Java Date & Time API

Java'da Zaman İşlemleri

- java.util.**Date** ve java.util.**Calendar** sınıfları Java'da zamanla ilgili işlemler için temel sınıflardır
- Zamanın akışı içerisinde **herhangi bir spesifik anı** ifade etmek için Date sınıfı kullanılır
- **Milisaniye** düzeyinde detay verir
- **Calendar** sınıfı ise date ve time alanları arasında **çevrim** yapmak için kullanılmaktadır

java.util.Date Kullanımı

```
Date now = new Date();

long timeElapsed = now.getTime();

Date then = new Date(timeElapsed);

System.out.println(now.equals(then));

then.setTime(1234567);

System.out.println(now.compareTo(then));
System.out.println(now.before(then));
System.out.println(now.after(then));
```

1 Ocak 1970 00:00:00 GMT'den
Bu yana geçen zamanı
Milisaniye cinsinden döner

```
true
1
false
true
```

java.util.Calendar Kullanımı

- Günümüzde dünyanın büyük bir bölümü **Gregorian takvimi**ni kullanmaktadır
- Java'da **GregorianCalendar** implemantasyonu ile bunu destekler
- Uzakdoğu takvimleri de Calendar üzerinden desteklenmektedir

java.util.Calendar Kullanımı

```
Calendar calendar = Calendar.getInstance();

Calendar calendar2 = Calendar.getInstance(
        TimeZone.getDefault(),Locale.getDefault());

calendar.setTime(new Date());

calendar2.set(Calendar.YEAR, 2015);
calendar2.set(Calendar.MONTH, 5);
calendar2.set(Calendar.DAY_OF_MONTH, 25);

Date time = calendar.getTime();

System.out.println(time);
System.out.println(calendar2.getTime());
```

java.util.TimeZone

- java.util.TimeZone sınıfı ile zaman dilimi, dün ışığından yararlanma ile ilgili hesaplamalar vs yapılabilir
- Sisteme ait varsayılan TimeZone nesnesi elde edilerek çalışılabilir,
- Ya da time zone ID'si ile spesifik bir TimeZone nesnesi elde edilebilir

java.util.TimeZone Kullanımı

```
TimeZone defaultTZ = TimeZone.getDefault();

TimeZone tzIst = TimeZone.getTimeZone("Europe/Istanbul");

TimeZone tz = TimeZone.getTimeZone("GMT+08:00");

boolean useDaylightTime = tzIst.useDaylightTime();

boolean inDaylightTime = tzIst.inDaylightTime(new Date());

System.out.println(useDaylightTime);
System.out.println(inDaylightTime);
```

```
true
true
```

Java 8 ve Neden Yeni Bir API?

- Mevcut **java.util.Date** ve **java.util.Time** sınıflarındaki bir takım **yetersizlikler**
  - Thread-safe olmamaları
  - Zayıf bir API tasarımına sahip olmaları
- **ISO-8601 standartı** (Gregorian takvim sistemi) dışında takvimlerin de desteklenme ihtiyacı
- **Joda Time** gibi harici kütüphanelerin sunduğu kabiliyetlerin JDK'ya dahil edilmesi gibi nedenlerden yeni bir Date & Time API'si JDK'ya eklenmiştir

Yeni API'nin Temel Özellikleri

- Nesnelerin **salt-okunur (immutable)** olması ve bu şekilde thread-safety'nin sağlanması
- Domain driven design yaklaşımı ile yeni sınıfların daha **açık ve anlaşılır bir yapı**da olması
- **ISO-8601 dışında** kalan Japonya, Tayland gibi bölgelerde kullanılan **takvim sistemleri** ile de daha kolay çalışmayı sağlaması
- **Extend edilebilir** olması (örneğin yeni bir takvim sistemi tanımlamak gibi)

Yeni Date Time API ile Çalışmak

JAVA Eğitimleri

- Bilgisayar sitemlerinde zaman iki farklı biçimde ele alınabilir
- **İnsana odaklı**
  - Genellikle yıl, ay, gün, saat, dakika, saniye biçiminde ifade edilir
- **Makinaya odaklı**
  - Belirli bir başlangıç noktasından (epoch) itibaren **nanosecond düzeyinde** süreklilik arz eden bir bilgi şeklinde ifade edilir
- Date Time API her **iki yaklaşım için ayrı ayrı sınıflar** sunmaktadır

Yeni Date Time API ile Çalışmak

- Öncelikle zamanın **hangi biçimde** (insan odaklı mı, makine odaklı mı?) ele alınacağına karar verilmelidir
- Daha sonra **time zone bilgisine ihtiyaç duyulup duyulmadığı** netleştirilmelidir
- Date ve Time'a birlikte mi ihityaç duyuluyor, ya da sadece Date yeterli olur mu?
- Gün, ay, yıl birlikte mi isteniyor, yoksa bunların sadece bir bölümü mü?

LocalDate & LocalTime

- **LocalDate**
  - Time zone bilgisi olmadan yıl, ay, gün içeren immutable bir nesnedir
- **LocalTime**
  - Time zone bilgisi olmadan saat, dakika, saniye, nanosaniye bilgisini ifade eden immutable bir nesnedir
- **LocalDateTime**
  - LocalDate ve LocalTime'ın bileşkesidir

LocalDate & LocalTime

```
LocalDate today = LocalDate.now();
System.out.println("Today's local date :" + today);

LocalTime now = LocalTime.now();
System.out.println("Time is now :" + now);

LocalDateTime localDateTime = LocalDateTime.now();
System.out.println("Today's local date & time :" + localDateTime);
```

```
Today's Local date          : 2016-02-24
Time is now                 : 14:00:05.001
Today's local date & time   : 2016-02-24T14:00:05.001
```

LocalDate & LocalTime

```
LocalDate date = LocalDate.of(2015, Month.AUGUST, 15);
System.out.println("Specific local date :" + date);

LocalTime time = LocalTime.of(12, 30, 45);
System.out.println("Specific time :" + time);

LocalDateTime localDateTime = LocalDateTime.of(date, time);
System.out.println("Date & time :" + localDateTime);
```

```
Specific local date :2015-08-15
Specific time :12:30:45
Date & time :2015-08-15T12:30:45
```

Time Zone Kavramı

- Dünya üzerinde aynı zaman değerini kullanan **belirli bir bölge**ye time zone adı verilir
- Her bir **time zone bir ID**'ye sahiptir
- Bölge/Şehir ve Greenwich/UTC zamanına göre bir offset değeri ile gösterime sahiptir
  - Europe/Istanbul +02:00
- Time zone ve offset bilgilerini ifade etmek için **ZoneId** ve **ZoneOffset** sınıfları kullanılır

Time Zone Sınıfları

- **ZonedDateTime**
  - LocalDateTime ve ZoneId sınıflarının bileşimidir
  - Yıl,ay,gün,saat,dakika,saniye,nanosaniye + zone id bilgisi tutan immutable bir nesnedir
- **OffsetDateTime**
  - LocalDateTime ve ZoneOffset sınıflarının bileşimidir
  - Yıl,ay,gün,saat,dakika,saniye,nanosaniye + zone offset bilgisi tutan immutable bir nesnedir
- **OffsetTime**
  - LocalTime ve ZoneOffset sınıflarının bileşimidir

Time Zone Örnekleri

```
ZoneId amerikaZoneId = ZoneId.of("America/New_York");

ZoneOffset amerikaOffset = ZoneOffset.of("-05:00");

LocalDateTime localDateTime = LocalDateTime.now();

ZonedDateTime zonedDateTime  = ZonedDateTime.of(
                                    localDateTime, amerikaZoneId );

OffsetDateTime offsetDateTime = OffsetDateTime.of(
                                    localDateTime, amerikaOffset);

System.out.println("Current date/time here :" + localDateTime);
System.out.println("Current date/time in New York :" + zonedDateTime);
System.out.println("Current date/time in New York :" + offsetDateTime);
```

+18:00 ile -18:00 aralığında değer girilebilir

```
Current date/time here :2016-02-24T18:00:38.033
Current date/time in New York :2016-02-24T18:00:38.033-05:00[America/New_York]
Current date/time in New York :2016-02-24T18:00:38.033-05:00
```

Instant

- Bir zaman doğrusunda **nano saniye düzeyinde belirli bir başlangıç noktası**nı ifade eder
- **Makina zamanı**nı ifade eden bir **timestamp** değeri oluşturmak için kullanılır
- Bu başlangıç noktası 1 Ocak 1970 00:00:00 zamanıdır (**epoch date**)
- **Epoch date öncesi** bir Instant değeri **negatif**, **sonrası** ise **pozitiftir**

Instant

```
Instant timestamp = Instant.now();

long numberOfSeconds = timestamp.getEpochSecond();

ZoneId amerikaZoneId = ZoneId.of("America/New_York");

ZonedDateTime zonedDateTime = timestamp.atZone(amerikaZoneId);

System.out.println("Current timestamp :" + timestamp);
System.out.println("Number of secs since 01/01/1970 :" + numberOfSeconds);
System.out.println("Current date/time in New York :" + zonedDateTime);
```

```
Current timestamp :2016-02-24T15:20:05.016Z
Number of secs since 01/01/1970 :1487949605
Current date/time in New York :2016-02-24T10:20:05.016-05:00[America/New_York]
```

Period & Duration

- **Period**
  - Zamanı miktar olarak saniye veya nano saniye şeklinde ifade etmeyi sağlar
- **Duration**
  - Zamanı miktar olarak yıl, ay veya gün şeklinde ifade etmeyi sağlar
- **ChronoUnit**
  - Zamanı ölçmek için değişik **TemporalUnit**'ler tanımlar

Period & Duration

```
LocalDate startDateInclusive = LocalDate.of(2005, Month.DECEMBER, 16);
LocalDate endDateExclusive = LocalDate.of(2017, Month.FEBRUARY, 24);

Period period = Period.between(startDateInclusive,
                               endDateExclusive);

LocalTime startTimeInclusive = LocalTime.of(13, 10, 30);
LocalTime startTimeExclusive = LocalTime.of(23, 10, 45);

Duration duration = Duration.between(startTimeInclusive,
                                     startTimeExclusive);

System.out.printf("Period years :%d months :%d days :%d\n",
            period.getYears(), period.getMonths(), period.getDays());

System.out.printf("Duration seconds: %d\n",duration.getSeconds());
```

```
Period years :11 months :2 days :8
Duration seconds: 36015
```

Clock

- **Sistem saati**ni ifade etmeyi sağlar
- Temporal nesnelerin hemen hepsi **now()** şeklinde şu andaki zamanı dönen metoda sahiptir
- Ayrıca **now(Clock)** şeklinde bir metot da vardır
- **Clock nesnesi sayesinde** uygulama içerisinde **doğru time zone da date/time bilgisi üretmek** mümkün hale gelir
- Uygulama kodunun **farklı time zone'larla test** edilmesini de kolaylaştırır

Clock

```
Clock clock = Clock.systemUTC();

System.out.println("DateTime for Clock UTC: " +
                                        LocalDateTime.now(clock));

Clock defaultClock = Clock.systemDefaultZone();

System.out.println("DateTime for Clock System DefaultZone: " +
                                    LocalDateTime.now(defaultClock));
```

```
Clock UTC: 2017-02-24T16:17:48.035
Clock System DefaultZone: 2017-02-24T19:17:48.117
```

Diğer Yardımcı Sınıflar

- **DayOfWeek**
  - Haftanın günlerini belirten bir enum'dur
  - MONDAY-SUNDAY (1-7) arasında enum değerler sunar
- **Month**
  - Yılın aylarını belirten bir enum'dur
  - JANUARY-DECEMBER (1-12) arasında enum değerler sunar

Diğer Yardımcı Sınıflar

- **MonthDay**
  - Ay ve ayın günü bilgilerini birlikte barındıran immutable bir nesnedir
  - Yıl, zaman veya time zone bilgisi içermez
  - Dolayısı ile Şubat 29 geçerli bir değerdir
- **YearMonth**
  - Yıl ve ay bilgilerini birlikte barındıran immutable bir nesnedir
  - Gün, zaman ve time zone bilgisi içermez

Diğer Yardımcı Sınıflar

- **Year**
  - Yıl bilgisini barındıran immutable bir nesnedir
  - ISO-8601/Gregorian takvim sistemine uygun yıl değerlerini ifade eder
  - Ay, gün, zaman veya time zone bilgisi içermez
  - 0. yıl, 1. yıldan, -1. yıl ise 0. yıldan önce gelir

Parse & Format

- Date & Time API sınıfları zaman bilgisi içeren **String değerleri parse edecek** metotlar içerir
- Ayrıca bu sınıflar mevcut değerlerini **gösterim amaçlı formatlayacak** metotlarda içerir

Parse & Format


```
String formattedLocalDate = "20160224";
String formattedLocalDateTime = "2016-02-24T14:00:05";
String formattedDate = "Feb 15 1976";

LocalDate localDate = LocalDate.parse(formattedLocalDate,
                                      DateTimeFormatter.BASIC_ISO_DATE);
LocalDateTime localDateTime = LocalDateTime.parse(formattedLocalDateTime,
                                      DateTimeFormatter.ISO_DATE_TIME);
LocalDate customDate = LocalDate.parse(formattedDate,
                                DateTimeFormatter.ofPattern("MMM dd yyyy"));

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");
String formattedDate2 = customDate.format(formatter);

System.out.println("LocalDate :" + localDate);
System.out.println("LocalDateTime :" + localDateTime);
System.out.println("Custom Date :" + customDate);
System.out.println("Custom Date formatted :" + formattedDate2);
```

```
LocalDate :2016-02-24
LocalDateTime :2016-02-24T14:00:05
Custom Date :1976-02-15
Custom Date formatted :15/02/1976
```

Eski Date/Time Sınıfları ile Dönüşüm

- Java 8 öncesinde Java'daki date/time kabiliyeti aşağıdaki sınıflar ile sağlanmaktaydı
  - java.util.Date
  - java.util.Time
  - java.util.TimeZone
  - java.util.GregorianCalendar
- Bu sınıflara Java 8 date/time API sınıfları arasında dönüşüm yapacak metotlar eklenmiştir

Eski Date/Time Sınıfları ile Dönüşüm

```
Date legacyDate = new java.util.Date();
java.sql.Date legacySqlDate = new java.sql.Date(legacyDate.getTime());
Timestamp legacyTimestamp = new java.sql.Timestamp(legacyDate.getTime());

Instant instantForLegacyDate = legacyDate.toInstant();
LocalDateTime localDateTime = LocalDateTime.ofInstant(instantForLegacyDate,
ZoneOffset.UTC);
LocalDate localDateForLegacySqlDate = legacySqlDate.toLocalDate();
LocalDateTime localDateTimeForLegacyTS = legacyTimestamp.toLocalDateTime();

System.out.println("LocalDateTime :" + localDateTime);
System.out.println("LocalDate from legacySqlDate :" + localDateForLegacySqlDate);
System.out.println("LocalDateTime from legacyTS :" + localDateTimeForLegacyTS);

System.out.println("Legacy date via Date.from :" + Date.from(instantForLegacyDate));
System.out.println("Legacy date via Time.from :" + Time.from(instantForLegacyDate));
System.out.println("Legacy date via Timestamp.from :" +
                                        Timestamp.from(instantForLegacyDate));
```

```
LocalDateTime :2017-02-24T17:06:57.020
LocalDate from legacySqlDate :2017-02-24
LocalDateTime from legacyTS :2017-02-24T20:06:57.020
Legacy date via Date.from :Fri Feb 24 20:06:57 EET 2017
Legacy date via Time.from :Fri Feb 24 20:06:57 EET 2017
Legacy date via Timestamp.from :2017-02-24 20:06:57.02
```

Date Time API'sindeki Sınıfların İçeriği ve Gösterimi

JAVA Eğitimleri

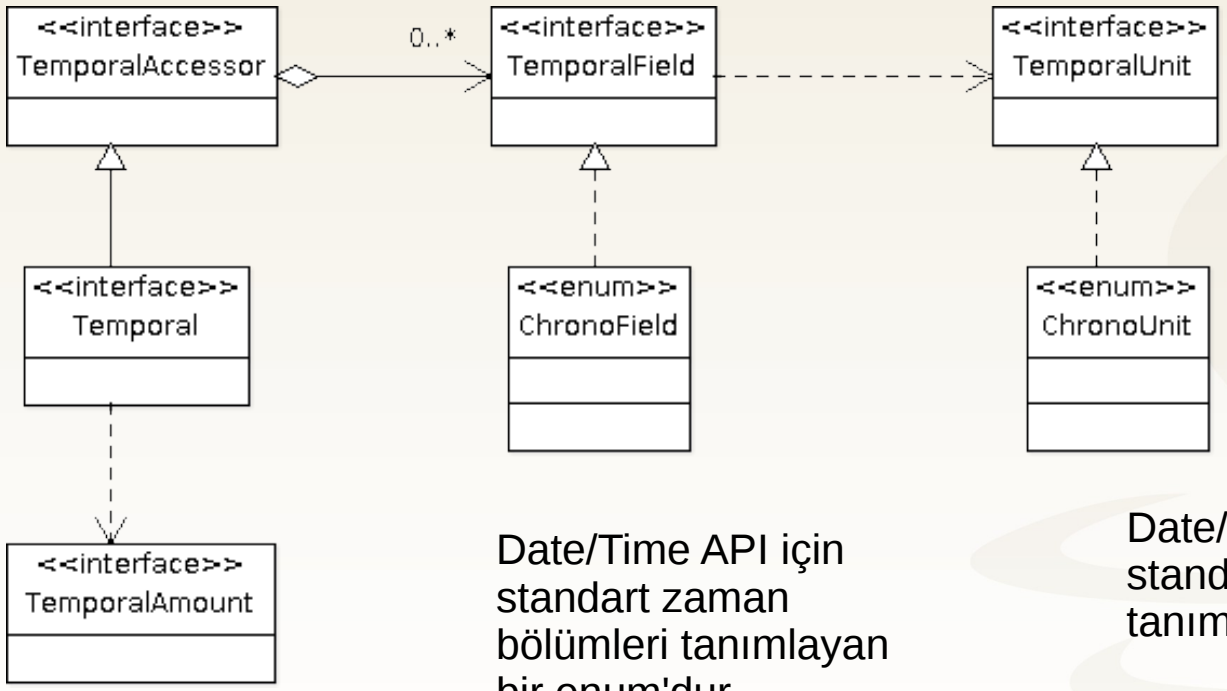
Class/Enum	Yıl	Ay	Gün	Saat	Dk	Sn (nano)	Zone Offset	ZoneID	String Gösterim
Instant						x			2013-08-20T15:16:26.355Z
LocalDate	x	x	x						2013-08-20
LocalDateTime	x	x	x	x	x	x			2013-08-20T08:16:26.937
ZonedDateTime	x	x	x	x	x	x	x	x	2013-08-21T00:16:26.941+09:00[Asia/Tokyo]
LocalTime				x	x	x			08:16:26.943
DayOfWeek			x						MONDAY
MonthDay		x	x						--08-20
Year	x								2013
YearMonth	x	x							2013-08
Month		x							AUGUST
OffsetDateTime	x	x	x	x	x	x	x		2013-08-20T08:16:26.954-07:00
OffsetTime				x	x	x	x		08:16:26.957-07:00
Duration						x			PT20H (20 hours)
Period	x	x	x						P10D (10 days)

Date/Time API ve Temporal Framework

Temporal nesneler için salt okunur bir arayüz tanımlar

Tarih/saat ifadelerindeki belirli bir bölüme karşılık gelir

Zaman bölümlerinin birimlerini ifade eder



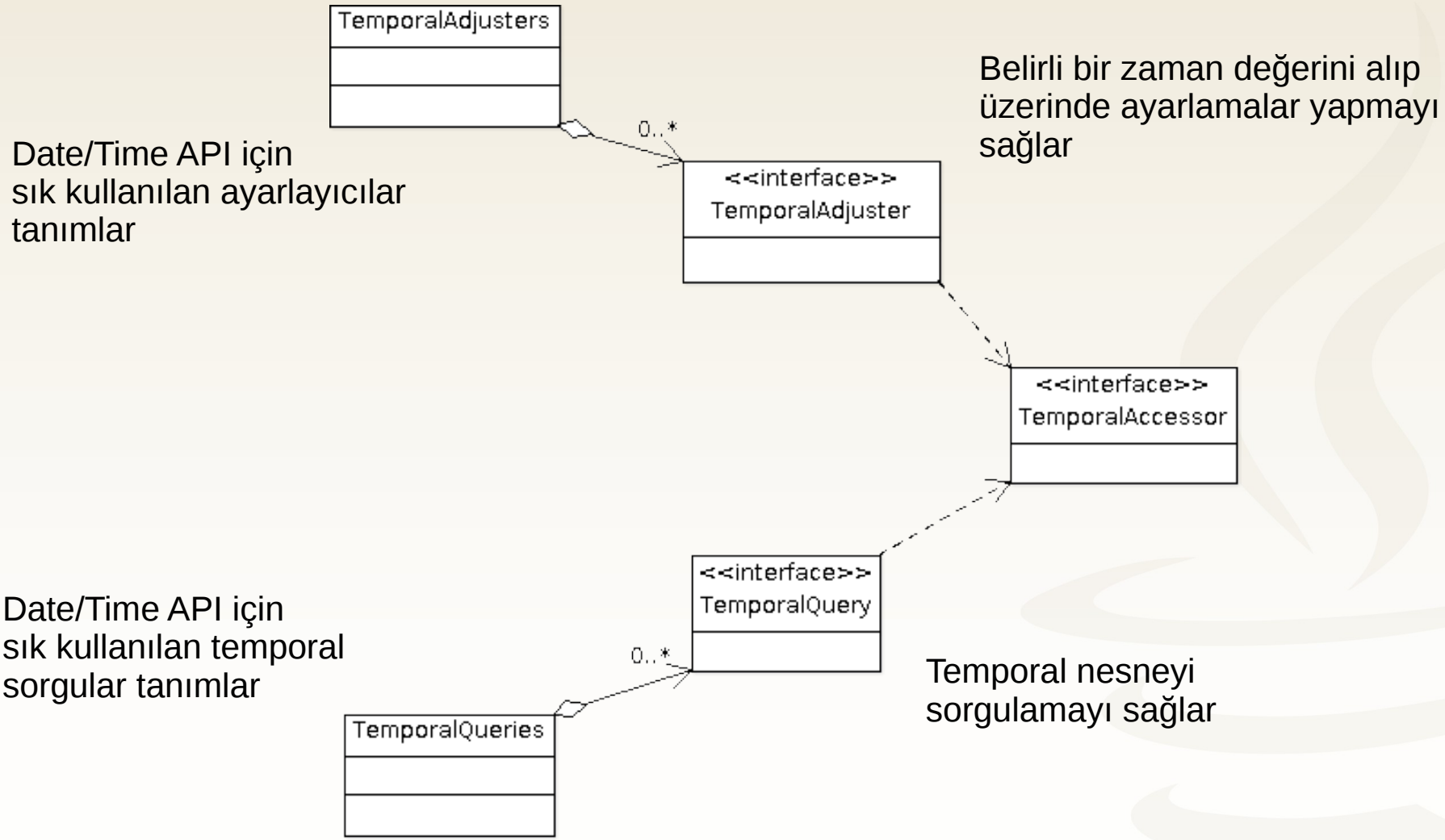
Temporal nesneler üzerinde zaman bazlı aritmetik işlemler yapmayı sağlar

Date/Time API için standart zaman bölümleri tanımlayan bir enum'dur

Date/Time API için standart zaman birimleri tanımlayan bir enum'dur

Zamana karşılık bir miktar tanımlar

Date/Time API ve Temporal Framework



Date/Time API için sık kullanılan ayarlayıcılar tanımlar

Belirli bir zaman değerini alıp üzerinde ayarlamalar yapmayı sağlar

Date/Time API için sık kullanılan temporal sorgular tanımlar

Temporal nesneyi sorgulamayı sağlar

İletişim

www.harezmi.com.tr

www.java-egitimleri.com

info@harezmi.com.tr

info@java-egitimleri.com

@HarezmiBilisim

@JavaEgitimleri