the window where the label lives.

b. A text string to be printed for the label.

We could make such a label with a statement like

```
Label(parent, text = "This is a string")
```

In our case we want the text of the label to be modifiable, so we go one step further. Instead of a static string as the text of the label we give a control variable that can be dynamically modified by our program. There is a separate class for each kind of control variable: IntVar, StringVar, and DoubleVar (a floating point control variable). Each has a value instance variable to hold the value of the control variable, and each class has set() and get( ) methods to manipulate this value. Accordingly, we create this control variable with

```
<name> = StringVar(value = "string")
```

and we assign it to the label as its textvar field.

The only other items our program needs are the callback functions to be called when the user selects the menu items. These are registered with the program when the menu items are created. Here is a typical one; it changes the label's textvar value to the string "Howdy".

```
def hi(self):
    self.outputString.set("Howdy")
```

We make this a method of the GUI class, though it could just as easily be a stand-alone function. As a method it needs self as a formal parameter; as a stand-alone function it would have no parameters.