Tarih İşlemleri

Java içerisinde tarih işlemleri bir kaç sınıf ile karşılanmaktadır. Bunların başlıcaları java.util.Date, java.sql.Date, java.util.Caledar, java.text.DateFormat sınıflarıdır diyebiliriz.

Date

Java da tarih ile ilgili bir ilkel yani primitive bir tip yoktur bunun için Date sınıfı kullanılır. Yukarıdaki açıklamada da geçtiği üzere bunlar java.util ve java.sql paketleri içerisinde olmak üzere 2 tanedir. Ancak biz standart yani java.util.Date sınıfını kullanıyor olacağız. java.sql.Date ise daha çok veritabanı işlemleri için tasarlanmış bir sınıftır.

Aslında Date sınıfı kendi içerisinde tarih bilgisini sayısal değer olarak tutmaktadır bunun nedeni tarih kavramının belirli bir formata sahip olmamasıdır. Peki bu sayısal değer neyi ifade etmektedir ? Java ve diğer dillerde tarih için belirlenen milat 1 Ocak 1970' tir yani elektronik tüm cihazlarda kullanılan başlangıç tarihi. Bu durumda eğer tarih anlamında elimizde 0 değeri varsa bu bizim için 1 Ocak 1970 i ifade edecektir. Bundan önceki ve sonraki tarihler milisaniye artı ve eksi olarak belirtilecektir.

Örnek verecek olursak aşağıdaki kod örneğinde çıktı şu anki zamanı 1337939429382 ile milisaniye olarak vermiştir. Yani 1 Ocak 1970 tarihinden beri bu kadar milisaniye zaman geçmiştir.

```
import java.util.Date;

public class Ornek {
	public static void main(String[] args) {
		Date tarih = new Date();
		System.out.println(tarih.getTime());
	}
}
```

Peki biz bu durumda tarih atamasını nasıl yapacağız ? Aşağıdaki örnekte iki adet tarih tanımı bulunmaktadır. Bunlardan t1 constructor parametresi olarak herhangi bir değer almamıştır. t2 ise constructor değeri olarak 0 almıştır.

```
import java.util.Date;

public class Ornek {
	public static void main(String[] args) {
		Date t1 = new Date();
		Date t2 = new Date(0);
		System.out.println(t1);
		System.out.println(t2);
	}
}
```

Bu örneğimizin çıktısı aşağıdaki gibidir.

```
Fri May 25 12:53:33 EEST 2012
Thu Jan 01 02:00:00 EET 1970
```

Sonuçtanda anlaşılacağı gibi Date tipinde bir nesne oluşturduğumuz zaman eğer constructor parametresi atanmamışsa direkt olarak bu günkü değeri alıyor. Ancak ben ikinci değişkende 0 değerini gönderdiğim anda buna eşit gelen 1 Ocak 1970 değerini yazdırıyor.

Not : Buradaki örnek çıktılarda tarih formatları anlamsız görünebilir. Bunu daha anlaşılabilir kılmak için DateFormat sınıfını kullanıyor olacağız.

Calendar

Date sınıfı içerisinde gün, ay ve yıl gibi bilgiler almak istediğimizde bu metodların aşağıdaki örnekteki gibi deprecated yani kullanımdan kalkmış olduğunu belirtir.

```
import java.util.Date;

public class Ornek {
	public static void main(String[] args) {
		Date tarih = new Date();
		System.out.println(tarih.getMonth());
		System.out.println(tarih.getYear());
	}
}
```

Bu metodların işlevleri Java 1.1 ile birlikte java.util.Calendar sınıfına bırakılmıştır. Calendar sınıfı Date tipindeki nesneler için yardımcı özellikler getirmektedir.

Aşağıdaki örnekte yıl bilgisini almaktayız. Buna göre yıl tanımını Calendar sınıfı içerisinedeki YEAR sabitiyle alıyoruz. Buradaki her sabit 0, 1 gibi bir sayısal değeri ifade etmektedir.

```
import java.util.Calendar;

public class Ornek {
	public static void main(String[] args) {
		Calendar takvim = Calendar.getInstance();
		int yil = takvim.get(Calendar.YEAR);
		System.out.println(yil);
	}
}
```

Not : Calendar sınıfı içerisinde constructor lar protected olduğundan direkt nesne oluşturamazlar. Bunun yerine getInstance() metodları kullanılır.

Takvim bilgisini almanın yanında aşağıdaki örnekte olduğu gibi değer ataması da yapabiliriz. Bu atama set metodu içerisinde tek tek yapılacağı gibi gün, ay, yıl, saat, dakika ve sn gibi bilgiler tek seferde de atanabilir.

```
import java.util.Calendar;

public class Ornek {
	public static void main(String[] args) {
		Calendar takvim = Calendar.getInstance();
		takvim.set(Calendar.YEAR, 2013);
		takvim.set(Calendar.MONTH, Calendar.SEPTEMBER);
		takvim.set(Calendar.DAY_OF_MONTH, 16);


		takvim.set(2013, Calendar.SEPTEMBER, 16);
		takvim.set(2013, Calendar.SEPTEMBER, 16, 22, 14, 00);
	}
}
```

Not : Calendar sınıfı içerisindeki getTime() metodu geriye java.util.Date döndürür.

Zaman Tanımları

Calendar.YEAR	Yıl değeri
Calendar.MONTH	Ay değeri (0-11)
Calendar.DAY_OF_MONTH	Ayın günü (1-31)
Calendar.DAY_OF_WEEK	Haftanın günü (0-6)
Calendar.HOUR	Saat değeri (0-12)
Calendar.AM_PM	Calendar.AM veya Calendar.PM değeri
Calendar.HOUR_OF_DAY	Saat değeri (0-24)
Calendar.MINUTE	Dakika değeri (0-59)
Calendar.SECOND	Saniye değeri (0-59)
Calendar.MILLISECOND	Milisaniye değeri (0-999)

Ay Değerleri

Calendar.JANUARY
Calendar.FEBRUARY
Calendar.MARCH
Calendar.APRIL
Calendar.MAY
Calendar.JUNE
Calendar.JULY
Calendar.AUGUST
Calendar.SEPTEMBER
Calendar.OCTOBER
Calendar.NOVEMBER
Calendar.DECEMBER

Gün Değerleri

Calendar.SUNDAY
Calendar.MONDAY
Calendar.TUESDAY
Calendar.WEDNESDAY
Calendar.THURSDAY
Calendar.FRIDAY
Calendar.SATURDAY

Takvim üzerinde gün, ay, yıl gibi değerler için ekleme çıkarma gibi sayısal bir işlem yapmak istiyorsak add metodunu kullanabiliriz. Örneğin aşağıda belirli bir tarihe gün ve saat değerleri ilave ediliyor.

```
import java.util.Calendar;

public class Ornek {
	public static void main(String[] args) {
		Calendar takvim = Calendar.getInstance();
		takvim.set(2012, Calendar.SEPTEMBER, 10, 0, 0, 0);

		System.out.println(takvim.getTime());
		takvim.add(Calendar.DAY_OF_MONTH, 10);
		takvim.add(Calendar.HOUR_OF_DAY, 6);
		System.out.println(takvim.getTime());
	}
}
```

Tarih Formatı

Tarihler belirsiz veya sayılsal bir formatta demiştik. Peki bu tarihleri istediğimiz formata çevirmek için ne yapmamız gerekiyor ? Bunun için DateFormat ve bundan türeyen SimpleDateFormat sınıfları kullanılabilir.

Aşağıdaki örnekte tarih nesnesi DateFormat sınıfı ile formatlanıyor. Buradaki önemli noktalardan biri formatlama sonrasında geriye String bir değer dönmesi. Yani formatlama işlemi sonrasında çıktı verilecek değerin tarih nesnesi ile bir alakası kalmıyor.

```
import java.text.DateFormat;
import java.util.Date;

public class Ornek {
	public static void main(String[] args) {
		Date tarih = new Date();

		DateFormat formatlayici = DateFormat.getDateInstance(
DateFormat.SHORT);
		String formatliTarih = formatlayici.format(tarih);
		System.out.println(formatliTarih);
```

```
	}
}
```

Not : DateFormat içerisinde SHORT haricinde DEFAULT, MEDIUM, LONG ve FULL şeklinde halihazırda formatlar bulunur.

Eğer çıktı alacağımız tarihi yerelleştirmek istiyorsak aşağıdaki örnekteki gibi Locale tanımı ile bunu gerçekleştirebiliriz.

```
import java.text.DateFormat;
import java.util.Date;
import java.util.Locale;

public class Ornek {
	public static void main(String[] args) {
		Date tarih = new Date();
		Locale yerel = new Locale("tr");


		DateFormat formatlayici = DateFormat.getDateInstance(
DateFormat.SHORT, yerel);
		String formatliTarih = formatlayici.format(tarih);
		System.out.println(formatliTarih);

	}
}
```

Eğer tarih formatımızı daha özel bir şekle getirmek istiyorsak bunun için DateFormat sınıfından türetilmiş SimpleDateFormat sınıfını kullanabiliriz. SimpleDateFormat' ın en önemli özelliği formatlama yaparken özelleştirilmiş şablonlar verilebilmesidir.

Aşağıdaki örnekte benzer bir şekilde SimpleDateFormat ile bir formatlama biçimi bulunmaktadır.

```
import java.text.SimpleDateFormat;
import java.util.Date;

public class Ornek {
	public static void main(String[] args) {
		Date tarih = new Date();

		SimpleDateFormat formatlayici = new SimpleDateFormat(
"dd.MM.yyyy");
		String formatliTarih = formatlayici.format(tarih);
		System.out.println(formatliTarih);

	}
}
```

Tarih Şablon Formatları

Anahtar	Tanım	Örnek
G	Çağ	AD
y	Yıl	1996; 96
M	Ay	July; Jul; 07
w	Yılın kaçıncı haftası	27
W	Ayın kaçıncı haftası	2
D	Yılın kaçıncı günü	189
d	Ayın kaçıncı günü	10
F	Haftanın kaçıncı günü	2
E	Haftanın hangi günü	Tuesday; Tue
a	Am/pm durumu	PM
H	Saat (0-23)	0
k	Saat (1-24)	24
K	Saat am/pm (0-11)	0
h	Saat am/pm (1-12)	12
m	Dakika	30
s	Saniye	55
S	Milisaniye	978
z	Saat Dilimi	Pacific Standard Time; PST; GMT-08:00
Z	Saat Dilimi	-0800

Not: Saat dilimlerinde şablon tanımlarken kullanacağınız anahtarların sayısı formatın çıktısını etkileyecektir. Örneğin yıl formatında y sayıları aşağıdaki gibi çıktı verir.

dd.MM.yyyy	25.05.2012
dd.MM.yy	25.05.12
dd.MM.yyyy HH:mm:SS	25.05.2012 15:35:856
dd.MMM.yyyy	25.May.2012
dd.MMMM.yyyy	25.Mayıs.2012
EEEE.MMMM.yyyy	Cuma.Mayıs.2012

Eğer elimizde tarih görünümlü ancak String tipinde bir değer varsa bununla herhangi bir tarih operasyonu gerçekleştiremeyiz. Bu durumda String değerin Date tipine çevrilmesi gerekmektedir. Buna parse yani ayrıştırma işlemi diyoruz ve yine DateFormat ve SimpleDateFormat sınıflarındaki parse metodunu kullanmamız gerekiyor.

Aşağıki örnekte String tipindeki değerin Date formatına çevrilmesi işlemi yer alıyor. Buna göre String olan s değeri java.util.Date tipine t değeri olarak çevriliyor.

```
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
```

```
public class Ornek {
	public static void main(String[] args) {
		SimpleDateFormat formatlayici = new SimpleDateFormat
("dd.MM.yyyy");

		String s = "16.09.2012";
		try {
			Date t = formatlayici.parse(s);
		} catch (ParseException e) {
			System.out.println("Formatlamada hata oluştu");
		}

	}
}
```

Not : Parse işleminde format uyumunda herhangi bir sorun oluşursa ParseException hatası fırlatılacaktır ve bu bloğun try – catch ile yönetilmesi zorunludur.