Core Date/Time Classes

วัตถุประสงค์

- สร้างและจัดการ events ตามวันและตามเวลา รวมไปถึงการรวมวันที่และเวลาให้อยู่ใน object เดียวกันโดยใช้ LocalDate, LocalTime, LocalDateTime, Instant, Period และ Duration
- กำหนดและสร้างและจัดการเหตุการณ์ตามวันและเวลาโดยใช้ Instant, Period, Duration และ TemporalUnit

A new Date/Time API

ตั้งแต่เริ่มต้นของ Java จะมี java.util.Date and java.util.Calendar แต่ class พวกนี้ยังไม่ค่อยดีและมีปัญหาบางอย่างคือ

- java.util.Date แสดงเวลาที่มีความแม่นยำเฉพาะ ”milliseconds” เท่านั้น (ซึ่งอาจไม่เพียงพอในบางแอพพลิเคชัน) ปีที่เริ่มต้นตั้งแต่ปี 1900 และเดือนเริ่มต้นที่ 0
- JVM’s จะ default time zone ให้
- ทั้ง java.util.Date and java.util.Calendar เป็น mutable class เมื่อเปลี่ยนไปแล้วจะห้ามสร้าง instant ด้วยการ new value

LocalDate

แสดงวันที่ ด้วย ปี, เดือน, วัน example, 2015-08-25

LocalTime

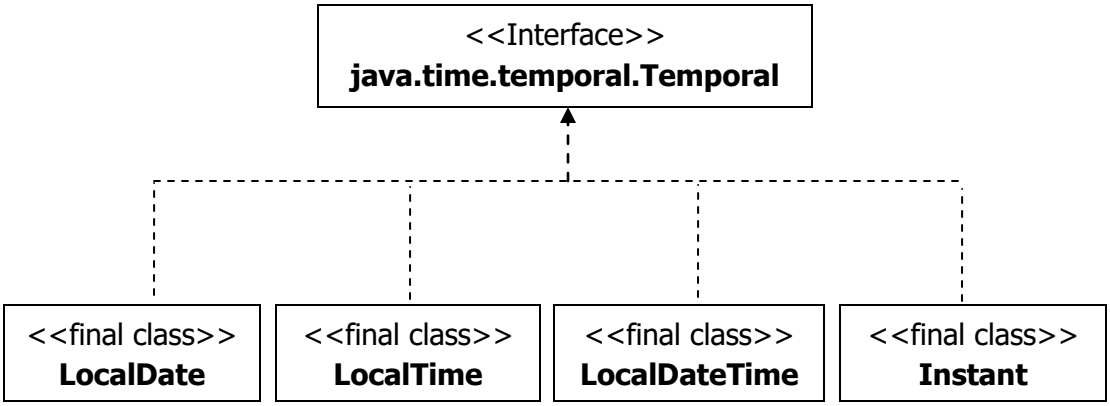
แสดงเวลา ด้วย ชั่วโมง, นาที, วินาที, nanoseconds example, *13:21.05.123456789*.

LocalDateTime

รวมวันที่และเวลาเข้าด้วยกัน example, *2015-08-25 13:21.05.12345*.

Instant

แสดงจุดเดียวในเวลาเป็น seconds and nanoseconds example *923,456,789 seconds and 186,054,812 nanoseconds ใช้กับการสร้าง timestamps*

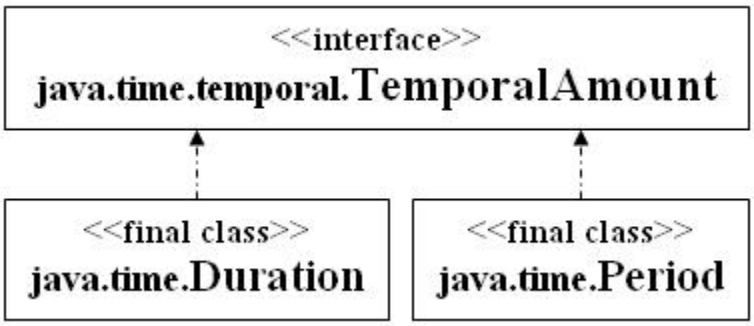


Period

แสดงจำนวนเวลาในแง่ของปี, เดือน, และวัน example, *5 years, 2 months and 9 days*.

Duration

แสดงจำนวนเวลาในแง่ของ seconds and nanoseconds example, *12.87656 seconds*



LocalDate Class

Create Instant โดยใช้ static method "of"

public static LocalDate of(int year,Month month, int dayOfMonth)

public static LocalDate of(int year,int month, int dayOfMonth)

public static LocalDate ofYearDay(int year, int dayOfYear)

public static LocalDate ofEpochDay(long epochDay)//นับตั้งแต่ 1970-01-01

```
// With year(-999999999 to 999999999),
// month (1 to 12), day of the month (1 - 31)}
LocalDate newYear2001 = LocalDate.of(2001, 1, 1);
// This version uses the enum java.time.Month
LocalDate newYear2002 = LocalDate.of(2002, Month.JANUARY, 1);
```

สังเกตว่าแตกต่างจาก java.util.Date ตรงที่เดือนเริ่มจากหนึ่ง

ถ้าสร้างวันที่ที่มีค่าไม่ถูกต้องเช่น 29 กุมภาพันธ์ จะ throw exception

ถ้าต้องการรู้ว่าวันนี้เป็นวันอะไรใช้ method “now”

```
LocalDate today = LocalDate.now();
```

เมื่อมี instant ของ LocalDate แล้วก็สามารถ getYear, month, and day ได้

```
int year = today.getYear();
int month = today.getMonthValue();
Month monthAsEnum = today.getMonth(); // as an enum
int dayYear = today.getDayOfYear();
int dayMonth = today.getDayOfMonth();
DayOfWeek dayWeekEnum = today.getDayOfWeek(); //as an enum
```

สามารถใช้ get ของ java.time.ChronoField เป็น enum ซึ่ง implement มาจา Interface java.time.TemporalField

```
int get(java.time.TemporalField field) // value as int
long getLong(java.time.TemporalField field) // value as long
```

```
int year = today.get(ChronoField.YEAR);
int month = today.get(ChronoField.MONTH_OF_YEAR);
int dayYear = today.get(ChronoField.DAY_OF_YEAR);
int dayMonth = today.get(ChronoField.DAY_OF_MONTH);
int dayWeek = today.get(ChronoField.DAY_OF_WEEK);
long dayEpoch = today.getLong(ChronoField.EPOCH_DAY);
```

The supported values for ChronoField are:

DAY_OF_WEEK	แสดงวันของสัปดาห์(Monday(1)-Sunday(7))	int
ALIGNED_DAY_OF_WEEK_IN_MONTH	แสดงลำดับวันที่ของสัปดาห์ในเดือนนั้นๆ (1-7) วนซ้ำไปเรื่อยๆ เช่น กำหนดวันที่เป็น 15 จะได้ 1 ซึ่งมาจาก (15%7)	int
ALIGNED_DAY_OF_WEEK_IN_YEAR	แสดงลำดับวันที่ของสัปดาห์ในปีนั้นๆ (Sunday(1)- Saturday(7)) วนซ้ำไปเรื่อยๆ	int
DAY_OF_MONTH	แสดงลำดับวันที่ในเดือนนั้นๆ(1-28, 1-29, 1-30, 1-31)	int
DAY_OF_YEAR	แสดงลำดับวันที่ในปีนั้นๆ(1-365, 1-366)	int

EPOCH_DAY	นับวันตั้งแต่ 1970-01-01 (ISO) โดยเริ่มที่ 0	long
ALIGNED_WEEK_OF_MONTH	นับสัปดาห์ภายใน 1 เดือนนั้น(1-7) เช่น กำหนดวันที่เป็น 17 จะได้ 3 ซึ่งมาจาก (17 / 7)	int
ALIGNED_WEEK_OF_YEAR	นับสัปดาห์ภายใน 1 ปี(0-52)	int
MONTH_OF_YEAR	แสดงเดือน(1-12)	int
PROLEPTIC_MONTH	แสดงจำนวนเดือนทั้งหมดตั้งแต่เริ่มต้น ค.ศ. โดยเริ่มจาก 0 เช่น กันยายน 2017 จะได้ 24212 ซึ่งมากจาก (year * 12 + month - 1)	long
YEAR	แสดงปี	int
ERA	ระบบปฏิทินที่ไม่ใช่ ISO จะใช้ฟิลด์นี้เพื่อกำหนดช่วงเวลา ถ้าเป็นระบบ ISO จะมีค่าเป็น 1	int

Example : compare date with another instance

```
boolean after = newYear2001.isAfter(newYear2002); // false
boolean before = newYear2001.isBefore(newYear2002); // true
boolean equal = newYear2001.equals(newYear2002); // false
boolean leapYear = newYear2001.isLeapYear(); // false
```

Example : การสร้าง instance อื่นจาก Instance ที่มีอยู่ได้โดยการใช้ method “with”

```
LocalDate newYear2003 = newYear2001.with(ChronoField.YEAR, 2003);
LocalDate newYear2004 = newYear2001.withYear(2004);
LocalDate december2001 = newYear2001.withMonth(12);
LocalDate february2001 = newYear2001.withDayOfYear(32);
// Since these methods return a new instance, we can chain them!
LocalDate xmas2001 = newYear2001.withMonth(12).withDayOfMonth(25);
```

Example: บวก ลบ ปี, เดือน, วัน, สัปดาห์ โดยการใช้ java.time.temporal.ChronoUnit

```java
// Adding
LocalDate newYear2005 = newYear2001.plusYears(4);
LocalDate march2001 = newYear2001.plusMonths(2);
LocalDate january15_2001 = newYear2001.plusDays(14);
LocalDate lastWeekJanuary2001 = newYear2001.plusWeeks(3);
LocalDate newYear2006 = newYear2001.plus(5, ChronoUnit.YEARS);

// Subtracting
LocalDate newYear2000 = newYear2001.minusYears(1);
LocalDate nov2000 = newYear2001.minusMonths(2);
LocalDate dec30_2000 = newYear2001.minusDays(2);
LocalDate lastWeekDec2001 = newYear2001.minusWeeks(1);
LocalDate newYear1999 = newYear2001.minus(2, ChronoUnit.YEARS);
```

The supported values for ChronoUnit are:

DAYS	บวกหรือลบกี่วัน
WEEKS	บวกหรือลบกี่สัปดาห์
MONTHS	บวกหรือลบกี่เดือน
YEARS	บวกหรือลบกี่ปี
DECADES	บวกหรือลบกี่ทศวรรษ(กำหนด 1 มีค่าเท่ากับ 10)
CENTURIES	บวกหรือลบกี่ปี(กำหนด 1 มีค่าเท่ากับ 100)
MILLENNIA	บวกหรือลบกี่ปี(กำหนด 1 มีค่าเท่ากับ 1000)
ERAS	Concept เดียวกันกับ ERA

Method “toString()” จะ return format uuuu-MM-dd

LocalTime Class

เป็น class ทำงานกับเวลา hour, minutes, seconds, and nanoseconds.

Create instance โดยการใช้ static method “of”

public static LocalTime of(int hour, int minute)

public static LocalTime of(int hour, int minute, int second)

public static LocalTime of(int hour, int minute, int second, int nanoOfSecond)

public static LocalTime ofSecondOfDay(long secondOfDay) //from 0 to 24 * 60 * 60 – 1

public static LocalTime ofNanoOfDay(long nanoOfDay) //from 0 to 24 * 60 * 60 * 1,000,000,000 – 1

```
// With hour (0-23) and minutes (0-59)
LocalTime fiveThirty = LocalTime.of(5, 30);
// With hour, minutes, and seconds (0-59)
LocalTime noon = LocalTime.of(12, 0, 0);
// With hour, minutes, seconds, and nanoseconds (0-999,999,999)
LocalTime almostMidnight = LocalTime.of(23, 59, 59, 999999);
```

ถ้าอยากรู้เวลาปัจจุบันก็ใช้ method “now()”

```
LocalTime now = LocalTime.now();
```

เมื่อมี instant ของ LocalTime แล้วก็สามารถ getHour, minute, and ข้อมูลอื่นมาดูได้

```
int hour = now.getHour();
int minute = now.getMinute();
int second = now. getSecond();
int nanosecond = now. getNano();
```

สามารถ get ข้อมูลด้วย ChronoField เหมือนกับ LocalDate

```
int hourAMPM = now.get(ChronoField.HOUR_OF_AMPM); // 0 - 11
int hourDay = now.get(ChronoField.HOUR_OF_DAY); // 0 - 23
int minuteDay = now.get(ChronoField.MINUTE_OF_DAY); // 0 - 1,439
int minuteHour = now.get(ChronoField.MINUTE_OF_HOUR); // 0 - 59
int secondDay = now.get(ChronoField.SECOND_OF_DAY); // 0 - 86,399
int secondMinute = now.get(ChronoField.SECOND_OF_MINUTE);// 0 - 59
long nanoDay = now.getLong(ChronoField.NANO_OF_DAY);//0-86399999999
int nanoSecond = now.get(ChronoField.NANO_OF_SECOND);//0-999999999
```

The supported values for ChronoField are:

NANO_OF_SECOND	0-999,999,999	int
NANO_OF_DAY	0-86,399,999,999	long
MICRO_OF_SECOND	0-999,999	int
MICRO_OF_DAY	0- (24 * 60 * 60 * 1,000,000) - 1	long
MILLI_OF_SECOND	0-999	int
MILLI_OF_DAY	0 to (24 * 60 * 60 * 1,000) - 1	int
SECOND_OF_MINUTE	0-59	int
SECOND_OF_DAY	0-86,399	int
MINUTE_OF_HOUR	0-59	int
MINUTE_OF_DAY	0-1439	int

HOUR_OF_AMPM	0-11	int
CLOCK_HOUR_OF_AMPM	1-12	int
HOUR_OF_DAY	0-23	int
CLOCK_HOUR_OF_DAY	1-24	int
AMPM_OF_DAY	0 (AM) to 1 (PM)	int

Example : compare time

```
boolean after = fiveThirty.isAfter(noon); // false
boolean before = fiveThirty.isBefore(noon); // true
boolean equal = noon.equals(almostMidnight); // false
```

Example : การสร้าง instance อื่นจาก Instance ที่มีอยู่ได้โดยการใช้ method “with”

```
LocalTime ten = noon.with(ChronoField.HOUR_OF_DAY, 10);
LocalTime eight = noon.withHour(8);
LocalTime twelveThirty = noon.withMinute(30);
LocalTime thirtyTwoSeconds = noon.withSecond(32);
// Since these methods return a new instance, we can chain them!
LocalTime secondsNano = nooon.withSecond(20).withNano(999999);
```

Example: บวก ลบ hours, minutes, seconds , หรือ nanoseconds โดยการใช้ java.time.temporal.ChronoUnit

```
// Adding
LocalTime sixThirty = fiveThirty.plusHours(1);
LocalTime fiveForty = fiveThirty.plusMinutes(10);
LocalTime plusSeconds = fiveThirty.plusSeconds(14);
LocalTime plusNanos = fiveThirty.plusNanos(99999999);
LocalTime sevenThirty = fiveThirty.plus(2, ChronoUnit.HOURS);

// Subtracting
LocalTime fourThirty = fiveThirty.minusHours(1);
LocalTime fiveTen = fiveThirty.minusMinutes(20);
LocalTime minusSeconds = fiveThirty.minusSeconds(2);
LocalTime minusNanos = fiveThirty.minusNanos(1);
LocalTime fiveTwenty = fiveThirty.minus(10, ChronoUnit.MINUTES);
```

The supported values for ChronoUnit are:

- NANOS
- MICROS
- MILLIS

- SECONDS
- MINUTES
- HOURS
- HALF_DAYS

Method toString() จะ return format *HH:mm:ss.SSSSSSSSS* ยกเว้นถ้า seconds และ nanoseconds เป็น 0 จะ return *HH:mm*

LocalDateTime Class

เป็นการรวมระหว่าง LocalDate และ LocalTime จะแสดงทั้ง date และ time ด้วย year, month, day, hours, minutes, seconds, and nanoseconds และสามารถเข้าถึง field อื่นได้เช่น day of year, day of week, and week of year

Example : Create Instant โดยใช้ static method “of” และการ combine LocalDate, LocalTime

public static LocalDateTime of(int year, Month month, int dayOfMonth, int hour, int minute)
public static LocalDateTime of(int year, Month month, int dayOfMonth,int hour, int minute, int second)
public static LocalDateTime of(int year, Month month, int dayOfMonth, int hour, int minute, int second, int nanoOfSecond)
public static LocalDateTime of(int year, int month, int dayOfMonth, int hour, int minute)
public static LocalDateTime of(int year, int month, int dayOfMonth,int hour, int minute, int second)
public static LocalDateTime of(int year, int month, int dayOfMonth,int hour, int minute, int second, int nanoOfSecond)
public static LocalDateTime of(LocalDate date, LocalTime time)
public static LocalDateTime ofInstant(Instant instant, ZoneId zone)
public static LocalDateTime ofEpochSecond(long epochSecond, int nanoOfSecond, ZoneOffset offset)

```java
// Setting seconds and nanoseconds to zero
LocalDateTime dt1 = LocalDateTime.of(2014, 9, 19, 14, 5);
// Setting nanoseconds to zero
LocalDateTime dt2 = LocalDateTime.of(2014, 9, 19, 14, 5, 20);
// Setting all fields
LocalDateTime dt3 = LocalDateTime.of(2014, 9, 19, 14, 5, 20, 9);
// Asumming this date LocalDate date = LocalDate.now();
// And this time LocaDate time = LocalTime.now();
// Combine the above date with the given time like this
LocalDateTime dt4 = date.atTime(14, 30, 59, 999999);
// Or this LocalDateTime dt5 = date.atTime(time);
// Combine this time with the given date. Notice that LocalTime
// only has this constructor to be combined with a LocalDate
LocalDateTime dt6 = time.atDate(date);
```

- ถ้าอยากรู้วันเวลาปัจจุบันก็ใช้ method “now()” เหมือนกับ LocalDate, LocalTime
- เมื่อมี instance ของ LocalDateTime เราสามารถ get ข้อมูลโดยใช้ method “get…()” หรือสามารถใช้ get ด้วย ChonoField

The supported values for ChronoField are:

NANO_OF_SECOND	Ref. LocalTime
NANO_OF_DAY	Ref. LocalTime
MICRO_OF_SECOND	Ref. LocalTime
MICRO_OF_DAY	Ref. LocalTime
MILLI_OF_SECOND	Ref. LocalTime
MILLI_OF_DAY	Ref. LocalTime
SECOND_OF_MINUTE	Ref. LocalTime
SECOND_OF_DAY	Ref. LocalTime
MINUTE_OF_HOUR	Ref. LocalTime
MINUTE_OF_DAY	Ref. LocalTime
HOUR_OF_AMPM	Ref. LocalTime
CLOCK_HOUR_OF_AMPM	Ref. LocalTime
HOUR_OF_DAY	Ref. LocalTime
CLOCK_HOUR_OF_DAY	Ref. LocalTime
AMPM_OF_DAY	Ref. LocalTime
DAY_OF_WEEK	Ref. LocalDate
ALIGNED_DAY_OF_WEEK_IN_MONTH	Ref. LocalDate
ALIGNED_DAY_OF_WEEK_IN_YEAR	Ref. LocalDate
DAY_OF_MONTH	Ref. LocalDate

DAY_OF_YEAR	Ref. LocalDate
EPOCH_DAY	Ref. LocalDate
ALIGNED_WEEK_OF_MONTH	Ref. LocalDate
ALIGNED_WEEK_OF_YEAR	Ref. LocalDate
MONTH_OF_YEAR	Ref. LocalDate
PROLEPTIC_MONTH	Ref. LocalDate
YEAR_OF_ERA	Ref. LocalDate
YEAR	Ref. LocalDate
ERA	Ref. LocalDate

** method toString() returns the date-time in the format *uuuu-MM-dd'T'HH:mm:ss.SSSSSSSSS* ยกเว้นถ้า seconds และ nanoseconds เป็น 0 จะ return *uuuu-MM-dd'T'HH:mm*

Instant Class

- แม้ว่าในทางปฏิบัติ LocalDateTime จะแสดงถึงช่วงเวลาในปัจจุบัน แต่ก็มีคลาสอื่นที่อาจเหมาะสมกว่า
- เหมาะสำหรับการสร้าง timestamp ซึ่งจะใช้ seconds และ nanoseconds
- เราสามารถสร้าง instance ของคลาสด้วย method “ofEpoch…”

```
// Setting seconds
Instant fiveSecondsAfterEpoch = Instant.ofEpochSecond(5);
// Setting seconds and nanoseconds (can be negative)
Instant sixSecTwoNanBeforeEpoch = Instant.ofEpochSecond(-6, -2);
// Setting milliseconds after (can be before also) epoch
Instant fifthyMilliSecondsAfterEpoch = Instant.ofEpochMilli(50);
```

ใช้ Instant.now(); สำหรับ get current system clock

เมื่อมี instance แล้วเราสามารถ get ข้อมูลอื่นด้วยการใช้ ChronoField

```
long seconds = now.getEpochSecond(); // Gets the seconds
int nanos1 = now.getNano(); // Gets the nanoseconds
// Gets the value as an int
int milis = now.get(ChronoField.MILLI_OF_SECOND);
// Gets the value as a long
long nanos2 = now.getLong(ChronoField.NANO_OF_SECOND);
```

The supported ChronoField values are:

- NANO_OF_SECOND
- MICRO_OF_SECOND
- MILLI_OF_SECOND
- INSTANT_SECONDS //แสดงจุดของ time-line ได้ด้วยตัวเองโดยไม่ต้องระบุ time zone

To check an `Instant` object against another one, we have three methods:

```
boolean after = now.isAfter(fiveSecondsAfterEpoch); // true
boolean before = now.isBefore(fiveSecondsAfterEpoch); // false
boolean equal = now.equals(fiveSecondsAfterEpoch); // false
```

นอกจากนี้เรายังสามารถสร้าง instance อื่นจาก Instance ที่มีอยู่ได้โดยการใช้ method “with”

```
Instant i1 = now.with(ChronoField.NANO_OF_SECOND, 10);
```

และยังสามารถ บวก ลบ seconds, milliseconds, หรือ nanoseconds โดยการใช้ java.time.temporal.ChronoUnit

```
// Adding
Instant dt10 = now.plusSeconds(400);
Instant dt11 = now.plusMillis(98622200);
Instant dt12 = now.plusNanos(3000138900);
Instant dt13 = newYear2001.plus(2, ChronoUnit.MINUTES);

// Subtracting
Instant dt14 = now.minusSeconds(2);
Instant dt15 = now.minusMillis(1);
Instant dt16 = now.minusNanos(1);
Instant dt17 = now.minus(10, ChronoUnit.SECONDS);
```

The supported ChronoUnit values are:

NANOS	Ref. LocalTime
MICROS	Ref. LocalTime
MILLIS	Ref. LocalTime
SECONDS	Ref. LocalTime
MINUTES	Ref. LocalTime
HOURS	Ref. LocalTime

HALF_DAYS	Ref. LocalTime
DAYS	Ref. LocalDate

Method toString() จะ return format *uuuu-MM-dd'T'HH:mm:ss.SSSSSSSSS*

Example : `1970-01-01T00:00:00.050Z`

สังเกตว่ามีข้อมูลโซนเวลา (Z) เนื่องจาก Instant แสดงถึงจุดในช่วงเวลาตั้งแต่ยุค 1970-01-01Z ในเขตเวลา UTC

Period Class

- แสดงให้เห็นถึงจำนวนเวลาในแง่ของปีเดือนและวัน

เราสามารถสร้าง instance ด้วย method “of”

```
// Setting years, months, days (can be negative)
Period period5y4m3d = Period.of(5, 4, 3);
// Setting days (can be negative), years and months will be zero
Period period2d = Period.ofDays(2);
// Setting months (can be negative), years and days will be zero
Period period2m = Period.ofMonths(2);
// Setting weeks (can be negative). The resulting period will
// be in days (1 week = 7 days). Years and months will be zero
Period period14d = Period.ofWeeks(2);
// Setting years (can be negative), days and months will be zero
Period period2y = Period.ofDays(2);
```

Example:

- ผลลัพธ์ของการสร้าง instance ด้วยการกำหนด ปี เดือน วันจะได้ PnYnMnD -> P5Y4M3D

Period เป็นแนวคิดที่ช่วยในการหาผลต่างระหว่าง 2 LocalDate

```
LocalDate march2003 = LocalDate.of(2003, 3, 1);
LocalDate may2003 = LocalDate.of(2003, 5, 1);
Period dif = Period.between(march2003, may2003); // 2 months
```

** The start date is **INCLUDED**, but **NOT** the end date.

ผลลัพธ์สามารถติดลบได้ถ้า end before start

```
// dif1 will be 1 year 2 months 2 days
Period dif1 = Period.between( LocalDate.of(2000, 2, 10),
LocalDate.of(2001, 4, 12));
// dif2 will be 25 days
Period dif2 = Period.between( LocalDate.of(2013, 5, 9), LocalDate.of(2013, 6, 3));
```

```
// dif3 will be -2 years -3 days
Period dif3 = Period.between( LocalDate.of(2014, 11, 3),
LocalDate.of(2012, 10, 31));
```

เมื่อมี instance แล้วเราก็สามารถดึงข้อมูลของ period ด้วย method “get” โดยสามารถใช้ ChronoUnit

The supported ChronoUnit values are:

- DAYS
- MONTHS
- YEARS

ซึ่งถ้าใช้นอกเหนือจากนี้จะ throw exception

นอกจากนี้เรายังสามารถสร้าง instance อื่นจาก Instance ที่มีอยู่ได้โดยการใช้ method “with”

```
Period period8d = period2d.withDays(8);
// Since these methods return a new instance, we can chain them!
Period period2y1m2d = period2d.withYears(2).withMonths(1);
```

และยังสามารถ ลบ หรือ บวก ปี เดือน วัน ได้ด้วย

```
// Adding
Period period9y4m3d = period5y4m3d.plusYears(4);
Period period5y7m3d = period5y4m3d.plusMonths(3);
Period period5y4m6d = period5y4m3d.plusDays(3);
Period period7y4m3d = period5y4m3d.plus(period2y);

// Subtracting
Period period5y4m3d = period5y4m3d.minusYears(2);
Period period5y4m3d = period5y4m3d.minusMonths(1);
Period period5y4m3d = period5y4m3d.minusDays(1);
Period period5y4m3d = period5y4m3d.minus(period2y);
```

Method toString() จะ return format *PNYNMND ถ้า period เป็น 0 จะแสดง P0D*

Duration Class

java.time.Duration จะคล้ายกับ Period แต่ Duration จะแสดงให้เห็นถึงการทำงานกับ Seconds และ nanoseconds

เราสามารถสร้าง instance ของ Duration ด้วย method “of” ร่วมกับการใช้ ChronoUnit

```
Duration oneDay = Duration.ofDays(1); // 1 day = 86400 seconds
Duration oneHour = Duration.ofHours(1); // 1 hour = 3600 seconds
Duration oneMin = Duration.ofMinutes(1); // 1 minute = 60 seconds
Duration tenSeconds = Duration.ofSeconds(10);
// Set seconds and nanoseconds (if they are outside the range
// 0 to 999,999,999, the seconds will be altered, like below)
Duration twoSeconds = Duration.ofSeconds(1, 1000000000);
// Seconds and nanoseconds are extracted from the passed milisecs
Duration oneSecondFromMilis = Duration.ofMilis(2);
// Seconds and nanoseconds are extracted from the passed nanos
Duration oneSecondFromNanos = Duration.ofNanos(1000000000);
Duration oneSecond = Duration.of(1, ChronoUnit.SECONDS);
```

ผลลัพธ์ที่ได้จากการจะมี format ดังนี้ *PTnHnMnS*

- PT เป็น prefix ของ Duration
- n แสดงจำนวน
- H หมายถึงชั่วโมง
- M หมายถึงนาที
- S หมายถึงวินาที

Valid values of ChronoUnit are:

- NANOS
- MICROS
- MILLIS
- SECONDS
- MINUTES
- HOURS
- HALF_DAYS
- DAYS

Duration ยังสามารถหาความแตกต่างระหว่าง 2 instace ด้วยการใช้ interface java.time.temporal.Temporal ซึ่ง class ที่สามารถใช้ได้มีดังนี้

- LocalTime

- LocalDateTime
- Instant

```
Duration dif = Duration.between( Instant.ofEpochSecond(123456789), Instant.ofEpochSecond(99999));
```

** ผลลัพธ์สามารถเป็นลบได้ถ้า end before start

** ถ้า object ต่าง type กันจะ convert base on first object แต่ first argument ต้องเป็น LocalTime และ second argument เป็น LocalDateTime

เมื่อมี instance ของ Duration แล้วก็สามารถใช้ method “get” เพื่อดึงข้อมูลโดยสามารถใช้ร่วมกับ ChronoUnit ได้

```
// Nanoseconds part the duration, from 0 to 999,999,999
int nanos = oneSecond.getNano();
// Seconds part of the duration, positive or negative
int seconds = oneSecond.getSeconds();
// Supports SECONDS and NANOS.Other units throw an exception
int oneSec = oneSecond.get(ChronoUnit.SECONDS);
```

นอกจากนี้เรายังสามารถสร้าง instance อื่นจาก Instance ที่มีอยู่ได้โดยการใช้ method “with”

```
Duration duration1sec8nan = oneSecond.withNanos(8);
Duration duration2sec1nan = oneSecond.withSeconds(2).withNanos(1);
```

และยังสามารถ บวก ลบ days, hours, minutes, seconds, milliseconds หรือ nanoseconds โดยสามารถใช้ java.time.temporal.ChronoUnit ได้

the method toString() returns the duration with the format *PTnHnMnS*.

Key Points

1. LocalDate, LocalTime, LocalDateTime, Instant, Period, Duration เป็น new Java Date/Time API ซึ่งมีอยู่ใน package java.time, เป็น immutable, thread-safe, ยกเว้น Instant จะไม่เก็บหรือแสดง time-zone
2. LocalDate, LocalTime, LocalDateTime and Instant จะ implement interface java.time.temporal.Temporal, ดังนั้นจะมี method ที่เหมือนๆกัน ในขณะที่ Period ฟีก Duration จะ implement java.time.temporal.TemporalAmount
3. LocalDate แสดงวันที่ด้วยข้อมูล year, month and day of month, เราสามารถสร้าง instance ด้วยการใช้ method “of”
4. Valid ChronoField values to use with the get() method are: DAY_OF_WEEK, ALIGNED_DAY_OF_WEEK_IN_MONTH, ALIGNED_DAY_OF_WEEK_IN_YEAR, DAY_OF_MONTH, DAY_OF_YEAR, EPOCH_DAY, ALIGNED_WEEK_OF_MONTH, ALIGNED_WEEK_OF_YEAR, MONTH_OF_YEAR, PROLEPTIC_MONTH, YEAR_OF_ERA, YEAR, and ERA.
5. Valid ChronoUnits values to use with the plus() and minus() methods are: DAYS, WEEKS, MONTHS, YEARS, DECADES, CENTURIES, MILLENNIA, and ERAS.
6. LocalTime แสดงข้อมูลเวลาด้วย hour, minutes, seconds, and nanoseconds, เราสามารถสร้าง instance ด้วยการใช้ method “of”
7. Valid ChronoField values to use with the get() method are: NANO_OF_SECOND, NANO_OF_DAY, MICRO_OF_SECOND, MICRO_OF_DAY, MILLI_OF_SECOND, MILLI_OF_DAY, SECOND_OF_MINUTE, SECOND_OF_DAY, MINUTE_OF_HOUR, MINUTE_OF_DAY, HOUR_OF_AMPM, CLOCK_HOUR_OF_AMPM, HOUR_OF_DAY, CLOCK_HOUR_OF_DAY, and AMPM_OF_DAY.

8. Valid ChronoUnits values to use with the plus() and minus() methods are: NANOS, MICROS, MILLIS, SECONDS, MINUTES, HOURS, and HALF_DAYS.
9. LocalDateTime คือการรวม LocalDate และ LocalTime, เราสามารถสร้าง instance ด้วย method “of”
10. Valid ChronoField and ChronoUnits values are a combination of the ones used for LocalDate and LocalTime.
11. Instant แสดงถึงจุดเดียวในเวลาเป็น seconds และ nanoseconds, เราสามารถสร้าง instance ด้วยการใช้ method “ofEpoch”
12. Valid ChronoField values to use with the get() method are: NANO_OF_SECOND, MICRO_OF_SECOND, MILLI_OF_SECOND, and INSTANT_SECONDS.
13. Valid ChronoUnits values to use with the plus() and minus() methods are: NANOS, MICROS, MILLIS, SECONDS, MINUTES, HOURS, HALF_DAYS, and DAYS.
14. Period แสดงระยะเวลาในแง่ของ years, months and days, เราสามารถสร้าง instance ด้วย method “of”
15. Valid ChronoUnits values to use with the get() method are: DAYS, MONTHS, YEARS.
16. Duration แสดงระยะเวลาในแง่ของ seconds and nanoseconds, สามารถสร้าง instance ด้วย method “of”
17. Valid ChronoUnits values to use with the constructor and the get(), plus(), and minus() methods are: NANOS, MICROS, MILLIS, SECONDS, MINUTES, HOURS, HALF_DAYS, and DAYS.