ACTIONSCRIPT 3
VE
SINIF KAVRAMI

HAZIRLAYAN: Ö. Faruk GÜNDÜZ (mapacarta.com)

Actionscript nesne yönelimli eventlerle(olaylarla) yönlendirilen bir programlama dilidir. Şimdilik bunun bir şey ifade etmesi gerekmiyor, sadece bu şekilde bilmemiz yeterli.

Kodlama için kullanabileceğimiz Adobe Flash, Flash Builder, FlashDevelop gibi bir çok alternatif mevcuttur. Ben bunların arasından hem kalitesi hem de tamamen ücretsiz olması sebebiyle FlashDevelop'u kullanmayı tercih ediyorum.

Kodlama kısmına geçersek actionscript'te kodları sınıf(class) dosyalarının içerisine yazarız. Sınıf dosyalarını kullanarak nesne örnekleri oluştururuz (başta belirttiğimiz nesne yönelimli kısmı)[1]. Bu noktada sınıflar ve nesneler arasında şu şekilde bir ilişki vardır; sınıf bir taslaktır, bu taslaktan istediğimiz sayıda nesne(obje) oluşturabiliriz. Örnek olarak elimizde bir Taş sınıfı olsun, biz bu sınıftan istediğimiz sayıda taş objesi oluşturabiliriz ve her taş objesinin kendine has özelliklerini (boyutu, rengi, ağırlığı vs.) belirleyebiliriz.

Taş sınıfını ilk oluşturduğumuzda hangi özelliklerin atanması gerektiğini belirleriz. Misal 3 özellik belirledik diyelim; boyutu, rengi, ağırlığı. Bir kaç nesne örneği oluşturduğumuzda:

1. nesne - taş1 : boyut: 10x10 / rengi: kırmızı / ağırlığı: 3 kg
2. nesne - taş2 : boyut: 15x10 / rengi: mavi/ ağırlığı: 4kg;
3. nesne - taş3 : boyut: 20x5 / rengi: sarı / ağırlığı: 2kg;
…. bu şekilde istediğimiz kadar obje oluşturabiliriz.

Peki bu ne işimize yarayacak? Mesela şu şekilde bir oyun geliştirdiğimizi düşünelim; basitçe ekranda hareket ettirdiğimiz bir karakterimiz var. Ekranın 4 tarafından zombiler geliyor, biz de bunlardan kaçmaya çalışıyoruz. Bu noktada bütün zombiler mesela Zombi ismini verdiğimiz bir sınıfın nesne örnekleridir. Her birisinin kendine has bir konumu, hızı vs. özellikleri olacaktır.

[1] Sadece sınıfları kullanmak nesne yönelimli programlama(kısaca OOP)'ye uyduğumuz anlamına gelmez. OOP'nin polymorphism, inheritance gibi çeşitli teknikleri/kuralları vardır. Ancak şimdilik sadece sınıfları kullanarak nesne örnekleri oluşturduğumuzu bilmek yeterli.

Yani oluşturduğumuz Zombi taslağından istediğimiz sayıda farklı özellikleri olan zombiler oluşturabiliriz.

Şimdi gelelim basitçe bir sınıf oluşturmaya. Sınıf dosyasının uzantısı .as'tır. Mesela örnek verdiğimiz Zombi sınıfının ismi Zombi.as şeklinde olabilir. Sınıf dosyasının en üstünde sınıfın dizinini(klasör yolunu gösteren) package kısmı bulunur. Package kısmının sınırları iki süslü parantez ile ({}) belirtilir. Bütün sınıf kodları bu iki süslü parantez arasında yer alır.

```
package
{
////Bütün sınıf kodları bu kısımda yer alacak
}
```

Aşağıda basitçe oluşturduğum Zombi.as sınıfını görebilirsiniz. Bu sınıf üzerinden sınıfın geri kalan özelliklerini, işlevlerini anlamaya çalışalım.

```as
package
{
    import flash.display.Sprite;

    public class Zombi extends Sprite
    {
        //SINIF GENELİ DEĞİŞKENLER BURADA TANIMLANIR
        public var hiz:Number = 2;

        //Constructor Fonksiyon
        public function Zombi()
        {
            //sınıfın nesne örneği oluşturulunca ilk
            //olarak bu kısım çalıştırılacak.

            //...Kodlar...//
        }


        public function zombiYuru():void {
            ///zombinin yürümezi için gerekli kodlar
        }

    }

}
```

Gördüğünüz gibi package kısmının hemen altında import kodu ile başlayan bir kısım var. Bu kodun işlevi basitçe sınıf içerisinde kullandığımız diğer sınıfları(örnekte Sprite sınıfı) import ediyor(sınıfımıza ekliyoruz diyebiliriz). Bu kısım şu an için çok önemli değil, zaten herhangi bir harici sınıfı kullandığımızda FlashDevelop otomatik olarak import kodunu yazar. Ancak Adobe Flash ile çalışıyorsanız -benim kullandığım versiyonda- otomatik olarak import kodunu yazmıyordu, bizim kendimiz yazmamız gerekiyordu.

Import kısmının altında artık sınıfımızı tanımlıyoruz. public kısmı şu an için çok önemli değil. Bütün sınıflarımızı, değişkenlerimizi ve fonksiyonlarımızı şimdilik public olarak tanımlayacağımızı bilmeniz yeterli. Public basitçe diğer bütün sınıfların bu tanımlanan değişkene, fonksiyona ulaşabileceği anlamına geliyor.[2]

Sınıf tanımı kısmında dikkat ettiyseniz sınıfın ismi(Zombi) ile sınıf dosyasının adı(Zombi.as) aynı. Eğer bu ikisi aynı olmazsa program hata verecektir. Sınıf tanımından sonra extends kodu kullanılmış. extends kelimesi genişlemek, uzamak anlamına gelir. Kodda kullandığımızda da anlamına benzer şekilde extend ettiğimiz sınıfın özelliklerini kendi sınıfımızda kullanabileceğimiz anlamına gelir. Örnekteki kodda Sprite sınıfının özelliklerini kendi sınıfımızda kullanıyoruz.

Bu noktada sınıfı anlatmaya biraz ara verip Actionscript'in built in class(programın içinde yer alan sınıflar) hakkında biraz bilgi verelim. Built in class'lar programın içerisinde yer alan, bir çok işlemi yaparken kullandığımız sınıflardır. Bu sınıflar olmadan Flash ile hiç bir işlem yapılamaz, yani programı oluşturan bu sınıflardır. Ekrana bir şey çizmek, bir text dosyası oluşturmak, klavye kontrollerini eklemek vs. aklınıza gelecek her işlem bu sınıflar kullanılarak yapılır. Biz kendi sınıflarımızı oluşturduğumuzda bu sınıfların özelliklerini kendi sınıflarımızda kullanırız, yani bu sınıfların üzerine özellikler ekleriz.

Mesela örnekte kullandığımız Sprite sınıfı. Sprite sınıfı en çok kullandığımız sınıflardan birisidir. Oluşturduğumuz sınıf içerisinde bir çizimi, grafiği içerecekse genelde bu sınıfı extends etmeyi tercih ederiz. Zombi sınıfının bir grafiği olduğu için (evet bir zombi grafiği) Sprite sınıfını tercih ettik. Sprite sınıfının ekrandaki bir grafiği(objeyi) kontrol etmek için x, y, width, height gibi bir çok özelliği vardır. Sprite sınıfını extends ettiğimiz zaman bu özellikleri kendi sınıfımıza aktarmış olduk yani artık bu özellikleri kendi sınıfımızda kullanabiliriz ve artık zombiyi bu özellikler sayesinde istediğimiz gibi ekranda hareket ettirebiliriz.

Sınıf tanımından hemen sonra da 2 adet süslü parantez yer alıyor. Sınıfla ilgili bütün fonksiyonlar, değişkenler bu iki süslü parantez arasında yer alır. Resme bakarsanız blokları görebilirsiniz. package bloğu bütün kodu kaplıyor: 2. satırdan 26. satıra kadar.Sınıf bloğu 6. satırdan 24. satıra kadar kısmı kaplıyor. Bu şekilde her sınıf, her fonksiyon belli bir bloğun içinde kalan kodları çalıştırır.

---

[2] OOP için konuşursak, her şeyi public olarak tanımlamak tercih edilen bir şey değildir. İleride OOP konusuna gelince bu konuda daha detaylı bilgi vermeye çalışacağım.

Sınıf tanımının hemen altında sınıf geneli değişkenler burada tanımlanır diye bir yazı yer alıyor. Bu yazı benim yazdığım bir yorum (yeşil ile yazılan her şey yorumdur). Yorumları program dikkate almaz, sadece programcıyı bilgilendirme amaçlıdır. Kodunuza yorum yazarsanız aynı kodu ileride açıp incelediğinizde neyin ne olduğunu daha rahat anlayabilirsiniz.

Yorum kısmını bir kenara bırakırsak değişken konusuna basitçe bir giriş yapmamız gerekecek. Değişkenler bilgi depoladığımız boş kutular gibidir, bunların içine istediğimiz türde istediğimiz bilgiyi koyabiliriz. Türden kastımız sınıftır. Bu built-in- class ya da bizim oluşturduğumuz herhangi bir sınıf olabilir. Değişkenler var kodu ile tanımlanır. Basitçe bir sayı değişkenini şu şekilde oluştururuz:

```
var degiskenIsmi : Number = 30;
```

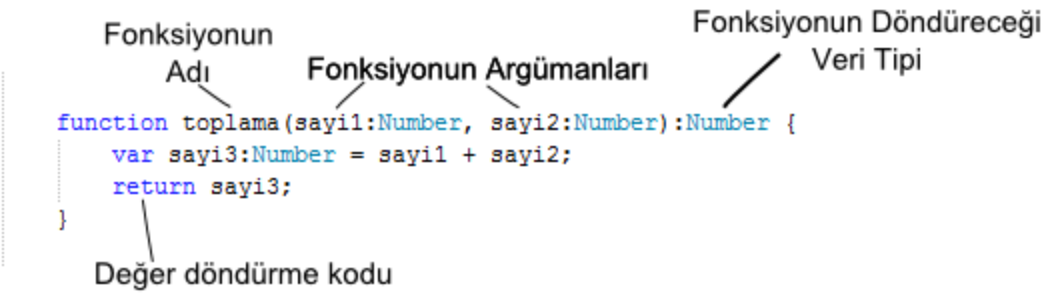
Mesela karakterimizin can(hp) değerini bir değişken içinde tutarız. Başlangıçta can değerini 100 olarak belirlediğimizi farzedelim. Etraftaki zombiler bize vurdukça bu can değerinden 10 eksiltiriz, can sıfır olunca oyun biter. Bu şekilde değişkenleri bir oyunun içerisinde her şey için kullanabiliriz.

Şimdilik kullanacağımız iki tip değişken var; sınıf geneli değişkenler ve lokal(yerel) değişkenler. Sınıf geneli değişkenlere sınıfın içinden her yerden ve diğer sınıflardan erişilebilir. Lokal değişkenlere ise sadece tanımlandıkları fonksiyon içerisinde erişebiliriz. Mesela örnekteki zombiYuru isimli fonksiyon içerisinde bir değişken tanımladığımızda bu değişkene sadece bu fonksiyon içerisinden erişebiliriz. Diğer yerlerden erişmeye çalışırsak program hata verir.

Oluşturduğumuz Zombi sınıfına dönersek; dikkat ederseniz tanımladığımız hiz değişkeninin başına public kodunu yazdık. Şimdilik sadece sınıf geneli değişkenlerin public var ile tanımlandığını bilmeniz yeterli. Bu değişkene sınıfın içerisinden her yerden ve bu sınıfın nesne örneğini oluşturan bütün sınıflar erişebilecek.[3]

Değişkenleri de anlattıktan sonra sıra geldi fonksiyonlara. Fonksiyonlar belli bir işlevi yapmak için oluşturulan kod gruplarıdır. En basitinden toplama işlemi için bile bir fonksiyon oluşturulabilir. Basitçe bir toplama fonskiyonu oluşturup fonksiyonları bu örnek üzerinden anlamaya çalışalım:

[3] Bu tarz bir şey OOP için istenmeyen bir durumdur ancak basit projelerde bu problem olmaz, hatta işinizi kolaylaştırır ve hızlandırır.



Örnekte gördüğünüz gibi fonksiyonu function anahtar kelimesiyle tanımlıyoruz. Bunun hemen ardından fonksiyonun adı(bu bizim belirlediğimiz herhangi bir isim) yer alıyor. Daha sonra parantez içerisinde fonksiyonun argümanları (fonksiyona gönderilen/atanan değerler) yer alıyor. Fonksiyon bu değerleri kullanarak bir sonuç üretir ya da bir işlem yapar. Bir fonksiyonun illaki argümanı olması gerekmez, fonksiyona göre argümana ihtiyacımız olup olmadığına biz karar veririz. Örnekteki fonksiyon belirlenen 2 sayının toplamını yapacağı için bu fonksiyonda argüman kullanmamız gerekiyor.

Parantezden sonra : koyup fonksiyonun döndüreceği veri tipi(sınıf türü) belirliyoruz. Toplama işleminin sonucu bir rakam olacağı için veri türü olarak Number belirledik. Hiç bir değer döndürmeyen fonksiyonlar da vardır, bu tür fonksiyonlarda döndüreceği veri tipi olarak void yazarız. Yani fonksiyon hiç bir değer döndürmeyecek demektir. Fonksiyon kendisinden beklenen işlemi yapar ve sonlandırılır. Mesela örneğimizdeki zombiYuru() fonksiyonu herhangi bir değer döndürmüyor, sadece işlemini yapacak(zombiyi yürütecek). Eğer fonskiyon void harici bir değer döndürüyorsa fonksiyonun içerisinde return(döndürmek anlamına gelir) anahtar kelimesi ile değer döndürmek zorundayız. Eğer return kullanılmazsa program hata verecektir.

Bunların haricinde gördüğünüz gibi fonksiyonun içerisinde sayi3 isminde yeni bir **lokal değişken** tanımladık ve sayi1 ile sayi2 değerlerinin toplamlarını buna atadık ve bu değeri fonksiyonun sonucu olarak döndürdük. Bu noktada sayi3 lokal bir değişken olduğu için sadece bu fonksiyon içerisinde erişilebilir durumdadır, diğer yerlerden erişmeye çalışırsak program hata verir.

Bu fonksiyonu kullanmak istediğimizde fonksiyonu şu şekilde çağıracağız:

toplama(10,20) ///30 sonucunu elde ederiz.

Toplama örneği fonksiyonların çok basit bir kullanım şekli. Fonksiyonları oyunlarımızdaki her türlü işlemde kullanacağız. Örneğin; zombinin hareket etmesi, saldırması, karakterimize

değmesi vs. her türlü işlev için fonksiyon oluşturabiliriz. Bu şekilde her işlevi kendi fonksiyonu altında konumlandırıp düzgün bir kodlama yapabiliriz.
Ayrıca en başta actionscript eventlerle(olaylarla) yönlendirilen bir dil diye belirtmiştik. İleride event konusunda geldiğimizde her event için de bir fonksiyon oluşturmamız gerekecek.

Bir sınıfın yapı taşı olan değişken ve fonksiyonlara değindik. Zombi sınıfımıza geri dönersek; 10. satırda constructor fonksiyon diye bir not düştüm. Constructor fonksiyon özel bir fonksiyon türüdür, her sınıfta sadece bir tane Constructor fonksiyon bulunabilir ve bu fonksiyon sınıf ismi ile aynı ismi taşır. Bu fonksiyon sınıfın nesne örneği oluşturulduğunda çağırılan ilk fonksiyondur, yani bir nesne örneği oluşturulunca yapılmasını istediğimiz şeyleri bu fonksiyonun içine yazarız. Zombi sınıfımız üzerinden konuşursak, bir zombi objesi oluşturduğumuzda ilk ne yapmak isteyebiliriz? İlk olarak yapmamız gereken zombinin çizimini oluşturmak olabilir mesela veya 1-2 değişken tanımlayabiliriz. Ya da bir event(olay) tanımlarız gibi.