Qt Pencereleri

Kaya Oğuz

<kaya@kuzeykutbu.org>

Pencere var, pencere var

Daha önce okumadıysanız, belgeler sayfasından tekrar bir Qt'ye Giriş belgesine bakın. Orada pencerelere bir giriş yapmıştık. Ben burada yine bir özet geçeceğim.

Grafik kullanıcı arayüzü programlamada TOP_LEVEL_WINDOW ya da MainWindow olarak geçen esas pencereler vardır. Bu pencereler, bu belgeyi okuduğunuz internet tarayıcısı gibi programların "ana" pencereleridir. Bu pencere içinde bütün işlemler olur ve genelde bu pencereyi kapattığınızda program da kapanıverir.

Diğer pencerelere Dialog Window deniyor, aynı şekilde Diyalog pencereleri olarak çevirebiliriz. Diyalog penceresi, adından da anlaşılacağı gibi bizimle kısa bir etkileşime girip kapanan pencerelerdir. Bunlara örnek menüden Dosya ve arkasından Aç dediğimiz zaman çıkan "Dosya Açma" penceresi. Burada işinizi gördükten sonra program kaldığı yerden devam eder.

Bu diyalog pencereleri de kendi aralarında ikiye ayrılırlar. Bunlardan ilki, Modal denen ve pencere işini bitirene kadar alttaki pencerelere erişime izin vermeyen pencerelerdir. "Dosya Aç" pencereleri genelde bu şekildedir. Diğerleri ise açıkken diğer pencerelere bir kısıtlama getirilmez. Bunlara da Modeless ya da Non-Modal pencereler denir. Örneğin bir metin düzenleyicideki "Ara" penceresi gibi. Bu Ara penceresi bir kelimeyi ya da ifadeyi ararken alttaki ana pencerede düzenlemeye izin verir...

Şimdi pencerelerle nasıl baş edeceğimizi öğrenelim.

Designer Dosyaları

Pencere tasarımı vs. işlemler için elle kod yazmanıza gerek yok. Designer denen araçla istediğiniz arayüzü hazırlayıp kaydedin. Uzantısı "ui" olan bu dosyalar basit birer XML dosyalarıdır. Qt'nin bizden habersiz çalışan uic (ui compiler, daha doğrusu user interface compiler) komutu bu XML dosyasını bir başlık, yani header dosyasına dönüştürür.

Bunu elle yapabilirsiniz, yani komutu verip bir header dosyası alabilirsiniz. Ama bunun yerine qmake kullanarak proje yönetimi ile beraber gitmeniz tavsiye edilir. Böylece bu dosyalarınız qmake tarafından otomatik olarak algılanıp uic'den geçirileceklerdir.

Designer ile üç tür pencere yaratabilirsiniz: Ana Pencereler (MainWindow), Diyalog pencereleri (Dialog) ve parçacıklar (Widget)... Ana pencere ve diyalog pencerelerini az çok biliyorsunuz ama parçacıklar ne işimize yarayacak? Parçacıkları dinamik sekme yaratmada kullanacağız. İlerleyen konularda değineceğiz :)

Orta ve küçük ölçekli programlarınız genelde bir ana pencere ve onun etrafında çalışan diyalog pencerelerinden oluşur. Bu yüzden öncelikle ana pencerenizi tasarlamanız ve yeni bir dizin altında kaydetmeniz, arkasından da ihtiyacınız oldukça diyalog pencerelerini yaratmanız izlenecek en olağan yollardan biridir. Var olan pencerelerle çalışırken qmake -project, qmake, make üçlüsünün sadece son adımını çalıştırmanız yetecekken, yeni bir pencere eklediğinizde sorun çıkabilir. Bunun için qmake üçlüsünden önce "make clean" diyerek ortamı temizleyin. Arkasından da Makefile ve ".pro" dosyanızı silin. qmake üçlüsünü tekrar edin ve tekrar sorunsuz derlenecektir.

Mama Kin

Mama Kin, Aerosmith'in güzel bir parçasıdır, Guns 'n Roses da "Lies" albümünde söyler bu parçayı. Sözlerinde "Keep in touch with mama kin" derler, yani annenizin akrabaları ile iletişimi koparmayın gibisinden.

Ne alakası var derseniz, pencere yaratmak için torunlara ihtiyacınız vardır. Designer ile oluşturduğunuz ui dosyaları uic'den geçtikten sonra bir header dosyası oluyor demiştik. Bu header dosyalarının içinde QMainWindow'dan türetilmiş bir sınıf vardır. Bu sınıf Ui denen bir namespace altındadır.

Şimdi yukarıda geçen türetilme, header, namespace gibi kelimeler size yabancı geliyorsa şöyle söyleyelim, o dosyanın içinde QMainWindow'un bir çocuğu var. Çocuk annesine pek benzemiyor, ondan daha renkli. Çocuğun da kendi bir çocuğu olunca daha da güzel olacak. İşte bu son çocuk bizimdir :)

Biz uic'den geçtikten sonra oluşan header dosyasındaki sınıftan ve QMainWindow'un kendinden türeteceğiz penceremizi...

```
#include <QMainWindow>
#include "ui_pencere.h"

class AnaPencere:public QMainWindow, Ui::MainWindow
{
        Q_OBJECT
        public:
        AnaPencere():QMainWindow()
        {
                setupUi(this);

        }
};
```

Burada pencere.ui dosyasından oluşacak olan ui_pencere.h dosyasını include ediyoruz önce. Kendi sınıfımızın adı AnaPencere ve bu sınıfı QMainWindow ile ui_pencere.h dosyasındaki Ui namespace'i altındaki MainWindow'dan türetiyoruz... Buraya kadar sorun yok. Ondan sonra kurucu (constructor) içinde, yine ui_pencere.h dosyasında Ui::MainWindow'un bir metodu olan setupUi'yi çağırıyoruz, bu Designer'da oluşturduğumuz tasarımın yapılmasını sağlıyor. Tabii bu arada kurucuyu çağırırken, QMainWindow'un kurucusunu da çalıştırıyoruz (

`AnaPencere():QMainWindow()`).

İşte penceremiz hazır! Aynı Designer'da yarattığımız şekilde... Buna biraz işlevsellik kazandırmak istiyorsak biraz sinyal / slot ilişkileri kurmamız yeterli olacaktır.

Dialog Pencereleri

Tek pencere ile sınırlı kalmak elbette sıkıcı. Bu yüzden bu ana pencereye biraz sinyal slot ile yeni işlevler kazandırıp yeni bir pencere açtıralım. Hatta, pencereden değerler alıp kullanalım!

Üstte ana pencereyi hiç göstermedik, ben size bir ekran görüntüsü sunayım:

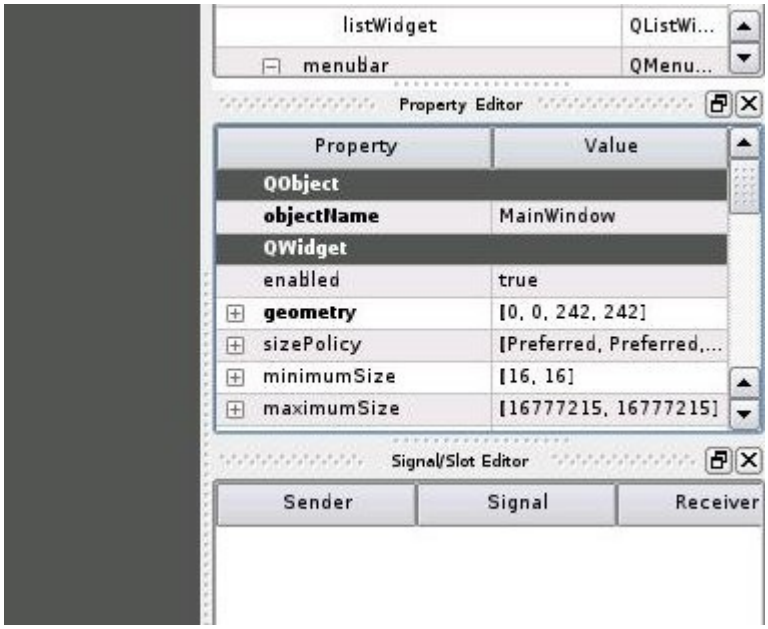
Figure 1. Pencere Demo

Burada gördüğünüz üzere bir "Yeni Pencere" QAction'ımız var. Menüye yazdığınız her satır bir QAction oluyor :) Bunun dışında beyaz alan bir QListWidget. Yani bir liste... Şimdi "Yeni Pencere"ye tıklayacağız, arkasından bir pencere açılacak. Oraya bir bilgi gireceğiz, girdiğimiz bilgi de listeye eklenecek :) Kolay değil mi? Basit bile olsa aslında birçok işlem bu şekilde yapılıyor. QAction'ları doldurmadan önce, diyalog penceremizi tasarlayalım:

Figure 2. Diyalog Demo

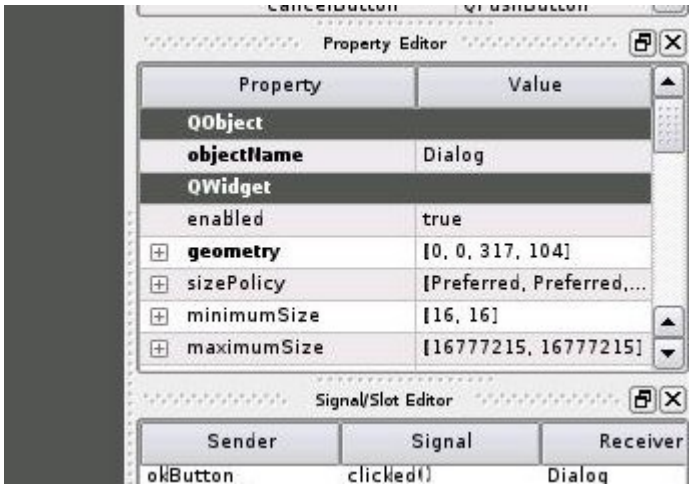
Buraya kadar her şey normal. Ama bu iki pencere hala Designer'da açıkken, size bu pencerelerin isimlerini göstermek istiyorum:

Figure 3. Designer'da Ana Pencere Özellikleri



Bakın, burada objectName olarak MainWindow duruyor. Bu, Ui::MainWindow ifadesindeki MainWindow. Şimdi diyalog penceremize bakalım:

Figure 4. Designer'da Diyalog Penceresi Özellikleri



Burada gördüğünüz gibi, bunun adı Dialog. Yani Ui::Dialog olacak. Ama birden fazla pencere kullanacaksanız bu isimler çakışacaktır. Bu yüzden, eğer çok form olacaksa bunlara yeni isimler vermeniz iyi olur. Burada başka penceremiz olmayacak, o yüzden öntanımlı bu değerler işimizi görecektir. Ama değiştirmek isterseniz, Designer'da pencerenin boş bir yerine sağ tıklayıp "Change Object Name" diyerek yeni ismi girebilirsiniz.

Pekala, şimdi diyalog penceremizi kaydettik, diyalog.ui, şimdi bir temizlik yapalım (make clean) ve diyalog.h dosyamızı yazalım:

```
#include <QDialog>
#include "ui_diyalog.h"

class dialog:public QDialog, Ui::Dialog
{
        Q_OBJECT
        public:
        dialog(QWidget *parent):QDialog(parent)
        {
                setupUi(this);
```

```
        }
};
```

Burada her şey sanki MainWindowdaki gibi. QMainWindow yerine QDialog'u ekledik, dikkat edin, bu sefer QDialog ve Ui::Dialog'tan türettik. Burada Ui::Dialog ifadesindeki Dialog, üstte ekran görüntüsündeki Dialog. Eğer onu formPencere yapsaydık, Ui::formPencere diyecektik ;)

Bunun dışında dikkat etmeniz gereken bir diğer nokta QWidget *parent ifadesi. Diyalog pencerelerinin bir parent nesnesi olmalı. QMainWindow ve QDialog, QWidget'tan türedikleri için biz oraya bizim AnaPencere'yi koyacağız. QDialog'un kurucusu (constructor) da bu QWidget'ı alır :)

Hemen hemen iki pencereyi de yarattık, ama bu pencereler nasıl açılacak. Elbette bu iş için ana pencereye biraz işlem yapmamız gerekecek. "Yeni Pencere" QAction'ını bir slot'a bağlayalım ve pencereyi açtıralım. Yeni anaPencere.h'ımız şu şekilde:

```
#include <QMainWindow>
#include "ui_pencere.h"
#include "diyalog.h"

class AnaPencere:public QMainWindow, Ui::MainWindow
{
        Q_OBJECT
        public:
        AnaPencere():QMainWindow()
        {
                setupUi(this);

                connect(actionYeniPencere, SIGNAL(activated()), this,
SLOT(slotPencere()));

        }

        public slots:
        void slotPencere()
        {
                diyalog *yeni = new diyalog(this);
                yeni->exec();
        }
};
```

Pekiii! Şimdi burada ne oldu? Öncelikle diyalog.h dosyasını include etmeyi unutmadık (tamam ilk başta unuttum gene, ama fark ettim hemen :)) Ondan sonra "actionYeniPencere"mizi, slotPencere denen hemen aşağıya yazdığım metoda bağladık. O metod napıyor peki? Basitçe diyalog denen sınıfımızdan (kendisi diyalog.h içindeydi) bir yeni pointer yaratıyor, adı yeni :) Bakın yaratırken new dialog(**this**) ifadesini kullandık. Yani this dediğimiz, bir QMainWindow gönderdik içine. O zaman ne oldu? Diyalog penceremiz ana penceremizin tam ortasında çıktı. Burada this yerine 0 ya da null derseniz ekranın ortasında çıkar. Ama pencereye ait diyalog pencerelerinin pencerenin ortasında çıkması daha iyidir :)

Peki bu pencere hiçbir işlem yapmıyor. Belirdikten sonra Kaydet / iptal neye basarsanız basın kapanıyor. Amacımız neydi? O QLineEdit'e bir şeyler yazdırıp, QListWidget'a eklemek!

O zaman size biraz QDialog'un yapabileceklerinden bahsedeyim. Bu iki düğme

Designer ile hazırlandığında iki tane Slot'a bağlılar aslında. Biri, Kaydet daha doğrusu, accept() denen slota bağlı, diğeri de, İptal yani, reject() denen slot'a bağlı. Bu iki slot QDialog'un slotları. Eğer kullanıcı Kaydet'e basarsa accept() slotu anaPencere.h içindeki slotPencere() fonksiyonuna QDialog::Accepted değerini döndürüyor. Orada yeni->exec() herhangi bir değere eşitlenmemiş, ama eşitlersek durumu, yani kullanıcının accept / reject ettiğini öğrenebiliriz.

Bütün dosya yerine değişen yer olan void slotPencere() metodunu tekrar inceleyelim:

```
void slotPencere()
	{
		diyalog *yeni = new diyalog(this);
		if (yeni->exec() == QDialog::Accepted)
		{
			// demek ki kabul etmiş!
		}
	}
```

Harika! accept() ifadesini artık kullanabiliyoruz, ama değeri nasıl alacağız? Eğer yeni->lineEdit->text() gibi bir düşünceniz varsa böyle değil. Çünkü QDialog'un accept() ifadesinden sonra ona ulaşamıyoruz. Bunun yerine diyalog.h dosyasındaki sınıfımıza bir değişken daha ekliyoruz, adı QString yazilan:

```
#include <QDialog>
#include "ui_diyalog.h"

class diyalog:public QDialog, Ui::Dialog
{
	Q_OBJECT
	public:
	QString yazilan;
	diyalog(QWidget *parent):QDialog(parent)
	{
		setupUi(this);
		connect(okButton, SIGNAL(clicked()), this, SLOT(slotKaydet()));
	}

	public slots:
	void slotKaydet()
	{
		yazilan = lineEdit->text();
	}

};
```

Neler oldu şimdi? okButton'u zaten bir slot'a bağlıydı hani? Evet, hala accept() slotuna bağlı. Ama biz onu bir de bu slot'a, slotKaydet()'e bağladık. Bu slot ile yazilan değişkenine lineEdit içindeki metni aldık! Artık bunu void slotPencere() içinde kullanabiliriz:

```
void slotPencere()
	{
		diyalog *yeni = new diyalog(this);
		if (yeni->exec() == QDialog::Accepted)
		{
			new QListWidgetItem(yeni->yazilan, listWidget);
		}
	}
```

Vay! Ne kadar basitmiş! :) QListWidget'a yeni bir satır eklemek için bir

QListWidgetItem kullanıyoruz. Detaylı bilgi için Assistant'a bakabilirsiniz. Burada new ile aslında yeni bir pointer dönüyor ama ona ihtiyacımız yok, o yüzden bıraktık. yeni->lineEdit'e ulaşamasak bile artık yeni->yazilan'a ulaşabiliyoruz ve işte listeye yazdığımız ifade eklendi.

Burada elbette kontrolleri de yapmak lazım, mesela boş bırakmışsa "lütfen bir metin giriniz" vs. yazmak lazım değil mi? Onları da yapalım. O zaman ilk işimiz okButton'dan accept() slot'unu kaldırmak olmalı. Bu iş için Designer'ı kullanabilirsiniz. Hemen oradaki SIGNAL SLOT EDITOR altında okButton, clicked, Dialog, accept satırını seçip kaldırın. Kaydedin. Artık accept olayını kendimiz yazacağız. Tek yapmamız gereken daha önce yazdığımız slotKaydet() metodunun sonunda bir kontrol edip accept() ifadesini yazmak. slotKaydet() metodunu tekrar yazalım:

```
void slotKaydet()
        {
                if (lineEdit->text() == "" || lineEdit->text() == QString::null)
                {
                        QMessageBox::critical(this, "Hata!",
QString::fromUtf8("Lütfen boş bırakmayınız!"));
                        return;
                }
                yazilan = lineEdit->text();
                accept();
        }
```

Burada, en başa, `#include <QDialog>` ifadesinden sonra `#include <QMessageBox>` eklemeyi unutmayın! Burada artık accept() slotumuz slotKaydet olduğu için güvenle kontrol ediyoruz lineEdit->text() ifadesini. Eğer boşsa veya null ise bir hata mesajı verip dönüyoruz. Değilse, yazilan değişkenine yazıp, accept() ile işlemi bitiyoruz :)

Artık bütün işlevleri yerine getiren basit ve etkili bir pencere / diyalog ikilimiz var, hayırlı uğurlu olsun! Sekmelerle ilgili yazıyı artık bir sonrakine anlatırım :) Bütün kaynak kodları belgeler sayfasından indirebilirsiniz.