
Pengenalan Date & Time API



Sebelum Date dan Time API

- Sejak awal, di Java representasi tipe data tanggal dan waktu adalah `java.util.Date` dan `java.util.Calendar`
- Java Date dan Time API merupakan fitur baru sejak Java versi 8

Masalah Dengan Class Date dan Calendar

- Thread Safety, class Date dan Calendar tidak thread safe, dalam artian berbahaya jika diakses secara paralel (beberapa proses mengakses object yang sama).
- Desain class Date dan Calendar tidak terlalu bagus di desain dari awal, hal ini menyulitkan ketika kita butuh melakukan operasi yang melibatkan tanggal dan waktu, seperti mencari durasi waktu, periode, menambah waktu, dan operasi lainnya.
- Desain tanggal dan waktu dengan timezone agak menyulitkan di class Date dan Calendar.

Package `java.time`

- Date & Time API yang baru sekarang berada dalam satu package, yaitu package `java.time`
- Sekarang ada banyak sekali class-class yang terdapat di package tersebut, dan kegunaannya berbeda-beda, yang akan kita bahas secara detail di chapter-chapter selanjutnya
- <https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/time/package-summary.html>

Perbedaan dengan Date dan Time API Baru

- Object di Date dan Time API baru bersifat immutable dan thread safe, artinya tidak bisa diubah, jika diubah, itu akan membuat object yang baru sehingga aman digunakan para proses paralel
- Terdapat pemisahan antara Tanggal dan Waktu, tidak digabung seperti pada class Date
- Mendukung data lain seperti durasi, periode dan lain-lain

Date



Date

- Sebelum membahas Date dan Time API yang baru, kita akan sekilas membahas class Date
- Date adalah representasi tanggal dan juga waktu di Java sejak versi awal
- Karena tidak ada pemisahan antara tanggal dan waktu di class Date, ini memang agak menyulitkan jika kita hanya butuh misal tanggal saja, dan waktu saja
- <https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/Date.html>



Date Constructor

Constructors

Constructor	Description
<code>Date()</code>	Allocates a Date object and initializes it so that it represents the time at which it was allocated, measured to the nearest millisecond.
<code>Date(int year, int month, int date)</code>	Deprecated. As of JDK version 1.1, replaced by <code>Calendar.set(year + 1900, month, date)</code> or <code>GregorianCalendar(year + 1900, month, date)</code> .
<code>Date(int year, int month, int date, int hrs, int min)</code>	Deprecated. As of JDK version 1.1, replaced by <code>Calendar.set(year + 1900, month, date, hrs, min)</code> or <code>GregorianCalendar(year + 1900, month, date, hrs, min)</code> .
<code>Date(int year, int month, int date, int hrs, int min, int sec)</code>	Deprecated. As of JDK version 1.1, replaced by <code>Calendar.set(year + 1900, month, date, hrs, min, sec)</code> or <code>GregorianCalendar(year + 1900, month, date, hrs, min, sec)</code> .
<code>Date(long date)</code>	Allocates a Date object and initializes it to represent the specified number of milliseconds since the standard base time known as "the epoch", namely January 1, 1970, 00:00:00 GMT.
<code>Date(String s)</code>	Deprecated. As of JDK version 1.1, replaced by <code>DateFormat.parse(String s)</code> .



Millisecond

- Date menggunakan milisecond setelah UNIX epoch (January 1, 1970 00:00:00 UTC)
- Untuk mendapatkan milisecond saat ini, di Java kita bisa menggunakan System.currentTimeMillis()
- <https://currentmillis.com/>



Kode : Membuat Date

```
var date1 = new Date();
var date2 = new Date(System.currentTimeMillis());

System.out.println(date1);
System.out.println(date2);
```

Calendar

Calendar

- Class Calendar adalah class yang digunakan sebagai pembantu class Date
- Pada class Date, hampir semua method untuk melakukan manipulasi tanggal dan waktu sudah ditandai sebagai @Deprecated, artinya tidak direkomendasikan digunakan lagi
- Untuk itu, kita butuh menggunakan class Calendar untuk memanipulasi tanggal dan waktu
- Class Calendar tidak memiliki public constructor, sehingga untuk membuat object Calendar, kita akan menggunakan static method milik calendar bernama getInstance()

Kode : Membuat Calendar

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

// Membuat Date dari Calendar
Date date = calendar.getTime();
```

Manipulasi Tanggal dan Waktu

- Salah satu fitur yang terdapat di Calendar adalah, kita bisa melakukan manipulasi tanggal dan waktu di object Calendar menggunakan method set(type, value)
- Dan untuk mengambil value tanggal atau waktu, kita bisa menggunakan method get(type)

Kode : Mengubah Data Calendar

```
Calendar calendar = Calendar.getInstance();
calendar.set(Calendar.YEAR, 1980);
calendar.set(Calendar.MONTH, Calendar.FEBRUARY);
calendar.set(Calendar.DAY_OF_MONTH, 10);

Date date = calendar.getTime();
```



Kode : Mengambil Data Calendar

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

int year = calendar.get(Calendar.YEAR);
int month = calendar.get(Calendar.MONTH);
int dayOfMonth = calendar.get(Calendar.DAY_OF_MONTH);
int hour = calendar.get(Calendar.HOUR_OF_DAY);
int minute = calendar.get(Calendar.MINUTE);
int second = calendar.get(Calendar.SECOND);
int millisecond = calendar.get(Calendar.MILLISECOND);
```

TimeZone

TimeZone

- Class TimeZone merupakan representasi dari data time zone
- Secara default, jika kita membuat object Calendar, dan tidak menggunakan TimeZone, secara otomatis objectnya akan menggunakan default TimeZone, yaitu timezone sistem operasi yang kita gunakan.
- Untuk mengetahui default time zone, kita bisa gunakan method TimeZone.getDefault()
- Sedangkan jika ingin membuat object TimeZone, kita bisa gunakan method TimeZone.getTimeZone("Zone ID")
- Untuk mengetahui semua zone id yang didukung oleh Java, kita bisa gunakan TimeZone.getAvailableIDs()



Kode : Membuat TimeZone

```
TimeZone timeZone1 = TimeZone.getDefault();
TimeZone timeZone2 = TimeZone.getTimeZone("Asia/Jakarta");

String[] availableIDs = TimeZone.getAvailableIDs();
```

TimeZone di Date

- Class Date tidak memiliki method apapun untuk mendapatkan time zone atau mengubah time zone
- Secara default, saat kita membuat object Date, object tersebut akan menggunakan time zone default

Kode : TimeZone di Date

```
Date date = new Date();
System.out.println(date.toString());
System.out.println(date.toGMTString());
```

TimeZone di Calendar

- Berbeda dengan Date, di Calendar, informasi TimeZone yang bisa kita ubah
- Untuk mengubah time zone di Calendar, kita bisa menggunakan method setTimeZone()



Kode : TimeZone di Calendar

```
Calendar calendar1 = Calendar.getInstance(); // default timezone  
Calendar calendar2 = Calendar.getInstance(TimeZone.getTimeZone("GMT"));  
  
System.out.println(calendar1.get(Calendar.HOUR_OF_DAY));  
System.out.println(calendar2.get(Calendar.HOUR_OF_DAY));  
  
calendar1.setTimeZone(TimeZone.getTimeZone("GMT"));  
System.out.println(calendar1.get(Calendar.HOUR_OF_DAY));
```

LocalDate



LocalDate

- LocalDate adalah class di Date & Time API baru
- LocalDate merupakan representasi untuk tipe data tanggal (tanpa waktu)
- Default format waktu untuk LocalDate adalah yyyy-MM-dd



Kode : Membuat LocalDate

```
LocalDate localDate1 = LocalDate.now();
LocalDate localDate2 = LocalDate.of( year: 1980, Month.JUNE, dayOfMonth: 10);
LocalDate localDate3 = LocalDate.parse("1980-06-10");

System.out.println(localDate1);
System.out.println(localDate2);
System.out.println(localDate3);
```

Mengubah LocalDate

- Object LocalDate juga bisa kita ubah tanggal nya jika kita mau
- Untuk mengubah tanggal, kita bisa menggunakan method with, seperti withYear, withMonth, dan lain-lain
- Perlu diingat, mengubah LocalDate akan menciptakan object LocalDate baru, artinya object aslinya tidak akan berubah, karena bersifat immutable



Kode : Mengubah LocalDate

```
LocalDate localDate1 = LocalDate.now();
LocalDate localDate2 = localDate1.withYear(2000);
LocalDate localDate3 = localDate1.withMonth(1);

System.out.println(localDate1);
System.out.println(localDate2);
System.out.println(localDate3);
```

Manipulasi LocalDate

- Object LocalDate juga bisa kita manipulasi, seperti menambah tanggal atau mengurangi tanggal.
- Untuk menambah tanggal, kita bisa gunakan method plus, seperti plusYears, plusMonths, plusDays, dan lain-lain
- Untuk mengurangi tanggal, kita bisa gunakan method minus, seperti minusYears, minusMonths, minusDays, dan lain-lain
- Perlu diingat, manipulasi LocalDate akan menciptakan object LocalDate baru, artinya object aslinya tidak akan berubah, karena bersifat immutable



Kode : Manipulasi LocalDate

```
LocalDate localDate1 = LocalDate.now();
LocalDate localDate2 = localDate1.plusYears(10);
LocalDate localDate3 = localDate1.minusMonths(10);

System.out.println(localDate1);
System.out.println(localDate2);
System.out.println(localDate3);
```

Mengambil Tanggal LocalDate

- LocalDate juga mendukung pengambilan detail data tanggal nya, seperti tahun, bulan, hari dalam bulan, hari dalam tahun, dan lain-lain
- Untuk mengambil data tanggal, kita bisa menggunakan method get, seperti getYear, getMonth, dan lain-lain



Kode : Mengambil Tanggal LocalDate

```
LocalDate localDate = LocalDate.now();  
  
int year = localDate.getYear();  
Month month = localDate.getMonth();  
int monthValue = localDate.getMonthValue();  
int dayOfMonth = localDate.getDayOfMonth();  
DayOfWeek dayOfWeek = localDate.getDayOfWeek();  
int dayOfYear = localDate.getDayOfYear();
```

LocalTime

LocalTime

- Berbeda dengan LocalDate, LocalTime merupakan representasi data waktu (tanpa tanggal)
- Cara penggunaanya dan method-method nya hampir sama dengan LocalDate
- LocalTime juga immutable, jadi kita tidak bisa mengubahnya setelah datanya dibuat, jika kita ubah, maka akan menghasilkan object LocalTime baru
- Format standard LocalTime adalah HH:mm:ss.nano, dimana second dan nano second nya optional



Kode : Membuat LocalTime

```
LocalTime localTime1 = LocalTime.now();
LocalTime localTime2 = LocalTime.of( hour: 10, minute: 10, second: 10);
LocalTime localTime3 = LocalTime.parse("10:10");

System.out.println(localTime1);
System.out.println(localTime2);
System.out.println(localTime3);
```



Kode : Mengubah LocalTime

```
LocalTime localTime1 = LocalTime.now();
LocalTime localTime2 = localTime1.withHour(10);
LocalTime localTime3 = localTime1.withHour(10).withMinute(10);

System.out.println(localTime1);
System.out.println(localTime2);
System.out.println(localTime3);
```



Kode : Manipulasi LocalTime

```
LocalTime localTime1 = LocalTime.now();
LocalTime localTime2 = localTime1.plusHours(10);
LocalTime localTime3 = localTime1.minusHours(5).plusMinutes(5);

System.out.println(localTime1);
System.out.println(localTime2);
System.out.println(localTime3);
```

Kode : Mengambil Waktu LocalTime

```
LocalTime localTime = LocalTime.now();  
  
int hour = localTime.getHour();  
int minute = localTime.getMinute();  
int second = localTime.getSecond();  
int nano = localTime.getNano();
```

LocalDateTime

LocalDateTime

- Seperti dari nama class nya, LocalDateTime, class ini digunakan sebagai representasi tanggal dan waktu, mirip seperti class Date
- LocalDateTime juga immutable, jadi tidak bisa diubah setelah dibuat, jika diubah otomatis akan membuat object baru
- Cara pembuatan, cara mengubah, memanipulasi nya pun sama seperti LocalDate dan LocalTime
- Dan format default untuk LocalDateTime adalah menggunakan format yyyy-MM-ddTHH:mm:ss.nano



Kode : Membuat LocalDateTime

```
LocalDateTime localDateTime1 = LocalDateTime.now();
LocalDateTime localDateTime2 = LocalDateTime.of( year: 2020, Month.DECEMBER, dayOfMonth: 10, hour: 10, minute: 10);
LocalDateTime localDateTime3 = LocalDateTime.parse("2020-12-10T11:12:13.14444");

System.out.println(localDateTime1);
System.out.println(localDateTime2);
System.out.println(localDateTime3);
```



Kode : Mengubah LocalDateTime

```
LocalDateTime localDateTime1 = LocalDateTime.now();
LocalDateTime localDateTime2 = localDateTime1.withYear(1990);
LocalDateTime localDateTime3 = localDateTime1.withMinute(40);

System.out.println(localDateTime1);
System.out.println(localDateTime2);
System.out.println(localDateTime3);
```



Kode : Manipulasi LocalDateTime

```
LocalDateTime localDateTime1 = LocalDateTime.now();
LocalDateTime localDateTime2 = localDateTime1.plusYears(1990);
LocalDateTime localDateTime3 = localDateTime1.plusMinutes(40);

System.out.println(localDateTime1);
System.out.println(localDateTime2);
System.out.println(localDateTime3);
```



Kode : Mengambil Data LocalDateTime

```
LocalDateTime localDateTime = LocalDateTime.now();  
  
int year = localDateTime.getYear();  
Month month = localDateTime.getMonth();  
int dayOfMonth = localDateTime.getDayOfMonth();  
int hour = localDateTime.getHour();  
int minute = localDateTime.getMinute();  
int second = localDateTime.getSecond();  
int nano = localDateTime.getNano();
```

Konversi dari dan ke LocalDate

- Kadang ada kebutuhan kita melakukan konversi data dari LocalDate ke LocalDateTime atau juga sebaliknya
- Untuk melakukan konversi dari LocalDateTime ke LocalDate, kita bisa menggunakan method `toLocalDate()`
- Sedangkan untuk melakukan konversi dari LocalDate ke LocalDateTime, kita bisa menggunakan method `atTime()`



Kode : Konversi ke dan dari LocalDate

```
LocalDateTime localDateTime = LocalDateTime.now();  
  
LocalDate localDate = localDateTime.toLocalDate();  
LocalDateTime localDateTimeBack = localDate.atTime(LocalTime.now());
```

Konversi dari dan ke LocalTime

- Selain melakukan konversi untuk tipe data LocalDate, kita juga bisa lakukan ke dan dari tipe data LocalTime.
- Untuk melakukan konversi dari LocalDateTime ke LocalTime, kita bisa menggunakan method toLocalTime()
- Sedangkan untuk melakukan konversi dari LocalTime ke LocalDateTime, kita bisa menggunakan method atDate()



Kode : Konversi dari dan ke LocalTime

```
LocalDateTime localDateTime = LocalDateTime.now();  
  
LocalTime localTime = localDateTime.toLocalTime();  
LocalDateTime localDateTimeBack = localTime.atDate(LocalDate.now());
```

Year, YearMonth dan MonthDay

Year, YearMonth dan MonthDay

- Di Date & Time API baru, terdapat class Year , YearMonth dan MonthDay
- Seperti dari nama class nya, Year digunakan untuk tanggal yang hanya berisi data tahun, dan YearMonth adalah tanggal yang berisi data tahun dan bulan, dan MonthDay adalah tanggal berisi bulan dan hari
- Kenapa menggunakan Year? Kenapa tidak langsung menggunakan angka saja? Misal 2020. Di Year, sudah banyak sekali method yang bisa kita gunakan untuk manipulasi data tanggal dan konversi ke tipe lain seperti LocalDate misal nya
- Begitu juga dengan YearMonth dan MonthDay
- Format default untuk Year adalah yyyy dan format untuk YearMonth adalah yyyy-MM dan format untuk MonthDay adalah --MM-dd



Kode : Membuat Year dan YearMonth

```
Year year1 = Year.now();
Year year2 = Year.of(1980);
Year year3 = Year.parse("2020");

YearMonth yearMonth1 = YearMonth.now();
YearMonth yearMonth2 = YearMonth.of( year: 2020, Month.DECEMBER);
YearMonth yearMonth3 = YearMonth.parse("2020-10");

MonthDay monthDay1 = MonthDay.of(Month.JUNE,   dayOfMonth: 10);
MonthDay monthDay2 = MonthDay.now();
MonthDay monthDay3 = MonthDay.parse("--10-10");
```

Kode : Konversi dari Year ke LocalDate

```
Year year = Year.of(1980);
YearMonth yearMonth = year.atMonth(Month.DECEMBER);
LocalDate localDate = yearMonth.atDay(dayOfMonth: 30);
MonthDay monthDay = MonthDay.from(localDate);
```

Kode : Mengambil Data Year dan YearMonth

```
Year year = Year.now();
int value = year.getValue();

YearMonth yearMonth = YearMonth.now();
int year1 = yearMonth.getYear();
Month month = yearMonth.getMonth();
int monthValue = yearMonth.getMonthValue();
```

Zoneld dan ZoneOffset

Zoneld dan ZoneOffset

- Sebelumnya kita sudah tahu bahwa terdapat class TimeZone di Java sebagai representasi time zone
- Namun di Java Date & Time API terbaru, terdapat class baru untuk mendukung time zone, yaitu Zoneld dan ZoneOffset

Zoneld

- Zoneld mirip dengan TimeZone, dimana ini merupakan representasi time zone id
- Untuk mendapatkan default time zone, kita bisa menggunakan method Zoneld.systemDefault()
- Untuk membuat Zoneld, caranya bisa menggunakan method Zoneld.of("Zone Id")
- Dan untuk mendapatkan semua daftar time zone id yang didukung oleh Java, kita bisa menggunakan method

Kode : Membuat Zoneld

```
ZoneId zoneId = ZoneId.systemDefault();
ZoneId zoneIdGMT = ZoneId.of("GMT");

Set<String> availableZoneIds = ZoneId.getAvailableZoneIds();
```

ZoneOffset

- Jika Zoneld merupakan representasi timezone menggunakan time zone id
- Pada class ZonenOffset, iin adalah format time zone dengan offset dari time zone Greenwich/UTC, misalnya +07:00 atau -01:00
- Untuk membuat ZoneOffset, kita bisa menggunakan method of
- ZoneOffset adalah turunan dari Zoneld, jadi semua parameter yang menerima object Zoneld, bisa kita isi dengan ZoneOffset



Kode : Membuat ZoneOffset

```
ZoneOffset zoneOffset1 = ZoneOffset.of("+07:00");
ZoneOffset zoneOffset2 = ZoneOffset.ofHours(-7);
ZoneOffset zoneOffset3 = ZoneOffset.ofHoursMinutes( hours: 5, minutes: 30);

System.out.println(zoneOffset1);
System.out.println(zoneOffset2);
System.out.println(zoneOffset3);
```

ZonedDateTime

ZonedDateTime

- Pada class LocalDateTime, secara default tidak ada informasi time zone sama sekali pada class tersebut
- Jika kita ingin membuat tanggal dan waktu yang mendukung time zone, kita bisa menggunakan class ZonedDateTime
- Class ZonedDateTime sama seperti LocalDateTime, perbedaannya hanyalah mendukung time zone
- Format default untuk ZonedDateTime adalah yyyy-MM-ddTHH:mm:ss.nano(+/-)ZoneOffset[Zoneld], dimana Zoneld tidak wajib, dan jika Zoneld diisi, maka nilai ZoneOffset akan dihiraukan



Kode : Membuat ZonedDateTime

```
ZonedDateTime zonedDateTime1 = ZonedDateTime.now(); // default zone
ZonedDateTime zonedDateTime2 = ZonedDateTime.now(ZoneId.of("GMT"));
ZonedDateTime zonedDateTime3 = ZonedDateTime.of(LocalDateTime.now(), ZoneOffset.ofHours(7));

System.out.println(zonedDateTime1);
System.out.println(zonedDateTime2);
System.out.println(zonedDateTime3);
```



Kode : Parsing ZonedDateTime

```
ZonedDateTime zonedDateTime1 = ZonedDateTime.parse("2020-10-09T08:07:06+07:00[Asia/Jakarta]");  
ZonedDateTime zonedDateTime2 = ZonedDateTime.parse("2020-10-09T08:07:06+07:00");  
  
System.out.println(zonedDateTime1);  
System.out.println(zonedDateTime2);
```

Mengubah Time Zone

- Untuk mengubah time zone pada ZonedDateTime, terdapat dua cara
- Pertama, mengubah time zone, tanpa merubah tanggal dan waktu. Caranya dengan menggunakan method withZoneSameLocal(Zonelid)
- Kedua, mengubah time zone, sehingga tanggal dan waktu mengikuti timezone yang baru. Caranya dengan menggunakan method withZoneSameInstance(Zonelid)



Kode : Mengubah Time Zone

```
ZonedDateTime zonedDateTime1 = ZonedDateTime.now(); // default zone
ZonedDateTime zonedDateTime2 = zonedDateTime1.withZoneSameInstant(ZoneId.of("GMT"));
ZonedDateTime zonedDateTime3 = zonedDateTime1.withZoneSameLocal(ZoneId.of("GMT"));

System.out.println(zonedDateTime1);
System.out.println(zonedDateTime2);
System.out.println(zonedDateTime3);
```

OffsetTime dan OffsetDateTime



OffsetTime dan OffsetDateTime

- Java Date & Time API memiliki class yang bernama OffsetTime dan OffsetDateTime
- Class OffsetTime adalah seperti LocalTime, namun memiliki time zone offset
- Dan Class OffsetDateTime adalah seperti LocalDateTime, namun memiliki time zone offset
- Sekilas OffsetDateTime mirip dengan ZonedDateTime, yang membedakan adalah kalo OffsetDateTime hanya bisa menggunakan ZoneOffset, tidak bisa menggunakan ZoneId
- Format default OffsetTime adalah HH:mm:ss(+/-)ZoneOffset
- Format default OffsetDateTime adalah yyyy-MM-ddTHH:mm:ss(+/-)ZoneOffset



Kode : Membuat OffsetTime & OffsetDateTime

```
OffsetTime offsetTime1 = OffsetTime.now();
OffsetTime offsetTime2 = OffsetTime.of(LocalTime.now(), ZoneOffset.ofHours(7));

OffsetDateTime offsetDateTime1 = OffsetDateTime.now();
OffsetDateTime offsetDateTime2 = OffsetDateTime.of(LocalDateTime.now(), ZoneOffset.ofHours(7));
```

Konversi ke dan dari Non Offset

- Apa yang bisa kita lakukan di LocalTime dan LocalDateTime bisa juga dilakukan di OffsetTime dan OffsetDateTime
- Selain itu kita juga bisa konversi data dari dan ke non offset



Kode : Konversi ke dan dari Non Offset

```
LocalTime localTime = LocalTime.now();
OffsetTime offsetTime = localTime.atOffset(ZoneOffset.ofHours(7));
LocalTime localTime1 = offsetTime.toLocalTime();

LocalDateTime localDateTime = LocalDateTime.now();
OffsetDateTime offsetDateTime = localDateTime.atOffset(ZoneOffset.ofHours(7));
LocalDateTime localDateTime1 = offsetDateTime.toLocalDateTime();
LocalDate localDate = offsetDateTime.toLocalDate();
LocalTime localTime2 = offsetDateTime.toLocalTime();
```

Instant



Instant

- Sebelumnya, milliseconds direpresentasikan dengan tipe data long, di Java Date & Time API baru, implementasi milliseconds direpresentasikan dalam class Instant
- Instant juga sama seperti class Date, dia menggunakan milliseconds setelah Unix EPOCH 1970-01-01T00:00:00Z
- Artinya secara time zone, Instant menggunakan time zone UTC (00:00)



Kode : Membuat Instant

```
Instant instant1 = Instant.now();
Instant instant2 = Instant.ofEpochMilli(System.currentTimeMillis());

System.out.println(instant1);
System.out.println(instant2);
```



Kode : Mengubah Instant

```
Instant instant1 = Instant.now();
Instant instant2 = instant1.plusMillis(1000);
Instant instant3 = instant1.plusSeconds(1000);
Instant instant4 = instant1.minusMillis(1000);
Instant instant5 = instant1.minusSeconds(1000);
```

Kode : Mengambil Data Instant

```
Instant instant = Instant.now();
long epochMilli = instant.toEpochMilli();
long epochSecond = instant.getEpochSecond();
int nano = instant.getNano();
```

Konversi dari Instant

- Karena Instant berisikan milisecond, jadi kita bisa lakukan konversi ke tipe data lainnya, seperti LocalDate, LocalTime dan ZonedDateTime
- Namun karena Instant menggunakan time zone UTC, jadi kita perlu memberi tahu time zone apa yang akan kita gunakan ketika kita lakukan konversi
- Untuk melakukan konversi dari Instant ke tipe data lainnya, kita bisa menggunakan method ofInstant(instant, zone) ketika membuat object nya



Kode : Konversi dari Instant

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

LocalDateTime localDateTime = LocalDateTime.ofInstant(instant, ZoneId.of("Asia/Jakarta"));
LocalTime localTime = LocalTime.ofInstant(instant, ZoneId.of("Asia/Jakarta"));
ZonedDateTime zonedDateTime = ZonedDateTime.ofInstant(instant, ZoneId.of("Asia/Jakarta"));
```

Konversi ke Instant

- Selain itu konversi juga bisa dilakukan ke Instant, dari tipe data LocalDateTime dan ZonedDateTime
- Namun karena Instant menggunakan time zone UTC, jadi kita perlu menambahkan ZoneOffset jika konversi dari tipe data yang tidak memiliki time zone



Kode : Konversi ke Instant

```
Instant instant1 = LocalDateTime.now().toInstant(ZoneOffset.ofHours(7));  
Instant instant2 = ZonedDateTime.now().toInstant();  
  
System.out.println(instant1);  
System.out.println(instant2);
```

Clock

Clock

- Seperti nama class nya, Clock adalah representasi tanggal dan waktu saat ini mengikuti time zone yang kita pilih
- Best practice nya sebenarnya jika kita ingin menggunakan tipe data di Date & Time API yang multi time zone adalah menggunakan Clock, sehingga jika kita ingin membuat data baru, kita bisa memanfaatkan Clock



Kode : Membuat Clock

```
Clock clockUTC = Clock.systemUTC();
Clock clockSystem = Clock.systemDefaultZone();
Clock clockJakarta = Clock.system(ZoneId.of("Asia/Jakarta"));
```

Mendapatkan Instant di Clock

- Kita sudah tahu bahwa Clock itu akan selalu berjalan, tidak pernah berhenti
- Dan representasi milliseconds di Date & Time API baru adalah Instant
- Oleh karena itu, untuk mendapatkan tanggal dan waktu saat ini sesuai dengan time zone di Clock, kita bisa menggunakan method instant(), dan otomatis akan mengembalikan Instant saat ini sesuai dengan Clock nya
- Perlu diingat, karena Instant itu selalu menggunakan time zone UTC, jadi jika clock nya tidak menggunakan UTC, maka akan secara otomatis dikonversi ke time zone UTC



Kode : Mendapatkan Instant di Clock

```
Clock clockJakarta = Clock.system(ZoneId.of("Asia/Jakarta"));

Instant instant1 = clockJakarta.instant();
System.out.println(instant1);
Thread.sleep( millis: 1_000);

Instant instant2 = clockJakarta.instant();
System.out.println(instant2);
Thread.sleep( millis: 1_000);
```

Membuat Tanggal dan Waktu dari Clock

- Karena jika menggunakan Instant kita harus melakukan konversi secara manual ke LocalDateTime atau ZonedDateTime, karena bisa saja time zone nya berbeda
- Untungnya, kita juga bisa membuat tipe data tanggal dan waktu langsung menggunakan Clock, caranya menggunakan method now(Clock)
- Secara otomatis informasi time zone Clock akan dibawa di tipe data tanggal dan waktu yang kita buat



Kode : Membuat Data dari Clock

```
Clock clockJakarta = Clock.system(ZoneId.of("Asia/Jakarta"));

Year year = Year.now(clockJakarta);
YearMonth yearMonth = YearMonth.now(clockJakarta);
LocalTime localTime = LocalTime.now(clockJakarta);
LocalDate localDate = LocalDate.now(clockJakarta);
LocalDateTime localDateTime = LocalDateTime.now(clockJakarta);
ZonedDateTime zonedDateTime = ZonedDateTime.now(clockJakarta);
```

Duration

Duration

- Class Duration adalah representasi dari data durasi waktu
- Durasi waktu yang terdapat di class Duration
- Dengan menggunakan Duration, kita bisa dengan mudah mengkonversi durasi ke waktu yang kita inginkan, misal hour, minute, second dan nano second



Kode : Membuat Duration

```
Duration duration1 = Duration.ofSeconds(10);  
Duration duration2 = Duration.ofMillis(10);  
Duration duration3 = Duration.ofHours(10);
```



Kode : Mengambil Data Duration

```
Duration duration = Duration.ofHours(10);  
  
long days = duration.toDays();  
long hours = duration.toHours();  
long minutes = duration.toMinutes();  
long seconds = duration.toSeconds();  
long nanos = duration.toNanos();
```



Kode : Menghitung Duration

```
Duration duration1 = Duration.between(LocalTime.of( hour: 10, minute: 10), LocalTime.of( hour: 20, minute: 20));  
Duration duration2 = Duration.between(LocalDateTime.now(), LocalDateTime.now().plusHours(10));
```

Period

Period

- Class Period mirip dengan Duration, yang membedakan adalah Period adalah durasi untuk tanggal
- Cara penggunaan Period hampir mirip dengan Duration

Kode : Membuat Period

```
Period period1 = Period.ofDays(10);
Period period2 = Period.ofWeeks(10);
Period period3 = Period.ofMonths(10);
Period period4 = Period.ofYears(10);
Period period5 = Period.of( years: 10, months: 10, days: 10);
```

Kode : Mengambil Data Period

```
Period period = Period.of( years: 10, months: 10, days: 10);

int years = period.getYears();
int months = period.getMonths();
int days = period.getDays();
```



Kode : Menghitung Period

```
Period period = Period.between(  
    LocalDate.of( year: 2020, month: 10, dayOfMonth: 10),  
    LocalDate.of( year: 2020, month: 12, dayOfMonth: 12)  
);  
  
System.out.println(period.getYears());  
System.out.println(period.getMonths());  
System.out.println(period.getDays());
```

Temporal

Temporal

- Di dalam package java.time, sebenarnya ada package java.time.temporal
- Di dalam package java.time.temporal terdapat banyak sekali interface yang merupakan kontrak dari Java Date & Time API



Interface Temporal

Interface	Keterangan
Temporal	Interface untuk temporal object, seperti date, time, dan lain-lain
TemporalAmount	Interface untuk jumlah waktu, seperti duration dan period
TemporalUnit	Interface untuk unit satuan temporal, seperti jam, menit, hari
TemporalField	Interface untuk field dalam temporal data
TemporalQuery	Interface untuk query data dari TemporalAccessor (super interface Temporal)
TemporalAdjuster	Strategi untuk menyesuaikan objek temporal.

Temporal

- Hampir semua tipe data tanggal dan waktu di Java Date & Time adalah implementasi dari interface Temporal
- Maka dari itu, jika diperhatikan, hampir semua tipe data nya memiliki method-method yang hampir sama



Kode : Temporal

```
Temporal temporal1 = LocalTime.now();
Temporal temporal2 = LocalDate.now();
Temporal temporal3 = LocalDateTime.now();
Temporal temporal4 = ZonedDateTime.now();
Temporal temporal5 = Year.now();
Temporal temporal6 = YearMonth.now();
Temporal temporal7 = Instant.now();
```

TemporalAmount

- Duration dan Period adalah implementasi dari interface TemporalAmount
- Salah satu method yang menggunakan TemporalAmount di Temporal adalah plus() da minus()
- Artinya, dengan ini kita bisa meggunakan object TemporalAmount untuk melakukan penambahan/pengurangan tanggal dan waktu



Kode : TemporalAmount

```
LocalDateTime localDateTime1 = LocalDateTime.now();
LocalDateTime localDateTime2 = localDateTime1.plus(Duration.ofHours(10));
LocalDateTime localDateTime3 = localDateTime1.minus(Period.of( years: 0, months: 10, days: 10));

System.out.println(localDateTime1);
System.out.println(localDateTime2);
System.out.println(localDateTime3);
```

TemporalUnit

- TemporalUnit adalah implementasi dari unit satuan waktu
- Implementasi TemporalUnit adalah sebuah enum ChronoUnit
- TemporalUnit selain sebagai informasi satuan waktu, bisa juga digunakan untuk menghitung durasi waktu



Kode : TemporalUnit

```
long between1 = ChronoUnit.SECONDS.between(LocalDateTime.now(), LocalDateTime.now().plusDays(10));
long between2 = ChronoUnit.MINUTES.between(LocalDateTime.now(), LocalDateTime.now().plusDays(10));
long between3 = ChronoUnit.DAYS.between(LocalDateTime.now(), LocalDateTime.now().plusDays(10));

System.out.println(between1);
System.out.println(between2);
System.out.println(between3);
```

TemporalField

- TemporalField adalah informasi field yang terdapat dalam sebuah tipe data
- Semua object Temporal memiliki method bernama get(TemporalField) atau getLong(TemporalField) untuk mendapatkan info seputar field pada object tersebut, sesuai dengan Field yang kita inginkan
- Implementasi TemporalField adalah enum bernama ChronoField



Kode : TemporalField

```
LocalDateTime localDateTime = LocalDateTime.now();
int year = localDateTime.get(ChronoField.YEAR);
int month = localDateTime.get(ChronoField.MONTH_OF_YEAR);
int day = localDateTime.get(ChronoField.DAY_OF_MONTH);
int hour = localDateTime.get(ChronoField.HOUR_OF_DAY);
```

TemporalQuery

- TemporalQuery merupakan lambda interface yang bisa kita gunakan untuk mengambil informasi dari data TemporalAccessor
- TemporalQuery adalah generic type, jadi kita bisa mengembalikan tipe data apapun pada hasil query yang kita lakukan di TemporalAccessor



Kode : TemporalQuery

```
LocalDateTime localDateTime = LocalDateTime.now();

List<Integer> integers = localDateTime.query(temporal → {
    ArrayList<Integer> list = new ArrayList<>();
    list.add(temporal.get(ChronoField.YEAR));
    list.add(temporal.get(ChronoField.MONTH_OF_YEAR));
    list.add(temporal.get(ChronoField.DAY_OF_MONTH));
    return list;
});
```

TemporalAdjuster

- TemporalAdjuster adalah strategi untuk menyesuaikan objek temporal.
- Kita bisa melakukan implementasi penyesuaian object sendiri, atau kita juga bisa menggunakan helper class beranam TemporalAdjusters
- Terdapat banyak static method di TemporalAdjusters yang bisa kita gunakan untuk mempermudah melakukan penyesuaian objek temporal



Kode : TemporalAdjuster

```
LocalDate localDate1 = LocalDate.now();
LocalDate localDate2 = localDate1.with(TemporalAdjusters.firstDayOfMonth());
LocalDate localDate3 = localDate1.with(TemporalAdjusters.lastDayOfMonth());
LocalDate localDate4 = localDate1.with(TemporalAdjusters.firstDayOfYear());
LocalDate localDate5 = localDate1.with(TemporalAdjusters.lastDayOfYear());
```

DayOfWeek

DayOfWeek

- Yang menarik di Java Date & Time adalah, memiliki implementasi data hari
- Data hari ini berupa Enum DayOfWeek, namun dia adalah turunan dari Temporal, sehingga operasi yang bisa dilakukan di Temporal bisa kita lakukan disini, seperti menambah atau mengurangi



Kode : DayOfWeek

```
DayOfWeek dayOfWeek1 = DayOfWeek.SUNDAY;  
DayOfWeek dayOfWeek2 = dayOfWeek1.plus(2);  
DayOfWeek dayOfWeek3 = dayOfWeek1.minus(2);  
  
System.out.println(dayOfWeek1);  
System.out.println(dayOfWeek2);  
System.out.println(dayOfWeek3);
```

Parsing dan Formatting

Parsing dan Formatting

- Sebelumnya di awal-awal kita sering sekali menggunakan method parse untuk melakukan parsing data String menjadi objek Temporal
- Sebenarnya proses parsing dan formatting tersebut dilakukan oleh class `java.time.format.DateTimeFormatter`



Pattern

- Pattern untuk membuat DateTimeFormatter hampir mirip dengan pattern yang sudah pernah dibahas di course tentang Java Internationalization
- Atau lebih detail bisa dilihat di halaman ini :
<https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/time/format/DateTimeFormatter.html>

Pattern Letters and Symbols

Symbol	Meaning	Presentation	Examples
G	era	text	AD; Anno Domini; A
u	year	year	2004; 04
y	year-of-era	year	2004; 04
D	day-of-year	number	189
M/L	month-of-year	number/text	7; 07; Jul; July; J
d	day-of-month	number	10
g	modified-julian-day	number	2451334
Q/q	quarter-of-year	number/text	3; 03; Q3; 3rd quarter
Y	week-based-year	year	1996; 96
w	week-of-week-based-year	number	27
W	week-of-month	number	4
E	day-of-week	text	Tue; Tuesday; T
e/c	localized day-of-week	number/text	2; 02; Tue; Tuesday; T
F	day-of-week-in-month	number	3
a	am-pm-of-day	text	PM
B	period-of-day	text	in the morning
h	clock-hour-of-am-pm (1-12)	number	12
K	hour-of-am-pm (0-11)	number	0
k	clock-hour-of-day (1-24)	number	24
H	hour-of-day (0-23)	number	0
m	minute-of-hour	number	30

S	fraction-of-second	fraction	978
A	milli-of-day	number	1234
n	nano-of-second	number	987654321
N	nano-of-day	number	1234000000
V	time-zone ID	zone-id	America/Los_Angeles; Z; -08:30
v	generic time-zone name	zone-name	Pacific Time; PT
z	time-zone name	zone-name	Pacific Standard Time; PST
O	localized zone-offset	offset-O	GMT+8; GMT+08:00; UTC-08:00
X	zone-offset 'Z' for zero	offset-X	Z; -08; -0830; -08:30; -083015; -08:30:15
x	zone-offset	offset-x	+0000; -08; -0830; -08:30; -083015; -08:30:15
Z	zone-offset	offset-Z	+0000; -0800; -08:00
p	pad next	pad modifier	1
'	escape for text	delimiter	
"	single quote	literal	'
[optional section start		
]	optional section end		
#	reserved for future use		
{	reserved for future use		
}	reserved for future use		

Parsing

- Sebelumnya kita sudah bahas tentang default pattern untuk parsing semua tipe objek temporal
- Jika kita ingin mengubah format pattern nya, kita bisa menggunakan DateTimeFormatter pada saat melakukan parsing, misal :
- `LocalDate.parse("yyyy MM hh", dateTimeFormatter)`



Kode : Parsing dengan DateTimeFormatter

```
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy MM dd");  
  
LocalDate localDate = LocalDate.parse( text: "2020 10 12", formatter);  
  
System.out.println(localDate);
```

Formatting

- Untuk formatting juga kita bisa menggunakan DateTimeFormatter
- Untuk melakukan formatting, kita bisa menggunakan method format(DateTimeFormatter) milik objek temporal



Kode : Formatting

```
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy MM dd");

LocalDate localDate = LocalDate.now();
String format = localDate.format(formatter);

System.out.println(format);
```

Default Formatter

- Selain kita bisa membuat formatter sendiri menggunakan pattern
- Sudah disediakan juga formatter default, sehingga jika kita ingin menggunakan formatter misal yang sudah standar internasional, kita tidak perlu membuat ulang menggunakan pattern

Formatter Default di Java

BASIC_ISO_DATE	Basic ISO date	'20111203'
ISO_LOCAL_DATE	ISO Local Date	'2011-12-03'
ISO_OFFSET_DATE	ISO Date with offset	'2011-12-03+01:00'
ISO_DATE	ISO Date with or without offset	'2011-12-03+01:00'; '2011-12-03'
ISO_LOCAL_TIME	Time without offset	'10:15:30'
ISO_OFFSET_TIME	Time with offset	'10:15:30+01:00'
ISO_TIME	Time with or without offset	'10:15:30+01:00'; '10:15:30'
ISO_LOCAL_DATE_TIME	ISO Local Date and Time	'2011-12-03T10:15:30'
ISO_OFFSET_DATE_TIME	Date Time with Offset	'2011-12-03T10:15:30+01:00'
ISO_ZONED_DATE_TIME	Zoned Date Time	'2011-12-03T10:15:30+01:00[Europe/Paris]'
ISO_DATE_TIME	Date and time with ZoneId	'2011-12-03T10:15:30+01:00[Europe/Paris]'
ISO_ORDINAL_DATE	Year and day of year	'2012-337'
ISO_WEEK_DATE	Year and Week	'2012-W48-6'
ISO_INSTANT	Date and Time of an Instant	'2011-12-03T10:15:30Z'
RFC_1123_DATE_TIME	RFC 1123 / RFC 822	'Tue, 3 Jun 2008 11:05:30 GMT'



Kode : Default Formatter

```
DateTimeFormatter formatter = DateTimeFormatter.ISO_LOCAL_DATE;  
  
LocalDate localDate = LocalDate.parse("2020-10-12", formatter);  
String format = localDate.format(formatter);
```

Internationalization

- DateTimeFormatter juga mendukung internationalization
- Caranya saat membuat formatter, kita perlu menambahkan Locale



Kode : Internationalization

```
Locale locale = new Locale(language: "in", country: "ID");
DateTimeFormatter formatter = DateTimeFormatter.ofPattern(pattern: "yyyy MMMM dd", locale);

LocalDate localDate = LocalDate.now();
String format = localDate.format(formatter);

System.out.println(format);
```

Legacy Date dan Time

Legacy Date dan Time

- Saat fitur Java Date & Time API baru keluar, Java juga menambahkan integrasi dengan fitur legacy date dan time yang menggunakan Date, Calendar dan TimeZone
- Kita bisa melakukan konversi dari tipe temporal legacy ke tipe temporal baru ataupun sebaliknya



Kode : Konversi Legacy ke Baru

```
Date date = new Date();
Instant instantDate = date.toInstant();

Calendar calendar = Calendar.getInstance();
Instant instantCalendar = calendar.toInstant();

TimeZone timeZone = TimeZone.getDefault();
ZoneId zoneId = timeZone.toZoneId();
```



Kode : Konversi New ke Legacy

```
ZonedDateTime zonedDateTime = ZonedDateTime.now();
Date date = Date.from(zonedDateTime.toInstant());

Calendar calendar = GregorianCalendar.from(zonedDateTime);

ZoneId zoneId = ZoneId.systemDefault();
TimeZone timeZone = TimeZone.getTimeZone(zoneId);
```