A Perl/Tk

Kölcsönözz felhasználóbarát külsőt a már megírt, hatékonyan működő Perl parancsállományaidnak!

A Linuxvilág hűséges olvasói biztosan emlékeznek még arra, amikor saját IRC-botot írtunk, vagy amikor elkészítettük önálló web pókunkat, hogy azzal gyűjtsük be az információkat a világhálóról. Ezeket a feladatokat azért Perlben oldottuk meg, mert a hálózatkezelés, és a vele járó szövegfeldolgozási feladatok ezen a nyelven tűnhetnek a legegyszerűbbnek. A Perl különlegesen rugalmas és ennek köszönhetően számos feladatra kitűnően alkalmazható. Biztosan feltűnt, hogy eddigi cikkeimben ugyan szó volt szövegfeldolgozásról, könyvtárkezelésről, TCP/IP programozásról, de sosem készült valódi felhasználói felület programjainkhoz. Csak alapvető be- és kimeneti módszereket használtunk, melyekkel parancssorhoz szokott szemmel egész jól el lehetett igazodni az alkalmazások használatában. Ám mindig gondolni kell arra a felhasználóra is, aki először ül a monitor előtt, és általában a grafikus felhasználói felületekért (GUI) rajong. Képes erre a Perl? A válasz természetesen: igen. A Tk-ról van szó, melyet eredetileg a Tcl nevű parancsnyelvhez készítettek el. Később azonban egy Perl modul formájában már ebből a nyelvből is elérhetővé vált ez a remek eszköztár. Mindenekelőtt szerezd be ezt a modult a CPAN oldaláról (➲ http://www.cpan.org). Több mint valószínű, hogy terjesztésed is tartalmazza csomag formájában, ekkor elég ezt telepítened. Debian alatt a `perl-tk` csomagra van szükséged, amely tartalmaz egy elég jó leírást is. A `Tk.pm` egy objektumközpontú felülettel áll a rendelkezésünkre. Látni fogjuk, hogy a grafikus programozás mindössze különféle osztályokból történő példányosításokból fog állni. Ha még nem barátkoztál meg az objektumközpontú programozással, feltétlenül olvasd el a ***perlboot (1)*** súgóoldalt, e nélkül ugyanis nem fogod tudni alkalmazni az itt bemutatásra kerülő módszereket. Vágjunk bele! Első lépésként hozzunk létre egy ablakot. Íme a kód:

```
#!/usr/bin/perl -w

use strict;
use Tk;

my $main_win = MainWindow -> new ();
$main_win -> title ("Hello világ!");

MainLoop ();
```

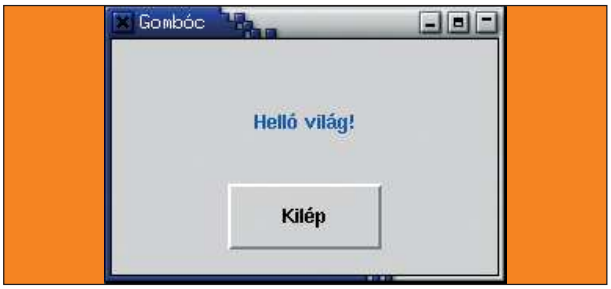
Bemelegítésként nézzük át sorról sorra, mit is csinál ez a program. Az első sor mindössze arra szolgál, hogy a parancshéjnak meghatározzuk a parancsállományunkat feldolgozó parancsértelmező elérési útját. Emellett megadunk neki egy további kapcsolót is, mellyel arra utasítjuk az értelmezőt, hogy a futtatás során a figyelmeztetéseket is jelenítse meg, ne kizárólag a hibaüzeneteket. A következő sor a szigorú feldolgozást kapcsolja be. Ha nem tudod, mi ez, és jó Perl programot akarsz írni, jobb, ha mindig használod. A harmadik sorban pedig a fentebb már emlegetett Tk modult vesszük használatba. Ezután létrehozunk egy új skalár változót, `$main_win` néven. A `my`, vagy `local` kulcsszóval történő változóbevezetés szigorú módban kötelező. A két kulcsszó a változó láthatóságában tesz különbséget, ám erre itt nem térnék ki. Új változónknak azonnal értéket is adunk. Jelen esetben a `MainWindow` osztály `new` nevű konstruktorát hívjuk meg, mely egy objektummal tér vissza. Ezt csak azért hangsúlyoztam, mert Perlben nincs megkötés egy osztály konstruktorára vonatkozólag, ezért hívhatnák másképp is. Annak, hogy a neve `new`, megvan viszont az az előnye, hogy egy már más programozási nyelvekből is ismerős formában is példányosíthatunk:

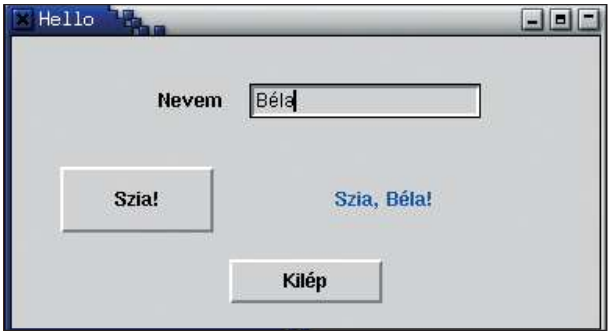
```
my $main_win = new MainWindow;
```

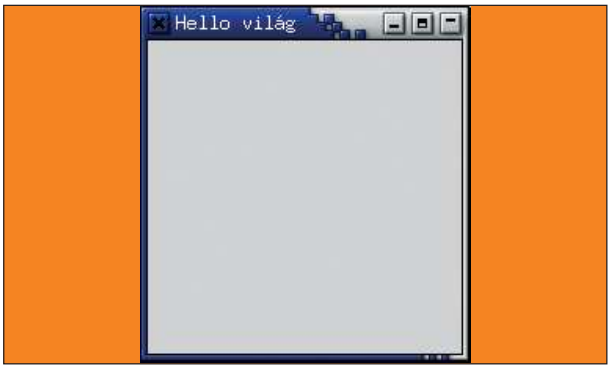
Visszatérve a példára, a következő sorban újonnan létrehozott objektumunk egy tagfüggvényét hívjuk meg. A `title ()` egy sztringet vár, és segítségével az ablak címét módosíthatjuk. Az utolsó sorban történő függvényhívás indítja el az eseményciklust. Ez lényegében egy végtelen ciklus, amely a különböző vezérlők (***widget***) eseményeinek lekezeléséért felelős. Ne ijedj meg, ha elsőre nem érted, a példák világossá fogják tenni ennek a rejtélyes függvényhívásnak a mibenlétét. A lényeg, hogy az ablak e nélkül a sor nélkül nem jelent volna meg. Továbbá az eseményciklus végtelen tulajdonsága miatt az ablak mindaddig él (és fut a program), amíg az ablakkezelőn keresztül be nem zárjuk.
Miután létrehoztuk első ablakunkat, semmi sem állíthat meg bennünket, hogy vezérlőket pakoljuk rá. Íme:

```
#!/usr/bin/perl -w

use strict;
use Tk;
```








```
my $main = new MainWindow;
$main -> title ("Hello");

my $label = $main -> Label (
     -text => "Helló világ!",
     -width => 25,
     -height => 3
);

$label -> pack;

MainLoop;
```

A Tk eszközkészletben számos vezérlő áll rendelkezésünkre. Ilyenek a gomb, jelölőnégyzet, rádiógomb, menü, vászon, és még sok más. A címke (*label*) vezérlő egy közönséges, a felhasználó által csak olvasható szövegmező. Mint minden vezérlőnek, ennek is vannak tulajdonságai. Ezek a tulajdonságok már a vezérlő létrehozásakor kaphatnak kezdeti értéket, de vannak alapértelmezett beállítások is. Általában igaz, hogy a vezérlőről a `Tk::vezérlő_neve (3pm)` súgóoldalon kaphatunk egy teljes leírást.

A hatodik sorban hozzuk létre az új címkét. Látható, hogy ehhez mindössze a főablak objektumának a megfelelő tagfüggvényét kell meghívnunk, és ez visszaadja az új címkeobjektumot. A Label tagfüggvény egy asszociatív tömböt vár paraméterként. Ez egy kulcs-érték párokból álló lista, másként fogalmazva egy olyan tömb, ami karakterláncokkal van indexelve. Mi itt egy névtelen tömböt adunk át a függvénynek. A kulcsok rendre egy kötőjellel kezdődnek, jelezve, hogy beállításokról van szó, ám ezek a kötőjelek bizonyos Tk változatokban elhagyhatók. Nagyon fontos, hogy a kulcsok és értékek között => áll és nem ->!

A következő sor nagyon fontos. E nélkül a címke meg sem jelenik az ablakban, hiába hoztuk létre az objektumot. A vezérlőobjektum ugyanis a létrehozás pillanatában független az ablaktól, annak ellenére, hogy a főablak tagfüggvényével hívtuk életre. Szükség van egy geometriakezelőre, amely felpakolja a vezérlőt az ablakra. Számos geometriakezelő létezik, mi most a legegyszerűbbet használjuk, a `pack ()` függvényt. Ezt nem szabad összetéveszteni a szabványos Perl könyvtárban található `pack ()` függvénnyel. Ez a vezérlőobjektum tagfüggvénye. Segítségével nagyon egyszerűen, a már meglévő elemekhez képest a négy irányban bárhova rakhatunk egy új vezérlőt. Paraméterek nélküli híváskor folyamatosan egymás alá pakol.

Nem élet az élet gomb nélkül. Íme:

```
#!/usr/bin/perl -w

use strict;
use Tk;

my $main = new MainWindow (
     -borderwidth => 15
);
$main -> title ("Gombóc");

$main -> Label (
     -text => "Helló világ!",
     -foreground => "blue",
     -width => 30,
     -height => 5
     ) -> pack;

$main -> Button (
     -text => "Kilép",
     -width => 10,
     -height => 2,
     -command => \&exit
     ) -> pack;

MainLoop;
```

Számos újdonságot fedezhetünk fel a már megismert elemek használatában is. Mint láthatod, a főablak objektum létrehozásakor is át lehet adni bizonyos paramétereket a konstruktornak. Itt az ablak belső szegélyének vastagságát adtuk meg képpontokban. Ennek a beállításnak az alapértelmezés szerinti értéke nulla. A félreértések elkerülése

végett, ez nem az ablakkezelő által szolgáltatott ablakkeret! Ez az a szegély, ami körbeveszi az összes általunk felpakolt vezérlőt az ablakban. A címkeobjektumot létrehozzuk, ám egy változónak sem adjuk értékül. Ez akkor nem gond, ha a programunkban később sehol sem akarunk hivatkozni erre a vezérlőre. Ha például később módosítani szeretnénk a szöveget, akkor ezzel a könnyelműséggel nem élhetünk. Ebben az esetben viszont elég arról gondoskodni, hogy az objektum geometriakezelőjét meghívjuk, ezt pedig a fent látható módszerrel megtehetjük. Kurta megoldásunknak nem látjuk kárát, mert a -> operátor balról jobbra értékelődik ki. Megadtuk továbbá a címke `-foreground` beállításában a szöveg színét. A gomb létrehozása ezek után már gyerekjáték. Itt is egy névtelen gombot hozunk létre, és meghívjuk rá a `pack ()`-et. Értelemszerűen a `-text` a gombon olvasható szöveget jelenti. Érdekes újdonság a `-command`. Ez egy olyan beállítás, amelynek egy függvény címét kell adni. Egy változó címét a \ operátorral képezzük, a & pedig a függvénytípust jelöli. A gombra történő kattintás hatására meghívásra kerül a megadott függvény.
A `MainLoop` tehát az ablak kirajzolása után folyamatosan vár a gomb megnyomására, és az eseményt kiváltó kattintást követően meghívja a `-command` függvényét.
Jöjjön végre egy igazi példa.

```
#!/usr/bin/perl -w

use strict;
use Tk;

my $main = new MainWindow (
     -title => "Hello",
     -borderwidth => 15
);

my $frame1 = $main -> Frame ( -borderwidth =>
➥15 );

$frame1 -> Label (
     -text => "Nevem",
     -width => 10,
     -height => 1,
     ) -> pack ( -side => "left" );

my $name;
$frame1 -> Entry (
     -textvariable => \$name,
     -width => 20,
     -validate => "key",
     -validatecommand => \&editentry
     ) -> pack ( -side => "right" );

$frame1 -> pack;

my $frame2 = $main -> Frame ( -borderwidth =>
➥15 );

$frame2 -> Button (
     -text => "Szia!",
     -width => 10,
     -height => 2,
     -command => \&printhello
     ) -> pack ( -side => "left" );

my $label;
$frame2 -> Label (
     -textvariable => \$label,
     -foreground => "blue",
     -width => 30,
     -height => 1
     ) -> pack ( -side => "right" );

$frame2 -> pack;

my $quit = $main -> Button (
     -text => "Kilép",
     -width => 10,
     -height => 1
     ) -> pack;
$quit -> bind ("<Button-1>", sub { exit; });

MainLoop;

sub printhello {
     my $hello = "Szia";
     if (defined $name and "" ne $name) {
        $hello .= ", " . $name;
     }
     $hello .= "!";
     $label = $hello;
}

sub editentry {
     $label = "";
     return 1;
}
```

Az első változtatás az, hogy milyen módon adtuk meg a főablak címét. Eddig az objektum `title ()` tagfüggvényét használtuk, most áttértünk a kezdetiérték-adásra. A második a keretek használata. A keret feladata egységbe foglalni a vezérlőket. Ez előnyös lehet nekünk, a programozónak, hiszen logikai egységekre bonthatjuk a vezérlőelemeket. Előnyös lehet a felhasználónak, mert ha láthatóvá tesszük a keret szélét, azzal áttekinthetőbbé tesszük a felületet. Továbbá előnyös lehet a vezérlők elhelyezésekor. Most ez utóbbi miatt alkalmazzuk a kereteket, mivel a `pack ()` nem túl okos. Első megközelítésben nyugodtan gondolhatsz úgy a keretekre, mint kisebb ablakokra a főablakon belül, hiszen nem mások, mint vezérlőtárolók. Ez a program egy nevet kér be, majd kiírja. Úgy építjük fel az ablakunkat, hogy lesz két keretünk egymás alatt, és egy kilépés gomb legalul. A felső keretben egymás mellett van egy címke, és egy szövegbeviteli mező. A címke természetesen utal arra, hogy mit kell beírni a mezőbe. Az alsó keretben pedig van egy gomb, és egy másik címke. A gomb megnyomására a címkében megjelenik egy üdvözlőszöveg, azzal a névvel, amely a felső keretbeli beviteli mezőben szerepel. Mivel a keret olyasmi, mint egy kisebb ablak, úgy is kell használni, mint egy ablakot. Létrehozásakor megadhatjuk a keret vastagságát a `-borderwidth`

beállítás segítségével csak úgy, mint a főablak esetében.
A kereten belül elhelyezésre kerülő vezérlőket természetesen a keretobjektum tagfüggvényeivel kell létrehozni. Ezért dolgoztunk a fenti példában `$frame1`, illetve `$frame2` objektumok függvényeivel. Továbbá nem elég az egyes vezérlőkre meghívni a `pack ()` függvényt, a vezérlő kipakolása után az egész keretre is ugyanúgy meg kell hívni. A következő meglepetés egy szövegbeviteli mező formájában érhet. A beviteli vezérlő (`entry`) egysoros, felhasználó által szerkeszthető szöveges mező. Hasonló beállításai vannak, mint a címkének, ám mivel egysoros, nincs `-height` beállítása.
A `-textvariable` által megadható egy változó címe, mely mindig a mező pillanatnyi értékét fogja tartalmazni, felülírása pedig a mező szerkesztésével egyenértékű. Ám a leírás szerint bizonyos esetekben nem javallott írni ezt a változót.
A `-validate` segítségével megadható egy, a mező szerkesztésével kapcsolatos esemény, amely bekövetkezésekor a `-validatecommand` által tárolt függvény meghívásra kerül. Ha ez utóbbi függvény igazat ad vissza, a szerkesztés végbemegy, ha hamisat, akkor nem. Ezzel szerkesztés közben koppinthatunk a felhasználó orrára, ha például egy csak számokat váró mezőbe betűket pötyögne. Itt csak arra használjuk ezt a lehetőséget, hogy töröljük az üdvözlő üzenetet, ha a felhasználó szerkeszti a nevet.
Azt, hogy az alsó keretben szereplő címkénkben mindig a helyes üdvözlő üzenet szerepeljen, egy gombhoz rendelt függvénnyel oldjuk meg. A függvényünk neve `printhello ()`. Továbbá használunk egy segédváltozót, melyet a `-textvariable` segítségével hozzárendelünk a címkéhez.

A címke először nem tartalmaz semmit, hiszen az objektum létrehozásakor nem adtunk meg alapértéket. Először akkor kaphat értéket, amikor a felhasználó a Szia! feliratú gombra kattint. Ekkor meghívásra kerül a `printhello`, amely, miután ellenőrizte, hogy a felső beviteli mezőben szerepel-e adat, a segédváltozónak értékül adja a megfelelő karakterláncot. A kilépést egy kicsit furcsán oldottam meg. Nem használtam a gomb `-command` beállítását, ehelyett meghívtam a `bind ()` tagfüggvényt. Utóbbi szintén eseményekhez rendel függvényeket, ám sokkal általánosabb. Különböző egérgombokhoz, egyszeres- és kétszeres kattintáshoz, billentyűkombinációkhoz más- és más függvényeket rendelhetünk. Itt az eredmény ugyanaz, mintha a `-command`-ot használtam volna, ám szerettem volna bemutatni, hogyan működik a `bind ()`. Első paraméterként meg kell adni az eseményt. Ez <> jelek közötti egyszerű események sorozata is lehet (gondolj az Emacs-ra). A Button-1 a bal egérgomb (ezt jelenti az 1-es) egyszeri megnyomását jelenti. Második paramétere pedig az a függvény, amelyet az esemény kiváltásakor meg kell hívni.
Rengeteg dolgot hallgattam el, amiért bocsánatot kérek, de sajnos a helyszűke nagy úr. Ajánlom figyelmedbe ***Steve Lidie, Nancy Walsh: Mastering Perl/Tk*** című könyvét, ami egy nagyon jól használható leírás. Az említett súgóoldalak pedig töménytelen információval szolgálnak az egyes objektum tagfüggvényeinek paraméterezéséről.

Fülöp Balázs
admin@guardware.com