Swift ile iOS Programlama temelleri

Özgen İmrağ

İçindekiler

1

Swift Programlama Dili ve XCode Ortamına Giriş

Bu Bölümde

Bu bölümde

Swift programlama dilinin teknik konularına girmeyeceğiz. Bunun yerine kitabın genelindeki bilgilerden nasıl daha iyi yararlanabileceğinize dair ipuçları vereceğim ve bazı açıklamalarda bulunacağım. Ayrıca Swift dilinin diğer programlama dilleri ile benzer ve farklı yanlarını genel hatları ile açıklayacağım.

SWIFT OBJECTIVE-C VE DİĞER DİLLER

Bir çoğunuzun Swift programlama dilinden önce Objective-C dilinin kullanıldığını bildiğini tahmin ediyorum. Objective-C dili ise isminden de dikkat edeceğiniz üzere C dilinden gelmektedir. Bu Objective-C dilini her ne kadar güçlü yapsa da hem öğrenilmesinin zorluğu hem de programcıların kodlama esnasında yaşadığı yönetim zorlukları oldukça fazlaydı. Kısmen de olsa Apple firması bu sebeplerden dolayı 2014 yılında Swift programlama dilini duyurdu.

Apple firmasının Objective C dilinden Swift diline geçmesine sebep olan diğer etmenleri anlayabilmemiz için C programlama dili hakkında biraz bilgi sahibi olmalıyız. C programlama dili günümüzde kullanılan bir çok programlama dilinin atası konumundadır. Bunlardan en bilindikleri JAVA ve Objective-C dilleridir. Tabi bu dillerden ayrı başka programlama dillerini de saymak mümkün.

C, her ne kadar çok güçlü bir programlama dili olsa da programcılar tarafından öğrenmesi kolay değildir. Ayrıca C ile büyük projeler geliştirmek ciddi uzmanlık gerektirecek kadar zor. Bunun sebebi ise hassas bir dil olması ve hata yönetiminin zor olmasıdır. Bu zorlukların üstesinden gelebilmek için Objective-C dilinde yapısal programlama kullanılmaktadır. Yapısal programlama problemlerin ufak parçalara bölünerek giderilmesi mantığına dayanan bir yapıdır. Günümüzdeki bir çok programlama dili hala yapısal programlamayı kullanmaktadır. Android platformu için geliştirilen uygulamalar ve Google Play üzerindeki uygulama zenginliği her geçen gün artarken App Store'un bu zenginliği yakalayamaması Apple firmasını geliştiriciler için bazı kolaylıklar sunmaya itti. Bunun sonucunda da biraz önce de belirttiğim üzere **Swift Programlama** dili ortaya çıktı.

> **NOT** Yapısal programlamanın en önemli yapı taşlarından birisi olan fonksiyonları kitabın son bölümlerine doğru öğreneceksiniz. Fonskiyonlar konusunu öğrendiğinizde yapısal programlamanın avantajlarını kendininiz anlayacaksınız.

Swift programlama dilinin ortaya çıkması her ne kadar bazı sorunları beraberinde getirse de Apple'ın bu konuda çözümleri oldukça esnekti. Öncelikle Swift dili Objective-C dilinin kütüphanelerini kullanabiliyor. Bu da en büyük sorunun çözümü niteliğindedir. Ayrıca her ne kadar Swift ile Objective-C dili çok farklıymış gibi görünse de mantıksal benzerlikler Apple adına ciddi mühendislik becerileridir. Swift programlama dili piyasaya çıktığında yeni bir programlama dili olmasından ötürü bazı korkuları da beraberinde getirdi. Yeni bir programlama dilinin güçlü olmaması konusunda insanların çekinceleri vardı. Fakat kısa bir süre sonra bu korku yerini güvene bıraktı.

Hem Swift dilinin güç ve esneklik olarak beklentilerin üstünde olması hem de Objective-C dilinin kütüphanelerini kullanabilmesi geliştiricileri mutlu etmeyi başardı. Apple mühendislerinin Swift üzerindeki başka bir becerisi de geliştiricinin Objective-C dilinde geliştirdiği modülleri Swift dilinde de problemsiz bir şekilde kullanabilmesidir.

Objective-C ile Swift dilinin en büyük ortaklığı belki de sadece iOS ve OS X platformları için uygulama geliştirilebilmesidir. Aslında bu biraz Apple firmasının dijital ortamdaki ırkçılığından kaynaklanmaktadır. Tabi buna anlam vermek az da olsa mümkün. Çünkü Swift dili ile diğer mobil platform için uygulama geliştirmemizi sağlayan diller arasında belirgin farklar var. Bunlardan benim en çok gözüme çarpan farklılık tasarımdaki farklılıklardır. Swift programlama dilinde geliştirdiğiniz uygulamalara harika görsellikler katabiliyorsunuz. Üstelik çok fazla zorlanmaya veya harici kütüphaneler kullanmanıza gerek kalmadan. Oysa günümüzün en güçlü dillerinden birisi olan Java'da tasarım konusunda biraz kendinizi kısıtlanmış hissedebilirsiniz.

Swift programlama dilini kullanmanızın bir başka avantajı ise hedef kitlenizdir. Yani iPhone veya iPad kullanıcıları hatta MAC kullanıcıları Windows veya Android kullanan kişilere nazaran uygulamalara daha fazla para harcamaktadır. Bu da sizin daha fazla kazanacağınız anlamına gelmektedir.

Öncelikle aşağıdaki tablodan programlama dillerinin 2014 ve 2015 yılına ait kullanım oranlarına bakalım. Tabi bu listede Swift dili yok. Swift yerine referans alacağımız dil Objective-C Programlama dilidir.

Feb 2015	Feb 2014	Change	Programming Language	Ratings	Change
1	1		C	16.488%	-1.85%
2	2		Java	15.345%	-1.97%
3	4	^	C++	6.612%	-0.28%
4	3	v	Objective-C	6.024%	-5.32%
5	5		C#	5.738%	-0.71%
6	9	^	JavaScript	3.514%	+1.58%
7	6	v	PHP	3.170%	-1.05%
8	8		Python	2.882%	+0.72%
9	10	^	Visual Basic .NET	2.026%	+0.23%
10	-	^^	Visual Basic	1.718%	+1.72%
11	20	^^	Delphi/Object Pascal	1.574%	+1.05%
12	13	^	Perl	1.390%	+0.50%
13	15	^	PL/SQL	1.263%	+0.66%
14	16	^	F#	1.179%	+0.59%
15	11	vv	Transact-SQL	1.124%	-0.54%

8

XCode Proje Ortamına Giriş

Bu Bölümde

Şimdiye kadar olan çalışmalarımızı hep playground proje ortamında yaptık. Bu bölümden itibaren artık XCode Proje türü ile çalışarak görsel uygulamalar geliştireceğiz. XCode projeleri ile geliştirdiğimiz uygulamalar mobil cihazlar dediğimiz iPhone ve iPad aygıtlarında çalışabilmektedir. Tabi bunun için bazı prosedürler gerekmektedir.

Bu bölümle birlikte gerekli prosedürleri öğrenerek ilerleyeceğiz.

YENİ PROJE OLUŞTURMA

Şimdiye kadar yaptığımız örnekleri playground ortamında yaptık. Bu bölümle birlikte artık gerçek proje ortamına geçiyoruz. Gerçek proje ortamına geçtiğimizde karşımıza çıkacak iki büyük farklılıktan bir tanesi görsel unsurlarla sıkça muhatap olmamız, diğeri ise birden fazla dosya ve modül ile muhatap olma zorunluluğumuzdur. Yeni bir proje oluşturduğumuz andan itibaren bizi playground ortamından daha karmaşık gibi görünen bir ortam karşılayacaktır. Sırayla yeni bir proje oluşturma ve proje oluşturduktan sonraki basamakları inceleyerek ilerleyeceğiz.

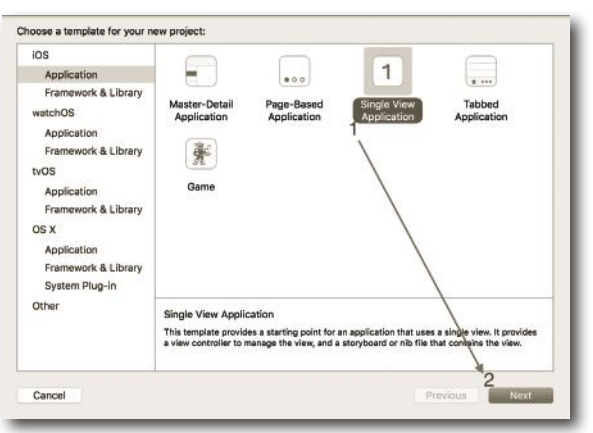
XCode programını açtığımızda bizi karşılayan ekranda **Create a new XCode project** seçeneğine tıklayarak yeni bir proje oluşturmak için ilk adımı atalım.

Karşımıza çıkan ekranda farklı seçenekler bulunmaktadır. Bu seçenekler içinde en fazla **Single View Application** ile muhatap olacağız. Eğer farkettiyseniz sol taraftaki panelde **tvOS**, **watchOS**, **OS X** gibi bölümler kategoriler bulunmaktadır. Bu kategorilerde geliştireceğiniz uygulamalar mobil cihazlar için şimdilik uygun değildir. Tabi swift dili yeni ve fazlasıyla gelişmekte olan bir dil olduğundan gelecekte ne oladağını bilemeyiz. Sol taraftaki kategorilere dönecek olursak **TV** ile Apple destekli televizyon sistemleri için, **watchOS** ile yine Apple destekli akıllı kol saatleri için ve OS X ile Apple firmasının Mac bilgisayarlarında (MacBook, iMac vb.) kullandığı masaüstü işletim sistemi olan OS X için uygulamalar geliştirebilirsiniz.

iOS kategorisi altında 5 seçenek yer almaktadır. Bu seçeneklerin ne işe yaradığını anladıktan sonra bir sonraki aşamaya geçeceğiz.

1. **Master-Detail Application:** Birden fazla sayfayla çalışmak için Apple mobil platformlar bize bir çok seçenek sunmaktadır. Bunlardan birisi Master-Detail Application'dur. Bu proje türünde sol taraftan açılan panelde seçenekler sunularak farklı sayfalar arasında geçiş imkanı sağlamaktadır. Sonraki bölümlerde farklı sayfalardan kastettiğim yapının ne olduğunu ve nasıl yapıldığını öğreneceğiz.
2. **Page-Based Application:** İster Android kullanın istereniz iOS kullanın ana ekrandayken ekranı sağa sola sürükleyerek farklı sayfalara geçiş yapabiliyoruz. Bu yapının geliştiriciler yani bizler tarafından kolayca uygulamaya dahil edilebilmesi açısından **Page-Based Application** seçeneği kullanılmaktadır.
3. **Single View Application:** En temel proje türüdür. Oluşturduğumuz bu proje türünde karşımıza tek sayfadan (view) oluşan bir yapı çıkmaktadır. Geliştirici istediği gibi bu view yapısını özelleştirebilir veya yeni view yapıları ekleyerek bu view yapıları arasında etkileşim kurabilir.
4. **Tabbed Application:** Sekmeli yapıya ve birden fazla sayfayla çalışma ortamına alternatiflerden birisi de bu proje türüdür. Aslında başka platformlarda sekmeli yapı tercihinde bu proje türü başı çekse de iOS ortamındaki diğer estetik türlerden dolayı kimi zaman ikinci plana düşebilmektedir. Fakat bu **Tabbed Application** seçeneğinin az kullanıldığı anlamına gelmemektedir.
5. **Game:** Eski XCode sürümlerinde **SpriteKit Game** olarak adlandırılan bu proje türü ile Apple'ın geliştiriciler için sunduğu oyun motoru ile oyun geliştirmek mümkün. Tabi oyun motoru dediğimde aklınıza **Unity** gibi güçlü motorler gelebilir. Apple'ın motoru ile Unity oyun motoru arasında ciddi bir güç farkı olduğunu unutmayalım. Unity günümüzde kullanılan en güçlü oyun motorlarından birisidir.

Biz **Single View Application** seçeneğini seçerek **Next** butonuna tıklayalım.



10

Storyboard ve View Controller

Bu Bölümde

İster mobil olsun ister masaüstü olsun programlarda birden fazla form ile çalışmamız gerekebilir.

Bu bölümde swift programlama dilinde form veya başka bir deyişle sayfalarla çalışma olayına detaylıca inceleyeceğiz. Ayrıca oluşturulan formların swift dilindeki karşılığına ve bu formlar arasında geçiş yapma, veri aktarımı gibi konuları inceleyerek daha esnek uygulamalar geliştirmek için sağlam adımlar atacağız.

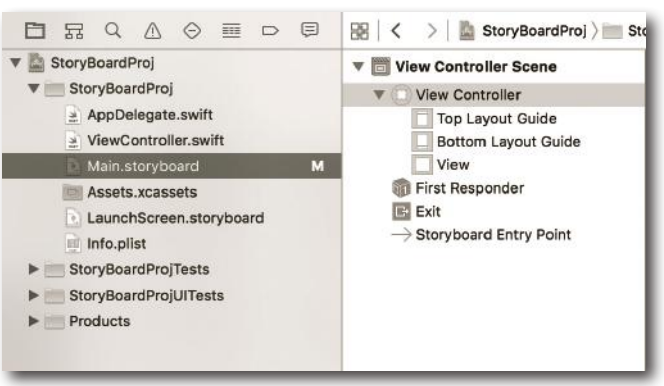
STORYBOARD VE BİZE SUNDUKLARI

Giriş kısmını okuduysanız bu bölümde göreceğimiz kavramların başında form gelmektedir. İşin aslına bakarsanız swift dilinde veya apple platformlarda form yerine View Controller kullanılmaktadır. Şimdiye kadar yaptığımız bütün örneklerin tamamında tek View Controller ile çalıştık. Label, button vb. kontrollerimizi View Controller üzerine ekleyerek işlemlerimizi yaptık. Bunların yanı sıra bir de **storyboard** diye bir kavram ile karşı karşıyayız.

Storyboard, projemizde kullandığımız View Controller nesnelerimizi üzerinde barındırıan düzenektir. Nasıl ki uygulamamıza bir button eklediğimizde bu buttonu View Controller kendi üzerinde barındırıyordu, eklediğimiz View Controller nesnelerini de storyboard kendi üzerinde barındırmaktadır. Ayrıca storyboard dediğimiz düzenin bize sunduğu ufak, fakat kullanışlı özelliklerde bulunmaktadır.

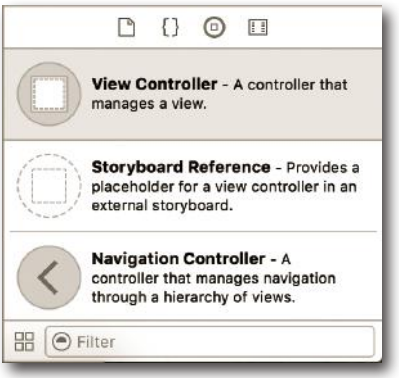
VIEW CONTROLLER

Şimdi storyboard dediğimiz düzeni biraz inceleyelim. Ekranın sol tarafındaki panel ile daha önceki bölümlerde sıkça muhatap olduk. Bu panelde bulunan **Main.storyboard** isimli bölümü açarak incelemeye başlayalım. Storyboard yapısını bir deftere benzetebiliriz. Birden fazla sayfayı storyboard üzerinde barındırarark bu sayfaları istediğimiz gibi özelleştirebiliriz.

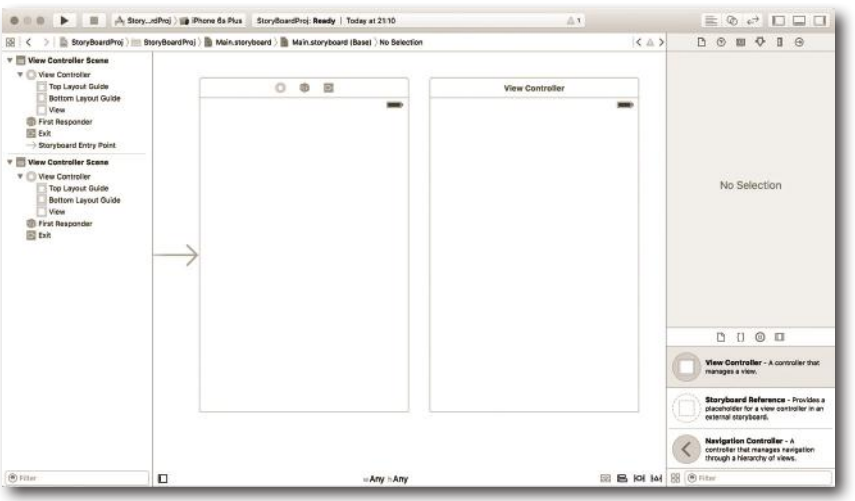


Main.storyboard'a tıkladığımızda projemizdeki storyboard ile karşlaşıyoruz. Storyboard üzerinde ise bir tane View Controller bulunmaktadır. Şimdi storyboard üzerine bir tane daha View Controller ekleyerek birden fazla View Controller üzerinde çalışalım. Bu işlemler View Controller ve Storyboard mantığını anlamamaızda işimizi kolaylaştıracaktır.

View controller eklemek için button vb. komponentleri eklediğimiz paneli kullanabiliriz.



Storyboard üzerine View Controller ekledikten sonra ekranımızdaki görüntü aşağı yukarı aşağıdaki gibi olacaktır.



Burada dikkatinizi çekmek istediğim nokta sol taraftaki View Controller. Sebebi ise sol tarafta bulunan View Controller üzerinde bulunan **ok** işareti bulunmaktadır.

15

Ekran Görüntüsü Alma ve Galeriye Resim Ekleme

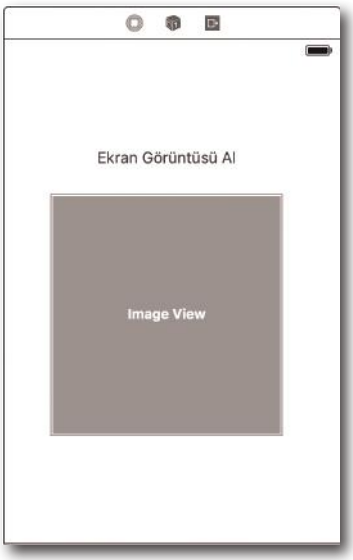
Bu Bölümde

Bir önceki bölümde galeriden ve webden resim çekme işleminin nasıl olduğunu öğrenmiştik. Webden çektiğimiz resimleri her ne kadar View Controller üzerinde görüntüleyebilsek de kaydetmediğimizden dolayı uygulama kapandığında resmin hafızadan silinmesi söz konusuydu.

Bu bölümde hem ekranın anlık görüntüsünü almayı hem de aldığımız ekran görüntüsünü veya webden çektiğimiz resmi cihaza nasıl kaydedeceğimizi öğreneceğiz.

EKRAN GÖRÜNTÜSÜ ALMA

Ekran görüntüsü almayı öğrenmek View Controller üzerine bir tane button bir tane de Image View ekleyelim. Eklediğimiz Image View için outlet button için ise action oluşturalım. Ayrıca aldığımız ekran görüntüsünün daha anlaşılır olabilmesi için isterseniz View Controller üzerine bir kaç tane daha Label vb. komponent ekleyebilirsiniz.



Button için oluşturduğumuz `action` fonksiyonunun içine aşağıdaki kodları yazarak başlayalım. Aynı zamanda kodların ne işe yaradığına bakalım.

```
var ekran:UIWindow = UIApplication.sharedApplication().keyWindow!
ekran = (UIApplication.sharedApplication().windows[0] as? UIWindow)!
```

Yukarıdaki kod ile ekran isminde bir nesne oluşturduk. Bu nesnenin türünü `UIWindow` olarak belirledik. İlk satırda uygulamamızın cihazın ekran görüntüsünü kullanabilmesi için gerekli ortamı hazırlıyoruz. İkinci satırda ise `windows[0]` ile en üstte bulunan sayfanı (formu) oluşturduğumuz ekran değişkenine atıyoruz.

```
UIGraphicsBeginImageContextWithOptions(ekran.frame.size, ekran.opaque, 0.0)
ekran.layer.renderInContext(UIGraphicsGetCurrentContext()!)
var resim = UIGraphicsGetImageFromCurrentImageContext()
UIGraphicsEndImageContext()
```

Yukarıdaki kodlarda bilinmesi gereken en genel noktalardan birisi şudur: `UIGraphicsBeginImageContextWithOptions` metodu ile grafiksel bir işlemi başlatmış oluyoruz. Ayrıca bu metot içerisine girdiğimiz parametreler ile ekranın hangi boyutları arasında çalışacağımızı vs. belirtebiliyoruz. `UIGraphicsEndImageContext` komutu ile grafiksel işlemi bitirmiş oluyoruz. Yani yapacağımız grafiksel işlemleri bu iki komut arasına yazmamız gerekmektedir. `UIGraphicsEndImageContext` komutunu yazmadığınızda, bazı durumlarda hata almayabilirsiniz fakat uygulamanız düzgün çalışmayacaktır. Yukarıdaki kodları çalıştıktan sonra **Resim** isimli değişkenin içinde ekranın görüntüsü olacaktır. Artık bize düşen bu ekran görüntüsünü aşağıdaki gibi Image View üstünde görüntülemek olacaktır.

```
imageView1.image = resim
```

Benim View Controller üzerine sonradan yerleştirdiğim komponentler ile ekranın görüntüsü aşağıdaki gibidir:

Ekran görüntüsü almak için button'a tıkladığımda ise Image View üzerinde ekranın görüntüsünü göreceğiz.

20

Fonksiyonlar

Bu Bölümde

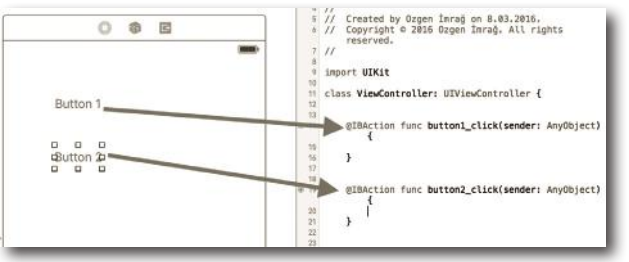
Nesne tabanlı programlamanın doğrudan olmasa da bir parçası olan bir konuya başlıyoruz.

Bu bölümde fonksiyonları en başından olabildiğince ileri seviyeye kadar öğreneceğiz. Bu bölümde neler öğreneceğimizden ziyade öğrendiklerimizin bize ne faydası olacağına kısaca değinmek istiyorum. Özellikle büyük projelerde, proje ilerledikçe kimi zaman kodlar içinden çıkılmaz bir hal alır. Böyle durumlarla karşılaşmamak için kodlamada olabildiğince temiz kod yazmalıyız. Bunun için de gerekli durumlarda fonksiyonlara hatta parametreli fonksiyonlara başvurmalıyız. Hangi durumlarda başvurmamız gerektiğini ise konuyu öğrendiğinizde ve deneyim kazandıkça kendiniz anlayacaksınız.

TEMEL FONKSİYONLAR

Fonksiyonlar için kaba bir açıklama yapacak olursak yapmak istediğimiz bir işi küçük bir program parçası haline getirerek kısayol oluşturur diyebiliriz. Tabi bu açıklama sadece parametresiz fonksiyonlar için geçerlidir. Çünkü işin içine parametreler girdiğinde fonksiyonlar çok daha kullanışlı bir hal alacaktır. Kısa kesmek gerekirse bir çok yerde kullanmanız gereken kodları fonksiyon kullanarak tek bir yere yazarak istediğiniz yerde kullanabilirsiniz.

Şimdi yapacağımız örnek için View Controller üzerine iki tane button yerleştirelim ve button'lar için action oluşturalım.



Örnekte yapmak istediğimiz olay ise her iki button'a tıklayınca da bir alert (*uyarı*) ekrana gelecek. Öncelikle fonksiyon kullanmadan bunu yapalım. Aşağıdaki resimde göreceğiniz üzere aynı kodları iki defa yazmış olduk:

```
import UIKit

class ViewController: UIViewController {


    @IBAction func button1_click(sender: AnyObject)
        {
        let alert = UIAlertView()
        alert.title = "Fonksiyonlar"
        alert.message = "Temel Fonksiyonlar"
        alert.addButtonWithTitle("Tamam")
        alert.show()

    }

    @IBAction func button2_click(sender: AnyObject)
        {
        let alert = UIAlertView()
        alert.title = "Fonksiyonlar"
        alert.message = "Temel Fonksiyonlar"
        alert.addButtonWithTitle("Tamam")
        alert.show()
    }
```

Oysa bu işlemi fonksiyon ile yapmış olsaydık bir kere yazmamız yeterli olacaktı. Nasıl temel bir fonksiyon oluşturacağımızı ve bu örneğimizi fonksiyon ile nasıl yapacağımızı öğrenelim. Aşağıda `alertGoster` adında bir fonksiyon oluşturdum ve alert için gerekli kodları bu fonksiyonun içine yazdım.

```
func alertGoster()
{
        let alert = UIAlertView()
        alert.title = "Fonksiyonlar"
        alert.message = "Temel Fonksiyonlar"
        alert.addButtonWithTitle("Tamam")
        alert.show()
}
```

Artık butonlar için oluşturduğum `action` fonksiyonların içine aynı kodları tekrar tekrar yazmama gerek kalmadı. Bunun yerine `alertGoster` diyerek fonksiyonumuzu çağırmış olduk.

```
import UIKit

class ViewController: UIViewController {

    func alertGoster()
    {
        let alert = UIAlertView()
        alert.title = "Fonksiyonlar"
        alert.message = "Temel Fonksiyonlar"
        alert.addButtonWithTitle("Tamam")
        alert.show()
    }

    @IBAction func button1_click(sender: AnyObject)
        {
        alertGoster()

    }

    @IBAction func button2_click(sender: AnyObject)
        {
        alertGoster()
    }
```

Parametreli Fonksiyonlar

Temel fonksiyonlar başlığı altında yaptığımız örneğe dikkat edecek olursak, diyelim ki alert'in başlık metnini değiştireceğiz. Ayrıca `button1` tıklandığında alert başlığı farklı `button2` tıklandığında da farklı olacaktır. Bunun için iki tane ayrı ayrı fonksiyon oluşturmaya gerek yok. Oluşturacağımız parametreli bir fonksiyon ile sonuca ulaşabiliriz.