

A'dan Z'ye Java

Java sayesinde; animasyonlar, kayan yazılar, web sayfalarında chat, hesap makinesi, oyunlar, şifreleme programları, interaktif web sayfaları, müzik, kelime işlemci, küçük internet appletleri, kocaman paket programlar, resim işleme programları, tercüme programları, sipariş sistemleri, saatler, yazım kontrol programları yapılabilir.

HTML'de Java

Bir applet hazırladığınızda, bunu bir HTML dökümanına yerleştirmelisiniz. HTML 3.2 ile birlikte, Java appletlerin kullanılmasını sağlayan iki yeni eleman çıktı. Bunlar APPLET ve PARAM. Diğer HTML elemanları gibi bu iki elemanın da birçok özellikleri mevcuttur. Fakat ideali, sadece gerekli olan özelliklerin kullanılmasıdır.

APPLET Elemanının kullanılması

APPLET elemanı, HTML dökümanına yerleştirdiğiniz appletin tanımlanmasını ve özelliklerinin tanımlanmasını sağlar. Her HTML elemanı gibi, APPLET elemanının da bir başlangıç tagı <APPLET> ve bitiş tagı </APPLET> vardır.

APPLET elemanı için gerekli olan özellikler: CODE, WIDTH ve HEIGHT'tir. CODE özelliği kullanılacak appleti, WIDTH özelliği appletin genişliğini (pixel), HEIGHT özelliği ise appletin yüksekliğini (pixel) belirlemenizi sağlar. Aşağıda gerekli olan özellikleri ile bir appletin HTML dökümanına bağlanmış şeklini görüyorsunuz.

```
<HTML>
<HEAD>
<TITLE> Maximum Bilgi </TITLE>
</HEAD>
<BODY>
<APPLET CODE="maximumbilgi.class" WIDTH=300 HEIGHT=100>
</APPLET>
</BODY>
</HTML>
```

Yukarıdaki örnekte HTML dökümanı ve "maximumbilgi.class" adlı java appleti aynı dizinde bulunmalıdırlar. Şayet java appleti farklı bir dizinde ise, opsiyonel CODEBASE özelliğini kullanmanız gerekir.

CODEBASE özelliği, appletin bulunduğu URL'yi belirtmenizi sağlar. Aşağıdaki örnek, yukarıdaki örneğin hemen hemen aynısıdır. Tek farkı, java appleti başka bir dizinde bulunduğu varsayılarak CODEBASE özelliği kullanılmıştır.

```
<HTML>
<HEAD>
<TITLE> Maximum Bilgi </TITLE>
</HEAD>
<BODY>
<APPLET CODE="maximumbilgi.class"
CODEBASE=""http://www.maximumbilgi.com/java/classlar/" WIDTH=300
HEIGHT=100>
</APPLET>
```

</BODY>
</HTML>

Diğer opsiyonel özellikleri şöyle sıralayabiliriz:

ALIGN

Appletin konumunu belirlemizi sağlar. Alabileceği değerler ABSBOTTOM, ABSMIDDLE, BASELINE, BOTTOM, CENTER, LEFT, MIDDLE, RIGHT, TEXTTOP, TOP'tır.

HSPACE

Appletin altında ve üstünde ne kadar boşluk bırakabileceğinizi belirleyebilirsiniz. HSPACE'e vereceğiniz değer, pixel olarak işlenir. Örnek: HSPACE=10

VSPACE

Appletin sağında ve solunda ne kadar boşluk bırakabileceğinizi belirleyebilirsiniz. VSPACE'e vereceğiniz değer pixel olarak işlenir. Örnek: VSPACE=20

ALT

Appletleri gösteremeyen browserlarda, appletin yerine alternatif bir yazı çıkmasını sağlayabilirsiniz.

NAME

Applete bir isim vermenizi sağlar.

PARAM Elemanının kullanılması

PARAM elemanı ile applete parametre gönderebilirsiniz. PARAM elemanının sadece başlangıç tagı vardır: <PARAM>. Applete göndermek istediğiniz her parametre için bir <PARAM> tagı kullanmalısınız. PARAM taglarını başlangıç tagı <APPLET> ile bitiş tagı </APPLET> arasına yerleştirmelisiniz.

Appletler, <PARAM> tagında belirtilen parametrelere "getParameter()" metodu ile ulaşırlar. <PARAM> tagının gerekli iki özelliği vardır. NAME ve VALUE. NAME özelliği ile parametrenin adı belirtilir ki "getParameter" metodu ile bu isme göre aranır. VALUE özelliği ise parametrenin değerini verir. Aşağıda, applette kullanılacak resmi, parametre ile girilen bir HTML dökümanı örneği verilmiştir.

```
<HTML>
<HEAD>
<TITLE> Maximum Bilgi </TITLE>
</HEAD>
<BODY>
<APPLET CODE="maximumbilgi.class" WIDTH=300 HEIGHT=100>
<PARAM NAME=Resim VALUE="logo.gif" >
</APPLET>
</BODY>
</HTML>
```

Java appletinde, resim parametresinin değeri; Resim_logo=getparameter("Resim") komutu ile okunur.

Veri Tipleri

Java'da her şey bir nesnedir. Tek istisna veri tipleridir. Java veri tipleri, bütün platformlarda standart büyüklüğe sahiptir. Bu standart Java'nın taşınabilirliğini sağlıyor. Aşağıda Java'da bulunan veri tipleri ve büyüklükleri listelenmiştir.

Veri tipi	Büyüklik
byte	8-bit
short	16-bit
int	32-bit
long	64-bit
float	32-bit kayan nokta
double	64-bit kayan nokta
char	16-bit Unicode

Eğer C/C++ programcısı iseniz "unsigned" tipinin olmadığı dikkatinizi çekmiştir. Byte tipi ise C/C++'daki "char" tipinin yerini almıştır. Java'daki char tipi 16 bittir. Çünkü Java karakter verisinde Unicode karakter setini temel alır.

Unicode, uluslararası karakterleri destekleyen bir standarttır. Programlarınızın değişik platformlarda ve ülkelerde çalıştırılacağında Unicode çok uygundur. Yukarıdaki tabloda olmayan diğer bir veri tipi ise boolean'dır. Bir boolean, değişken numerik değere çevrilemez ve sadece iki değer alabilir. Bu değerler "true" ve "false" tur.

Literaller

Literaller, değişkenlere değer atamak için kullanılır. Integer'lara C/C++ benzer biçimde değer atanır. 25 gibi tamsayı atayabilirsiniz. Hexadecimal bir tamsayı atamak için, sayının önüne "0x" ifadesi koymanız gerekir. Örneğin, 15 sayısını hexadecimal olarak 0xF şeklinde atamalısınız. 8'lik tabanda bir sayı atamak için "0" ifadesini, sayının önüne koymalısınız.

Kayan noktalı sayıları ise direkt 5.2345 şeklinde atayabilirsiniz. Bunlar 32 bit float veya 64 bit double olarak saklanabilirler. Belirtmezseniz, varsayılan 64 bit double'dır. Belirtmek için yapacağınız tek şey, sayının arkasına float için F, double için D koymaktır. Örneğin 5.323 F veya 5.323 D.

Karakterler, tek tırnak içinde atanırlar. Örneğin 'a' gibi. Escape karakterleri için slash (\) kullanılır. Bunlar da tırnak içinde belirtilir. \t=tab, \n=satır atlama gibi. Stringler için çift tırnak kullanılır. "Maximum Bilgi" gibi. Satır atlaması yapmak istiyorsanız "Maximum Bilgi\n" kullanmalısınız.

Değişkenler

Java'da 3 tip değişken vardır: instance, class ve local. Lokal değişkenler, metodların ve blokların içinde tanımlanabilirler. Blok, "{" ile başlayan ve "}" ile biten ifadeler topluluğudur. Blok içinde tanımlanan lokal değişkenler, blok sonuna kadar geçerlidir. Genel formatı <tip> <değişken adı> şeklindedir. Örneğin double tipinde pi adlı değişkeni tanımlayalım: double pi; bir değer de atayabildik: double pi=3.1415.

Değişkenler; harf, sayı, dolar işareti, alt çizgi ile başlayıp bu karakterlerle devam edebilir. Ama komutlar, operatörler değişken ismi olarak kullanılamazlar.

Açıklamalar

Programlarınıza açıklama koymak için, Java'da kullanabileceğiniz iki stil vardır. Birincisi çift slash (//). Çift slash'tan satır sonuna kadar olan kısımda bulunan her şey açıklama olarak kabul edilir.

İkincisi ise slash ve yıldız (/*) ile başlar yıldız ve slash ile biter. Bunlar arasındaki her şey, açıklama olarak kabul edilir. Örnek 1: double pi; // pi değişkeni double olarak tanımlandı. Örnek 1: /* Bu programın amacı sayısal loto tahmini yapmaktır.

Operatörler

Java'da bulunan operatörler aşağıdaki tabloda verilmiştir.

Kategori	Operatör
Aritmetik	+ - * / %
İlişkisel	< > >= <= == != &&
Bit işlemleri	& ^ << >> >>> ~ &= = ^=
Artırma	++
Azaltma	--

Aşağıda iki değişken tanımlanıp, bunlara değer atanıyor. Daha sonra ise bu iki değişkenin içeriği toplanıyor:

```
int x,y ;
x= 3 ;
y=4 ;
int z=x+y ;
```

Z 'nin değeri yedidir. Operatörlerin öncelik sırası aşağıdaki tabloda gösterilmiştir.

[] ()
++ - ! ~
* / %
+ -
<< >> >>>
< > <= >=
== !=
& ^
&&

= ve diğerleri

Bit işlemleri

Ayrılmış Kelimeler

Aşağıdaki tabloda, Java'nın komutları olarak kullanılan kelimeler listelenmiştir. Bu kelimeler, Java için ayrılmıştır. Yani bu kelimeleri değişken ismi gibi şeyler için kullanamazsınız. Bunlar sınıf tanımlaması, değişken tipi belirleme, koşul, döngü gibi işlemler için kullanılır.

Abstract	boolean
break	byte
byvalue	case
catch	char
class	const
continue	default
do	double
else	extends
false	final
finally	float
for	goto
if	implements
import	instanceof
int	interface
long	native
new	null
package	private
protected	public
return	short
static	super
switch	synchronized
this	threadsafe
throw	transient
true	try
void	while

Karşılaştırmalar

Karşılaştırmalar, genelde bir karşılaştırmamanın sonucuna göre belirtilen komut veya komutları çalıştırlar. Eğer birden fazla komut kullanılacak ise blok içinde olmalıdır. Tek bir komut için buna gerek yoktur. Örneğin; a değişkeninin değerinin bir olup olmadığını kontrol ediyoruz. Eğer a=1 ise a=2, değil ise a=1.

Görüldüğü üzere karşılaştırma, boolean bir değer döndürmelidir. Yani true (doğru) veya false (yanlış) değerini döndürmelidir ki bu değere göre işlenecek komutlar belirlenebilsin. Java'da karşılaştırma yapmak için "if...else..." kullanılır. Formatı şu şekildedir:

```
if (karşılaştırma)
{ karşılaştırmanın sonucu
doğru ise işlenecek komutlar
}
else
{ karşılaştırmanın sonucu
yanlış ise işlenecek komutlar
}
```

Yukarıdaki örneği java formatında yazarsak :

```
if (a==1) // Java'da = operatörü atama için kullanılır.
// Karşılaştırma için == operatörünü kullanılır.
a=2;
else
a=1; // Tek komut olduğu için blok içine alınmasına gerek
yoktur.
```

Eğer birçok karşılaştırma varsa, iç içe karşılaştırmalar (if) kullanılabilir. Sonucu nümerik olan karşılaştırmalar için "switch...case..." komutu kullanılır. Bu komut, sadece nümerik değerler için kullanılır. Formatı şu şekildedir:

```
switch (değişken)
{
case 1: // değişkenin değeri 1 ise
break;
case 2:{
// değişkenin değeri 2 ise
break;
}
default: // yukarıdaki değerler haricindeki değer ise
break;
}
```

Döngüler

Java 'da üç tane döngü işlemi vardır. "for" döngüsünün yapısı aşağıdaki şekildedir. for (ilk değer; test; değer artırma/azaltma) Bu yapıda görülen ilk değer ifadesi, değişkeninizin alacağı ilk değerdir. Test ifadesinde ise basit veya kompleks bir karşılaştırma kullanabilirsiniz. Değer arttırma veya azaltma ise değişkeninizin o anki değerini değiştirecek bir ifadedir. "for" döngüsünü örnek kullanımı:

```
for (sayac=0;sayac<3;sayac++)
```

Bu örnekte, sayaç değişkenine ilk değer olarak sıfır verilmiş ve her döngüde bir arttırılması istenmiş. Döngüden çıkma koşulu ise sayaç değişkeninin 3 ve daha yukarı değerleri olarak belirlenmiş. Bu durumda sayaç değişkeni 0,1,2 değerleri için döngüye girecektir. "for" döngüsünden sonra bir komut veya blok komutları gelebilir. "while" döngüsünün yapısı şu şekildedir: while (test). Test ifadesi "for" döngüsündeki ile eşdeğerdir. Örnek:

```
sayac=0;
while (sayac<3)
{
//komutlar
sayac++;
}
```

"do" döngüsünün yapısı:

```
do
{
//komutlar
} while (test);
```

"do" döngüsünün "while" döngüsünden farkı en az bir kere döngüye girmesidir.

Tüm döngülerde "break" komutu döngüden çıkılmasını, "continue" komutu ise bir sonraki değere geçmeyi sağlar.

Diziler

Java'da diziler nesnedirler, C'deki gibi hafızada yer kaplayan pointer değillerdir. Java 'da diziler, C 'ye göre daha güvenilirdir. Dizilerin elemanlarına gelişigüzel değerler atayamazsınız. Java, dizi elemanlarını sırasıyla kontrol eder. Dolayısıyla arada değer atanmamış bir elemana rastlanırsa hata oluşur. Bu da C'de çıkan hafıza bozulmalarını önler. Java'da dizi kullanmak için ilk önce dizinin tipi verilir. Tip verilirken dizinin büyüklüğü belirtilmez.

```
int numbers[]; // Integer diziler için
String myStrings[]; // String diziler için
```

Diğer metod ise tipin arkasına köşeli parantez koymaktır.

```
String[] myStrings; // String diziler için
```

İkisi de tanımlamada aynıdır. Her ikisi de kullanılabilir. Size hangisi daha kolay geliyorsa onu kullanın. Java dizileri örneklerden de görüldüğü üzere tüm veri tipleri için kullanılabilir. Sonraki adım ise dizinin "new" operatörü kullanılarak yaratılmasında, bu adımda dizinin büyüklüğü de belirtilir.

```
int numbers[] = new int[5]; // 5 büyüklüğünde integer dizi
String myStrings[] = new String[20]; // 20 büyüklüğünde string dizi.
```

Bu adımdan sonra diziler yaratılmış oldu. Dizinin her bir elemanına varsayılan değer atanır. Varsayılan değerler integer veri tipi için "0 (sıfır)", string veri tipi için "null" değeridir. Dizi elemanlarına bu adımdan sonra değer atamak çok kolaydır. C++ 'da olduğu gibi Java 'da da dizilerin elemanları sıfırdan başlayan tamsayı değerleridir. Dizinin ilk elemanına değer atamak için:

```
myStrings[0] = "ilk dizim";
numbers[0] = 10;
```

Java'da bir dizinin eleman sayısını (büyüklüğü) bulmak için "length" metodu kullanılır. Bu metodun döndürdüğü değer "int" tipindedir.

int boyut=numbers.length // boyut=5 olur

Java'da çok boyutlu diziler yoktur. Fakat bu tür diziler dizi içinde dizi oluşturularak simüle edilebilir.

**int k[][] = new int[5][4]; // 5-4 boyutunda bir dizi
k[1][3] = 999; // Değer atama**

Bilgi Alma ve Yazdırma

Ekrandan bilgi almak için BufferedReader sınıfını kullanacağız. BufferedReader giris=new BufferedReader(new InputStreamReader (System.in)); Bu sınıfın readLine() metodunu kullanarak bilgiyi alıyoruz:

```
try {  
    String girilen=giris.readLine();  
} catch (IOException e ) { System.out.println(e);}
```

try-catch bloğu olası bir hatayı yakalamak için kullanılıyor. Eğer bir hata oluşursa hatayı ekrana yazacak. Şimdi de girileni yazdıralım. Ekran birşey yazdırmak için System.out sınıfının println metodunu kullanacağız :

System.out.println ("Girdiğiniz şey : " + girilen);

println metodu ekrana verilen parametredeki yazıyı yazdıktan sonra imleci bir alt satıra geçirir. Şayet imlecin bir alt satıra geçmesini istemiyorsanız print metodunu kullanın. Ama bu metodu kullandığınızda ekrana bilginin çıkması için akabinde flush metodunda kullanmalısınız. Şu şekilde :

```
System.out.print("Ali ");  
System.out.flush();
```

Java Sertifikası

Java internet ortamına girdiğinde, javayı destekleyen browserlardaki buglar yüzünden çoğu kullanıcı java özelliğini kapatmak durumunda kalmıştı. Günümüzde javanın güvenliği artmış durumda. Bunda browserların buglarının azalmasının rolü olduğu gibi sertifika kavramının da büyük rolü vardır.

Sertifikasız/İşaretsiz Appletler Neler Yapabilir?

Kullanıcıların bilgisayarında istenmeyen işlemlerin yapılmasını engellemek için, her browserın güvenlik sınırlamaları vardır. Bu güvenlik sınırlamasının türü browsera göre değişir. Günümüzdeki browserların java appletlerin için aşağıdaki güvenlik sınırlamaları vardır:

- Appletler kullanıcının bilgisayarından veri okumaz ve yazamazlar.
 - Kullanıcının bilgisayarında program çalıştıramazlar.
- Ağ bağlantısı sadece appletin bulunduğu server ile kurulabilir.
 - Sistem ayarlarını değiştiremezler.

Bu güvenlik sınırlamalarına bakıldığı zaman normal bir applet sadece grafik veya yazı gösterme, ses ve animasyon işlemlerini yapabilir. Büyük işler için normal appletler uygun değildir ki küçük bir veri girişi dahi olamadan dış dokunur uygulama yapmak mümkün değildir.

Sertifikalı ve İşaretli Nedi?

Appletlerin web sayfalarını daha görsel hale getirmekten başka bir işe yaramadığı çabuk anlaşıldı. Ama daha fazlasını; mesela kullanıcıdan bilgi almak ve yazmak; yapabilmek için güvenlik sınırlamalarını kaldırmak gerekiyordu. Fakat bunun mümkün olduğu güvenli olması için, appletler yeni browserlar için işaretlenebiliyor.

Bunun için applet geliştiricisinin amacını belirten sertifikaya ihtiyacı vardır. Sertifika browser okuyabileceği şekilde açıktır ve geliştiricinin adını ve kontrol kodu içerir. Bu sertifikaya ile geliştirici appleti güvenli olarak işaretler. Başka hiç kimse bu sertifikayı kullanarak bir appleti işaretleyemez. Çünkü geliştirici appleti işaretlemek için sertifika ile birlikte verilen özel kodu kullanır.

Kullanıcı, appleti çalıştırdığında (applet olan bir web sayfasını çağırdığında) browser appletin işareti ile sertifikayı karşılaştırır. Böylece appletin işareti sertifikaya ile uyumlu ise çalıştırır. Bu şu anlama geliyor: Doğru işaretli bir appletin geliştiricisini kullanıcı applet yüklenirken görür. Ama sadece applet yüklenirken.

İşaretli Appletler Neler Yapabilir?

İşaretli bir applet yüklendiğinde kullanıcının karşısına bir pencere gelir. Bu pencerede sertifika hakkında bilgiler vardır ve kullanıcı onayı istenir. Kullanıcı bu bilgiler ışığında appletin çalışmasına izin verir veya vermez. Kullanıcının çalışmasını onaylamış bir applet normal programların yapabildiği her şeyi yapabilir.

Bu Durumdaki Güvenlik

Buradaki problem onaylama işleminin ya hep ya hiç mantığında işlemesidir. Bunun anlamı onayladığınız bir appletin gerçek amacının dışındaki verilerde okuyabildiği veya istenmeyen bir yere veri yazabildiğidir. Tam anlamıyla erişim haklarının belirlenmesi programcılara ağır geldiği için çoğu programcı tarafından yapılmıyor. Yine de Microsoft ve Netscape'in browserları için erişim haklarının değişik şekilde belirlenmesi karşısında programcılar minimum düzeyde erişim hakkıyla yetiniyorlar.

İkinci bir problem ise sertifikanın gerçekliğidir. Gerçek sertifikalar, programcı tarafından değil, programcının güvenilirliğine kefil olan şirketler tarafından oluşturulmuş sertifikalardır. Bu şirketler para karşılığında sertifika oluşturmaktadır. Bir sertifika en az \$200 başlayıp sertifikaya göre artmaktadır. Programcı yıllık olarak \$100 başlayan fiyatları da ödemek zorundadır. Çoğu programcı için bu fiyatlar aşırı pahalı gelmektedir.

En azından herkes bedava kendi sertifikasını oluşturabilir. Fakat böyle bir sertifika, programcının gerçekte kim olduğunu ve amacının doğruluğunu belirtmez. Hatta test sertifikaları diye anılanlar istenen herhangi bir isim ile oluşturulabilir. Her ne kadar insanı korkutsada, bilinmeyen bir shareware programı denemekten daha güvenlidir. Appleti onaylamak veya onaylamamak bizim elimizde değil mi?

Sertika Şirketleri

Gerçek sertifikalar sertifika şirketleri (Certificate Authority kısa CA) tarafından veriliyor. Bu şirketler programcının yada şirketlerin amacını kontrol ettikten sonra sertifika veriyorlar. En tanınmış sertifika şirketleri VeriSign, Thawte ve BelSign.

Her CA, vereceği sertifikaları kendi sertifikası altına tanımlar ki daha önceden güvenli belirlenmiş kendi sertifikaları gibi işlem görsünler. CA'lar kendilerine ait sertifikaları daha önceden browserlara tanımlanmış ve kurulmuştur. Maalesef bu şirketlerin verdikleri

hizmetin bedeli yüksek. Bir defaya mahsus ücret \$200 dan başlıyor, yıllık ücretler ise 100\$ dan. Şuana kadar işaretli appletlerin yaygınlaşmamasının en büyük nedeni bu olsa gerek.

Test Setifikaları

Test sertifikası herkes tarafından beş kuruş ödenmeden oluşturulabilen sertifikalardır.

Ama programcının amacını gerçek olarak ispatlamayan sertifikalar bunlar. Bu sertifikalarda browsera göre değişiyor. Bir Netscape ve birde Microsoft için oluşturmak gerekiyor. Sun'ın HotJava browserına hiç girmeyeceğim burda, belki ileride onuda ele alırım.

Microsoft Browser İçin Test Sertifikası

Microsoft browser sertifikası oluşturmak için, Microsoft'un sitesinden bedava indirebileceğiniz " Software Developer's Kit (SDK) " e ihtiyacınız var. Bu yazılım kitinin içindeki programların tek tek indirilememesi tabii ki çok kötü bir durum. Bu kitin içindeki herhangi bir program için 20 MB büyüklüğündeki bu kiti indirmek zorunda kalıyorsunuz.

Ama bu kiti CD olarakta sipariş edebiliyorsunuz.

Bu kit ile gelen ingilizce dökümantasyon tam olarak okunmalı, nitekim olmazsa olmaz birçok önemli parametre mevcut. Burada bu kitin içindeki DOS programlarıyla sertifika oluşturulmasını kısaca anlatmaya çalışacağım. Bu sadece başlangıç için yardım niteliğindedir. Gerisini dökümana bırakıyorum.

makecert ile Private/Public Key oluşturun. Sertifika için gerekli olan .cer uzantılı bir dosya oluşacaktır.

cert2spc ile sertifikayı oluşturun. Birinci adımda oluşturduğunuz .cer uzantılı dosyadan .spc uzantılı bir dosya oluşturur.

cabarc ile class dosyalarınızı cab dosyası haline getirin. Bu cab dosyası Internet Explorer tarafından indirilecek dosyadır.

Oluşturduğunuz cab dosyasını **signcode** ile işaretleyin. Bu işlem sırasında oluşturduğunuz .spc uzantılı dosyada kullanılacaktır.

Netscape Browser İçin Test Sertifikası

Netscape programcılarının işini biraz daha kolaylaştırmış Microsoft'a göre, sertifika için gerekli olan "Signtool" uğraşmadan indirebilecek ve Microsoft'un kitine göre daha kolay kullanımı var. Signtool Win95 ve Unix sürümleri var. Kısaca adımlar:

- signtool ile Private ve Public Key oluşturmak için -G parametresini kullanın.
- Bu işlem ile x509.cacert adlı bir dosya oluşacaktır.
- signtool ile class dosyalarınızı jar dosyası haline getirin.

Bu işlemleri batch dosyası ile de yapabilirsiniz.

JAVA PLUG-IN

Bundan birkaç yıl önce Sun, HotJava adlı web browserı piyasaya çıkardı. Sun'ın bu web browserı, Java'yı destekliyordu. Buda içine java nesneleri gömülebilen uygulamaların ilk örneğini teşkil ediyordu. Bunun ardından Netscape kendi web browserında Java desteğini ekledi. Ardından da MS.

Gün geçtikçe Java'nın yeni sürümleri çıktı. Bununla birlikte Netscape ve MS kendi ihtiyaçlarına uygun sınıf kütüphanelerini eklediler browserlarına. Böylece her browserın kendine özel JVM (Java Virtual Machine) 'i oldu. Dolayısıyla browserların içine gömülen JVM ile browserdan bağımsız olarak JVM 'i güncellemek imkansızdı.

Sun, herhangi bir ortamda Java'nın çalışabilmesi için gerekli olan her şeyi içeren (Java VM ve sınıf kütüphaneleri) JRE 'yi çıkardı. Eğer JRE bilgisayarınızda yüklüyse, herhangi bir uygulamaya (örneğin browser) gömülmüş java nesneleri çalışıyor. Sorun, browserların sadece kendi JVM lerine bakmaları ve makinada yüklü olan JRE 'ye ulaşmak için bir mekanizmaya sahip olmamalarıydı. Sun olaya el attı : Java Plug-In

Sun'ın geliştirdiği Java Plug-In, JRE 'ye HTML sayfalarından browserı kullanarak ulaşmak için bir mekanizma. Java Plug-In, Sun'ın web sitesinden [bedavaya indirilebiliyor](#). Sayfaya girildiğinde, istenen java sürümü kullanıcının makinasında yüklü değilse, kullanıcı Java Plug-In sayfasına yönlendirilir. Java Plug-In, Netscape'de plug-in olarak, IE 'de ise ActiveX Control olarak otomatik yüklenir.

Java Plug-In Kullanmak

Java Plug-In, Netscape'de EMBED, IE 'de ise OBJECT tagı sayesinde kullanılabilir. MIME tipi ile istenen java sürümünü belirtmelisiniz. Örneğin Java 1.2 sürümünün özelliklerini kullanan bir appletiniz varsa, MIME tipinde bunu belirterek appletinizin doğru çalışmasını sağlarsınız.

MIME Tipi	Nesne Tipi / Sürüm
application/x-java-applet;	Applet / Mevcut sürüm
application/x-java-applet;version=1.1	Applet / Java 1.1
application/x-java-applet;version=1.2	Applet / Java 1.2
application/x-java-bean;	Bean / Mevcut sürüm
application/x-java-bean;version=1.1	Bean / Java 1.1

HTML sayfasına appletinizi geleneksel yöntemlerle koymak isterseniz, yani Java Plug-In kullanmak istemiyorsanız APPLET tagını kullanmanız gerekir. Java Plug-In kullanmak istiyorsanız;

Netscape için EMBED tagını kullanmalısınız :

<HTML>
<BODY>

<EMBED type="application/x-java-applet;version=1.1" width="200" height="200" code="Appletim.class" codebase="/appletler/" parametre1="2" pluginspage="http://java.sun.com/products/plugin/1.1/plugin-install.html">

<NOEMBED>
Applet Desteklenmiyor
</NOEMBED>

</EMBED>

</BODY>

</HTML>

PLUGINSOURCE ve TYPE alt tagını kullanmanız gerektiğini unutmayınız. Çünkü browser Java Plug-In yükleme sayfasını ve sürümünü bilmesi gerekir. APPLET tagında kullanılan PARAM alt tagını bu durumda kullanmazsınız. Parametreleri direkt, örnekte görüldüğü gibi (parametre1="2") direkt yazmalısınız.CODE, CODEBASE,WIDTH,HEIGHT gibi alt taglar APPLET tagında olduğu gibi kullanılır.

Internet Explorer için OBJECT tagını kullanmalısınız :

<HTML>

<BODY>

```
<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
width="200" height="200" codebase="http://java.sun.com/products/plugin/
1.1/jinstall-11-win32.cab#Version=1,1,0,0">
  <PARAM NAME="code" VALUE="Appletim.class">
  <PARAM NAME="codebase" VALUE="/appletler/">
  <PARAM NAME="type" VALUE="application/x-java-applet;version=1.1">
  <PARAM NAME="parametre1" VALUE="2">
```

Applet desteklenmiyor

</OBJECT>

</BODY>

</HTML>

Burada CODEBASE alt tagı gömülü nesnenin gerektirdiği ActiveX Controlünün yeridir. Java'ya ait codebase, type, code gibi taglar PARAM alt tagı ile belirtilir.

Netscape ve IE için farklı yöntemlerin kullanılması, her biri için ayrı web sayfalarımız olacağı anlamına gelmiyor. HTML kodu kullanarak her ikisi içinde geçerli bir sayfa hazırlayabilirsiniz. Fakat böyle bir HTML kodu yazmak için kendinizi kasmayın. Çünkü Sun, Java Plug-In ile beraber HTML Çeviricisi (HTML Converter) sunuyor. Bu çevirici sayesinde HTML sayfasında kullanılan APPLET tagları her iki browserda da kullanılabilecek şekilde çevriliyor.

Applet Neden Çalışmıyor?

Appletler, dinamik web tabanlı içerik için en ideal programcıklardır. Teorikte bir kere yaz, her bilgisayarda, her browserda ve her işletim sisteminde çalıştır; mantığını ileri süren Sun bunu "Bir kere yaz, her yerde çalıştır (Run Once, Run Anywhere) " sloganıyla lanse ediyor.

Pratikte, birçok java programcısı ve birçok kullanıcı Java appletlerinin başağrısına sebep olduğunu söylüyorlar. "Neden applet çalışmıyor ?" içerikli birçok mail geliyor.

Applet çalıştığı zaman tam çalışıyor, çalışmadığı zaman ise büyük bir sorun halini alıyor. Hatayı bulmak ve hatayı düzeltmek gerçekten başağrısına neden olabiliyor. Hatta Java ile program geliştirmeyi bu yüzden bırakanlar bile var. Değişik browserlar, browser sürümlerinin farklılıkları, hatta browser konfigürasyonlarındaki farklılıklar appletlerin

çalışmasında rol oynayabiliyor. Browserlara gömülü Java Sanal Makinasındaki (Java Virtual Machine) buglar ve bilgisayarın performansında etkili oluyor.

Şayet browser'ınız hiçbir applet çalıştırmıyor ise, java applet ihtiva eden bir sayfayı çağırdığınızda takılıp kalıyorsa browser yazılımını tekrar kurmak çoğu zaman problemi ortadan kaldırıyor. Bu söz konusu değilse, gri bir dikdörtgenden başka bir şey göremiyorsanız bunun bir kaç sebebi olabilir.

Neden Gri Dikdörtgen?

Appletin yüklenmesi sırasında veya yüklendikten sonra çalıştırılması sırasında meydana gelen hatalardan dolayı oluşur. Java programlama dilinde bunun teknik adı "Exception" dir. Java'da hataları yakalama imkanı olduğuna biliyoruz, daha doğrusu bilinen çoğu hatayı desek daha iyi olur. Oluşan hatalar bilinmedik, beklenmedik hata veya appletten kaynaklanmayan hatalar olabildiği için bu hataları yakalama imkanı yok. Şimdi en çok karşılaşılan ve bilinen hataları inceleyelim :

Class dosyasına ulaşamama veya yükleyememe

Bilindiği gibi applet dediğimiz programcıklar, derlendikten sonra class dosyası halini alıyorlar. Appletlerin çalışması için bu class dosyalarının yüklenmesi gerekiyor. Şayet class dosyasının yeri bulunamaz veya dosya yüklenirken hata oluşur ise gri kutunun içinde class dosyasını bulunamadığına dair veya yüklenirken hata oluştuğuna dair bir hata mesajı çıkar ve applet çalışmaz. Applet çalışmadığı için, ki hata daha applet yüklenirken oluştuğu için, applet kodu içinden bu hatayı yakalamanız ve engellemeniz mümkün değildir. Bu hata çoğu durumda, <applet> tagı içinde class dosyasının bulunduğu yeri yanlış olarak belirtildiği zaman ortaya çıkıyor.

Ağ kaynaklarına erişememe

Bazı zamanlar ağ üzerinde bulunana server göçer veya herhangi bir sebepten bağlantı kurulamaz. Kullanıcı firewall arkasındadır ve Socket, DatagramSocket istekleri çalışmaz.

Eğer ağ kaynaklarına erişim gerektiren bir applet yazıyorsanız, iletişim için TCP/UDP protokülü yerine HTTP protokülünü kullanın. Çünkü firewall arkasında olan kullanıcı sayısı gün geçtikçe artıyor. Karşılaşılan diğer bir hata sebebi ise, browserların appletler ile ilgili kısıtlamalarından kaynaklanıyor.

Appletviewer'da sorunsuz çalışan applet, browserda çalışmıyor. Applet sadece bulunduğu server ile bağlantı kurabilir. Birçok applet ise domain isimlerini IP adresine çevirirken hata oluşturuyorlar. Bunu önlemek için domain ismi yerine IP adresi kullanın.

Null Hataları

Çok karşılaşılan hatalardan biriside NULL hatalarıdır. Bu appletin kodundan kaynaklanan ve appletin çalışmamasını sağlayan bir hatadır. Bir değişkene bir nesne atarken, nesnenin değerinin Null (değersiz) olmamasına dikkat edin. Aşağıdaki örneği inceleyin:

Örnek:

```
String sayac = getParameter ("sayac"); //sayac parametresini al  
if ( sayac == null ) // Eğer sayac null ise  
    sayac = "1"; // sayac değerini "1" yap.
```

Biraz dikkat ile, web sayfalarında önümüze çıkan gri dikdörtgen kutucukları azaltabiliriz.

JavaScript

Java, C++ gibi derleme tabanlı bir dil olmasına rağmen, tek bir derlenmiş programın her tip bilgisayarda çalışabilmesi sağlanacak şekilde değiştirilmiştir. Fakat JavaScript bu tür bir programlama dili değildir. Yorumlanması için bir tarayıcıya ihtiyaç duyar. Bu yüzden script dilidir. Html dosyasını içine gömülüdür. Javascript , Netscape firması tarafından C dilinden esinlenilerek yazılmıştır. Yazılma amacı Html'in sahip olmadığı bazı özelliklerin web sayfalarında kullanılmak istenmesidir. Netscape firması bu konuya ağırlık vererek JavaScript script dilini internet ortamına kazandırmıştır. Kısaca Java derleme tabanlı bir dil, JavaScript ise küçük program parçalarıdır.

Netscape ve Internet Explorer tarayıcıları JavaScript kodlarını farklı yorumlar. Netscape firması JavaScript dilini hazırladığında Microsoft firması bu dilin özelliklerini veya yazılım tarzını tam anlamıyla Internet Explorer'a eklemeyi. Kendi yazım kurallarını belirledi. Bu yüzden JavaScript kodu yazarken bu iki tarayıcı özelliklerini de göz önünde bulundurulmalıdır. Fakat bu her kodda karşımıza çıkmaz.

2.2.1 JavaScript'in genel Özellikleri

- Javascript kodlarını yazmak için Windows kullanıcıları için NotePad , Mac. kullanıcıları için Simple Text yeterlidir.
- JavaScript kodları etiketi ile biter.
- Etiket JavaScript'i anlamayan eski sürüm tarayıcıların bu kısmı geçmeleri içindir.
- Genellikle yazım tarzı

```
<script>
<!--
JavaScript kodları
-->
</script>
```

şeklindedir. Tüm JavaScript kodları HTML ye gömülü olarak bu deyimler arasına yazılmalıdır.

- JavaScript'te açıklama işaretleri tek satır ise // ile başlar. Eğer açıklamanız bir satırdan fazla ise /* ile başlar */ ile biter.

```
// bu satır kullanılacak değişkenlerin tanımlanması
/* açıklama satırı 1
açıklama satırı 2
açıklama satırı 3 */
```

- JavaScript kodları Html kodların arasında yer alır. Veya uzantısı js olan dosyalarda saklanarak yine Html içerisinden çağırılır. Java Appletleri gibi Html'den ayrı bir unsur değildir. Javascript Html'in bir parçasıdır.
- Kullanılacak yere göre Html'in içerisinde kullanılır. Fakat genelde < head ></head > etiketleri arasında kullanılır.
- Javascript kodları bittiğinde elinizde asla kendi başına çalışan uzantısı exe veya com olan bir dosya olmaz. Her zaman için tarayıcı tarafından yorumlanması gerekir. Yorumlanması demek Javascript kodunun çalışması anlamındadır.
- Nesne ve buna uygulanan olaylar ile ilgili bir takım görevleri vardır. Javascript kullandığı her unsuru nesne olarak algılar. Siz bu nesneleri tıklamak , üzerine gelmek , üzerinde çıkmak gibi olaylar ile çalıştırırsınız ki bu da Javascript'in ziyaretçi ile etkileşmesi demektir.
- Genel öğrenim yapımız diğer programlama dillerine nazaran biraz farklı olacaktır. Bu Javascript'in bir script dili olmasında ileri gelir.

2.2.2 Değişkenler

Genel Değişken Özellikleri

Değişkenler Javascript'te ve diğer programlama dillerinde olduğu gibi bilgi depolamak bu bilgiyi kullanmak amacıyla kullanılırlar. Değişkenler `var` komutu ile oluşturulurlar. Karakter olarak kullanıldıklarında işlem yapılamazlar. Nümerik olarak kullanıldıklarında ancak işlem yapabilirler.

```
var sayi;
var sayi1=10;
var yazil="10";
```

Burada birinci satırdaki sayı değişkeni script kodunun herhangi bir yerinde kullanılmak üzere oluşturulmuştur. İkinci satırda `sayi1` adındaki değişkenin değeri hemen o satırda `=` ifadesinden sonra verilmiştir. Böyle değişken tanımlı da yapılabilir. Üçüncü satırda ise değişkenin karakter ifadesi olarak kullanımını göstermektedir. Burada önemli olan karakter değişkenlerin alıntı `" "` ifadesinin arasında kullanılmasıdır. Her değişkenden sonra `;` işareti konulmalıdır. Tarayıcı, bir başka komut satırına geçtiğini bu yol ile anlar.

```
var sayi1=10;
var sayi2=20;
var sayi3=sayi1+sayi2;
```

Birinci ve ikinci satırlarda değişkenler oluşturulmuştur. Üçüncü satırdaki ise `sayi3` değişkeni ile diğer iki değişken toplanmıştır. Burada önemli olan işlem yapmak istediğimizde değişken değerinin alıntı `" "` işaretlerinin arasına *konmamasıdır*. Üçüncü satır - ileride göreceğimiz `write()` fonk-

siyonu ile - tarayıcıda yazdırırsak göreceğimiz değer 30'dur.

```
var sayi1="10";
var sayi2="20";
var sayi3= sayi1+sayi2 ;
```

Bir önceki örnekten farklı olarak değişken değerlerinin alıntı işaretleri içerisinde yazılmıştır. Eğer `sayi3` adlı değişken tarayıcıda yazdırılırsa göreceğimiz ifade 1020 ifadesidir. Yani tarayıcı karakter olarak tanımladığımız değişkenleri ardarda ekledi. Burada unutulmaması gereken şey bunun sadece `+` işleminde geçerli olmasıdır. Diğer işlem türlerinde bu tür bir sonuç alınmaz.

Değişkenlere Ad Verirken Uyulması Gereken Kurallar.

1) Değişken isimleri harf veya `_` karakteri ile başlayabilir. Rakam kullanmak istersek 2. karakterden sonra kullanılabilir. Yani değişkenin ilk karakteri rakam olamaz. Değişken isimlerine örnekler;

```
var url="turkport";           // doğru
var _rakam=12;                // doğru
var a1=123;                   // doğru
var 3uzler="üçüzler"         // yanlış
var x1;                       // doğru
```


2) Değişken tanımlarken bir veya birden fazla boşluk bırakmak tanımlama açısından herhangi bir sorun teşkil etmez.

3) Değişken adı verirken kullandığımız harflerin büyük veya küçük olması bazı tarayıcılarda fark etmezken çoğu tarayıcıda farklı bir değişken anlamındadır. Yani;

```
var say=1;  
var Say=1;
```

Birçok tarayıcıda farklı değişkenler olarak algılanır.

Değişkenlerin işlem operatörleri ile kullanımı

Değişkenlere işlem yaptırabilecek işlemcilere operatör denir. JavaScript'te 4 çeşit operatör vardır.

- Aritmetik operatörler
- Karşılaştırmak operatörleri
- Mantıksal operatörler
- Özel operatörler

Aritmetik Operatörler

Her zaman kullandığımız bu operatörler `+`, `-`, `*`, `/`, `%` 'dir.

```
var i=10;  
var j=11;  
var k=12;
```

```
var m,n;  
m=i*j+k;  
n=i*(j+k);
```

Değişkenler i,j,k,m,n

i=10; j=11; k=12; m, n ye başlangıçta bir değer atanmamış.

m=i*j+k = 10*11+12 = 122

n=i*(j+k)= 10*(11+12)= 230

```
var a=100; var b=9;  
  
var c=100%9; //c=100/9 dan kalan değerdir.(Yani 100 Mod(9)  
göre değeri)
```

Burada c değişkeninin değeri 100/9'un kalanı 1'dir. Yani c değişkeninin değeri 1 olacaktır. Diğer -(eksi) ve / (bölme) operatörlerinin işlemleri kendilerine atanan çıkartma ve bölme işlemidir. Bu operatörlerin kısa kullanımı içinde Javascript bize kolaylık sağlar. Bu operatörleri sıralamak istersek;

-- : *= : /= : %= : ++ : --

```
x+=y; //x=x+y anlamında  
x-=y; //x=x-y anlamında  
x*=y; //x=x*y anlamında  
x/=y; //x=x/y anlamında  
x%=y; //x=x%y anlamında  
x++; //x=x+1 anlamında  
x--; //x=x-1 anlamında
```

```
var x,y,z;  
x=10; y=20; z=30;  
x++; x+=y; z--; y*= z;
```

Değişkenler: x,y,z

x=10 ; y=20 ; z=30 ;

x++ x=x+1 x=10+1=11

x+=y x=x+y x=11+20=31

z-- z=z-1 z=30-1=29

y*=z y=y*z y=20*29=580

Karşılaştırma operatörleri

Bu operatörler değişkenlerin birbirleri ile karşılaştırılmak istendiğinde kullanılır.

Bu operatörler ise;

== operatörü iki değişkenin birbirine eşitliğini kontrol eder.

!= operatörü iki değişkenin birbirine eşit olmadığı durumlarda kullanılır.

< operatörü bilindiği üzere küçüktür operatörüdür. Soldaki değişkenin sağdakine küçüklüğünü

kontrol eder.

<= soldaki değişkenin sağdaki değişkene küçük eşitliğini kontrol eder.

> soldaki değişkenin sağdaki değişkene göre büyük olup olmadığını kontrol eder.

>= soldaki değişkenin sağdaki değişkene büyük eşitliğini kontrol eder.

Mantıksal Operatörler

Bu tip operatörler iki değişkene bağlı karşılaştırmaların yapılmak istendiği durumlarda kullanılır.

Operatörler && , || , ! operatörleridir.

&& And (ve) operatörü iki değişkenin de değeri doğru olması istendiğinde kullanılır.

|| Or (veya) operatörü iki değişkenden en az birinin doğru olması durumu istendiğinde kullanılır.

! Not (değil) operatörü değişkenin değeri doğru ise yanlış , yanlış ise doğru olması istendiği

durumlarda kullanılır.

Özel karşılaştırma Operatörü

Bu operatör iki değişken(deg) arasında karşılaştırma yapmanın en sade ve kısa yoludur. Operatörün kullanım biçimi :

```
deg1 [istenen karşılaştırma operatörü] deg2 ? deg3 :deg4
```

```
a < b ? c : d
```

Burada a değişkeninin b değişkeninden küçük olup olmadığı karşılaştırılıyor. Buna göre cevap doğruysa işlemin sonucu c değişkeninin değeri değilse d değişkeni oluyor.

Örnek 1:

```
<html>

<head><title>ornek01.html</title></head>

<body>

Değişkenlerin ilk değerleri<br>

i=1 j=2 k=3 m=4 n=5 p=6 q=7<br>

Değişkenlerin son değerleri<br>

<script Language="JavaScript 1.2">
<!--
var i=1; var j=2;           // Değişkenler tanımlanıyor...

var k=3; var m=4;
var n=5;

var p=6; var q=7;

i+=j;                       // i=i+j anlamında
j++;                        // j=j+1 anlamında
k--;                        // k=k-1 anlamında
m=m+k;
n*=j;                       // n=n*j anlamında
i < j ? 3 : 1 ;             // i<j ise işlemin sonucu 3 değilse 1

k <= n ? 0 : 1 ;
k=2 && j=5 ? p : q ;
```

```

i=2 || j=3 ? m : n ;

p!=2 ? k : 10 ;           // p,2 den farklı ise işlemin sonucu k
yoksa 10

document.write("i=",i,"j=",j,"k=",k,"m=",m,"n=",n,"p=",p,"q=",q);

-->
</script>

```

Değişkenler i,j,k,m,n,p,q i=1 ; j=2 ; k=3 ; m=4 ; n=5 ; p=6 ; q=7 ;

<u>İfade</u>	<u>Anlamı</u>	<u>Sonucu</u>
i+=j	i=i+j	i=1+2=3
j++	j=j+1	j=2+1=3
k--	k=k-1	k=3-1=2
m=m+k	-	m=4+2=6
n*=j	n=n*j	n=5*3=15
i<j?3:1	3<3 mü?	Hayır,1
k<=n?0:1	2>=15 mi?	Evet, 0
k=2 && j=5?p:q	k=2 ve j=5 mi?	Hayır,7
i=2 j=3?m:n	i=3 veya j=3 mü?	Evet, 6
p!=2?k:10	p,2 den farklı mı?	Hayır,10

2.2.3 Ekranı Çıktı ve Klavyeden Bilgi Girişi

prompt()

Ziyaretçiden bilgi alma iki tür JavaScript komutuyla gerçekleşir. Birisi prompt diğeri ise Form yoluyla bilgi alınması. Form yoluyla alınan bilgiler formun Html üzerinde yer alması yüzünden prompt komutu ile birbirinden ayrılır. prompt komutu ile Html sayfasından hariç bir pencere açılır. Alınmak istenen bilgi ziyaretçiye bu yol ile sorulur ve hemen altındaki boşluk yardımıyla cevap alınır.

```
prompt ("Sorulan soru" , "Cevap örneği")
```

Bu komutun yorumu şudur. Html üzerinde Html'den bağımsız bir pencere aç. (bu prompt komutu ile yapılır) İlk çift tırnak içerisinde olan kelime veya kelime grubu, pencerenin üst kısmında ve değiştirilemeyen kısımdır. Burada soru veya pencerenin niçin açıldığı ile ilgili bir açıklama verilir. İkinci çift tırnakta ise doldurulacak olan cevap satırının içinde seçili bir haldedir. Bu ise genel olarak

cevap örneği olarak ziyaretçiye sunulur. Kullanılması zorunlu değildir. Kullanılmadığınızda *undefined* gibi tanımlanmamış uyarısı alınır.

```
prompt("Tarayıcınızın Türü ?" ,"Cevabı ie veya nn olarak veriniz");
```

Şimdi kullanıcıdan nasıl bilgi alınacağını gördük fakat bu bilgiyi nasıl kullanabiliriz ? Bu sorunun cevabı prompt komutunu var ile bir değişkene atmak suretiyle kullanabiliriz.

```
var tara=prompt ("Tarayıcınızın Türü?" ,"Cevabı ie veya nn olarak veriniz");
```

Biz bu satır ile, ziyaretçiye prompt komutu ile tarayıcısı soruldu ve bunu var değişken tanımlama komutuyla tara değişkenine atandı. Ziyaretçiden alınan bu bilgi sonucunda tara değişkeni ya ie yada nn değerini alacaktır. Biz daha sonraki satırlarda bu değişkeni belli bir koşul koyarak kullanabiliriz.

write()

Html dosyasına yazı yazdırma komutu write dir. Bu kodun yazım kuralları;

```
document.write ("görüntülenmek istenenler" , değişken_ismi );
```

Javascript html üzerinde yazı yazmak istediğinde write komutunu tek başına kullanamaz. Bunun için document fonksiyoneli (yardımcısı manasında) ile birlikte kullanılır. document.write komutundan sonra parantez açılır. Daha sonra yazılmak istenen sıraya göre değişken ismi veya görüntülenmek istenenler yazılır. *Değişkenler çift tırnak içerisinde yazılmazlar. Sadece görüntülenmek istenenler çift tırnak içerisinde yazılır.*

Şimdi prompt komutu ile write komutunu birleştirerek bir kod hazırlayalım. Bu kodumuzda prompt aracılığıyla ziyaretçiye adını sorup ad değişkenine atanıyor. Daha sonra bu değişkeni write komutu yardımıyla ziyaretçiye Merhaba deniyor.

-

Örnek 2:

-

```
<html>
<head><title>ornek02.html</title></head>
<body>

<script>
  var isim = prompt ("Adınız" , "Küçük harf veya büyük harf
farketmez" );
  document.write ("Merhaba " , isim , " !" );
</script>

</body>
</html>
```

Aynı örnekte Merhaba dedikten sonra alınan ismin bir alt satırda görüntülenmesini istiyorsak bunu Javascript'e şu şekilde yaptırabiliriz.

```
document.write ("Merhaba" , "<br>" , isim) ;
```

Bu tür (yani
 türünde) Html deyimlerinin tümünü Javascript'e yaptırabilirsiniz.

alert()

Ekranın girilen iletiyi yazar.

```
alert("ileti...")
```

Örnek 3:

```
<html>

<head><title>ornek03.html</title></head>

<body>

    <script>

        alert("Sayfama Hoşgeldiniz")

    </script>

</body>

</html>
```

2.2.4 Koşul Yapıları

Hiçbir bilgisayar kendine göre yorum yapamaz. Bizim verdiğimiz belli kıstasları göz önünde bulundurarak seçim yapar. Diğer programlama dillerinde olduğu gibi JavaScript'te de koşul yapıları mevcuttur.

if,if-else

Javascript'te çoğu dilde olduğu gibi koşul yapısının kodu `if` deyimidir. Yani `if` deyimi koşullu işlem yapma deyimidir. `if` ve `else` tek bir karşılaştırma deyimini olup `else` in kullanımı isteğe bağlıdır. Eğer koşul olumlu ise küme yürütülür ve `else` den sonraki küme atlanır; olumsuz ise, `if` den sonraki küme atlanır ve eğer varsa, `else` den sonraki küme yürütülür.

if deyiminin kullanımı:

```
if (koşul){ ← küme başlangıcı  
...  
deyimler;[küme]  
...  
} ← küme sonu
```

if-else deyimlerinin kullanımı:

```
if(koşul){ /* koşul olumlu ise [küme1] */  
... /* koşul olumsuz ise [küme2] */  
deyimler;[küme1] /* yürütülür. */  
}  
else{  
...  
deyimler;[küme2]  
...  
}
```

Örnek 4:

```
<html>

<head><title>ornek04.html</title></head>

<body>

<script>

    var parola=1999;

    var gir=prompt("Parola:", "Parolayı girin?");

    if(gir==parola){

        document.write("Parola kabul edildi", "<br>", "Sayfa
Açılıyor...")

    }

    else{

        document.write("Parola kabul edilmedi", "<br>", "İyi
düşün!..")

    }

</script>

</body>

</html>
```

Örnek 5:

```
<html>
<head><title>ornek05.html</title></head>
<body>
<script language="JavaScript">
<!-- //eski sürüm tarayıcılardan kodumuzu saklayalım
var gun = prompt ("Bugün günlerden ne ?" , "lütfen küçük harf
kullanınız");
```

```

if (gun=="pazar")
{
document.write ("Bugün ",gun ," olduğuna göre hafta
sonundayız" ,"<br>")
document.write ("<b>" , "İyi tatiller.." , "</b>")
}
else
{
document.write ("Bugün pazar olmadığına göre Hafta sonu
değil!" ,"<br>")
document.write ("<u>","İyi çalışmalar..","</u>")
}
//saklamayı bitir-->
</script>
</body>
</html>

```

switch()

Bir değişkenin içeriğine bakarak, programın akışını bir çok seçenektan birine yönlendiren bir karşılaştırma deyimidir. Bu deyim BASIC dilinde ONGOSUB ve Pascal dilinde Case deyimine benzerdir. C dilinde ise karşılığı ayıdır. Bu deyimın genel yazım biçimi;

```

switch(değişken){

    case "sabit1" :

        ...deyimler;

    case "sabit2" :

        ...deyimler;

    ...

    case "sabitN" :

        ...deyimler;

}

```

```
var sec;
sec = prompt ("Çıkmak istiyor musunuz "
,"Evet(E/e);Hayır(H/h)")
switch (sec){
    case "e" : case "E" :
        document.write ("Tekrar hoşgeldiniz")
//yapılması istenen işlemler
    case "h": case "H" :
        document.write ("Bizi tercih ettiğiniz için teşekkürler")
break;

//Çıkılması istendiği için döngüyü kesmek için break komutunu
//kullanıyoruz. İleride break deyimi açıklanacaktır.
```

2.2.5 Döngü Deyimleri

Bu tip deyimler bir kümenin belli bir koşul altında yinelenmesi için kullanılır. while,do...while ve for olmak üzere üç tip döngü deyimi vardır.Javascript'te diğer programlama dillerinde olduğu gibi istediğiniz işlemi 2 veya daha fazla kez yaptırmak için belli program kodları mevcuttur.

while

Tekrarlama deyimidir. Bir küme ya da deyim while kullanılarak bir çok kez yinelenebilir. Yinelenmesi için koşul sınaması çevrim başında yapılır. Genel yazım biçimi;

```
while(koşul){
    ...
döngüdeki deyimler; [küme]
    ...
}
```

Koşul olumlu olduğu sürece çevrim yinelenir. İki veya daha çok koşul mantıksal operatörler birleştirilerek verilebilir.

Örnek 6:

```
<html>

<head><title>ornek06.html</title></head>

<body>

<script Language="JavaScript">

<!--
var yil=0;var para=1000;

    while(para<50000){

        para+=para*0.75; // para=para+para*0.75;

        yil++;

    }

    document.write("1000 TL ",yil," sonra ",para," TL olur.")

-->

</script>

</body>

</html>
```

do...while

Bu deyim **while** dan farkı, koşulun döngü sonunda sınanmasıdır. Yani koşul sınanmadan çevrime girilir ve döngü kümesi en az bir kez yürütülür. Koşul olumsuz ise döngüden sonraki satıra geçilir.

Genel yazım biçimi;

```
do{  
    ...  
    döngüdeki deyimler;  
    ...  
}while(koşul);
```

Örnek 7:

-

```
<html>  
  
<head><title>ornek07.html</title></head>  
  
<body>  
  
<script Language="JavaScript">  
  
<!--  
  
    do{  
  
        var sayi=prompt("Girilen Sayının Karesi","Bir sayı  
giriniz");  
  
        document.write("Sayı=",sayi," Karesi=",sayi*sayi,"<br>")  
  
    }while(sayi>0);  
  
  
    document.write("Çevrim sona erdi...")
```

```
-->

</script>

</body>

</html>
```

for

Diğer döngü deyimleri gibi bir öbeği bir çok kez tekrarlamakta kullanılır. Koşul sınaması `while` da olduğu gibi girmeden yapılır. Bu döngü deyiminin içinde diğerlerinden farklı olarak başlangıç değeri ve döngü sayacına sahip olmasıdır.

Genel yazım biçimi;

```
for(başlangıç;koşul;artım){

    ...

    döngüdeki deyimler;

    ...

}
```

-

Örnek 8:

```
<html>
<head><title>ornek08.html</title></head>
<body>
<script language="JavaScript">
<!--

    for(sayi=0;sayi<=10;){
        sayi++;
        document.write( "5 * ",sayi," =",5*sayi,"<br>")
    }
-->
```

```
</script>
</body>
</html>
```

-

Örnek 9:

```
<html>

<head><title>ornek09.html</title></head>

<body>

<script>

var i;var fact=1;

var sayi=prompt("Faktoriyel Hesabı","Faktoriyeli hesaplanacak sayı")

for(i=1;i<=sayi;i++){

    fact*=i;

}

document.write(sayi," != ",fact)

</script>

</body>

</html>
```

Not: iç-içe bir çok döngü kullanılabilir.

break ve continue İfadeleri

Döngü deyimleri içindekiler yürütülürken, çevrimin, koşuldan bağımsız kesin olarak sonlanması gerektiğinde veya döngünün bir sonraki çevrime geçmesi istendiğinde bu deyimler kullanılır.

Örnek 10:


```
<html>

<head><title>ornek10.html</title></head>

<body>

<script>

var x=0;var y=1;var z=6;

do{

    y++;

    x+=y;

    if(x>=z) {break; }    // while deki koşula bakılmaksızın döngü sonuna gider.

    if(y<=3) {continue;}// y<=3 olduğu sürece döngü bir sonraki çevrime girer.

}while(x<10);           // x<10 olduğu sürece çevrime devam et.

document.write("x=",x ,"y=",y ,"z=",z);

</script>

</body>

</html>
```

2.2.6 Fonksiyon Kavramı

Çoğu zaman Javascript kodunuzda bir işlemin birden fazla yapılması gerekebilir. Hatta kimi zaman Javascript'e bir işlem yaptırmadan önce başka bir işlemi yaptırmak istenebilir. İşte bu tür tekrarlanan işin yapılması için gerekli işlem ve komut gruplarına Fonksiyon adı verilir. Fonksiyonlar genelde, filanca isimli gruptaki işlemleri yap oradan bir değer al bunu filanca isimli gruba götür gibi işlemler için kullanılır. Bu tür komut sistemleri Javascript'te en çok kullanılan komut türlerindendir. Fonksiyonun yazım kuralları şu şekildedir ;

```
function fonksiyon_ismi(parametre1, parametre2 , .... )
{ yapılması istenen işlemler;}
```

Fonksiyona Değer Gönderme ve Değer Alma

Bir fonksiyonun Javascript içerisindeki ilk önemli görevi diğer fonksiyonlardan veya herhangi bir yerden bir değer alıp onu kendi içerisinde işletip sonra istenilen fonksiyona veya yere göndermektir. Bir sonucu bir fonksiyondan çağırmak için `return` deyimi kullanılır.

Örnek 11:

```
<html>

<title>ornek11.html</title>

<head>

    <script language="JavaScript">

        function carp(a,b){

            var c=a*b;

            return t;

        }

    </script>

</head>

<body>

    <script language="JavaScript">

        var x = prompt("İki Sayının Çarpımı","Birinci Sayı");

        var y = prompt("İki Sayının Çarpımı","İkinci Sayı");

        var z = carp(x,y);

        document.write( x,"*",y,"= ",z);

    </script>

</body>

</html>
```

2.2.6 Nesneler ve Özellikleri

Günümüzde sıkça kullanılan bir deyim *Nesneye Yönelik Programlama*. Nedir bu Nesneye Yönelik programlama ? Bu tip programlamada kullanılan her öge bir nesne olarak kabul edilir. Bu nesnelerin özelliklerini kullanarak onları değiştirerek program yazılır. Javascript'te bu tür bir programlama dilidir. Örneğin webde sörf yaparken herkesin karşısına çıkan formlar birer nesnedir. Bu nesnelerin tepkiye göre cevap vermesi gibi özellikler de onun yani nesnenin özellikleridir.Örneğin şimdiye kadar çoğu kez kullanılan `document.write` komutu aslında bir nesnenin özelliğine atıfta bulunmaktan başka bir şey değildir. Yani `document` nesnesinin `write` özelliğini kullanarak html sayfamıza yazı yazdırıyoruz.

window Nesnesi

Genel olarak pencere özellikleri ile ilgili bir nesnedir.

Pencere açmak ve kapamak

Birçok yerde gördüğünüz pencere açma pencerelerin çeşitli özelliklerini değiştirme bu nesne yardımıyla yapılmaktadır.

Pencere açmak için :

```
window.open("Url_adı" , "pencere_adı" , "pencere_özellikleri");
```

Pencere kapatmak için :

```
window.close();
```

Pencere kapatmak için `window.close()` komutu vermek yeterlidir. Burada kapatılan pencere ona kullanılmakta olan penceredir.Pencere açma işleminde `window.open()` ile pencerenin açılmak istendiği belirtilir. Parantez içerisinde verilenler ise açılması istenen pencerenin özelliklerini belirtir.

`Url_adı` : Buraya yazılacak dosya ismi açılacak pencerenin içerisinde olacaktır.

```
window.open( "http://www.gantep.edu.tr" )
```

veya;

```
window.open( "index.html" )
```

Pencere_adı : Bu açılacak pencerenin adını belirtir. Birden çok pencere ile işlemler yapıyorsanız hangi pencereye bir komut gönderdiğinizin belli olması için gereklidir.

```
window.open( "index.html" , "ana" );
```

Pencere_özellikleri : Bu özellikte adından belli olduğu ölçüde pencerenin özellikleri ile ilgilidir. Bir pencerenin değiştirilebilir özellikleri şunlardır :

menubar : Tarayıcıların en üst kısmında bulunan File(Dosya) , Edit(Düzen) vb. menülerin bulunduğu satırdır.

toolbar : Tarayıcılarda üst kısımda Back(Geri) , Forward(İleri) vb. tuşların bulunduğu kısımdır.

location : Tarayıcılarda ziyaret etmek istediğiniz web adresini yazdığınız kısım.

status : Tarayıcıların en alt kısmında hangi dosyanın yüklendiği ile ilgili bilgi veren kısımdır.

scrollbars : Sağ tarafta bulunan sürgü çubuklarıdır.

resizable : Pencerenin boyutlarının kullanıcıya bırakılması veya kesin değerler almasıyla

ilgilidir.

width : Açılacak olan pencerenin piksel cinsinden genişliğidir.

height : Açılacak olan pencerenin piksel cinsinden boyudur.

left : Açılacak olan pencerenin ekranın sol tarafından kaç piksel uzaklıkta olacağını

belirler.

top : Açılacak olan pencerenin ekranın üstünden kaç piksel aşağıda olacağını belirler.

Eğer pencere özellikleri kısmında değişmesini istemediğiniz bir özellik varsa onu

yazmanıza gerek yoktur. Bu değerler tarayıcının banko(default) değerlerinden alınır.

200x300 ebatlarında <http://www.gantep.edu.tr> adresine bağlanan bir pencere açmak için;

```
window.open  
("http://www.gantep.edu.tr", "ana" , " menubar=no, toolbar=no,  
scrollbars=yes, location=yes, width=200, height=300");
```

window.location.protocol

window nesnesinin location.protocol nesnesi ise yüklenen dosyanın sabit diskten mi yoksa internetten mi yüklendiğini gösterir.

```
if (window.location.protocol == "http:" )  
{ document.write ("Bu belge Internet'ten geliyor.") }  
else  
{ document.write ("Bu belge sabit diskten geliyor") }
```

http: Dosyanın internetten yüklendiğini belirtir.
file: Dosyanın sabit diskten yüklendiğini belirtir.

window.history.go

window un history özelliği ile bir önceki sayfaya erişim sağlanabilir. Örneğin kullanıcı herhangi bir formu doldurmadı ve işlem yapılamadı bu durumda bir hata mesajı ile kullanıcıyı uyardıktan sonra history nesnesinin kullanarak bir önceki sayfaya kullanıcı gönderilebilir.

```
window.history.go(-1)
```

Bir önceki sayfaya -1 ile ulaşabilirsiniz. Bu değeri arttırarak daha önceki sayfalara da ulaşabilirsiniz.

window.status.bar kullanımı

status.bar, window nesnesinde belirttiğimiz gibi tarayıcıların en alt kısmında yer alan hangi dosyaya gidileceği veya yüklendiği ile ilgili bilgi veren kısımdır.

```
window.status="Ahmet Hoca iyi günler diler!";
```

window.moveTo(x,y)

IE veya NN penceresini bir x,y ile belirlenmiş noktaya taşımak için kullanılır. Taşınan nokta ekranın sol üst köşesidir. Normalde bu noktanın varsayılan değeri (0,0).

```
Window.moveTo(100,100); // Internet penceresini 100,100  
noktasına taşır.
```

Döngü deyimleri kullanılarak IE veya NN penceresi hareketli duruma getirilebilirsiniz.

Örnek 12:

```
<html>

<head><title>ornek12.html</title></head>

<body>

<script>

<!--

    var y;

    for(y=400;y>=0;y-=8){ // x değeri her zaman 0, y değeri  
400. pikselden

        window.moveTo(0,y); // başlıyor.

    }

-->

</script>
```

```
<center>Web Sayfama Hoşgeldiniz...</center>

</body>

</html>
```

Tarayıcı Nesnesi

Tarayıcılar Javascript tarafından bir nesne olarak algılanır. Bu nesnenin özelliklerini şöyle sıralayabilir.

appName : Tarayıcı adı
appVersion : Tarayıcının Versionu
appName: Tarayıcının kod adı
userAgent : Tarayıcının ana makinaya(server) kendini tanıtırken verdiği isim

Örnek 13:

```
<html>
<head><title>ornek13.html</title></head>
<body>
<script language="javascript1.2">
<!--
    document.write("Kullandığınız tarayıcının özellikleri : " ,
" <br> " );
    document.write(navigator.appname + navigator.appVersion +

    navigator.appCodeName + navigator.userAgent );
-->
</script>
</body>
</html>
```

2.2.7 Internet Explorer & Netscape Navigator

Giriş kısmında belirtildiği gibi Javascript kodlarında MSIE (Microsoft Internet Explorer) ve NN (Netscape Navigator) yönünden farklılık vardır. Bu tarayıcının html dökümanı nasıl modellediğine

bağlıdır. Tarayıcının nesne döküman modeli, bir Html sayfasındaki çeşitli elemanların tarayıcı tarafından nasıl algılanıp yorumlandığı ile ilgilidir. Tarayıcı farkının nasıl ayırt edilebileceğini aşağıda anlatılmıştır.

```
ie4 = (document.all) ? true : false ;  
nn4 = (document.style) ? true : false ;
```

Bu iki satırla ,önceki ders olan değişkenler ve mantıksal operatörler yardımıyla, iki tarayıcıyı da kontrol altına alınmıştır.

```
<script language="Javascript">  
<!-- // Kodları eski sürüm tarayıcılardan saklayalım.  
ie4 = (document.all) ? true : false ;  
nn4 = (document.style) ? true : false ;  
if (ie4){  
    // MSIE 4.0 için uygun kodları buraya  
  
}  
else{  
    // NN 4.0 için uygun kodları buraya  
  
}  
// Saklamayı bitir -->  
</script>
```

If(ie4) ve if(nn4)

Bu kodları Javascript'in anlayış tarzı şu şekilde olacaktır. Eğer nn4 , ie4 değişkenlerinden doğru olanı ie4 ise -ki bunu true ve false değerlerinden algılar- alt satıra geçip ona uygun kodu uygulayacaktır. Şayet ie4=false yani nn4=true ise diğer if koşulu yorumlanarak işleme konulacaktır. Bu da nn4 için gerekli kodun çalıştırılması demektir. Aşağıda verilen örnekleri dikkatlice inceleyin.

Örnek 14-1:

```
<html>  
<head><title>ornek14-1.html</title><head>  
<script language="Javascript">  
<!--  
  
function tarayici() {
```



```
ie4 = (document.all) ? true : false ;
nn4 = (document.style) ? true : false ;

if (ie4){window.location="ornek14-2.html";}
else{ window.location="ornek14-3.html"; }
} // tarayici

//Saklamayı bitir -->
</script>
</head>

<body onLoad=tarayici(>
</body>
</html>
```

-

-

Örnek 14-2:

```
<html>
<head><title>ornek14-2.html</title></head>
<body><h3>Tarayıcınız Internet Explorer</h3></body>
</html>
```

Örnek 14-3:

```
<html>
<head><title>ornek14-3.html</title></head>
<body><h3>Tarayıcınız Netscape Navigator</h3></body>
</html>
```

Önemli! : Bu üç (14-1-2-3) Html dosyasında aynı klasör de olması gereklidir.

2.2.8 Olaylar

Ziyaretçiye sunulan bir web sayfası üzerinde ziyaretçinin yaptığı her tür hareket bir bağlantıyı tıklaması , bir resmin üzerine gelmesi , resmin üzerinde ayrılması , bir formu yanlış doldurup hataya yol açması hep bir olaydır.

onMouseOver , onMouseOut

Bu tür nesne olayları ,ingilizce adı onMouseOver = fare işaretçisi(imleç) üzerindeyken , onMouseOut = fare işaretçisi üzerinden ayrıldığında, fare işaretçisinin istenen bir linkin üzerindeyken ve değilken açıklama yapmak için kullanılır.

-

Örnek 15:

```
<html>
<head><title>ornek15.html</title>
<script language="javascript1.2">
<!--
function uzerinde()
{window.status="Tıklayın ve Gaziantep Üniversite sine
bağlanın" }
function disinda()
{window.status="Gaziantep Üniversitesi ne bağlanmak
istiyormusun?" }
-->
</script></head>
<body>
<a href="http://www.webteknikleri/index.htm" onMouseOver =
uzerinde()

onMouseOut =disinda()> Webteknikleri.com </a>
</body>
</html>
```

onLoad , onUnload

Bu olaylar bize sayfanın yüklenmeye başlamasında (onLoad) sayfadan ayrılınca (onUnload) kadar olan yapılacak işlemler için gereklidir. Bir Javascript fonksiyonun web sayfası yüklenmeye başladığında otomatik olarak çalışmasını istiyorsak onLoad olayını kullanırız. Autoexec.bat dosyası nasıl makine açıldığında yapılmak istenenleri yapıyorsa onLoad olayında da sayfa yüklenmeye başladığında nelerin otomatik olarak başlatılacağını belirleyebiliriz. Mesela sayfa yüklenmeye başladığında ziyaretçiye Web sitemiz hoş geldiniz diyebiliriz. Sayfadan ayrıldığında ise Hoşçakalın diyebiliriz. Web sayfası kod açısından iki kısma ayrılır. Bunlar head ve body kısmıdır. Tarayıcı açısında body kısmı asıl kısımdır. head kısmında sayfanın nasıl görüntüleneceği gibi bölümler yer alır. Bu yüzden onLoad ve onUnload kısmı body etiketleri arasında yer alır.

Örnek 16:

```
<html>
<head>
<title>ornekl6.html</title>
<script language="javascript1.2">
<!--
function hosgeldiniz()

{alert("Web Sitemize Hosgeldiniz")}
function gulegule()

{alert("Hoşçakalın")}
-->
</script>
</head>
<body onLoad="hosgeldiniz()" onUnload="gulegule()">
</body>
</html>
```

onError

Ziyaretçi sayfayı herhangi bir neden yüzünden tam haliyle yükleyememiş olabilir. Bu nedenler aktarım hızı veya tarayıcının Javascript kodunu tam manasıyla yorumlayamamış olmasıdır. İşte bu durumda Error(hata) oluşur. Html üzerinde oluşan en sık error(hata) resim haritalarının (image-map) tam anlamıyla yüklenmemesinden kaynaklanır. Çünkü bu durumda resim tam yüklenmemiştir. Bu da ziyaretçinin resim üzerinde tıklayacağı yerlerin yorumlanmamasını doğurur.

```

```

onmousedown ve event.button

Web sayfanızı ziyaret eden bir kişinin farenin sağ tuşu ile işlem yapmasını istemiyorsanız bu iki deyimi kullanmalısınız.

Örnek 17: Bu program parçasını mutlaka web sayfanıza ekleyin!..

-

```
<html>

<head>

<title>ornek17.html</title></head>

<body>

<script language="JavaScript">

<!--

    document.onmousedown=click;

    function click(){

        if((event.button==2) || (event.button==3)){
alert("Oynama...");}

        }

-->

</script>

</body>

</html>
```

2.2.8 Javascript ile DHTML (Dinamik HTML)

Bu kısımda Javascript ile Katman(layer) özelliklerinin nasıl değiştirilebileceğini göreceğiz. Javascript bize html sayfamızı oluşturan önemli unsurlardan biri olan layer(katman) ların tüm özelliklerini değiştirmemize olanak sağlar. Ayrıca hemen her yerde gördüğünüz resim değiştirme tekniğini de bu işlemle yapılır.

Katman Özelliklerini Değiştirme

Katman sayfanın üzerinde ne sayfadan bağımsız ne de her yönüyle sayfaya bağlı bir unsurdur. Katman kullanarak istediğimiz herhangi bir yapıyı (yazı,resim,video,form) sayfamızın istediğimiz yerine koordinatları vermek koşulu ile yerleştirebiliriz. Zaten katmanın kullanım alanı en çok budur. Şimdi bir katman oluşturalım ve değiştirilebilir özelliklerini görelim.

Örnek 18:

```
<html>
<head><title>ornek18.html</title></head>
<body>
<div id="deneme" style="position:absolute ; left:100px ;
top:200px;
width:300px ; height:400px ; visibility:visible" >
Su anda bir katman(layer)in icerisindeyim

</div>
</body>
</html>
```

Layer oluşturmak istediğinizde <div> deyimi ile başlar </div> deyimi ile kodunuz tamamlarsınız.

id : Katmanın ismi

style : Katmanın özelliklerini belirtmek için

absolute : Katmanın koordinatlarının kesin olacağını belirler

left : Katmanın soldan kaç piksel sonra başlayacağını belirler

top : Katmanın üstten kaç piksel sonra başlayacağını belirler

width : Katmanın kaç piksel genişliğinde olacağını belirler

height : Katmanın kaç piksel boyunda olacağını belirler

visibility : Katmanın görünür mü görünmez mi olacağını belirler

Şimdi de Javascript komutlarıyla bu özelliklerin nasıl değiştirildiğini görelim. Fakat burada karşımıza bir sorun çıkmaktadır. IE ve NN tarayıcılarının doküman nesne modelleri farklı olduğundan katmana ulaşma teknikleri de farklıdır.

Internet Explorer kod tekniği

```
katman_adı.style.değiştirilmesi_istenen_özellik=yeni_değer;
```

Örnek :

```
deneme.style.left=50px;
```

Netscape Navigator kod tekniği

```
document.katman_adı.değiştirilmesi_istenen_özellik=yeni_değer;
```

Örnek :

```
document.deneme.left=50px;
```

Örnek 19: Bu örnek yukarıdaki iki örneği de içermektedir.

```
<html>
<head><title>ornek19.html</title>
<script language="javascript1.2">
<!--
  function tara(){

    var tarayici= navigator.appName
    if (tarayici=="Netscape") degisim = document.katman;
    if (tarayici=="Microsoft Internet Explorer") degisim =
katman.style;

  }//tara

  function hareket1(){
```

```

degisim.left=100

degisim.top=100

} //hareket1

function hareket2(){

degisim.left=300
degisim.top=300

} //hareket2
-->
</script></head>
<body onLoad="tara()">
<div id="katman" style="position:absolute ; left:400px;
top:10px">
Bu katmanın yeri degisecek
</div>
<p><p><p>
<a href="javascript:hareket1()">Burayı tıklayın ve katmanınız
100x100'e gitsin</a><br>
<a href="javascript:hareket2()">Burayı tıklayın ve katmanınız
300x300' gitsin</a>
</body></html>

```

Buradaki örnekte olduğu gibi sizde katmanın diğer özelliklerini (width,height)değiştirebilirsiniz. Fakat görünebilirlik özelliği için özel bir durum vardır. Katman özelliklerine erişimde olduğu gibi bu özellikte de Internet Explorer ve Netscape Navigator farklılıkları vardır.

Internet Expolorer için görünebilirlik özelliği Katmanı görünebilir kılmak için:

```
katman_adı.style.visibility="visible"
```

Katmanı gizleyebilmek için.

```
katman_adı.style.visibility="hidden"
```

Netscape Navigator için Görünebilirlik özelliği Katmanı görünebilir kılmak için:

```
document.katman_adı.visibility="show"
```

Katmanı gizleyebilmek için.

```
document.katman_adı.visibility="hide"
```

Örnek 20:

```
<html>
<head><title>ornek20.html</title>
<script language="javascript1.2">
<!--
    function sakla(){

    var tarayici= navigator.appName
    if (tarayici=="Netscape")

        document.katman.visibility="hide"
    if (tarayici=="Microsoft Internet Explorer")

        katman.style.visibility="hidden"

    }//sakla

    function goster(){

    var tarayici= navigator.appName
    if (tarayici=="Netscape")

        document.katman.visibility="show"
    if (tarayici=="Microsoft Internet Explorer")

        katman.style.visibility="visible"

    }//goster
-->
</script></head>
<body>
<div id="katman" style="position:absolute ; left:400px;
top:10px">
Bu katmanin tikladiginizda yok olacak
</div><p><p><p>
<a href="javascript:sakla()">Burayi tiklayin ve katmaniniz
yok olsun</a><br>
<a href="javascript:goster()">Burayi tiklayin ve katmaniniz
geri gelsin</a>
</body></html>
```

Sizde bu tıklama özelliklerin değil de onMouseOver ve onMouseOut olay yönlendiricilerini kullanarak çok daha güzel şeyler üretebilirsiniz.

ÖNSÖZ

Server-client mimarisine sahip programların birisi de Java Programlama diline ait veri tabanı uygulamaları JDBC'dir. 1980 yılı ve sonrası, İnternet ve ağ uygulamaların da Java ve Java uygulamaları büyük gelişme göstermiştir. JDBC veri tabanına bağlanmakta kullanılan arayüzdür.

Bu tezin hazırlanmasındaki amaç, JDBC konusunda örnek vermenin yanında, JDBC nin çalışma mantığını anlatmak ve konuyla ilgili ortaya çıkan kaynak sıkıntısı biraz olsun azaltabilmektir.

Gerek ders konusunda, gerekse sosyal hayatta bizlerden yardımını esirgemeyen değerli hocam Zafer DİNÇ'e sonsuz teşekkürlerimi sunarım.

ÖZET

Java programlama dilinde veri tabanına bağlantı için JDBODBC ara yüzü kullanılır. Temel mantık ana server-client mimarisinde server üzerinde bağlantı nesnelerinin oluşturulup, veri tabanında kayıtlara ulaşılması ve istenen değişikliklerin yapılmasıdır.

JDBC, ilişkisel veri tabanlarına ara yüz olarak tanımlanabilir. Java Veri tabanı bağlantısı (JDBC) sınıfları SQL cümlelerini çalıştırarak ilişkisel veri tabanlarına erişimi ve bu veriler üzerinde işlem yapmayı sağlar.

JDBC ile veri tabanı bağlantısını sağlamak için server üzerinde bağlantı nesnesi oluşturulur, JDBC sürücüleri yüklenir, SQL cümleleri ile veri tabanı üzerinde istenen değişiklikler yapılır ve bağlantı kapatılır.

BÖLÜM 1

GİRİŞ:

Bilişim teknolojilerinin gelişim hızı son 15 yılda inanılmaz bir ivme kazanmıştır. Bu gelişimde bilişim sistemleri kuruluşlarının başarılarının büyük payı vardır. Ancak bu gelişimle birlikte bu kurumlarının istek ve ihtiyaçları da değişmiştir, artık bilişim dünyası ana bilgisayardan (Mainframe) istemci_sunucu mimarisine geçiş gibi köklü bir değişiklik yaşamaktadır. Bu gün artık ihtiyaçlar veri tabanları, işletim sistemleri, bilgisayar ağları gibi varolan teknolojilerle uyumlu çalışabilecek ve aynı zamanda yeni teknolojilerin kullanılmasına olanak verebilecek mimarilerin geliştirilmesine yol açmıştır.

Bilişim teknolojilerinde ana bilgisayar tabanlı uygulamaların sakıncalarında dolayı ilişkisel veri tabanları ve istemci/sunucu modeline geçilmiştir. Bilindiği gibi ana bilgisayar tabanlı uygulamalarda depolanan bilginin paylaşılması büyük bir sorun olmaktaydı ve bu yöntem verimsizdi. Geliştirilen istemci/sunucu modelinde, uygulamalar istemci ve sunucu tarafında iki ayrı tek parçalı uygulama modelinin çıkmasına yol açmıştır. Ancak bu modelde, uygulamalarda değişiklik ya da geliştirmeler yapılmasını kolaylaştıran “kodun yeniden kullanılabilmesi özelliği” özelliğinin uygulanması, çoğu zaman ayrı ayrı ele alınması gereken bir çok program modülünün ortaya çıkmasına sebep olarak işlemi güçleştirir.

İşte yukarıda sayılan bu sebeplerden dolayı, dağıtık bilgi işlem modeli, CORBA ve IIOP’ nin geliştirilmesini sağlamıştır. 1980 yılında standart olarak kabul edilen bu teknolojide zamanla ortaya çıkan bir sorun, birbirlerinden kesin olarak çizilmiş sınırlarla ayrılmış ara yüzlerle etkileşim içerisinde olan ve tekrar kullanılabilen küçük yazılımlar modülü, nesneye yönelik programlama yaklaşımı ile çözülmüştür.

Dağıtık bilgi işleme modeli, farklı bilgisayar ağları ve farklı işletim sistemlerinde çalışabilir. Farklı programlama dillerinde yazılmış olan uygulamalar birbirleriyle sorunsuz olarak çalışabilmektedir.

İşte burada asıl konumuz olan Java veri tabanı uygulamaları geliştirme ara yüzü olan JDBC ortaya çıkmıştır. Birbirinden bağımsız olarak geliştirilmiş CORBA(Common Object Request Broker) ve Java dili yukarıda sayılan güçlüklerin aşılmasını sağlamıştır. Java farklı işletim sistemleri ve ağlar üzerinde sorunsuz çalışmasını, ortamdan bağımsız olması yani, diğer programlama dilleri gibi işletim yada donanım üzerinde çalışmak yerine, Java Sanal Makinesi (Java Virtual Machine JVM) adı verilen bir teknolojiyle sağlamıştır.

AMAÇ

Bu çalışmada Java programlama dili kullanılarak ağ ortamında çalışabilecek veri tabanı uygulamaları geliştirilmesi amaçlanmaktadır. Bu yüzden öncelikle veri tabanı uygulamaları geliştirebilecek yeterliliği sahip olmak için Java programlama dili veri tipleri, sınıf yapısı ve sınıf oluşturma, uygulamaların görsel olması için grafik kütüphanesi ve grafik uygulamaları, en son olarak da veri tabanı uygulamaları anlatılmıştır. Uygulamaların çalıştırılmasında herhangi bir Web Browser 'a (Web tarayıcısı) bağlı kalmadan Java ile birlikte hazır olarak gelen Appletviewer kullanılmıştır.

BÖLÜM 2

2.1 JAVA NEDİR?

Java™ platformu güçlü ağ uygulamaları için geliştirilmiş ve değişik bilgisayarlarda çalışabilen bir programlama dilidir.

Java™ teknolojisi ile aynı uygulamaları bir kişisel bilgisayar, Macintosh bilgisayarda, ağ bilgisayarında ve görüntülü Internet telefonu gibi bir çok yeni teknolojiye kullanabilirsiniz. (www.java.sun.com)

2.2 JAVA PROGRAMLAMA DİLİNİN ÖZELLİKLERİ

2.2.1 Her ortamda çalışabilir

Java teknolojisinin en önemli özelliği her ortamda, en küçük bilgisayarlardan super bilgisayarlara kadar, çalışabilmesidir. Java teknolojisi bileşenleri nasıl bir bilgisayar, telefon, televizyon, veya işletim sistemi olduğuna bakmaksızın Java platformunu destekleyen her türlü ortamda çalışır. Java teknolojisinin yaratılış amacı daha önce hiç kolay olmadığı kadar kolay bir şekilde bilgisayar ve diğer iletişim araçları arasında etkileşim kurmaktır. Java ilk ortaya çıktığında asıl amacı elektronik ev aletlerinin birbirleriyle iletişim kurabilmesini sağlamaktır. Java hızlı bir şekilde gelişerek bu gün ki halini almıştır.

2.2.2 Basit

Java ile program hazırlamak oldukça kolay ve zevklidir. Daha önce bir programlama dili ile çalışmış veya en az bir nesneye programlama dili ile çalışmak yeterlidir. Ayrıca Java programlama dilinin C++ programlama diline olan benzerliği sebebiyle C++ programlama dili çalışmış olanlar için çok zevkli yeni bir deneyim olacaktır.

2.2.3Object – Oriented

Giriş bölümünde bahsedildiği gibi nesneye dayalı bir programlama dilidir. İleri ki bölümlerde anlatılacağı gibi sınıf ve nesne yapılarıyla daha kolay, defalarca kullanılabilen modüller yaratılabilir.

2.2.4 Güvenli

Java programlama dili hazırlanan programlar bugün güvenliğin çok önemli üst düzey devlet kuruluşlarında ve NASA da kullanılmaktadır. Java ile hazırlanan programlara virüslere ve hackerlara karşı geliştirilen en güvenli programlar olarak düşünülmektedir.

2.2.5 Yüksek Performans

Java ile hazırlanan programlar hızlı çalışmaları ve bilgisayara az yük getirmeleri nedeniyle İnternet ve intranet uygulamalarında tercih edilmektedir.

2.2.6 Server Üzerine Az Yük

Java diğer Web tabanlı programlama dillerine göre bilgisayara daha az yük getirmektedir.

2.3 JAVA VE CGI KARŞILAŞTIRMASI

2.3.1 İnteraktif Standartların Savaşı :CGI ve JAVA

İnternet üzerinde veri işlemeye yönelik bir çok standart vardır. Bunlardan biride CGI (Commom Gateway Interface) ‘dır. CGI global değişkenler ve dosyaları kullanmak

gibi programlama dili özelliklerini kullanarak istemci ve ana bilgisayar arasında veri değişimi sağlayan başarılı bir standarttır.

CGI standardı basit çözümü olan uygulamalarda kullanılır. Örneğin; “Ben sana adımı ve e-mail adresimi göndereceğim, sen bu bilgiyi al,sakla ve bana bu bilgileri aldığını bildir” tarzında uygulamalar için kullanılır. Bu seviyenin üzerindeki uygulamalar ise Java programlama dilinin yeteneklerine ihtiyaç duyarlar. Örneğin İnternet üzerinde kullanılan bir satranç oyunu. Kullanıcıların her hamlesinde CGI standardı yeni istek ve cevap nesnesi oluşturması gerekirken Java programlama dili bunu oyunu girişte kullanıldığı tek bir applet ile sağlamaktadır.

Java programlama dili ile CGI standardı arasındaki diğer önemli fark ise CGI ‘ın ortama bağımlı olmasıdır. Yani CGI scriptlerinin istemci bilgisayarda çalışması için uygun bir işletim sisteminin ana bilgisayardaki scripti destekler olması gerekmektedir. Halbuki Java uygulamaları ‘Java-aware’ destekleyen herhangi bir browser da sistemden bağımsız olarak çalışabilmektedir.(Manger,1988)

2.4 JAVA PROGRAMLAMA DİLİYLE İLGİLİ SIKÇA SORULAN SORULAR

2.4.1 Programla ilgili sorular

2.4.1.1 Niçin Java ismi?

Neden Java olmasın? Java ismi Web de insanların aklında hemen yerleşebilecek bir kelimenin arandığı beyin fırtınasında ortaya çıkmıştır. İlk olarak HotJava ve Java isimleri akla gelmiştir. Daha Java ile ilgili her programa kahve isimleri verilmiştir. Bu yüzden ki Java’nın sembolü kahvedir.

2.4.1.2 Java platformuna nasıl ulaşabilirsiniz?

Eğer İnternet ulaşabiliyorsanız muhtemelen Java ya da ulaşabilirsiniz. Java Web tarayıcılarıyla uyumludur.

Bunun yanında Java platformu yeni nesil her türlü telefon, internete bağlanabilen televizyon, akıllı kartlar ve diğer internet cihazlarıyla uyumlu olarak çalışabilmektedir.

2.4.2 JDK (Derleyici) ile ilgili sıkça sorulan sorular:

2.4.2.1 Tam olarak JDK nedir?

Jdk (Java geliştirme kiti), Java dilinde yazılan programları geliştirme araçlarını içinde barındıran bir programdır. Java derleyicisi (Java.exe), Java yorumlayıcısı (Javac.exe) ve appletleri test etmemizi sağlayan (Appletviewer.exe) içermektedir.

2.4.2.2 Applet derleyicisi olan Javac.exe' yi nasıl kullanabilirim?

Çalıştırmak istediğiniz applet adı ve uygulama adı ile kullanılarak programlarımızı derleyebiliriz. Javac.exe ile hem Appletviewer hem de Netscape 2.0 da çalışabilecek bir class dosyası yaratılır.

Javac Murat.java

Javac.exe Ms-dos komut isteminde ad vererek veya Windows ortamında çalışabilir.

2.4.2.3 Java yorumlayıcısı olan Java.exe'yi nasıl kullanabilirim?

Bu uygulama yalnızca Java uygulamalarında kullanılabilir, Java appletlerinde kullanılamaz. Java yorumlayıcısı dosyanın derlenmesinden sonra oluşan .class dosyası ile kullanılır. Dikkat dosya adından sonra .class yazılmaz. Örneğin Murat.java dosyasının derlenmesinden sonra oluşan Murat.class dosyasını yorumlamak için

Java Murat komut satırı yeterli olacaktır.

Javac.exe Ms-dos komut isteminde ad vererek veya Windows ortamında çalışabilir.

2.4.2.3 Java uygulamalarını Appletviewer ile görüntüleyebilir miyim?

Hayır , çünkü Java uygulaması **static void main (args [0])** metodu ile program kod satırları başlatılır. Bir applet ise main uygulama bloğu ile başlar.

2.4.2.4 Java Applet ve Java uygulamaları arasındaki fark nedir?

Applet, Netscape 2.0 web browser' ı ile kullanılabilen çalıştırılabilir dosyalardır. Java uygulamaları benzer olmasına rağmen, browser gibi, Dos kabuğunda tek başına çalışan uygulamalarla çalıştırılabileceği gibi Java yorumlayıcısı ile de çalışabilir. Appletler çalıştırılabilmesi için Netscape 2.0 gibi web browser 'ına ihtiyaç duymalarına rağmen, uygulamalar Java.exe yorumlatıcısıyla çalışabilir. Appletler bunu yanın da Appletviewer gibi araçlar ile de çalıştırılabilir.

2.4.2.5 Awt nedir?

Awt, Java uygulamalarında ve Java appletlerinde grafik uygulamalarını gerçekleştirmek için kullanılan bir araçtır. Awt bileşenleri denilince butonlar, pencereler, checkbox lar , pull-down mönüler akla gelmelidir.

2.4.2.6 Java programlarıma kullanıcı bilgilerini nasıl iletebilirim?

Java appletlerinde Awt uygulamalarını (text kutusu) kullanarak bunu gerçekleştirebiliriz. Böylece kullanıcı giriş bilgileri programımıza aktarılmış olur. Awt kullanmadan bunu gerçekleştirmek için klavyeden giriş bilgilerini okuyan read() komutunu kullanabiliriz.

2.4.2.7 Applet ve uygulamalarına kullanıcı tanımlı parametreler nasıl aktarılır?
Appletler <param> tag ını kullanarak değerleri alabilirler. Applet 'in getParameter ("ad") parametre nesnesini kullanarak, **ad** adlı değeri applet 'e alırlar. HTML dosyaları <applet>..**</applet>** tagları arasında applete değer gönderirler. (Manger,1988)

BÖLÜM 3

SINIF VE NESNELER

Nesneye dayalı programcılık son yıllarda çok büyük gelişme gerçekleştirmiştir. Bu konunun anlaşılması belki de seneler alabilir. Bilinmesi gereken bazı noktalar;

- Sınıf ve nesne kavramı ve bunların birbirleriyle ilişkisi,
- Sınıf ve nesnelerin davranışları ve bunlara ait sınıflar,
- Sınıfa ait miras ve program dizaynı üzerine etkisi,
- Paket ve ara yüz kavramlarına ait temel bilgiler.

Nesneye dayalı programlama birbiriyle uyumlu bağımsız modüllerden oluşan yapı anlamına gelir.

Sınıf ve diğer sınıf kavramlarının anlatılmasından önce standart kavramlarının Türkçe karşılıklarının verilmesi yararlı olacaktır.

Class (sınıf): Ana nesne kökenli programlama elemanı.

Object (nesne): Sınıf (Class) yapıları bilgisayar belleğinde bir işlem için kullanıldığında aldıkları (özel) isim.

New :Sınıfların kurucu metodunu çağırıp nesne tanımını ve bilgisayar adreslerini oluşturan deyim

Method (metot): Sınıfların içindeki işlevsel program parçacıkları.

Constructor (kurucu metod): Nesne ilk defa oluşturulurken hangi sınıf değişkenlerinin hangi değerleri alacağını belirten metod.

3.1 Sınıflar ve Nesneler

Sınıf benzer özellikteki nesnelerin oluşturduğu yapıya verilen addır. Nesne denince de gerçek dünyada olduğu gibi diğer nesnelerden oluşan yapılar akla gelebilir.

Nesneye dayalı programlamada nesne değil nesnenin ait olduğu sınıf özellikleri göz önünde tutulur. Aynı bilgisayarlar da olduğu bilgisayar işlemci hızı ve ana bellek kapasitesi göz önüne alınmaz, bilgisayar deyince bilgisayar sınıfının özellikleri belirtilir. Her bilgisayar da bulunan monitör, klavye, mouse bilgisayar sınıfının özellikleridir.

Sınıf örneği, güncel nesne için kullanılan bir başka deyimdir. Sınıf nesne için kullanılan genel terim ise; örnek, seçilmiş bir nesneyi belirten terimdir.

Sınıf örneği ve nesne deyince aynı şey akla gelmelidir. Bir buton sınıfı ele alalım. Programcının yapması gereken programın gereği buton özelliklerini (buton rengi, şekli vb.) belirlemek ve butona ihtiyacı olduğunda bu sınıfı çağırmaktır. Sınıf deyince C+ programlama dilindeki struct ve typedef tanımlamaları akla gelebilir, ama unutulmamalıdır ki sınıf bunun çok ilerisindedir.

3.1.1 Davranış ve sınıflar

Java da oluşturulan her sınıf iki özellikten oluşur;

1. Sıfat,
2. Davranış.

Sıfatlar

Sıfatlar nesneleri birbirinden ayırmamızı sağlayan özelliklerdir. Bilgisayar örneğimize devam edelim.

İşlemci hızı: Pentium 166 Mhz, Pentium 350 Mhz, Pentium 1000 Mhz

Marka: Escort, Vestel, Toplama bilgisayar

Ana bellek: 16 MB Edo Ram, 64 MB Sd Ram, 256 MB Sd Ram

Sıfatlar durumla ilgili bilgi de verebilir. Örneğin bilgisayarın çalışıp çalışmadığı durumu da bir sıfat.

Sıfatlar için değişkenler tanımlanır. Sıfatlar nesneye ait global değişkenler olarak da tanımlanabilir. Sıfatlar her örnekte değişiklik gösterebildiği için örnek değişkenleri veya örneğe ait değişkenler olarak ta tanımlanabilir. (Çoban,2000)

3.1.1.1 Davranış

Sınıf davranışları, sınıfa ait hangi örneklerin durumunun ne şekilde değiştirebileceklerini gösterir. Bilgisayar örneğimize devam edersek şu davranışlar örnek verilebilir.

Bilgisayarı çalıştır

Bilgisayarı durdur.

Bilgisayarı düşük güce geçir.

3.1.1.2 Sınıf oluşturulması

Eğer herhangi bir editörde şu örneği yazarsak

```
class Bilgisayar{  
String marka;  
Int hız;  
Boolean bildurumu;  
}
```

Böylece Bilgisayar sınıfı oluşturulmuş olur. Bu sınıfa ait özellikler string tipinde marka, int tipinde hız, mantıksal değişkenler (true, false) bildurumu tanımlanmıştır.

```
void bilcalistir( ){  
if (bildurumu==true)  
System.out.println("Bilgisayar çalışıyor");  
else{  
bildurumu=true;  
System.out.println("Bilgisayar şimdi çalıştırıldı");  
}  
}
```

Yazdığımız program parçası bilgisayarın çalışıp çalışmadığı kontrol ediliyor. Eğer çalışıyorsa “Bilgisayar çalışıyor” mesajı veriliyor, aksi durumda bilgisayar durumu değişkeni true yapılarak bilgisayar çalıştırılıp, “Bilgisayar şimdi çalıştırıldı” mesajı verdiriliyor.

```
void ozelliklerebak( ){  
System.out.println(“Bu bilgisayar” + marka + ” “ +hız);  
if (bildurumu== true)  
System.out.println(“Çalışıyor”) ;  
else System.out.println(“Duruyor”);  
}
```

Yukarıdaki program parçası bilgisayarın markasını, hızını ve çalışıp çalışmadığını gösterir. Bu program yazılıp Javac Bilgisayar.java komutu ile derlendiğinde tanımladığımız sınıf bir ana programdan çalışmadığı için ;

In class Bilgisayar: void main(String argv (J) is not defined hatası verir.

Bu yüzden önce Bilgisayar sınıfını kullanan bir program yazmamız gerekir.

Bilgisayar.java (Class Örneği)

```
class Bilgisayar{  
String marka;  
int hiz;  
boolean bildurumu;  
  
void bilcalistir( ){  
if (bildurumu==true)  
System.out.println("Bilgisayar çalışıyor");  
else{
```

```

        bildirumu=true;
        System.out.println("Bilgisayar simdi □alistirildi");
    }
}

```

```

void ozelliklerebak(){
    System.out.println("Bu bilgisayar" + marka + " " +hiz);
    if (bildurumu==true)
        System.out.println("Çalışıyor") ;
    else System.out.println("Duruyor");
}

```

```

public static void main(String args[]){
    Bilgisayar m=new Bilgisayar();
    m.marka="Vestel";
    m.hiz=667;
    System.out.println("™zelliklere bak");
    m.ozelliklerebak();
    System.out.println("_____");
    System.out.println("Bilgisayar çalıştırılıyor");
    m.bilcalistir();
    System.out.println("_____");
    System.out.println("™zellikler bak");
    m.ozelliklerebak();
    System.out.println("_____");
    System.out.println("Bilgisayar çalıştırılıyor");
    m.bilcalistir();
}
}

```

```

C:\jdk1.3.0_02\bin>java Bilgisayar
özelliklere bak
Bu bilgisayarVestel 667
Duruyor

Bilgisayar alistiriliyor
Bilgisayar simdi Çalistirildi

özellikler bak
Bu bilgisayarVestel 667
Çalışıyor

Bilgisayar Çalistiriliyor
Bilgisayar Çalışıyor

```

Şekil 1

Programdaki void kısmında m=new Bilgisayar() ile Bilgisayar sınıfı özelliklerine sahip m adlı bir değişken tanımlanmış olur.

3.2 Miras Ara yüz ve Paket kavramları

Miras ile anlatılmak istenen yeni bir sınıf oluşturulduğunda diğer sınıflardan farkının belirtilmesidir. Her sınıfa ait bu sınıfın özelliklerini taşır.

Bir sınıfın daha üstünde bulunan sınıfa o sınıfın **süper sınıfı**, altında bulunana ise **alt sınıf** denir. Alt sınıflar üstünde bulunan sınıfın özelliklerini taşır, bu yüzden süper sınıfın değişkenlerini alt sınıfta yeniden tanımlamaya gerek yoktur. Çoğu kez bir sınıf oluşturmak için varolan diğer sınıfların özelliklerine yenilerini eklemeye gidilir. Yapacağımız işe uydun buton sınıfını tanımlamaka için varolan buton sınıfından yararlanılır. Buton sınıfına yeni özellikler eklenerek yeni bir buton sınıfı oluşturulabilir. Eğer yeni bir sınıf bir sınıf oluşturmak istersek, tıpkı Bilgisayar sınıfında olduğu gibi, var olan bir sınıftan yararlanacağımız için Java bu sınıfı nesne altındaki bir sınıf olarak kabul eder.

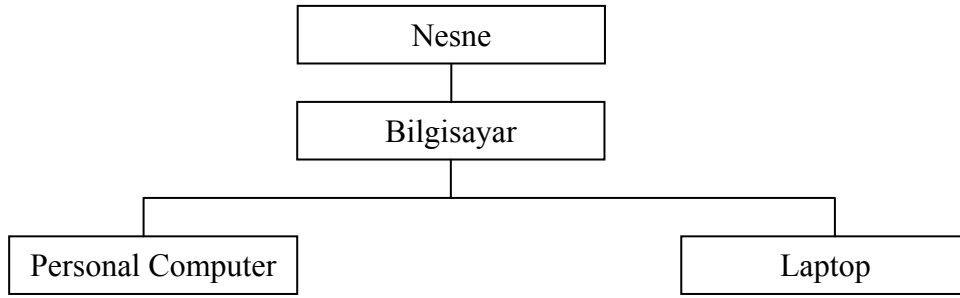
3.2.1 Sınıf hiyerarşisi:

Yeni bir sınıf oluşturulurken yapılacak en mantıklı şey sınıf hiyerarşisini kontrol etmektir. Bu da sınıfın hangi sınıfların alt sınıfları olabileceği ve bunun bize ne avantaj sağlayacağıdır.

Sağlanabilecek en önemli avantajlar:

1. Yukarıdaki sınıflara ait bilgileri alt sınıflarda istenildiği kadar kullanabilmek.
2. Bir sınıfın hiyerarşi içindeki yerini değiştirerek yeniden derlemeye gerek kalmadan davranışını değiştirebilmektir.

Bilgisayar sınıfına bir alt sınıf oluşturmak istediğimizde bir laptop bilgisayar uygun olabilir. Her şeyden önce her ikisi de bilgisayardır. İkisinin de benzer özellikleri (marka, hız) vardır. Ancak bir çamaşır makinesinin bilgisayar sınıfının alt sınıfı olmasıyla bize ne yarar sağlayacağı konusu üzerinde fazla düşünmeye gerek yoktur.



Şekil 2

3.2.2 Alt sınıf oluşturma

Alt sınıfların en çok kullanıldığı Java programları Java Appletlerdir. Bu sayede Applet sayfa içinde istenen yere kolaylıkla yerleştirilebildiği gibi sistemle etkileşimi de kolaylıkla sağlanabilir.

Örnek olarak kendimizi tanıtan bir applet oluşturalım.

```
public class Murat extends java.applet.Applet{  
}
```

bu komut satırıyla applet sınıfın alt sınıfı olan Murat adında bir alt sınıf oluşturduk. Komut satırındaki bir diğer önemli nokta ise public kelimesidir. Public kelimesi bu sınıfın diğer sınıflar tarafında kullanılabileceği anlamına gelmektedir. Applet özellikleri gereği public tanımlanmak zorundadır.

} imlecinden sonra

```
Font yeni = new Font("Time Roman", Font.BOLD,50);
```

Komut satırını eklemek ile java.awt sınıfına sahip Font fonksiyonu ile 50 puntoda, Times New Roman stilinde, Bold olan yeni adlı bir nesne tanımlamış olduk.

Diğer metot da painttir. Bu metot tek başına bir şey yapmamasına rağmen applet sınıfı içinde kullanarak yazılan metnin rengi vb. özelliklerinin belirlenmesinde kullanılır.

```
public void paint( Graphic g){  
g.setFont (yeni);  
g.setColor(Color, red);
```



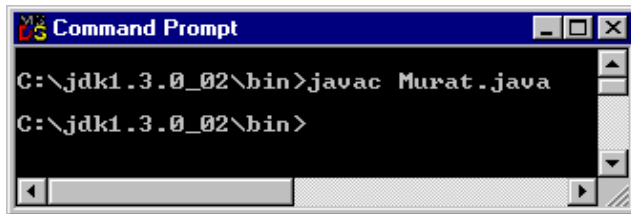
```
g.drawString("Merhaba ben Murat Çatmal",5,25);
```

Paint() sınıfı global olarak tanımlanmıştır. Bu komut satırlarıyla yapılan;

1. Graphics nesnesine fontlar için erişeceği yer belirtilmiştir.
2. Graphics nesnesine kullanacağı renk belirtilmiştir.
3. Merhaba ben Murat Çatmal yazısı 5 satır, 25. Sütuna yazdırılmıştır.

```
public class Murat extends java.applet.Applet{  
    Font yeni = new Font("Time Roman", Font.BOLD,50);  
    public void paint( Graphic g){  
        g.setFont (yeni);  
        g.setColor(Color, red);  
        g.drawString("Merhaba ben Murat Çatmal",5,25);  
    }  
}
```

appletimizi bu şekilde yazıp derlediğimizde bir hata ile karşılaşırız.



Şekil 3

MuratApplet.java: 7:Class Graphic not found in type declaration.

Bu hatanın sebebi kullanılan sınıfların tanımlanmamasıdır. Default olarak sadece java.lang paketi program tarafından içerilir.

Bizim ise örneğimizde üç sınıfımız vardır: Graphics, Font ve Renk.

Programımızın en son hali:

```
Murat.java (Class Örneği)
```

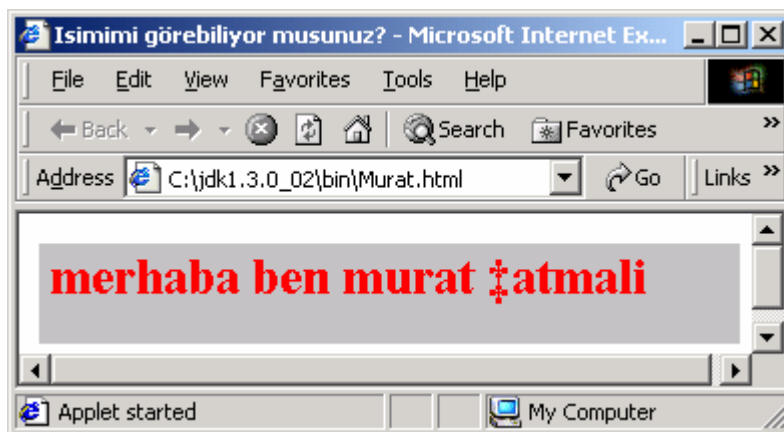
```
import java.awt.Graphics;  
import java.awt.Font;
```

```
import java.awt.Color;

public class Murat extends java.applet.Applet{
Font yeni=new Font("TimesRoman",Font.BOLD,36);
public void paint(Graphics g){
g.setFont(yeni);
g.setColor(Color.red);
g.drawString("Merhaba ben Murat ☐ atmali",5,25);
}
}
```

Murat.htm (Class Örneği)

```
<html>
<head>
<title>Isimimi görebiliyor musunuz?</title>
</head>
<body>
<applet code="Murat.class" width=350 height=50>
</applet>
</body>
</html>
```



Şekil 4

3.3 Kurucu (Costructor) Metot ve New() deyimi

Konumuzu bir örnek üzerinde anlatmaya çalışalım. Bu örneğimizde kisitesti adlı bir sınıf tanımlayalım. **kisitesti** sınıfında **ad** ve **soyad** adlı değişkenleri tanımlayalım.

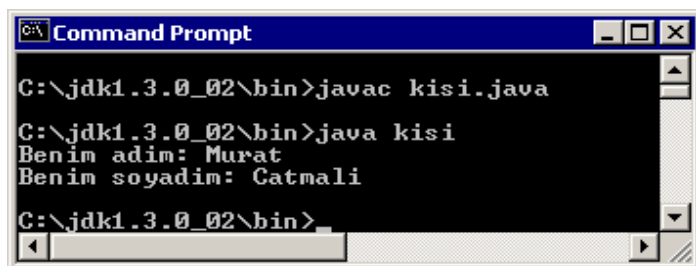
kisi.java (Class Örneği)

```
import java.io.*;

class kisitesti
{
    String ad;
    String soyad;
}

class kisi
{
    public static void main(String args[])
    {
        kisitesti benim=new kisitesti();
        benim.ad="Murat";
        benim.soyad="Catmali";
        System.out.println("Benim adim: "+benim.ad);
        System.out.println("Benim soyadim: "+benim.soyad);

    }
} Şekil 5
```



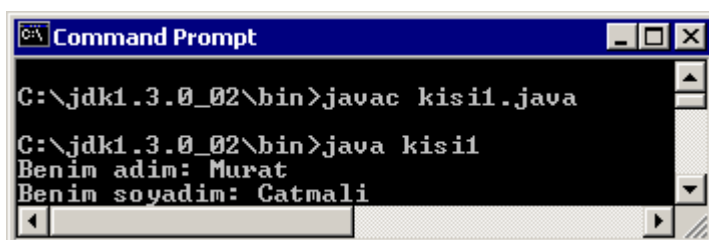
Bu programda kisitesti sınıfında tanımlanan ad ve soyad değişkenlerine ilk değer atamak yerinde iki defa çağrıldığını düşünelim. Bu kadar kısa bir program kısa bir program bile gereksiz yere çok uzayacaktır. Bu iş için Java programlama dilinde kurucu sınıfı kullanılmıştır. Bu metot diğer metotlardan biraz farklıdır. Önce metot önüne metot değişken türü gelmez ve metot dışında hiçbir değişken göndermezler. İsimleri de her zaman sınıf ismiyle aynı olur. Şimdi aynı örneği kurucu metodu kullanarak yapalım.

kisi1.java (Class Örneği)

```
import java.io.*;

class kisitesti
{
    String ad;
    String soyad;
    kisitesti(String a,String b)
    {
        ad=a;
        soyad=b;
    }
}

class kisi1
{
    public static void main(String args[])
    {
        kisitesti benim=new kisitesti("Murat","Catmali");
        System.out.println("Benim adim: "+benim.ad);
        System.out.println("Benim soyadim: "+benim.soyad);
    }
}
```



Şekil 6

3.3.1 This deyiminin kullanımı

Bir metodun ait olduğu sınıftan yaratılacak nesneyi veya o nesnenin bir alt değişkenini tanımlamak gerekir. Nesne daha tanımlanmadığından direk olarak nesne ismini kullanamayız. Bunun yerine Java this deyimini kullanılır. this deyimi özellikle sınıfa ait değişken isimlerinin aynısı metotta kullanılmışsa işe yarar. Bu durumda değişkenler this.değişken_adı komutu çağırabilir.

Deyim.java (Class Örneği)

```
import java.awt.Graphics;
import java.applet.Applet;
public class deyim extends Applet
{
//bu degiskeler tum sinifa aittir
double x;
int y;
void metot1(Graphics g)//metota hiçbir degisken girmiyor
{
double x;
int y;
x=5.5;
y=4;
//tum sinifa ait degiskenler this kelimesi ile birlikte kullanılabılır
g.drawString("metot 1 in dis degiskenleri this ile ulasimi:x="+this.x+"y="+this.y,25,25);
g.drawString("metot 1 in ic degiskenleri:x="+x+"y="+y,25,40);
}
public void paint(Graphics g)
{
x=2.5;y=3;
metot1(g);
}
```

BÖLÜM 4

SEQUENTIAL (ARDAŞIK) DOSYALAR

Bu dosyalarda veriler arka arkaya okunan byte blokları olarak kabul edilir. Her dosya dosya-bitiş işaretiyle sonlandırılır. Yeni bir dosya açıldığında bu dosyayı temsil eden bir nesne oluşturulur.

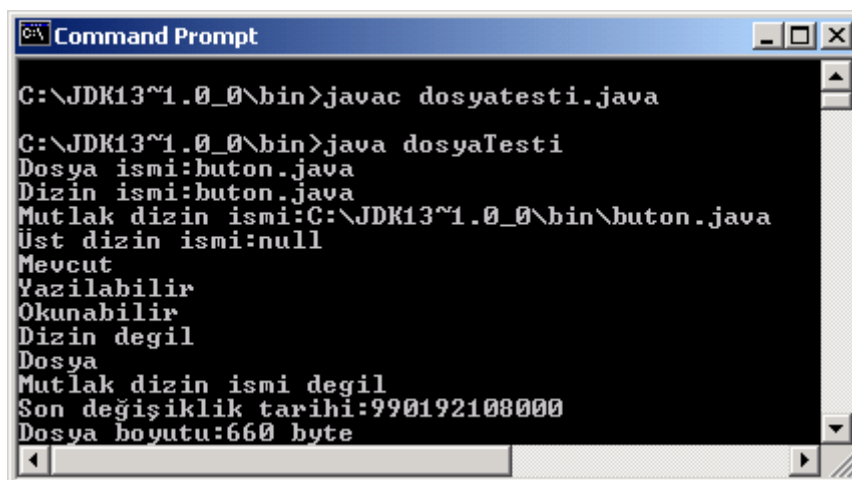
Dosya sınıfı, giriş çıkış işlemlerini sağlamanın yanı sıra dosya isimleri ve bulundukları dizinlerle ilgili bilgi verir. Kurucu metodları bir örnekte açıklamaya çalışalım.

(Çoban,2000)

dosyatesti.java

```
import java.io.*;

class dosyaTesti
{
    public static void main(String args[])
    {
        (1) File dosya=new File("buton.java");
        (2) System.out.println("Dosya ismi:"+dosya.getName());
        (3) System.out.println("Dizin ismi:"+dosya.getPath());
        (4) System.out.println("Mutlak dizin ismi:"+dosya.getAbsolutePath());
        (5) System.out.println("Şst dizin ismi:"+dosya.getParent());
        (6) System.out.println(dosya.exists()? "Mevcut":"Mevcut degil");
        (7) System.out.println(dosya.canWrite()? "Yazilabilir":"Yazilamaz");
        (8) System.out.println(dosya.canRead()? "Okunabilir":"Okunamaz");
        (9) System.out.println(dosya.isDirectory()? "Dizin":"Dizin degil");
        (10) System.out.println(dosya.isFile()? "Dosya":"Dosya degil");
        (11) System.out.println(dosya.isAbsolute()? "Mutlak dizin ismi":"Mutlak dizin ismi
degil");
        (12) System.out.println("Son değışiklik tarihi:"+dosya.lastModified());
        (13) System.out.println("Dosya boyutu:"+dosya.length()+" byte");
    }
}
```



```
Command Prompt
C:\JDK13~1.0_0\bin>javac dosyatesti.java
C:\JDK13~1.0_0\bin>java dosyaTesti
Dosya ismi:buton.java
Dizin ismi:buton.java
Mutlak dizin ismi:C:\JDK13~1.0_0\bin\buton.java
Üst dizin ismi:null
Mevcut
Yazilabilir
Okunabilir
Dizin degil
Dosya
Mutlak dizin ismi degil
Son değışiklik tarihi:990192108000
Dosya boyutu:660 byte
```

Şekil 7

1.File değişken adı : Dosya sınıfında yeni bir dosya tanımlamak için kullanılır. **dosya** adında bir değişkene tanımlanmış, buton.java dosyası bu değişkene atanmıştır.

2. getName(): Dosyanın ismini verir.

3. getPath(): Dosyanın ismini ve içinde bulunduğu dizinin ismini verir.

4.getAbsolutePath():Dosyanın tam dizinini ve dosya ismini verir.

5. getParent():Dosyanın içinde bulunduğu dizinin ismini verir.

6. exists(): Dosyanın mevcut olup olmadığını kontrol eder.

7. canWrite(): Tanımlanan dosyaya bilgi yazılabildiğini kontrol eder.

8. canRead(): Tanımlanan dosyadan bilgi olunabildiğini kontrol eder.

9. isDirectory(): Verile ismin bir dizin olup olmadığını kontrol eder.

10. isFile(): Verilen ismin bir dosya olup olmadığını kontrol eder.

11. isAbsolute(): Dosya isminin mutlak isim olup olmadığını kontrol eder. (Eğer dosya ismi co/java/prog/Buton.java olarak verilmişse true değeri döndürür.)

12. lastModified(): Dosyanın en son değiştirildiği tarihi verir.

13.length(): Dosyanın boyutunu Byte olarak verir.

Bunun yanı sıra kullanılan diğer dosya fonksiyonları:

delete(): Dosya siler.

equals(nesne): nesne de verilen dosya adıyla, dosya ismini karşılaştırır.

list(): Verilen dizinin içindeki dosyaların listesini verir.

mkdir(): Yeni bir dizin oluşturur.

mkdirs(): o anda tanımlı olan dizin içinde bir alt dizin oluşturur.

renameTo(dosya) : Dosyanın ismini değiştirir.

toString(): Dosya ve dizin topluğunun String değişkeni içerisindeki eşdeğerini verir.

Bir girdi-çıkıtkı akış nesnesi (dosya) oluşturmak istediğimizde FileInputStream veya FileOutputStream sınıfında bir nesne oluştururuz. Eğer değişkenleri byte byte yerine Double veya Integer veri tipinde okumak istersek DataInputStream veya DataOutputStream nesnelerini kullanmamız gerekir. Aşağıdaki Java programı sequential tipte bir dosya yaratır.

(1)Giriş-çıkış ve grafik fonksiyonların içeren kütüphaneler programa dahil ediliyor.

(2) dosyayarat adlı bir sınıf tanımlanıyor.

(3) hisim, ,isim, soyisim, hesap adlı metin kutuları tanımlanıyor.

(4) enter,done adlı butonları tanımlanıyor.

(5) io sınıfı nesnelerinden DataOutputStream ile dosyamıza veri yazılmasında kullanılacak nesne tanımlanıyor ve adı cikti veriliyor.

(6) H,I,S,P adlı etiketler tanımlanıyor.

(7) dosyayarat adlı tanımladığımız fonksiyon kodları yazılmaya başlanıyor.

(8) ("Musteri dosyasi Ac") başlıklı bir üst sınıf tanımlanıyor.

(9) musterit.txt adlı dosya açılmaya çalışılıyor.

(10) Eğer dosya açılmamışsa kullanıcıya mesaj verilerek programdan çıkılıyor.

(11) Formumuzun boyutları ve ızgara boyutları tanımlanıyor.

(12) H adlı etiket içeriği “Hesap numarası” oluyor.

(13) hisim adlı tanımladığımız text nesnesi oluşturuluyor.

(14) I adlı etiket içeriği “İsim” oluyor.

(15) isim adlı tanımladığımız text nesnesi oluşturuluyor.

(16) S adlı etiket içeriği “Soyisim” oluyor.

(17) soyisim adlı tanımladığımız text nesnesi oluşturuluyor.

(18) P adlı etiket içeriği “Hesap” oluyor.

(19) hesap adlı tanımladığımız text nesnesi oluşturuluyor.

(20) enter adlı tanımladığımız “Gir” etiketli buton oluşturuluyor.

(21) done adlı tanımladığımız “Çıkış” etiketli buton oluşturuluyor.

(22) Formumuzun görünür olması sağlanıyor. (Visible özelliği false yapılarak formumuzun görünmemesini sağlayabiliriz.)

(23) hesapekle adlı fonksiyon kodları başlıyor.

(24) Değişkenlerimiz tanımlanıyor.

(25) Eğer hismi adlı text nesnesi boş değilse yani bu alana veri girilmişse,

(26) Hesap alanı 0 dan farklı bir değer ise,

(27) Veriler dosyaya yazılıyor.

(28) Veri girişi yapılan alanlar bir sonraki kullanım için boşaltılıyor.

(29) Eğer hesap numarası alanı tamsayı girilmediyse kullanıcıya hata mesajı verdiriliyor.

(30) Dosyaya verileri yazarken hata oluşmuşsa kullanıcıya mesaj verdiriliyor.

(31) Tanımladığımız kullanıcı fonksiyonları çağrılıyor, buraya anlatılan kontroller yaptırılıyor.

(32) Dosya kapatılıyor.

(33) Eğer dosya kapatılırken hata oluşmuşsa kullanıcıya mesaj verdiriliyor.

(34) Oluşturduğumuz fonksiyon ana programdan çağrılıyor.

(1)import java.io.*;

import java.awt.*;

import java.awt.event.*;

(2)public class dosyayarat extends Frame implements ActionListener

{

(3) private TextField hismi,isim,soyisim,hesap;

(4) private Button enter,done;

(5) private DataOutputStream cikti;

(6) private Label H,I,S,P;

(7) public dosyayarat()

{

(8) super("Musteri dosyasi Ac");

(9) try

{

cikti=new DataOutputStream(new FileOutputStream("musteri.txt"));

}

catch(IOException e)

{

(10) System.err.println("Dosya dogru acilamadi\n"+e.toString());

```
System.exit(1);  
}
```

```
(11)    setSize(300,150);  
setLayout(new GridLayout(5,2));  
(12)    H=new Label("Hesap numarasi");  
        add(H);  
(13)    hismi=new TextField();  
        add(hismi);  
(14)    I=new Label("Isim");  
add(I);  
(15)    isim=new TextField(20);  
add(isim);  
(16)    S=new Label("Soyisim");  
        add(S);  
(17)    soyisim=new TextField(20);  
        add(soyisim);  
(18)    P=new Label("Hesap");  
add(P);  
(19)    hesap=new TextField(20);  
add(hesap);  
(20)    enter=new Button("Gir");  
enter.addActionListener(this);  
add(enter);  
(21)    done=new Button("Cikis");  
done.addActionListener(this);  
add(done);  
(22)    setVisible(true);  
}
```

```
(23)    public void hesapekle()  
{  
(24)    int accountNumber=0;  
Double d;
```

```

(25)    if(!hismi.getText().equals(""))
    {
try
    {
        accountNumber=Integer.parseInt(hismi.getText());
(26)    if(accountNumber!=0)
    {
(27)    cikti.writeInt(accountNumber);
    cikti.writeUTF(isim.getText());
    cikti.writeUTF(soyisim.getText());
    d=new Double(hesap.getText());
    cikti.writeDouble(d.doubleValue());
    }
(28)    hismi.setText("");
    isim.setText("");
    soyisim.setText("");
    hesap.setText("");
    }
    catch(NumberFormatException nfe)
    {
(29)    System.err.println("Hesap numarasi tamsayi degisken olmalidir");
    }
    catch(IOException io)
    {
(30)System.err.println("Dosyaya yazarken hata
    olustu\n"+io.toString());
    System.exit(1);
    }
    }
    }
(31)    public void actionPerformed(ActionEvent e)
    {
    hesapekle();
    if(e.getSource()==done)

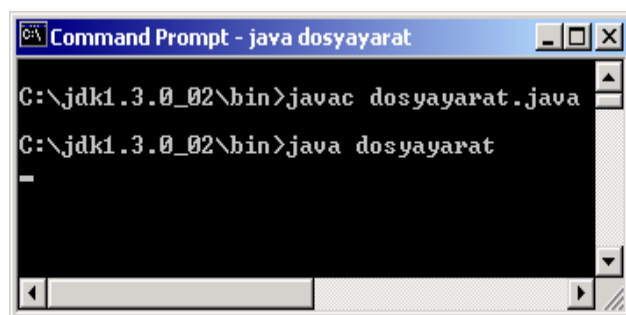
```

```

{
try
{
(32)    cikti.close();
}
catch(IOException io)
{
(33)    System.err.println("Dosya kapatilamadi\n"+io.toString());
}
System.exit(0);
}
}

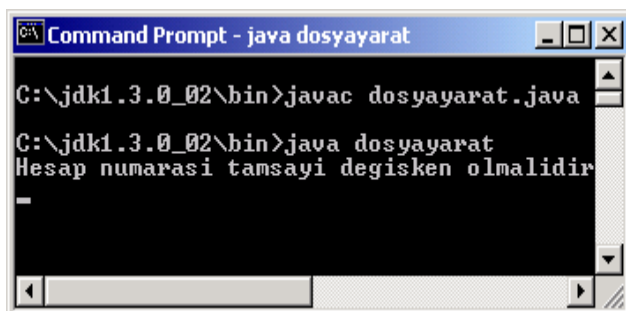
(34) public static void main(String args[])
{
new dosyayarat();
}
}

```



Şekil 8

Şekil 9



Şekil 10

Şekil 11

Eğer hesap numarasını bir tamsayı değişken girmezsek yukarıdaki gibi bir hata mesajıyla karşılaşırız.

Bu Java programı ise yarattığımız sequential dosyadan verileri okumamızı sağlar.

dosyaoku.java

(1) Giriş-çıkış ve grafik fonksiyonların içeren kütüphaneler programa dahil ediliyor.

(2) dosyaoku adlı bir sınıf tanımlanıyor.

(3) hisim, ,isim, soyisim, hesap adlı metin kutuları tanımlanıyor.

(4) next,done adlı butonları tanımlanıyor.

(5) io sınıfı nesnelerinden DataInputStream ile dosyamızın okunmasında kullanılacak nesne tanımlanıyor ve adı oku veriliyor.

(6) H,I,S,P adlı etiketler tanımlanıyor.

(7) dosyaoku adlı tanımladığımız fonksiyon kodları yazılmaya başlanıyor.

(8) ("Musteri dosyasi oku") başlıklı bir üst sınıf tanımlanıyor.

- (9) musteritxt adlı dosya açılmaya çalışılıyor.
- (10) Eğer dosya açılmamışsa kullanıcıya mesaj verilerek programdan çıkılıyor.
- (11) Formumuzun boyutları ve ızgara boyutları tanımlanıyor.
- (12) H adlı etiket içeriği “Hesap numarası” oluyor.
- (13) hisim adlı tanımladığımız text nesnesi oluşturuluyor.
- (14) I adlı etiket içeriği “İsim” oluyor.
- (15) isim adlı tanımladığımız text nesnesi oluşturuluyor.
- (16) S adlı etiket içeriği “Soyisim” oluyor.
- (17) soyisim adlı tanımladığımız text nesnesi oluşturuluyor.
- (18) P adlı etiket içeriği “Hesap” oluyor.
- (19) hesap adlı tanımladığımız text nesnesi oluşturuluyor.
- (20) next adlı tanımladığımız “İleri” etiketli buton oluşturuluyor.
- (21) done adlı tanımladığımız “Çıkış” etiketli buton oluşturuluyor.
- (22) Formumuzun görünür olması sağlanıyor. (Visible özelliği false yapılarak formumuzun görünmemesini sağlayabiliriz.)
- (23) kayitoku adlı fonksiyon kodları başlıyor.

(24) Değişkenlerimiz tanımlanıyor.

(1)import java.io.*;

import java.awt.*;

import java.awt.event.*;

(2)public class dosyaoku extends Frame implements ActionListener

{

(3) private TextField hismi,isim,soyisim,hesap;

(4) private Button next,done;

(5) private DataInputStream oku;

(6) private Label H,I,S,P;

(7) public dosyaoku()

{

(8) super("Musteri dosyasi oku");

(9) try

{

oku=new DataInputStream(new FileInputStream("musteri.txt"));

}

catch(IOException e)

{

(10) System.err.println("Dosya dogru acilamadi\n"+e.toString());

System.exit(1);

}

(11) setSize(300,150);

setLayout(new GridLayout(5,2));

(12) H=new Label("Hesap numarasi");

add(H);

(13) hismi=new TextField();

add(hismi);

(14) I=new Label("Isim");

add(I);

```

(15) isim=new TextField(20);
    add(isim);
(16) S=new Label("Soyisim");
    add(S);
(17) soyisim=new TextField(20);
    add(soyisim);
(18) P=new Label("Hesap");
    add(P);
(19) hesap=new TextField(20);
    add(hesap);
(20) next=new Button("ileri");
    next.addActionListener(this);
    add(next);
(21) done=new Button("cikis");
    done.addActionListener(this);
    add(done);
(22) setVisible(true);
    }

(23) public void kayitoku()
    {
(24) int account;
    String first,last;
    double balance;
    try
    {
        account=oku.readInt();
        first=oku.readUTF();
        last=oku.readUTF();
        balance=oku.readDouble();
        hismi.setText(String.valueOf(account));
        isim.setText(first);
        soyisim.setText(last);
        hesap.setText(String.valueOf(balance));
    }
    catch (Exception e) {
    }
    }

```

```

    }
    catch(EOFException eof)
    {
        dosyakapat();
    }
    catch(IOException io)
    {
        System.err.println("Dosya okurken hata olustu\n"+io.toString());
        System.exit(1);
    }
}

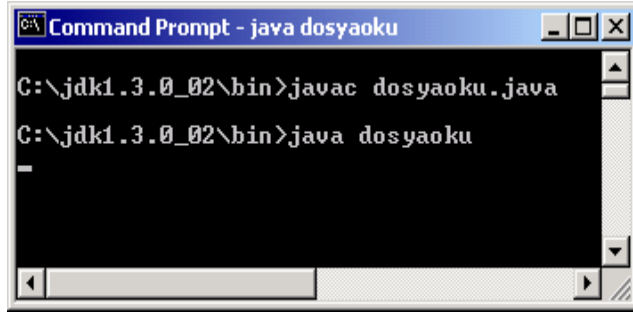
public void actionPerformed(ActionEvent e)
{
    if(e.getSource()==next)
        kayitoku();
    else
        dosyakapat();
}

private void dosyakapat()
{
    try
    {
        oku.close();
        System.exit(0);
    }
    catch(IOException e)
    {
        System.err.println("Dosya Kapama Hatasi olustu\n"+e.toString());
        System.exit(1);
    }
}

public static void main(String args[])

```

```
{  
    new dosyaoku();  
}  
}
```



Şekil 12

Şekil 13

Yazdığımız Java programı derlendiğinde yukarıdaki form ile dosyadan verilerimizi okuyabiliriz.

BÖLÜM 5

RANDOM (RASGELE ERİŞİMLİ) DOSYALAR

Java programlama dilinde kullanılan bir diğer dosya tipi de random (rasgele erişimli) dosyalardır. Bu dosyaların sıralı dosyalardan farkı erişilmek istenen kayıta, kayıt numarası verilerek erişilebilmesidir. Halbuki sıralı dosyalarda, istenen kayıta ulaşmak için diğer kayıtlar üzerinden kontrol edilerek geçilmesidir, bu da tabii ki zaman kaybına yol açmaktadır.

Aşağıdaki Java programı random dosyaya kayıt girilmesi sırasında kullanılacak metotların tanımladığı kısımlardır. Bu program diğer Java programlarında sınıf olarak

kullanılıp, burada tanımlanan metotların diğer programlarda da kullanılması sağlanmıştır. (Çoban,2000)

kayit.java

```
import java.io.*;
```

```
public class kayit
{
    private int hesap;
    private String soyisim;
    private String isim;
    private double hesaptakipara;
```

```
    public void oku(RandomAccessFile dosya) throws IOException
    {
        hesap=dosya.readInt();
        char first[]=new char[15];
        for(int i=0;i<first.length;i++)
        {first[i]=dosya.readChar();}
        isim=new String(first);
        char last[]=new char[15];
        for(int i=0;i<first.length;i++)
        {last[i]=dosya.readChar();}
        soyisim=new String(last);
        hesaptakipara=dosya.readDouble();
    }
```

```
    public void yaz(RandomAccessFile dosya) throws IOException
    {
        StringBuffer buf;
        dosya.writeInt(hesap);
```

```

if(isim!=null)
    buf=new StringBuffer(isim);
else
    buf=new StringBuffer(15);
buf.setLength(15);
dosya.writeChars(buf.toString());
if(soyisim!=null)
    buf=new StringBuffer(soyisim);
else
    buf=new StringBuffer(15);
buf.setLength(15);
dosya.writeChars(buf.toString());
dosya.writeDouble(hesaptakipara);
}

```

```

public void yazhesap(int a){hesap=a;}
public int okuhesap(){return hesap;}
public void yazisim(String f){isim=f;}
public String okuisim(){return isim;}
public void yazsoyisim(String f){soyisim=f;}
public String okusoyisim(){return soyisim;}
public void yazhesaptakipara(double b){hesaptakipara=b;}
public double okuhesaptakipara(){return hesaptakipara;}
public static int boyut(){return 72;}
}

```

Aşağıdaki program random tipli bir dosyayı bellekte yaratmak için kullanılır. kayıt isimli biraz önce tanımladığımız sınıfın import edilip, sahip olduğu metotların kullanıldığına dikkat ediniz.

rasdosyarat.java

```

import java.io.*;
import java.awt.*;

```

```

import java.awt.event.*;
import kayit;

public class rasdosyarat
{
    private kayit hesapdosyasi;
    private RandomAccessFile girdi;

    public rasdosyarat()
    {
        hesapdosyasi=new kayit();
        try{
            girdi=new RandomAccessFile("musteri1.dat","rw");
            for(int i=0;i<100;i++)
                {hesapdosyasi.yaz(girdi);}
        }
        catch(IOException e)
        {
            System.err.println("Dosya açma hatası\n"+e.toString());
            System.exit(1);
        }
    }

    public static void main(String args[])
    {
        rasdosyarat H=new rasdosyarat();
    }
}

```

Aşağıdaki program bir form ara yüzü tanımlanarak random dosyaya bilgi girişi yapılmasını sağlar.

dosyaz.java

```

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import kayit;

public class dosyaz extends Frame implements ActionListener
{
    private TextField hesapalani, isimalani, soyisimalani, hesaptakiparaalani;
    private Button birsonraki, kapat;
    private RandomAccessFile girdi;
    private kayit hesapdosyasi;

    public dosyaz()
    {
        super("Dosyaya Yaz");
        hesapdosyasi=new kayit();
        try
        {
            girdi=new RandomAccessFile("musteri1.dat", "rw");
        }
        catch(IOException e)
        {
            System.err.println("Dosya Acma Hatası\n"+e.toString());
            System.exit(1);
        }
        setSize(300,150);
        setLayout(new GridLayout(5,2));
        add(new Label("Hesap numarasi"));
        hesapalani=new TextField();
        add(hesapalani);
        add(new Label("~sim"));
        isimalani=new TextField(20);
        add(isimalani);
        add(new Label("Soyisim"));
        soyisimalani=new TextField(20);
        add(soyisimalani);
    }
}

```



```
add(new Label("Hesaptaki Para"));
hesaptakiparaalani=new TextField(20);
add(hesaptakiparaalani);
birsonraki=new Button("Gir");
birsonraki.addActionListener(this);
add(birsonraki);
kapat=new Button("€ □ k □ ¤");
kapat.addActionListener(this);
add(kapat);
setVisible(true);
}
```

```
public void eklekayit()
{
int accountNumber=0;
Double d;
if(!hesapalani.getText().equals(""))
{
try
{
accountNumber=Integer.parseInt(hesapalani.getText());
if(accountNumber<=100)
{
hesapdosyasi.yazhesap(accountNumber);
hesapdosyasi.yazisim(isimalani.getText());
hesapdosyasi.yazsoyisim(soyisimalani.getText());
d=new Double(hesaptakiparaalani.getText());
hesapdosyasi.yazhesaptakipara(d.doubleValue());
girdi.seek((long)(accountNumber-1)*kayit.boyut());
hesapdosyasi.yaz(girdi);
}
hesapalani.setText("");
isimalani.setText("");
soyisimalani.setText("");
}
```

```

hesaptakiparaalani.setText("");
}
catch(NumberFormatException nfe)
{
System.err.println("Hesap numarası tamsayı değilken olmalı");
}
catch(IOException io)
{
System.err.println("Dosyaya yazarken hata oluştu\n"+io.toString());
System.exit(1);
}
}
}

```

```

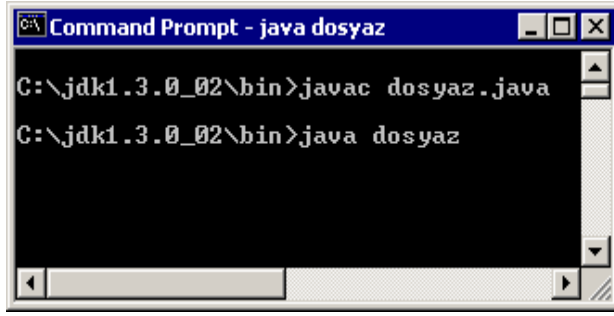
public void actionPerformed(ActionEvent e)
{
eklekayit();
if(e.getSource()==kapat)
{
try
{girdi.close();}
catch(IOException io)
{
System.err.println("Dosya kapatılamadı\n"+io.toString());
}
System.exit(0);
}
}

```

```

public static void main(String args[])
{
new dosyaz();
}
}

```



Şekil 14

Şekil 15

dosyaz adlı Java programı derlendiğinde random tipli dosyaya veri girişi yapılması sağlayan yukarıdaki form ekrana gelir.

BÖLÜM 6

JDBC

Bundan sonraki iki bölümde JDK ile gelen sınıflar sayesinde, ilişkili veri tabanlarına veya SQL(Yapısal sorgulama dili) ile ulaşılan veri tabanlarına göz atacağız.

İlk olarak veri tabanlarının veri nasıl depolandıklarını öğrenelim. Java bunu basit olarak SQL, veri tabanlarının yönetimi ve sorgulanmasında kullanılan dil, ile yapar. Daha sonra da Java

Database Connectivity (JDBC) –Java veritabanı bağlantısı- ,verilerin sunulması ve alınmasında kullanılan standart, sınıflarına göz atacağız. Bir kere veri tabanına bağlandıktan sonra, artık SQL dilini kullanarak verilere işleyebilirsiniz.

Bu bölümde işleyeceklerimiz özeti, SQL ve JDBC dir. Bir sonraki bölümde ise JDBC ile sağlanan özellikler ve veri tabanı sunumunun nasıl yapılacağını öğreneceğiz.bu bölümde öğreneceklerimiz:

- Veri tabanı nedir?
- Basit SQL cümlecikler nelerdir ve bunları nasıl uygularız.
- JDBC ile veri tabanı bağlantısı nasıl gerçekleştirilir?
- JDBC programları nasıl yazılır?
- JDBC uygulamalarındaki anahtar noktalar nelerdir?

(www.java.sun.com)

6.1 JDBC İçeriği ve Terminolojisi

İlk olarak veri tabanı terminolojisine bakalım. Veri tabanına ulaşma, uzaktaki veya yerel bir veri tabanına ulaşım, buradaki verilerin alınması veya işlenmesi anlamına gelir. Veri tabanları ilişkisel olmak zorunda değildir. Bir çok değişik formda veri tabanları olabilir, bunlardan bazıları;

Uzaktan erişimli ilişkisel server üzerinde bulunan veri tabanları; örneğin SQL Server.

Yerel bilgisayarınızda bulunan ilişkisel veri tabanları; örneğin Personal Oracle veya Microsoft Access.

Bilgisayarınızdaki metin dosyaları.

Tablolama programları.

Uzaktan veri erişimine izin veren mainframe bilgisayarlar.

Canlı bilgi servisleri; örneğin Dow Jones vb.

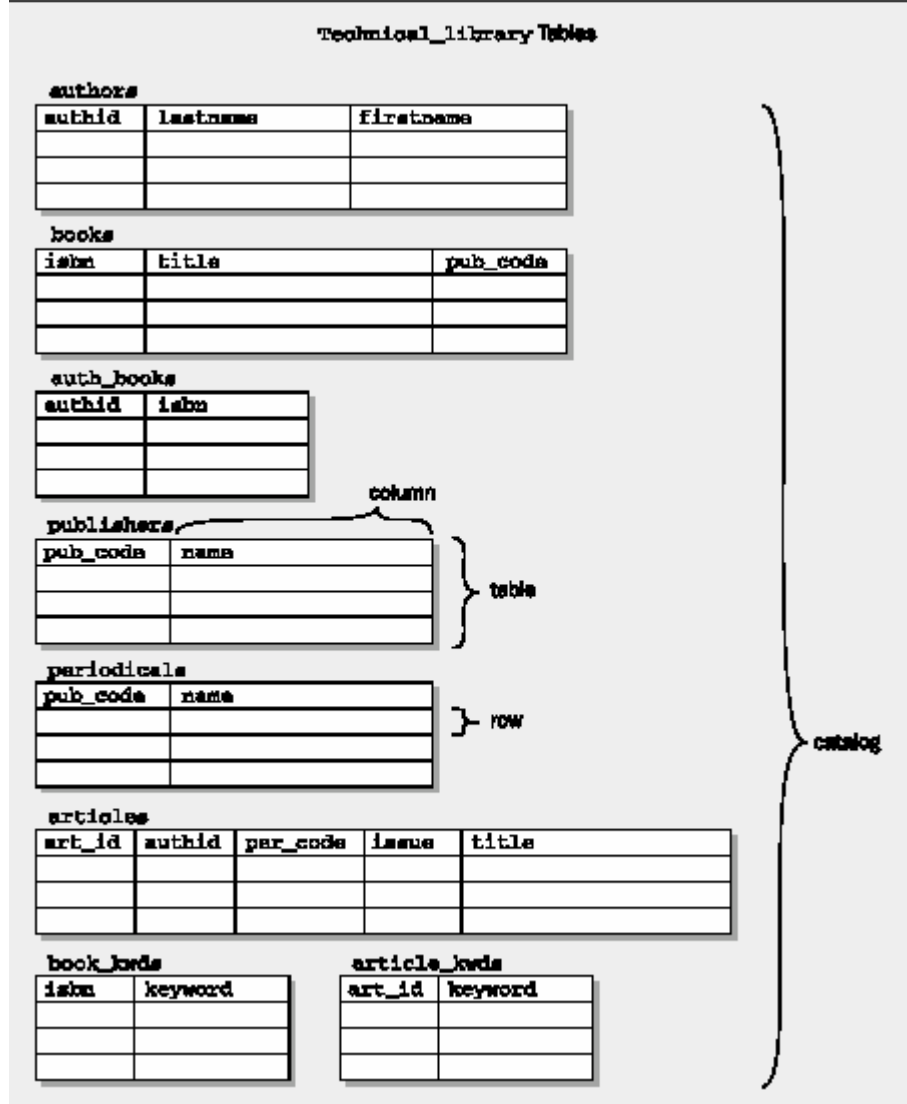
JDBC, ilişkisel veri tabanlarına ara yüz olarak tanımlanabilir. JDBC ilişkisel veri tabanlarına ulaşmada kullanılabilmesine rağmen ben bu bölümde ve bir sonraki bölümde JDBC ile ilişkisel veri tabanlarına ulaşmayı anlatacağım. Eğer daha önce ilişkisel veri tabanlarıyla ilgilenmediyseniz, şu tavsiyeleri dinlemelisiniz. İlişkisel veri tabanları mantıksal olarak öğrenmesi oldukça kolay ve zevklidir.

Java Veri tabanı bağlantısı (JDBC) sınıfları SQL cümlelerini çalıştırarak ilişkisel veritabanları erişimi ve bu veriler üzerinde işlem yapmayı sağlar. JDBC nesneye dayalı, veri tabanına erişim için geliştirilen uygulama programları ara yüzü (Application Programming Interface - API), ve Java geliştiricileri ve veri tabanları için standart olarak geliştirilmiştir.

JDBC, diğer bir veri tabanı ara yüzü olan, X/Open SQL CLI(Call Level Interface), özellikleri temel alınarak geliştirilmiştir. Fakat bu standardın bilgisine sahip olmak , JDBC öğrenmek için yeterli değildir. Buna rağmen, veri tabanına ulaşmak için yazılmış bir veri tabanı programınız varsa, JDBC ye kullanmadan önce bu deneyime sahip olmanız sizin için çok iyi olacaktır.

Java.sql paketi, veri tabanına ulaşmada kullanılan API ler içeren sınıflar barındırmaktadır.

Aşağıdaki şekil technical_library veritabanı (Wrox Web sitesinden ulaşılabilir) temel bileşenlerini göstermektedir. Bu şema bu sonraki anahtar noktaların anlatmakta ve örneklerde kullanılacaktır.



Şekil 16

Bu tablo basit bir veri tabanı örneğidir. Sizde bunun benzer, ilişkili veri tabanları yaratabilirsiniz.

İlişkisel veri tabanlarını üzerinde yapacağınız her türlü işlem, SQL (Structured Query Language –Yapısal Sorgulama Dili) ile sağlanacaktır. SQL Java gibi uygulama dili değildir. SQL daha çok sorgulama ve deklare için kullanılır. Bu da veri tabanı sunucusuna ne yapmak istediğini söylemesi fakat bunu nasıl yapılacağını söylememesi anlamına gelir. Her bir SQL cümlecği veri tabanı sunucusu tarafından analiz edilir, işlemler bu tanımlama doğrultusunda parçalara ayrılarak veri tabanı motoru tarafından

sağlanır. Veri tabanı motoru çeşitli istekleri birleştirir. Değişik kurumsal veri tabanı istekleri genel veri tabanı motorlarını kullanabilir.

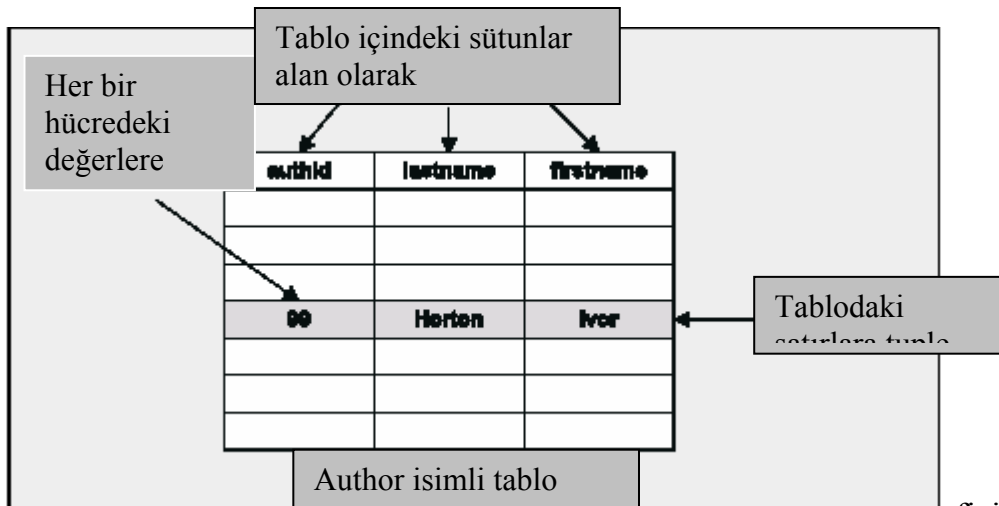
6.2 Tablolar

İlişkisel veri tabanları tablolardan oluşur. Tablo, yukarıdaki authors adlı tabloda olduğu gibi, ilişkiyi sağlayan bir adet primary (birincil) veritabanı olacaktır. Her hangi bir zamanda yapacağınız Define, create, update, delete işlemlerinde tablo içerisindeki verilerle çalışacaksınız.

Bir veri tabanında tablo yarattığınızda, verilerinizi içerecek dikdörtgen şekilli bir ızgara yaratmış olacaksınız. Kısacası tablo satır ve sütunlardan oluşan veri tabanı yapısıdır. Her bir satır, birbirleriyle ilişkili verileri içerir, kısacası veri nesneleri arasında bağlantıyı sağlar.

Satırlar için teknik terim **tuple** dir. Her bir sütun satırı alan denilen bölümlere ayırır, ve bu sütunlar, satırlardaki nesneler ait özellikler olarak tanımlanır. Böylece tabloda bir çok sütun karışık halde yerleştirilmiştir.

Şekil 17



Mantıksal olarak tablo, satır ve sütunlardan oluşan bir yapı olsa da aslında fiziksel olarak böyle değildir. Ama biz verilere ulaşırken bu yapı ile karşılaşacağımızdan bu kadarını bilmemiz yeterlidir.

6.2.1 Tablo sütunları

Daha öncede söylediğim gibi, tablolar dikdörtgen şekilli hücrelerden oluşan ızgaralar olarak düşünülebilir. Izgara keyfi olarak verilebilecek sayıda satır ve sütuna bölünebilir. Izgara içerisindeki, her bir sütundaki hücre çeşitli veriler içerir. Sadece çeşitli tipte veri içermez, aynı zamanda alana isimlerine kategorilere ayrılmış verilerde içermiş olur. Örneğin bir önceki şekildeki **lastname** adlı adlı alan authors adlı tablodaki üç alandan biridir.

6.2.2 Tablo satırları

Tablo içerisindeki her bir satır, tuple olarak adlandırılan ilişkili verilerin oluşturduğu yapıdır. Bazı veri tabanlarında her bir satıra kayıt denir.

Author adlı tablodaki bir satır (kayıt).

Şekil 18

20	Celko	Joe
----	-------	-----

6.3 SQL ile tanışalım

Yapısal sorgulama dili (Structured Query Language) ilişkisel veri tabanlar için geliştirilmiş bir standart olarak kabul edilmektedir. SQL dilinin ilişkisel veri tabanları için sorgu dili olarak kabul edilmesi 1980 li yıllarda istemci/sunucu mimarisinin ortaya çıkmasıyla kabul görmüştür.

SQL dilinde ilginizi çekecek ilk özellik kolay okunabilmesidir. Her bir sorgulamadaki yapı İngilizce diline çok yakındır. Syntax çok kaolay öğrenilebilir ve

anlaşılabilir. İkinci olarak, SQL de kullandığınız komutlar veri tabanında en çok kullanacağınız sorgulamalar için geliştirilmiştir. Veri tabanına isteğinizle ilişkili bir sorgu gönderirsiniz, veri tabanı size bir sonuç gönderir veya bir işlem gerçekleştirir.

6.4 Veri tabanı tabloları geliştirme

Daha önce örnek verdiğimiz, **technical_library** tablosunda aşağıdaki alanları oluşturmak isteyelim.

- Books
- Article
- Authors
- Publishers

Bu alanlarında aşağıdaki bilgileri içereceği düşünelim.

Books

- ISBN
- Kitap adı
- **Yazarlar**

Publishers

- **Yazarlar**
- Başlık
- Yayımlanma tarihi

Authors

- Soyadı
- Adı

- Yayımlanan kitapları
- Yayımlanan makaleleri

Yayımcı

- Yayımcı kodu
- Adı

(www.java.sun.com)

Şimdi veri tabanındaki tabloları oluşturmaya başlayalım. Bu tablomuzun adına authors diyelim. Bu tablodaki sütunlar ise aşağıdaki şekilde olsun.

Sütun başlığı	Açıklaması
Authid	Bir çok yazar aynı isme sahip olabilir.
Lastname	Aile adı
Firstname	İlk adı
Address1	Adres için ilk satır
Address2	Adres için ikinci satır
City	Şehir
State_prov	Devlet
Postcode	Postakodu
Country	Ülke
Phone	Telefon
Fax	Fax
Email	Email adresi

Tablo 1

Şimdi yapmamız gereken her sütunun başlığını uygun bir veri tipi seçmektir. Tabii ki burada geçerli olan SQL dili Java programlama dilinin kullandığı değişken yapılarıdır. Tanımlanan bu veri tipleri SQL tarafından kabul edilir ve işleme konulur. Şimdi SQL in desteklediği bir kaç veri tipine bakalım.

SQL Veri Tipleri	Tanımlama
CHAR	Değişebilen uzunluktaki string
VARCHAR	Değişebilen uzunluktaki string
SQL Veri Tipleri	Tanımlama
BOOLEAN	Mantıksal değerler –true veya false
SMALLINT	Küçük tamsayı değeri, -127 +127
INTEGER	Büyük tamsayı değeri, -32767 +32767
NUMERIC	Noktalı sayılar
FLOAT	Kayan noktalı sayılar
CURRENCY	Para birimi
DOUBLE	Yüksek duyarlıklı kayan noktalı sayılar
DATE	Tarih
TIME	Zaman
DATETIME	Tarih ve zaman
RAW	Binary veri(nesneleri veri tabanında tutar)

Tablo 2

Şimdi de oluşturacağımız tablodaki alanların veri tiplerini belirleyelim.

Sütun adı	Veri tipi
Authid	INTEGER
Lastname	VARCHAR
Firstname	VARCHAR
Address1	VARCHAR
Address2	VARCHAR
City	VARCHAR
State_prov	VARCHAR

Pzpcode	VARCHAR
Country	VARCHAR
Phone	VARCHAR
Sütun adı	Veri tipi
Fax	VARCHAR
Email	VARCHAR

Tablo 3

6.4.1 Tablomuz nasıl çalışacak

Sütun adları alan içerisinde saklanacak bilgi hakkında bilgi verir, fakat bilgisayar alan adlarından hangi tip verilerin saklanacağını anlayamaz. İşte bunu belirlemek için her bir alan bir veri tipine sahiptir

Books tablosu

Sütun başlığı	Açıklaması
Isbn	Kitap için global bir tanımlama
Title	Kitabın adı
Pub_code	Kitabın yayımlayıcısının kodu

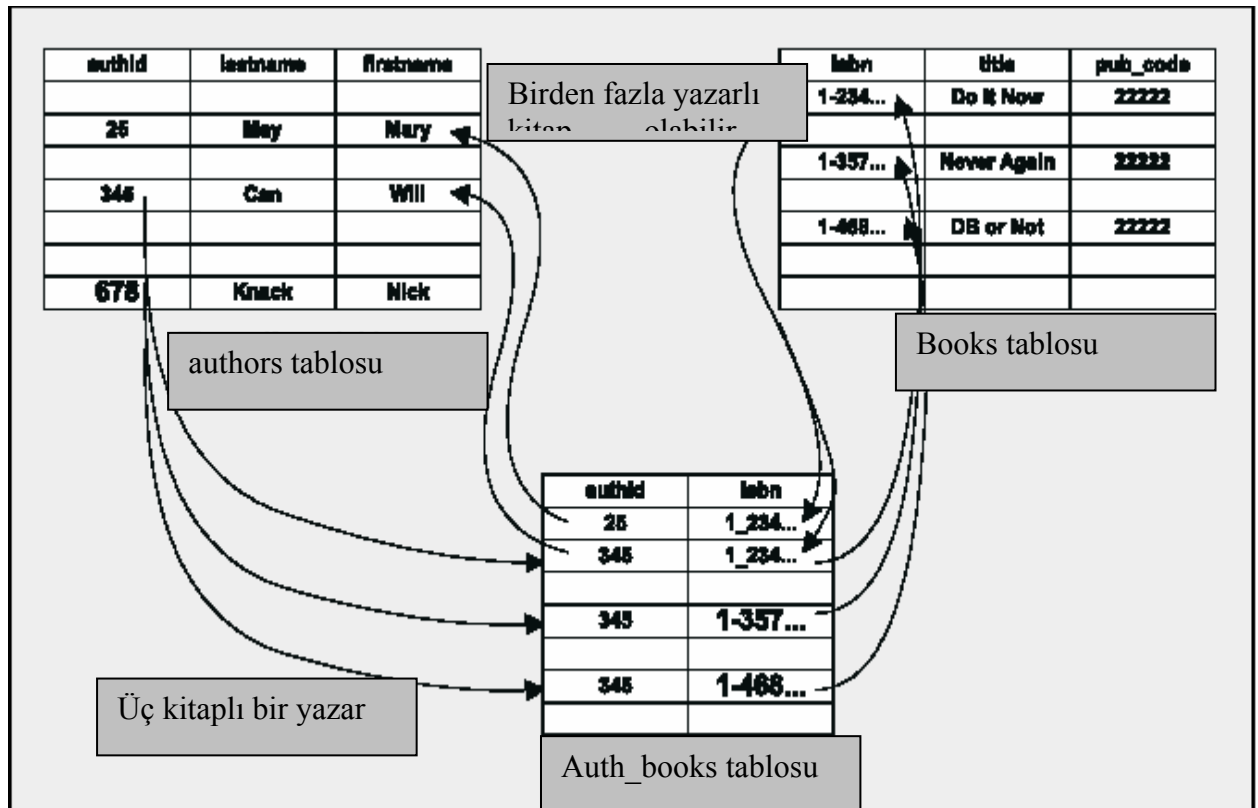
Tablo 4

Sütun adı	Veri tipi
Isbn	VARCHAR
Title	VARCHAR
Pub_code	CHAR(8)

Tablo 5

6.4.2 Tabloların dizayn edilmesi ve ilişkilendirilmesi

Author tablosu ile book tablolarının isbn(kitap tanımlayıcısı) ve authid alanlarıyla ilişkilendirildiği görmek çok da zor değildir. Ancak bu iki alanın birbiriyle ilişkili olabilmesi için veri tiplerinin de aynı olması gerekmektedir.



Şekil 19

Yukarıdaki oluşturulan **auth_books** tablosuna kesişim tablosu denir. Şimdi tablolarımızı oluşturduğumuza göre artık SQL cümleleri kullanarak veri tabanımızdaki tablolara bilgi girişi yapabiliriz.

6.5 SQL Yapıları

Çoğu SQL yapıları 2 grupta toplanır.

- **Veri tanımlama dili(DDL-Data Definition Language) :**Tablo ve içindeki verileri tanımlamada kullanılır.
 - **Veri işleme dili(DML-Data Manipulating Language):**Veriler üzerinde işlem yapmakta kullanılır.
- a. SELECT cümlecği:** Bir dizi sonuç çevirir.
- b. Everything else cümlecği:** Sonuç çevirmez

Tablo oluşturmak için, örneğimizde DDL kullanacağız .DDL CREATE TABLE ve ALTER TABLE gibi SQL cümleleri içermektedir. Projemizin ilerleyen kısımlarında DML yapılarını kullanarak tablolarımıza veri ekleme,veri arama gibi işlemler yapacağız.

DDL için bir örnek:

```
CREATE TABLE authors{  
authid INT NOT NULL PRIMARY KEY,  
lastname CHAR(15) NOT NULL,  
firstname CHAR(15),  
address1 CHAR(25),  
address2 CHAR(25),  
city CHAR(25),  
zipcode CHAR(10),  
country CHAR(15),  
phone CHAR(20),  
fax CHAR(20),  
email CHAR(25));
```

SQL cümleciklerinin büyükle yazıldığına dikkat ediniz. Daha kullanışlı olması açısından VARCHAR veri tipi tanımlaması yerine CHAR veri tipi tanımlanmıştır.

NOT NULL PRIMARY KEY söz dizimi veri tabanına iki ey belirtmektedir. Birincisi authid sütunundaki, her bir satırdaki alanın değeri NULL olamaz. Her bir satırda bu alanın bir değeri girilmelidir. İkinci olarak, PRIMARY KEY açıklaması bu alanın tabloda index alanı olacağı anlamına gelmektedir. Bu alan için girilen kayıt veri tabanında sadece bir adet olabilir. Yani authid alanındaki değer başka bir kayıta tekrarlanamaz.

Bu anlattıklarım, veri tabanın index alanının en önemli özelliklerindendir. Create deyimiyle veri tabanının oluşturduktan sonra sırada INSERT deyimiyle tabloya veri girmeye gelmiştir.

6.5.1 INSERT Cümlecği

Insert cümlecği için 3 önemli bölüm vardır.

- Veri eklenecek hedef tablo belirlenir.
- Verilerin atanacağı sütunlar belirlenir.
- Bu sütunlar için kullanılacak veriler belirlenir.

Insert cümlecği INSERT INTO, kelimeleriyle başlar ve hedef tablo adıyla devam eder.

INSERT INTO authors

Daha sonra hedef tabloda verilerin atanacağı sütun adları belirtilir.

(authid,lastname,firstname,email)

Son olarak bu sütunlara atanacak verileri belirlemeye sıra gelmiştir.

VALUES (99,'catmali','murat','catmali@hotmail.com')

Böylece INSERT cümlecği tamamlanmış oldu.

```
INSERT INTO authors (authid,lastname,firstname,email) VALUES  
'99','catmali','murat','catmali@hotmail.com')
```

6.5.2 SELECT Cümlecği

Veri tabanından verileri almak için kullanılır. SELECT cümlecği için 4 önemli bölüm vardır.

Hangi verilerin alınacağı belirlenir.

Verilerin nereye alınacağı belirlenir.

Şartlar ve filtreler belirlenir.

Hangi sırayla verilerin alınacağı belirlenir.

Select cümlecğinden sonra ikinci sırada hangi sürunların alınacağının karar vermektedir.

SELECT firstname, lastname,authid

Şimdi de verilerin hangi tablodan alacağımıza karar vermeliyiz. Belki bir kaç tabloda aynı alanlar olabilmesi olasılığı vardır. Burada FROM deyiminden sonra tablo adı belirtilerek bu sorun aşılabilir.

FROM authors

Sonuç olarak ;

SELECT firstname, lastname,authid FROM
authors

Veri tabanından aldığımız alanlara takma isim (alias) verebiliriz. Böylece birden fazla alanı tek bir alan altında toplayabiliriz.

Örneğin;

```
SELECT    firstname, lastname,authorid AS  
author_id FROM authors
```

Bu örnekte veri tabanından alınan üç alan author_id alanında birleştirilmiştir.

Eğer tablodaki bütün alanları seçmek istersek;

```
SELECT * FROM authors cümleciği işimizi görecektir.
```

İstediğimiz bir özellikteki alanlardaki kayıtları seçebileceğimiz SQL SELECT cümleciği örneği;

```
SELECT    lastname,firstname    FROM  
authors WHERE country='UK'
```

Bu cümleyle country='UK' olan kayıtlardaki lastname, firstname alanları authors adlı tablodan seçilmektedir.

6.5.3 UPDATE Cümleciği

Bu SQL cümleciği, tablodaki varolan veriyi değiştirmek için kullanılır. Update cümleciğinden önce işlem yapılacak alanın SELECET cümleciği ile seçilmesi gerekmektedir. Update kelimesini işlem yapılacak tablo adı takip eder.

```
UPDATE authors
```

Set kelimesiyle belirtilen tablodaki alanın yeni değerini belirlenebilir.

SET lastname='catmali'

Son olarak da filtre (şart) verilerek hangi kayıtn UPDATE işleminden etkileneceği belirlenir.

WHERE authid=27

Bu cümleyle authid alanı 27 olan kayılar UPDATE cümleciğinde belirtilen işleme tabi tutulur.

Sonuç olarak UPDATE cümleciğinin son hali;

UPDATE authors SET lastname='catmali' WHERE authid=27

6.5.4 DELETE Cümleciği

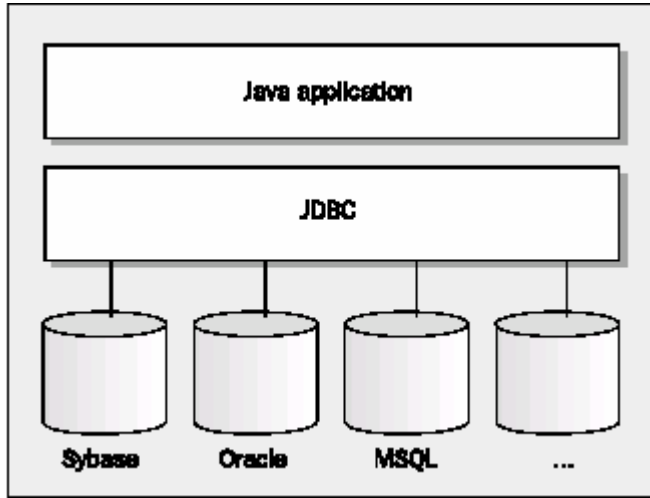
Veri tabanındaki herhangi bir tablodan istenilen şarta uygun kayıtların silinmesinde kullanılır.

DELETE cümleciği DELETE, FROM, WHERE anahtar kelimelerinden oluşmaktadır.

**DELETE FROM books WHERE
isbn='015415454'**

Bu SQL cümleciği ile books adlı tablodan isbn alanı 015415454 olan kayıtlar silinmektedir.

6.6 JDBC Paketi



JDBC paketi SQL cümlelerini alıştırmak için hazırlanmış yüksek seviyeli bir veri erişim ara yüzüdür. Yani, JDBC veri tabanı satır ve sütun işlemlerini otomatik olarak düzenlemek üzere geliştirilmiş bir ara yüzüdür. JDBC uygulamaları değişik veri tabanları için geliştirilmiştir. Bu

yüzden diğer veri tabanlarına geçiş hiç de

Şekil 20

zor olmayacaktır.

Kullanıcı için JDBC java uygulamaları üstte görüldüğü gibidir.

JDBC veri tabanı yönetimini her bir veri tabanı sürücüsüne JDBC ara yüzü ile ilişki kurarak yapmaktadır. Bu sayede JDBC sınıfları veri tabanı olarak diğer Java sınıfları tarafından işlenmesine olanak vermektedir. Bundan noktadan sonra bu konu üzerinde durulacaktır.

6.6.1 JDBC ve ODBC ilişkisi

JDBC uygulamalarının bu kadar pratik olmasının ana nedeni diğer veri tabanı uygulamalarıyla uyumlu olmasıdır. JDBC ve ODBC 'nin ortak yanı da, daha önce bahsedildiği gibi ikisinin de SQL X/Open CLI tabanlı olmasıdır.

Bu noktadan sonra kullanılacak bazı teknik terimlerin açıklanmasında yarar vardır.

6.6.2 Sürücü yöneticisi (Driver manager): Veri tabanı sürücülerini yükler, ve uygulama ile sürücü arasındaki bağlantıyı yönetir.

6.6.3 Sürücü (Driver): Uygulama veri tabanı ile ilgili isteklerini, veri tabanının anlayacağı çağrılara dönüştürür.

6.6.4 Bağlantı (Connection): **Uygulama ve veri tabanı arasındaki oturum.**

6.6.5 Cümle (Statement): **Sorgulama ve güncelleme işlemlerinin yapılmasında kullanılan SQL cümleleri**

6.6.6 Metadata: Veri, veri tabanı ve sürücü hakkında alınan bilgiler.

6.6.7 Sonuç seti (Result Set) : **SQL cümlelerin işlenmesi sonucu ortaya çıkan, mantıksal satır ve sütun yapıları.**

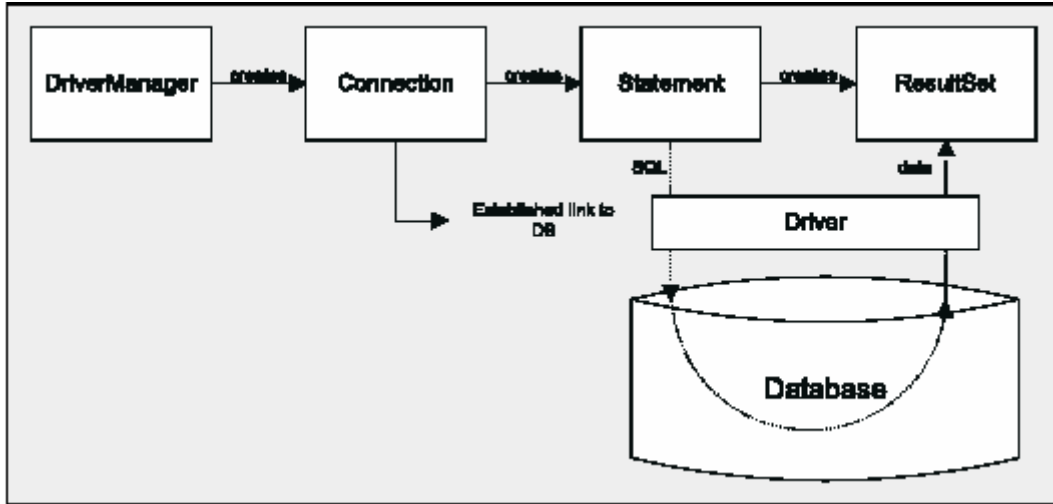
6.6.1.1 JDBC Temelleri

Bu bölümde buraya kadar anlattıklarımı ve gerekli sürücülerin bilgisayarınızda yüklü olduğu kabul edilmektedir.

- Gerekli olan sınıfları yükleyin.
- JDBC sürücüsünü yükleyin.
- Veri tabanını belirleyin.
- Nesne ile bağlantıyı sağlayın.
- SQL cümleciği ile bağlantıyı sağlayın.
- SQL cümleciğini kullanarak sorguyu çalıştırın.
- Sonuç seti ile verileri alın.
- Sonuç setini kapatın.
- SQL cümleciğini kapatın.
- Bağlantı nesnesini kapatın

JDBC mimarisi Java ara yüzleri ve sınıfları sayesinde veri tabanı ile iletişim kurup, SQL cümlecikleri sayesinde veri tabanında

Create, Execute gibi işlemleri yerine getirmeye dayanmaktadır.



Bu işlemler aşağıdaki şemada gösterilmektedir.

Sürücü yöneticisi Bağlantı SQL cümlecği Sonuç kümesi

Şekil 21

Sürücü

Her bir kutu veri tabanına erişimde ve verilerin sunumunda kullanılan ana noktalardır; JDBC sınıfları, ara yüzleri vb. JDBC mimarisinde ilk kullanılan Sürücü yöneticisi sınıfıdır. İlk olarak technical_library veri tabanına bakarak başlangıç noktamızı belirleyelim.

6.6.1.1.1 Veri tabanının hazırlanması

Sürücü yöneticisinin öğrenilmesinden önce incelenmesi gereken önemli bir nokta, JDBC sürücülerinin kodlarının kullanılarak bağlanılacak veri tabanının belirlenmesidir. Technical_library adlı veri tabanımız, teknik kitaplarla ilgili bilgilerin saklandığı Ms Access veri tabanı olsun. Bu veri tabanı Wrox Web sitesinden download edilebilir.

Windows işletim sisteminde, Start->Settings->ControlPanel den ODBC Data Sources ile kullanılacak veri tabanının bağlantısının yapılması gerekmektedir. Bir sonraki bölümde bu konu ayrıntılı olarak işlenmiştir.

Eğer veri tabanıyla bağlantı sağlandıysa Java sınıfları, build tables, tablo yaratmak için kullanılabilir.

java build_tables buildlibrary_access.sql

Bundan noktadan sonraki bütün örneklerde JDBC-ODBC köprüsü sun.jdbc.odbc.JdbcOdbcDriver ile sağlanacaktır.

6.6.2 DriverManager (Sürücü Yöneticisi)

Jdbc veri tabanı sürücüleri sürücü ara yüzlerini içeren sınıflar tarafından tanımlanmıştır. DriverManager JDBC sürücüleri sayesinde veri tabanı kaynağı ile bağlantıyı kuran sınıftır. Eğer her hangi bir Jdbc sürücüsü “jdbc.drivers” olarak bilgisayarınızın sistem özelliklerinde kabul edilmişse, DriverManager sınıfı bu veri tabanı alıp yükleyecektir.

Sistem özellikleri genel olarak Properties sınıfında kayıtlıdır. Properties sınıfı java.util paketi içerisinde. Bu sınıfın özellikleri ayrıntılı olarak bir sonraki bölümde incelenecektir.

Sistem özellikleri System sınıfının setProperty () metodu ile “jdbc.drivers” olarak düzenlenebilir.

System.setProperty(“jdbc.drivers”, “sun.jdbc.odbc.JdbcOdbcDriver”);

Birinci parametre özelliklerin set edilmesi için, ikinci değer ise değeri gösterir. Bu ifade system property nesnesinde JDBC ODBC sürücüsü tanımlar. Bu sürücü ODBC destekli bütün veri tabanları tarafından destekler. Eğer birden fazla sürücü kullanmak istersek, birden fazla sürücü ismi ifadede arka arkaya yerleştirilir.

Eğer sistem yöneticiniz izin verirse, getProperties() metodu ile Properties nesnesi hakkında bilgi edinebilirsiniz. Properties() sınıfı, List () metodu ile aşağıdaki gibi sistem özelliklerini listeler;

System.getProperties().list(System.out);// bütün özellikleri listeler

Kullanacağınız sürücü adını belirlemek için;

Class.forName(“sun.Jdbc.Odbc.JdbcOdbcDriver”);//ODBC sürücüsünü yükler.

kullanılır.

forName() bloğunda belirtilen sürücünün açılmaması durumunda try-catch bloğu içerisinde hataya ilişkin mesaj kullanıcıya verilir. Artık bir JDBC sürücüsüne bağlanmak istediğinizde yeniden bağlantı nesnesini oluşturmanıza gerek yoktur. DriverManager sınıfı bu işlemi sizin için yerine getirecektir.

6.6.4 Veri tabanına bağlantıyı kurma

Bir SQL cümleciğini çalıştırmadan önce yapmanız gereken, bir Connection (bağlantı) nesnesi yaratmaktır. Connection nesnesi veri tabanı ile bağlantıyı sağlayarak, SQL cümleciğinin çalıştırılması için ortam hazırlar. Bunun yanında, Connection() nesnesi, veri tabanı (metadata), kullanıma uygun olan tablolar, satırlar ve sütunlar hakkında bilgi almamızı sağlar.

Connection() nesnesi veri tabanı ile aşağıdaki şekilde bağlantıyı kurar;

Connection databaseConnection=DriverManager.getConnection(source);

Source olarak belirtilen kısım, veri tabanının bilgisayardaki yerini (URL) gösteren String tipi ifadedir.

6.6.4.1 URL ve JDBC

Veri tabanına bağlantıyı sağlamak için kullanılan URL, WWW da, FTP server da veya ağ üzerindeki veri tabanın elektronik olarak yolunu gösterir.

Jdbc:<sub protocol>://<veri tabanı tanımlayıcısı>

Bu kod satırı JDBC veri tabanının yolunu tanımlar. Sub protocol ile gösterilen kısım kullanılacak JDBC sürücüsünü tanımlar. Örneğin JDBC-ODBC köprüsü odbc sürücüsünü kullanır.

Daha önceden tanımladığımız, technical_library veri tabanı için kullanılacak URL formatı; jdbc:odbc:technical_library.

6.6.4.2 Bağlantıyı kuralım

Aşağıdaki jdbc programı Connection nesnesi sağlar. URL ile veri tabanına bağlantı

```
import java.sql.*;

public class MakingTheConnection
{
    public static void main(String[] args)
    {
        // Load the driver
        try
        {
            // Load the driver class
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

            // Define the data source for the driver
            String sourceURL = "jdbc:odbc:technical_library";

            // Create a connection through the DriverManager
            Connection databaseConnection =
                DriverManager.getConnection(sourceURL);

        }
        catch (ClassNotFoundException cnfe)
        {
            System.err.println(cnfe);
        }
        catch (SQLException sqle)
        {
            System.err.println(sqle);
        }
    }
}
```

gerçekleştirilmiştir.

Örneğimiz nasıl çalışır:

Doğal olarak JDBC kütüphanelerinin, sınıflarının ve ara yüzlerinin yüklenmesi gerekmektedir. Bu sınıflar java.sql paketinde tanımlanmıştır. main() bloğunun başında forName () metoduyla programımız için gerekli olan, JDBC sürücüsü için sınıfları yüklenir. Daha önce de belirtildiği gibi, forName() metoduyla belirtilen sürücü sınıfının bulunamaması durumunda ClassNotFoundException ile kullanıcıya hata mesajı verilir. Çoğu JDBC metodu hata kontrolünü SQLException, hata tespitiyle sağlarlar.

6.6.4.3 Complex bağlantılar

Eğer veri tabanına bağlantı için kullanıcı adı ve şifresi gerekiyorsa, getConnection() ikinci metodu kullanılır.

DatabaseConnection=DriverManager.getConnection(URL,UserName,Password);

Belirtilen bütün parametrelerin tipi String ifadedir.

```
import java.util.Properties;

// ...

String driverName = "sun.jdbc.odbc.JdbcOdbcDriver";
String sourceURL = "jdbc:odbc:technical_library";

try
{
    Class.forName (driverName);
    Properties prop = new Properties();
    prop.setProperty("user", "ItIsMe");
    prop.setProperty("password", "abracadabra");
    Connection databaseConnection = DriverManager.getConnection(sourceURL, prop);
}
catch(ClassNotFoundException cnfe)
{
    System.err.println("Error loading " + driverName);
}
catch(SQLException sqle)
{
    System.err.println(sqle);
}
```

Aşağıdaki program ODBC için JDBC sürücü bağlantısını gerçekleştirir.

6.6.4.4 Bağlantı kesilme süresinin tespiti:

DriverManager sınıfı, login timeout (bağlantı kesim süresi) periyodunu belirler. Bu metot saniye olarak bağlantı süresinin belirlenmesini sağlar. Bu süre içerisinde veri tabanına bağlantının sağlanması gerekir.

public static void setLoginTimeout(int saniye)

public static void getLoginTimeout(int saniye)

Bu metodlar veri tabanına uzaktan bağlantı sırasında, server makinenin meşgul olması ihtimaline karşı tanımlanmıştır. Eğer belirtilen sürede veri tabanına bağlanılamazsa bağlantı girişiminin askıya alınması gerekir. Aşağıdaki program 60 saniye içerisinde veri tabanına bağlantı sağlanamaması durumunda bağlantı girişimini askıya alır.

```
String driverName = "sun.jdbc.odbc.JdbcOdbcDriver";
String sourceURL = "jdbc:odbc:technical_library";
```

```
try
{
    Class.forName(driverName);
```

```
// fail after 60 seconds
DriverManager.setLoginTimeout(60);

    Connection databaseConnection = DriverManager.getConnection(sourceURL);
}
catch(ClassNotFoundException cnfe)
{
    System.err.println("Error loading " + driverName);
}
catch(SQLException sqle)
{
    System.err.println(sqle);
}
```

6.6.3 Sürücü çeşitleri

DriverManager sınıfı yüklendikten sonra bir çok çeşit sürücü ile veri tabanına bağlanılabilir.

Bunlar;

- JDBC-ODBC köprü sürücüsü;
- Yerel uygulama bölümü-Java
- Bütün ağ protokolleri-Java istemci
- Bütün yerel protokoller-Java

Sürücülerinin nasıl çalıştığı hakkında bir mantık gelişmesi için yukarıda belirtilen sürücüler kısaca anlatılacaktır.

6.6.3.1 JDBC-ODBC köprü sürücüsü (JDBC-ODBC Bridge Driver)

JDBC-ODBC köprüsü- “sun.jdbc.odbc.JdbcOdbcDriver” ile birlikte JDK içerisinde gelmektedir. Bu sürücü sayesinde ODBC standardına uygun olarak Java uygulamalarının veri tabanına bağlanması sağlanır. Sürücü kullanıma uygun olan veri tabanlarına ulaşım için kullanılır.

Bu sürücünün en büyük avantajı çok sayıdaki ODBC sürücüsüne sorunsuz olarak bağlanmasıdır. Dezavantajı ise sadece Microsoft Windows ve Sun Solaris işletim sistemleriyle uyumlu olarak çalışmasıdır.

6.6.3.2 Yerel uygulama bölümü Java sürücüler (Native API)

Bu sürücü köprü sürücüsüne çok benzemektedir. Yerel bir ağdaki veri tabanına ulaşmada kullanılan sınıfları içeren Java program kodlarından oluşur.

6.6.3.3 Bütün ağ protokolleri-Java istemci (Net Protocol All Java Client)

Bu sınıf sürücüsü genellikle TCP/IP protokollerini kullanarak ağ uygulamalarında, istemci JDBC isteklerini dağıtık veri tabanına iletmeye kullanılır.

6.6.3.4 Bütün yerel protokoller-Java (Native Protocol All Java)

Server makinenin yerel ağ protokollerini kullanarak veri tabanına erişimde kullanılan sürücülerdir. Bir önceki sürücü tipinin tersine herhangi bir dönüşümü gerek yoktur. İstemci direk olarak veri tabanına erişebilmektedir. Eğer sürücü sınıfı uygunsa, kullanılması gereken tek sürücü tipidir.

Gerekli olan JDBC sürücüleri <http://www.javasoft.com/jdbc.drivers.html> adresinden download edilebilir.

6.6.5 SQL cümleleri nesneleri (Statement Objects)

Statement nesneleri, Statement ara yüzünü geliştirmek için kullanılan sınıfları kullanır. Statement nesnesi yaratıldığında SQL cümleciklerin çalıştırılması, sorgulanması ve sonuçlarının geri çevrilmesi için bir alan yaratır.

Statment nesnesi, createStatement() metoduyla oluşturulur. Statment nesnesi bir kere oluşturulmasıyla, SQL cümleciklerinin çalıştırılması için gerekli olan executeQuery() metodu ile sorgulamalar yapılabilir.

Sorgulama sonuçları ResultSet (Sonuç kümesi) nesnesi şeklinde geri döner. Eğer Statment nesnesine sahip olmak istiyorsanız aşağıdaki sintaksı kullanmalısınız.

ResultSet results=statement.executeQuery

(“SELECT lastname, firstname FROM authors“)

6.6.7 Sonuç kümesi nesneleri(ResultSet Objects)

SQL cümlesinin çalıştırılması sonucu oluşturulan tablo sonuçlarını içeren ara yüzdür.

ResultSet nesnesi ile elde edilen sonuçlara uygun olarak imleç veri tabanı kayıtları üzerinde dolaşır. First () ve last() metotları kullanılarak ilk ve son kayıtlara konumlanır. Bunun yanında beforeFirst(), beforeLast() ve previous() gibi bir çok metot seçeneğimiz vardır.

Çoğu zaman veri tabanı kayıtları üzerinde ileri ve geri hareket etmemiz gerekebilir. Bu gibi durumlarda aşağıdaki kod satırları kullanılabilir.

```
while(resultSet.next() )
```

```
{
```

```
//kayıt işlemleri
```

```
}
```

bunun yanında ilk ve son kayıtlarda olduğumuz kontrol etmemiz için kullanılan isLast() ve isFirst() metotları da vardır.

6.6.7.1 Resultset nesnesindeki verilere ulaşma

Resultset nesnesi kullanılarak kayıtların kendisine veya kayıt numarasına ulaşılabilir. Bunun yanında sütunlardaki veri tipleri sütun sayıları gibi bilgilere de ulaşabiliriz. Resultset nesnesi sütun ve satır veri tipleri için aşağıdaki temel metotlara sahiptir.

GetAsciiStream()

GetTimestamp()

GetTime()

GetBoolean()

GetBinaryStream()

GetString()

GetDate()

GetBytes()

GetByte()

GetInt()

GetFloat()

GetDouble()

GetShort()

GetObject()

Getlong()

Aşağıdaki örnekte tchnical_library adlı veri tabanına bağlantıyı sağlamak için MakingTheConnection adlı bir sınıf tanımlamıştır.

```

import java.sql.*;

public class MakingAStatement
{
    public static void main(String[] args)
    {
        // Load the driver
        try
        {
            // Load the driver class
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

            // This defines the data source for the driver
            String sourceURL = new String("jdbc:odbc:technical_library");

            // Create connection through the DriverManager
            Connection databaseConnection =
                DriverManager.getConnection(sourceURL);

            Statement statement = databaseConnection.createStatement();

            ResultSet authorNames = statement.executeQuery
                ("SELECT lastname, firstname FROM authors");

            // Output the resultset data
            while(authorNames.next())
                System.out.println(authorNames.getString("lastname")+" "+
                    authorNames.getString("firstname"));
        }
        catch(ClassNotFoundException cnfe)
        {
            System.err.println(cnfe);
        }
        catch(SQLException sqle)
        {
            System.err.println(sqle);
        }
    }
}

```

Yukarıdaki program tabloya bağlantı kurulduktan sonra SQL sorgusu sayesinde lastname ve firstname alanları authors adlı tablodan alıp, kayıtlardaki firstname ve lastname alanları ekrana listelemektedir.

BÖLÜM 7

JDBC ve UYGULAMA PROGRAMLARI

Java veri tabanı uygulamaları için Java Bin klasörün de JdbcOdbc.dll dosyasının bulunması gerekir.

7.1 Jdbc nedir?

Jdbc, Microsoft'un ODBC (open database connectivity- açık veri tabanı bağlantısı) arayüzünü taban alan SQL API' lerinin bir kümesidir. Jdbc kullanılarak elde edilen veriler SQL komutlarıyla süzgeçten geçirilerek istenilen amaca yönelik kullanılabilir.

7.2 Java veri tabanı bağlantısının kurulması:

Java yı yalnızca internette animasyon, buton hazırlamakta hazırlanmasında kullanılan bir dil olarak düşünmek yanlış olur. Java SQL komutları sayesinde veri tabanı bağlantısı gerçekleştirerek istenilen fonksiyonları gerçekleştiren bir dildir.

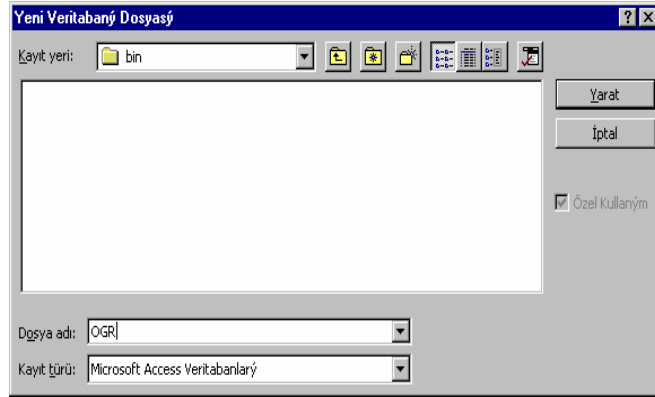
7.2.1 Access Veri Tabanı İle Jdbc Kullanımı:

Örneğimiz okulumuzun Bilgisayar Eğitimi bölümü öğrencilerine yönelik kayıt ekleme,düzeltilme,silme kısımlarından oluşacaktır. İncelenecek konunun okuyucu açısından anlamlı olması öğrenmeyi kolaylaştırıp kalıcı olacağı unutulmamalıdır. Tezimin bir amacıda benden sonraki arkadaşlarımin yararlanabileceği bir kaynak yaratmaktır.(Grup Java,2000)

İlk örneğimiz için Access veri tabanı programı ile boş bir veri tabanı oluşturup, bu boş veri tabanını ODBC nesnesi ile bağlantı oluşturacağız.

7.7.2.2 ODBC bağlantısının kurulması:

Adım1 :Microsoft Access programı ile OGR isimli boş bir veri tabanı oluşturuyoruz.



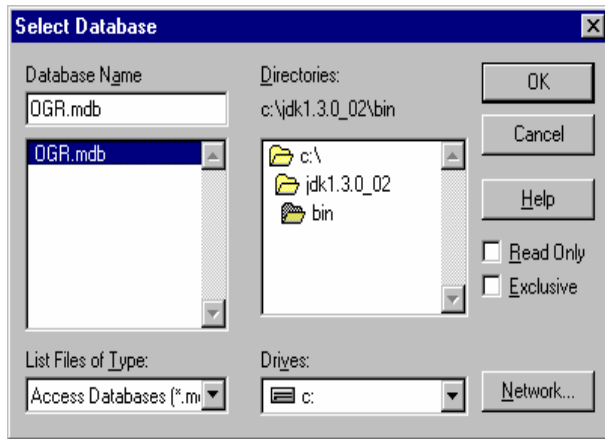
Şekil 22

Adım2:Boş veri tabanı ile ODBC bağlantısının oluşturulması.

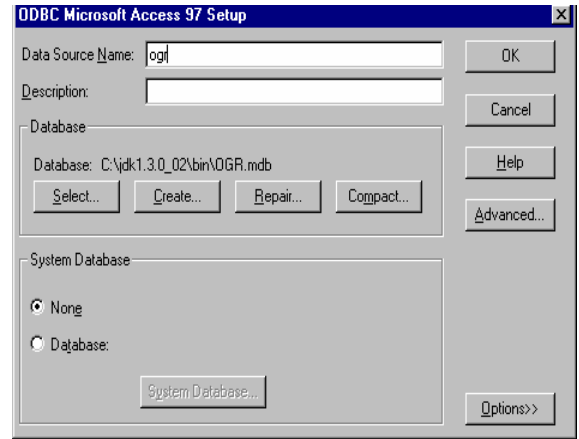
Denetim Masası altında ODBC nesnesi açılarak System dns sekmesinden Add butonu ile daha önceden oluşturmuş olduğumuz veri tabanını bağlayalım. Data Source Name alanına isim verirken dikkatli olalım, çünkü bu isim daha sonra programlarda kullanacağımız veri tabanı dosyası adı olacaktır

Veri tabanımızı bağladıktan sonra artık ilk örneğimizi yapabiliriz.

Şekil 23



Şekil 24



Aşağıdaki program Java programı -veri tabanı arası bağlantıyı kurara .mdb dosyasında bir tablo oluşturur.

```
import java.sql.*;

public class giris{

    public static void main(String args[]){

        String url="jdbc:odbc:ogr";
        Connection con;
        String createString;
        createString = "create table KIMLIK"+
        "(NO varchar(11)," +
        "AD varchar(15)," +
        "SOYAD varchar(15)," +

        "BOL_KOD varchar(3)," +
        "DONEM varchar(2)," +
        "D_TAR int)";

        Statement st;

        try{
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            System.out.println("Veri tabanina baglananildi");

        } catch(java.lang.ClassNotFoundException e){
            System.err.print("ClassNotFoundException: ");
            System.err.println(e.getMessage());
        }

        try{
            con=DriverManager.getConnection(url,"Murat","Murat");

            st=con.createStatement();

            st.executeUpdate(createString);
            System.out.println("Tablo yaratildi");
```

```

        st.close();
        con.close();

    } catch(SQLException ex){
        System.err.println("SQLException:" + ex.getMessage());
    }

}

```



Şekil 25

Aşağıdaki program .mdb dosyasına kayıt eklemek için kullanılır. Önce veri tabanı bağlantısı kurulduğuna ve bu bundaki önce bölümde anlatılan bağlantı nesnelerinin yaratıldığına dikkat ediniz.

ogrekle.java (Dosyalama Örneği)

```

import java.sql.*;
public class ogrekle{

    public static void main(String args[]){
        String url="jdbc:odbc:OGR";
        Connection con;
        Statement stmt;
        String query="select * from KIMLIK";
    }
}

```

```

try{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
}catch(java.lang.ClassNotFoundException e){
System.err.print("ClassNotFoundException:");
System.err.println(e.getMessage());
}

        try{
            con=DriverManager.getConnection(url,"Murat","Murat");
            stmt=con.createStatement();
            stmt.executeUpdate("INSERT INTO KIMLIK
values('1','murat','catmali','240','8','1979')");
            stmt.executeUpdate("INSERT INTO KIMLIK
values('2','deniz','kivrak','240','8','1979')");

            ResultSet rs=stmt.executeQuery(query);

            System.out.println("OGRENCI LISTES~");

            while (rs.next()){
                String s1=rs.getString("NO");
                String s2=rs.getString("AD");
                String s3=rs.getString("SOYAD");
                String s4=rs.getString("BOL_KOD");
                String s5=rs.getString("DONEM");
                int n=rs.getInt("D_TAR");
                System.out.println(s1 +" "+s2+" "+s3+" "+s4+" "+s5+" "+n);
            }
            stmt.close();
            con.close();
        }catch(SQLException ex){
            System.err.println("SQLException:"+ex.getMessage());
        }
    }
}

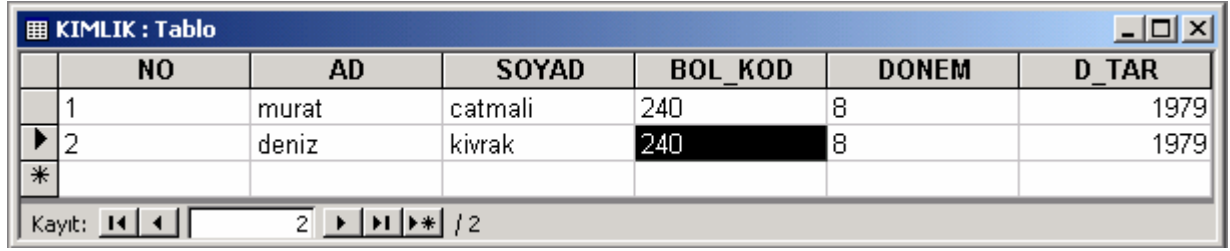
```

```

C:\JDK13~1.0_0\bin>javac ogrekle.java
C:\JDK13~1.0_0\bin>java ogrekle
OGRENCI LISTESİ
1 murat catmali 240 8 1979
2 deniz kivrak 240 8 1979
C:\JDK13~1.0_0\bin>

```

Şekil 26



	NO	AD	SOYAD	BOL_KOD	DONEM	D_TAR
	1	murat	catmali	240	8	1979
▶	2	deniz	kivrak	240	8	1979
*						

Kayıt: 2 / 2

Şekil 27

Aşağıdaki örnek .mdb dosyasına bir form ile bilgi girişini sağlar.

Giris1.java (Dosyalama Örneği)

```
import java.awt.*;
import java.awt.event.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.Dimension;
import java.sql.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.text.*;
import java.text.*;
import java.util.*;

public class giris1 extends JFrame{

    JButton Bkaydet,Biptal;
    JTextField Tbarkod,Tadi,Tadet,Tfiyat;
```

```

JLabel Lbarkod,Ladi,Ladet,Lfiyat,Lcinsi;
JComboBox Ccinsi;
Connection con1;
ResultSet rs;
PreparedStatement sorgu,urunInsert,stokInsert;
Statement cinsbul;
String[] cinsler;

public giris1(){
super("sr n Giri Yi",
false,
true,
false,
false);
Container contentPane = getContentPane();
contentPane.setLayout(null);

Bkaydet=new JButton("Kaydet");
contentPane.add(Bkaydet);
Bkaydet.setMnemonic('K');
Bkaydet.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent e){
    kaydet();
    try{
        rs.close();
        cinsbul.close();
        urunInsert.close();
        stokInsert.close();
        con1.close();
        dispose();
    }catch(SQLException ex){
        System.err.println("Kay tta Hata Var");
        System.err.println(ex);
    }
}
});

Biptal=new JButton("~ptal");
contentPane.add(Biptal);
Biptal.setMnemonic('~');
Biptal.addActionListener(new ActionListener(){
public void actionPerformed(ActionEvent e){
    try{
        rs.close();
        cinsbul.close();
        urunInsert.close();
        stokInsert.close();
        con1.close();
        dispose();
    }catch(SQLException ex){

```

```

        System.err.println("Iptalde Hata Var");
        System.err.println(ex);
    }
}
});
    Tbarkod=new JTextField(20);
    contentPane.add(Tbarkod);
    Tbarkod.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e){
            Tadi.requestFocus();
        }
    });

```

```

Tadet=new JTextField(5);
contentPane.add(Tadet);
Tadet.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
        if(Tadet.getText().length()==0){
            Tadet.setText("0");
            Bkaydet.requestFocus();
        }
        else
            Bkaydet.requestFocus();
    }
});

```

```

    Tfiyat=new JTextField(11);
    contentPane.add(Tfiyat);
    Tfiyat.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e){
            if(Tfiyat.getText().length()==0)
                JOptionPane.showMessageDialog(getContentPane(),
                    "Lütfen Fiyat Giriniz.", "Uyarı",
                    JOptionPane.ERROR_MESSAGE);
            else
                Tadet.requestFocus();
        }
    });

```

```

    Tadi=new JTextField(11);
    contentPane.add(Tadi);
    Tadi.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e){
            Ccinsi.requestFocus();
        }
    });

```

```

Lbarkod=new JLabel("Bar kod");
contentPane.add(Lbarkod);

```

```
Ladi=new JLabel("Adi");
contentPane.add(Ladi);
```

```
Lcinsi=new JLabel("Cinsi");
contentPane.add(Lcinsi);
```

```
Lfiyat=new JLabel("Fiyat");
contentPane.add(Lfiyat);
```

```
Ladet=new JLabel("Adet");
contentPane.add(Ladet);
```

```
cinsler=new String[15];
DBBaglan();
Ccinsi=new JComboBox(cinsler);
contentPane.add(Ccinsi);
```

```
Lbarkod.setBounds(30,30,70,25);
Ladi.setBounds(30,60,70,25);
Lcinsi.setBounds(30,90,70,25);
Lfiyat.setBounds(30,120,70,25);
Ladet.setBounds(30,150,70,25);
Tbarkod.setBounds(105,30,110,25);
Tadi.setBounds(105,60,110,25);
Ccinsi.setBounds(105,90,110,25);
Tfiyat.setBounds(105,120,110,25);
Tadet.setBounds(105,150,110,25);
```

```
Bkaydet.setBounds(40,195,80,25);
Biptal.setBounds(140,195,80,25);
```

```
setSize (260,270);
setLocation (160,70);
```

```
Tbarkod.setRequestFocusEnabled(true);
Tbarkod.requestFocus();
}
```

```
protected void DBBaglan(){
i=1;
try{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
con1=DriverManager.getConnection("jdbc:odbc:mal","DEN~Z",
"DEN~Z");
```



```

cinsbul=con1.createStatement();
rs=cinsbul.executeQuery("SELECT cins FROM cins");
    while(rs.next()){
        cinsler[j]=rs.getString("Cins");
        j++;
    }

sorgu=con1.prepareStatement("SELECT adi,adet,fiyati"+
"FROM urun,stok"+
"WHERE urun.Barkod = ?"+
"AND stok.Barkod = ?");
urunInsert= con1.prepareStatement("INSERT INTO urun"
+"VALUES(?,?,?,?)");
stokInsert=con1.prepareStatement("INSERT INTO stok"
+"VALUES(?,?)");

} catch(ClassNotFoundException ex){
System.err.println("Veri tabani s□r□c□leri bulunamad□.");
System.err.println(ex);
} catch(SQLException ex){
System.err.println("Veri taban□na baglan□lamad□");
System.err.println(ex);
}
}

```

```

protected void kaydet() {
if(Tfiyat.getText().length()==0||
Tbarkod.getText().length()==0){
JOptionPane.showMessageDialog(getContentPane(),
"Fiyat ve Barkod alani mutlaka doldurulmalisiniz!",
"Hatal□ Giriş",
JOptionPane.ERROR_MESSAGE);
}
else{
try {
String u1=Tbarkod.getText();
String u2=Tadi.getText();
String u3=(String)Ccinsi.getSelectedItem();
tmp4=Integer.valueOf(Tfiyat.getText());
int fiyat1=tmp4.intValue();

urunInsert.setString(1,u1);
urunInsert.setString(2,u2);
urunInsert.setString(3,u3);
urunInsert.setInt(4,fiyat1);

if(Tadet.getText().length()==0) Tadet.setText("0");
stokInsert.setString(1,Tbarkod.getText());
tmp4=Integer.valueOf(Tadet.getText());
int adet1=tmp4.intValue();

```

```
stokInsert.setInt(2,adet1);

urunInsert.executeUpdate();
stokInsert.executeUpdate();
}catch(SQLException ex){
System.err.println("şr□n giriÝinde hata var");
System.err.println(ex);
}try
}else
}kaydet

String adi,barkod,tmp1;
int fiyat,adet,i=0,j=0;
int fiyat1,adet1;
String tmp3;
Integer tmp4;
```

```
public static void main(String args[])
{
new giris1();
}
}
```

Özet

JDBC, ilişkisel veri tabanlarına ara yüz olarak tanımlanabilir. Veri tabanına ulaşma, uzaktaki veya yerel bir veri tabanına ulaşp, buradaki verilerin alınması veya işlenmesi anlamına gelir. Peki bu işlemi Java programlama dili nasıl gerçekleştirmektedir.

Java Veri tabanı bağlantısı (JDBC) sınıfları SQL cümlelerini çalıştırarak ilişkisel veri tabanlarına erişimi ve bu veriler üzerinde işlem yapmayı sağlar. JDBC nesneye dayalı, veri tabanına erişim için geliştirilen uygulama programları ara yüzü (Application Programming Interface - API), ve Java geliştiricileri ve veri tabanları için standart olarak geliştirilmiştir.

JDBC uygulamalarının bu kadar pratik olmasının ana nedeni diğer veri tabanı uygulamalarıyla uyumlu olmasıdır. JDBC ve ODBC 'nin ortak yanı da, ikisinin de SQL X/Open CLI tabanlı olmasıdır.

JDBC ile veri tabanı bağlantısını sağlamanın temel noktaları;

- Gerekli olan sınıfları yükleyin.
- JDBC sürücüsünü yükleyin.
- Veri tabanını belirleyin.
- Nesne ile bağlantıyı sağlayın.
- SQL cümleciği ile bağlantıyı sağlayın.
- SQL cümleciğini kullanarak sorguyu çalıştırın.
- Sonuç seti ile verileri alın.
- Sonuç setini kapatın.
- SQL cümleciğini kapatın.
- Bağlantı nesnesini kapatın

