2

Tk Sınıfı

2.1 Ana Taşıyıcı

Top level widget

Widget'leri[1]. incelemeye başlamadan önce, GUI alet çantalarının çalışma ilkelerini açıklamak yararlı olacaktır. Bir görsel arayüz hazırlarken, öncelikle widgetleri taşıyacak bir zemine gerekseme vardır. Bu zemin en alta konulan widgettir. Tkinter'de en alttaki zemin yerine kullanılan widget, genellikle, *Tk* sınıfından türetien bir nesnedir. Görsel arayüzü monitörde açılan bir pencere olarak düşünürsek, açılan bu pencere görsel arayüzün zeminidir. Düğmeler, etiketler, yazı alanları gibi öteki görsel arayüz aletlerini (widgets) `Tk` sınıfından türetilecek bu zemin üzerine koyacağız.

Bütün widgetleri taşıyan bu zemine *anataşıyıcı* (top level widget) diyoruz. Benzetmek gerekirse, bir galeride rasimlerin üzerine konulduğu duvar (ya da pano) anataşıyıcıdır. Duvar üzerine çeçeveler, çerçevelerin içine resimler, resimlerin üzerine yazılar konulabilir. Görsel arayüz araçlarının (widgets) anataşıyıcı üzerine konuşlandırılmaları da böyledir. En altta anataşıyıcı vardır. Onun üzerine widgetler konulur. Bazen widgetler üst üste olabilir. Böyle olduğunda, alttaki widget üstündeki widget'in taşıyıcısıdır. Ama hepsini taşıyan anataşıyıcıdır.

Anataşıyıcıya İngilizce'de *Window*, *root*, *top-level widget*, *basic* gibi adlar verilir. Türkçe kaynaklar, çoğunlukla, *pencere* der. Anataşıyıcı Tk()

[1]Kaynak kodları yazarken widgetlerin özgün (original) adlarını kullanmak zorundayız. Alışkanlık yaratmak için, konuyu anlatırken de widgetler'in orijinal adlarını kullanmayı tercih edeceğiz. Ancak, gerekli olduğunda, Türkçe anlamlarını parantez içinde yazacağız

kurucu metodu ile yaratılan nesneye verilen ad; yani onun ana bellekteki yerini gösteren pointerdir. Ona, örneklerimizde *w, win, window, ana, pencere* gibi adlar vereceğiz. Anataşıyıcıyı Tk() kurucusu ile yaratacağımız için, onun hangi adla anıldığı her örnekte apaçık belirli olacaktır.

Bazen widgetler üst üste konulabilir. Bir widget'in altında olan onun taşıyıcısıdır. Dolayısıyla, tkinter ile hazırlanan bir görsel arayüzde bütün widgetlerin ana taşıyıcısı *Tk* sınıfından türetilen bir tek nesnedir. Onu taşıyan; yani onun altına konulan başka bir widget yoktur. Vurgulamak için ona *anataşıyıcı* diyoruz. Ama uygulamada window ya da pencere terimleri alışkanlıklarımıza daha çok uymaktadır. Arayüz hazırlayan kişi *anataşıyıcı* yerine istediği bir adı kullanabilir. Zaten hazırlanan görsel arayüzü kullanan kişi, anataşıyıcıya ve öteki widgetlere kaynak programda verilen adları bilmeyecektir.

2.2 Nasıl Çalışır?

Widgetlein boyutlarını, konuşlanacakları yerleri, renklerini, vb niteliklerini istediğimiz gibi ayarlayabiliriz. Hazırlayacağımız görsel arayüzün anataşıyıcısının boyutlarını, ekranın boyutlarına eş kılabileceğimiz gibi, daha küçük boyutlar da seçebiliriz. O nedenle, uygulamalarımızda *Window* terimi, monitörün penceresini değil, görsel arayüzümüzün anataşıyıcısını (root, pencere) ifade edecektir.

Arayüz kullanırken, örneğin bir düğmeye tıkladığımızda, bilgisayar bir iş yapar. O işi yaptıran kodlar, düğmeye basılınca kendiliğinden çalışmaya başlar. Buna olay yönetimi (event handling) diyoruz. Yaratacağımız görsel arayüzlerde, olay yönetiminin nasıl yapıldığını, ilerleyen bölümlerdeki örneklerimizde ele alacağız. Ama öncelikle, bir pencere yaratıp, onun üzerine widgetleri nasıl yerleştireceğimizi öğrenmeliyiz.

2.3 Anataşıyıcıyı Yaratma

`Tk`, tkinter modülü içinde bir sınıftır (class). Anataşıyıcı Tk sınıfından türetilen bir nesnedir. Öyleyse, `Tk` sınıfına ait bir nesne yaratmamız gerekecektir. Bu işi yapmak için, `Tk` sınıfına ait bir nesne yaratan `Tk()` *kurucu metodunu* çalıştırmak yetecektir. Tabii, daha önce, *Tk* sınıfını içeren *tkinter* modülünü çağırmalıyız.

tkinter modülü, arayüz hazırlamak için gerekli olan sınıfları, metotları ve başka öznitelikleri içerir. Genellikle, arayüz yaratırken *tkinter* modülün-

deki her şeye gerekseme duyar ve dolayısıyla hepsini çağırırız. Çağırma işini

```
>>> import tkinter
>>> import tkinter as T
>>> from  tkinter import *
```

deyimlerinden birisiyle yapabiliriz. Sonuncusu tkinter modülüne ait bütün aduzayını çağırır. İlk ikisi bazı sistemlerde çalışmayabilir. Çoğunlukla üçüncüyü tercih edeceğiz.

Şimdi `Tk` sınıfına ait bir nesne yaratmak için, IDLE'da şu iki deyimi yazalım:

Liste 2.1.

```
>>> from  tkinter import *
>>> Tk()
```

Ekrana

```
<tkinter.Tk object at 0x02763D70>
```

yazısı ile birlikte Şekil 2.1'de görülen pencere gelecektir. Sondaki hex sayısı, yaratılan Tk nesnesinin bellekteki adresidir; sistemden sisteme değişir.

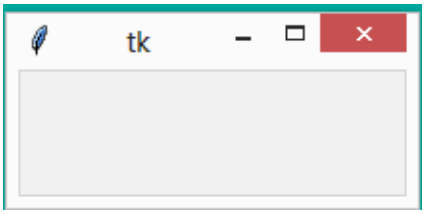
Bu ileti `Tk()` kurucusunun *Tk* sınıfına ait bir nesne yarattığını söylüyor ve onun ana bellekteki adresini gösteriyor. Görünen pencereye anataşıyıcı diyeceğiz. Önceden söylediğimiz gibi, ona *root, top-level, basic, parent, window, pencere* gibi adlar verilir.

`Tk()` metodu, sınıfa ait bir nesne yarattığı için, ona *kurucu metot* diyeceğiz. Python'da sınıftan nesne yaratma eylemine *initialize* denilir.

Nesne tabanlı programlama dillerinde, sınıftan bir nesne yaratılınca, onun ana bellekteki adresini işret eden bir pointer tanımlanır. Liste 2.1'de `Tk()` kurucusunun yarattığı nesneyi işaret eden bir pointer tanımlanmamıştır. Böyle yapmak iyi değildir. Çünkü, onu gösteren bir değişken (pointer) olmadığı için, programın başka yerlerinde o nesneye ulaşan kodlar yazmak olanaksızdır. O nedenle, yaratılan her nesneye bir ad vermeyi alışkanlık edinmeliyiz. Örneğin, Liste 2.1'in ikinci satırı yerine `p = Tk()` yazabiliriz. O zaman `p` pointeri yaratılan Tk nesnesini (pencere) işaret eden pointer olur; yani nesnenin adıdır. Tabii, `p` yerine istediğimiz başka bir ad kullanabiliriz.

Ekranda açılan Şekil 2.1 penceresini inceleyelim.

Şekil 2.1, Windows işletim sistemindeki pencerelere benziyor. Dikdörtgenel bir biçemi var. Üst kısımdaki şeritte, soldan sağa doğru şunları görüyoruz:



Şekil 2.1: Windows Nesnesi

> Tk logosu, **tk** etiketi, pencereyi küçültüp en alttaki görev şeridine indiren ikona dönüştürme düğmesi (_), pencereyi büyütüp küçültmeye yarayan dikdörtgen biçimindeki düğme , pencereyi kapatan en sağ üst köşede (X) işaretli düğme.

Pencerenin üst şeridinde gördüklerimiz Windows işletim sistemindeki pencerelerde standart olarak var olan öğelerdir. İşletim sistemimiz Linux ya da OS X olsaydı, yaratılan pencere o işletim sistemlerindeki pencerelere benzer olacaktı. Örneğin, burada en sağda beliren üç düğme, Linux ya da OS X işletim sistemlerinde pencerenin sol üst kısmına konuşlanacaktır.

Üst şeridin altında dikdörtgenel boş bir alan vardır. Bu alana görsel arayüzde kullanılacak widgetler konuşlandırılır. Bundan sonraki paragraflarda bu işi yapacağız.

> Buradan çıkaracağımız sonuç şudur: *tkinter* ile yarattığımız pencereler, kullandığımız işletim sisteminin pencerelerine benzer olur.

Pencerede Olay Yönetimi

Şimdi yarattığımz pencerede faremizle biraz oyun oynayalım. Yarattığımız pencereyi, üst şeritte bir yere tıklayıp, faremizin sol düğmesini basılı tutarak, monitörde istediğimiz konuma götürebiliriz. Gene, köşelerinden birisine tıklayıp, faremizin sol düğmesini basılı tutarak, pencerenin buyutunu büyültüp küçültebiliriz. Benzer eylemi kenarlardan birisi ile de yapabiliriz. İkonlaştırma düğmesine (_) tıklayınca pencerenin küçülerek görev çubuğuna bir ikon olarak indiğini, büyültme-küçültme düğmesine tıklayınca pencerenin ekranımızın boyutuna eşit boyuta geldiğini ya da tekrar küçüldüğünü, kapatma düğmesine tıklayınca pencerenin kapandığını (yok olduğunu) görebiliriz.

Bütün bu eylemleri yapması için, biz, hiç komut yazmadık. Ama, bu eylemleri yaptıran komutlar (metotlar) *Tk* sınıfında zaten vardır. O metotlar, kalıtsal olarak `Tk()` kurucusunun yarattığı nesneye geçmiştir. O metotları yeniden yazmaya gerek kalmadan kullanıyoruz. Gerçekte, burada yaptığımız eylemlerin her biri bir olay yönetimidir (event handling). Ama bu konuyu ele almadan önce widgetleri pencerede konuşlandırmayı öğrenmeliyiz.

Tk modülünün aduzayındakileri görmek için IDLE arayüzünde

Liste 2.2.

```
>>> from tkinter import *
>>> dir(Tk)
```

yazınız. Çok uzun bir listenin yazıldığını göreceksiniz.

2.3.1 Pencerenin Yapısı

tkinter ya da başka bir GUI alet çantası ile yaratılan bir pencerenin yapısı şöyledir. En üst şerit logo, başlık ve pencerenin eylem düğmelerini içerir. Bu satıra pencerenin *başlık şeridi* (title bar) diyeceğiz. *Başlık şeridi* dememizin nedeni açıktır; pencereye istediğimiz adı verebiliriz. Verdiğimiz ad, yukarıdaki `tk` yerine geçer. `tk`, pencerenin öntanımlı (default) başlık adıdır. Pencereye başka bir ad verdiğimizde, yeni ad `tk`'nın yerini alır. Ad şeridinin altındaki dikdörtgenel boşluğa, istediğimiz widgetleri konuşlandırabiliriz. Çoğunlukla, ad kuşağının hemen altına `menü` kuşağı konulur. Menü kuşağının altına başka widgetler konulur. O bölgeye içerik bölmesi (content pane) adı verilir. Bazı durumlarda, pencerenin en alt satırına *görev çubuğu* denilen şerit konulabilir. İçerik bölmesine widgetleri konuşlandırmak için, konuşlandırma yöneticilerini *(Layout Managers)* kullanacağız.

2.3.2 Pencereye Başlık Verme

`title()` Metodu

Şimdi penceremize ön tanımlı `tk` başlığı yerine başka bir ad verelim. Pencereye ad vermek için Windows sınıfının `title()` metodunu kullanacağız. Tabii, bu metodun kalıtsal olarak Windows sınıfından yaratılan nesnelere geçtiğini anımsıyoruz.

Liste 2.3.

```
from tkinter import *

pencere = Tk()
pencere.title('ilk Pencere')

pencere.mainloop()
```

Açıklamalar:

1.satır, tkinter modülünün aduzayını (namespace) çağırır. Böylece tkinter modülüne ait her şeyi programda kullanabiliyor oluruz. 3.satırda, Tk sınıfından `Tk()` kurucusu ile yaratılan nesneye *pencere* adı verildi. İstenirse başka bir ad da verilebilir. Tabii, verilen ad, yaratılan nesnenin ana bellekteki adresini gösteren bir pointerdir. Yaratılan pencere nesnesi anataşıyıcıdır. 4.satırda, Tk sınıfının `title()` metodu, yaratılan nesneye ad veriyor. Bu ad öntanımlı `tk` yerine geçer. Liste 2.3'in üçüncü satırı, pencere adlı nesneye *'ilk Pencere'* adını verdi. Bu ad, Şekil 2.1'deki `tk` yerine geçer ve ad şeridindeki yerini alır. `title()` metodunda ad olarak yazılan string görünmeyecek kadar uzunsa, hepsini görmek için fare ile pencerenin genişliğini büyütebilirsiniz. title() metodunun Tk sınıfının aduzayında olduğunu görmek için

```
>>> type(Tk.title)
```

deyimini yazarsanız

```
<class 'function'>
```

yanıtını alırsınız.

Son satırdaki *pencere.mainloop()* deyimi, bir sonsuz döngüdür. Kapatılana kadar pencerenin ekranda görünmesini sağlıyor. Bu döngü yazılmazsa, pencere, monitörde bizim göremeyeceğimiz hızla bir kez çıkar ve yokolur. Pencereyi sürekli görebilmemiz için, onu sonsuz *mainloop()* metodu ile ekranda tutuyoruz. Pencere kapatıldığında *mainloop()* döngüsü sona erer.

2.4 Sınıf Kullanma

Nesne tabanlı programlamada, her işi sınıflarla yapmayı alışkanlık edinmeliyiz. Onun için, yukarıda yaptıklarımızı sınıflarla terarlayalım. Önce bir sınıf yaratmalıyız.

```
class Basit(tkinter.Tk):
```

deyimi `Basit` adlı bir sınıf bildirimidir. Bu sınıf, `tkinter.Tk` sınıfının altsınıfıdır. Başka bir deyişle, *tkinter.Tk* sınıfı `Basit` adlı sınıf için üstsınıftır

(ata, parent). Dolayısıyla, `Basit` sınıfı, `tkinter.Tk` sınıfının bütün öğelerini kalıtsal (inheritance) olarak alır.

Şimdi Basit sınıfının bir kurucusunu (constructor) tanımlayalım:

```
class Basit(tkinter.Tk):
    def __init__(self,parent):
        tkinter.Tk.__init__(self,parent)
```

Açıklamalar:

2.satırda tanımlanan `__init__(self,parent)` metodu, 3.satırda üst-sınıfın `tkinter.Tk().__init__()` kurucusunu çağırıyor. Başka bir deyişle, Basit sınıfın içindeki `__init__()` metodu, üst sınıfın `__init__()` metodu imiş gibi tanımlanıyor.

GUI'de widgetler içiçe yuvalanabilir; yani bir widget başka birisinin taşıyıcısı olabilir. Bir widget başka birisinin üzerine konulmuş ise, alttaki *taşıyıcı*, üstteki *taşınan*'dır. Taşıyıcı, taşınanın atasıdır (parent). Bu açıdan bakınca, anataşıyıcı dışındaki her widget'in bir atası (taşıyıcısı) vardır. O nedenle, bir widget'in kurucusunda *ata ya da parent* parametresi onun taşıyıcısıdır. Bir grup widget'in görünür ya da görünmez kılınabilmesi için, onların taşıyıcısını bilmemiz gerekir. Bunu yapmak için *self.parent = parent* deyimini kullanırız. Bu deyimi de eklersek, programımız,

```
import tkinter

class Basit(tkinter.Tk):
    def __init__(self,parent):
        tkinter.Tk.__init__(self,parent)
        self.parent = parent
```

biçimini alır.

Arayüz hazırlarken bütün *label, button, text field,...* gibi pencere araçlarının hepsini (widgets) elimizin altında tutmak iyidir. Bunun için, programımıza *initialize()* metodunu ekleyelim.

```
import tkinter

class Basit(tkinter.Tk):
    def __init__(self,parent):
        tkinter.Tk.__init__(self,parent)
        self.parent = parent

    def initialize(self):
        pass
```

Son satırdaki `pass` deyimi, Python'da bir iş yapmadan geç anlamında kullanılan bir komuttur. Dolayısıyla, Module 2.1'nin 11.satırında tanımlanan `initialize()` metodu şimdilik bir iş yapmayacaktır.

Son olarak `main` metodunu ekleyerek programımızı tamamlayalım.

Module 2.1.

```
1 #!/usr/bin/python
  # -*- coding: utf-8 -*-

  import tkinter

6 class Basit(tkinter.Tk):
      def __init__(self,parent):
          tkinter.Tk.__init__(self,parent)
          self.parent = parent

11    def initialize(self):
          pass

  if __name__ == "__main__":
      uygulama = Basit(None)
16    uygulama.title('İlk Uygulamam')

      uygulama.mainloop()
```

Açıklamalar:

1.satır, Unix tabanlı sistemlerde Python'un kurulu olduğu dizini bildirir.

2. satır, Unix tabanlı sistemlerde, karakter kodlamasının utf-8 olduğunu bildirir. Windows işletim sisteminde ilk iki satırın işlevi yoktur.

6.satır bir sınıf bildirimidir.

14.satırdaki main metodu, Python programlarını yürütmek için kullandığımız metottur. Kava'daki main() metodunun işlevine benzer bir işleve sahiptir.

15.satır Basit sınıfının kurucusunu None parametresi ile çağırıyor. Dolayısıyla `parent` sınıfı `None` oluyor. Bilindiği gini Python dilinde `None` boş sınıf yerine geçer. O nedenle, Basit sınıfı üst sınıftan kalıtsal bir şey alamaz. Bu satırda yaratılan nesneye *uygulama* adı verildi. İstenirse başka bir ad da verilebilirdi.

16.satır, uygulama adlı nesneye "ilk uygulama" başlığını veriyor.

18.satır mainloop() döngüsü ile, yaratılan nesneyi (pencere), kapatılana kadar ekranda görünür kılıyor. Arayüzün görünmesi için tkinter'de bu döngünün gerekli olduğunu daima anımsamalıyız. Başka GUI araç kutularında `mainloop()`'un adı farklı olabilir.