**СТАРЫЕ КЛАССЫ**

**Date –** старый класс, инкапсулирующий дату и время.

**Конструторы:**

* *Date()* – объект с текущей датой и временем.
* *Date(long miliseconds)* – количество миллисекунд с 1 января 1970.

**Методы:**

* **date1.after(Date date2)**– true если date1 позже date2
* **date1.before(Date date2)**– true если date1 раньше date2
* **date.getTime()**– вернуть количество миллисекунд
* **date.setTime(long ms)** – установить новое время

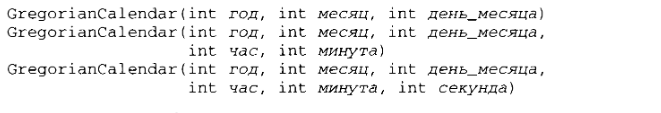
**Calendar –** абстрактный класс, предоставляет ряд методов для преобразования время в миллисекундах в разных составляющих (год, месяц, день, часы, минуты, секунды). Реализацией является **GregorianCalendar**.

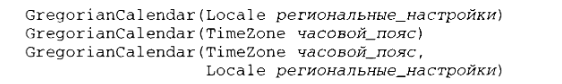
Содержит внутри себя **массив целых чисел**, каждая ячейка отвечает за год, день и т.д.

**Методы:**

* **cal.add(int составляющая, int значение)** – добавить (можно передать отрицательное) к составляющей указанное значение. В первый параметр можно передавать константы, определенные в классе Calendar ( например Calendar.HOUR).
* **cal.after(Object calendarObject)**
* **cal.before(Object calendarObject)**
* **cal.clear()** – обнулить все составляющие даты
* **cal.clear(int составляющая)** – обнуляет указанную составляющую
* **cal.get(int составляющая)** – возвращает значение указанной составляющей.
* **static Locale[] getAvailableLocales()** – возвращает массив объектов Locale, которые содержат в себе региональные настройки, для которых в системе доступны календари.
* **static Calendar getInstance(TimeZone, Locale)** – возвращает Calendar для региональных настроек и указанной временной зоны (можно передавать один из).
* **cal.set(int состовляющая, int значение)** – установить составляющую.

**Конструкторы GrigorianCalendar**:





**Методы GregorianCalendar**:

* **isLeapYear(int год)** – проверить, является ли год високосным
* **toZonedDateTime()** – преобразовать в ZonedDateTime

**TimeZone –** позволяет оперировать временем с учетом часовых поясов. В этом классе учитывается также летнее время.

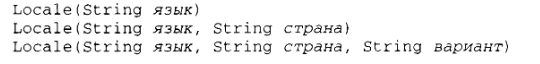
Методы:

* **static String[] getAvailableIDs()** – массив с названиями всех часовых поясов
* **static TimeZone getDefault()** – получить часовой пояс по умолчанию.
* **tz.getID()** – возвращает имя часового пояса
* **tz.getOffset()** – возвращает смещение, которое нужно добавить к времени по Гринвичу, чтобы рассчитать местное время с учетом летнего времени
* **static TimeZone getTimeZone(String name)** – получить объект TimeZone по названию.
* **tz.toZoneId()** – преобразует объект в ZoneId пакета java.time

реализацией TimeZone является **SimpleTimeZone** который позволяет работать с часовыми поясами для григорианского календаря.

Класс **Locale** – объект Локали. Локаль влияет на формат даты, времени, чисел.

**Конструкторы**:



**Методы:**

* **locale.setDefault(Locale locale)** – установить локаль по умолчанию.

**Недостатки старого API:**

**• не потокобезопасный,**

**• изменяемые объекты, ,,**

**• временная зона даты - это временная зона JVM по умолчанию,**

**• месяца начинаются с нуля.**

**НОВЫЕ КЛАССЫ**

* **Неизменяемые => потокобезопасные** (все вычисления создают новый объект)
* Нумерация месяцев начинается с 1
* В старом Api классы раскиданы по разным пакетам, util, sql, util.concurrent.TimeUnit, java.text.DateFormat. В новом api все в пакете java.time
* Точность повышена с мс до нс
* Все операции, требующие временной зоны требуют ее явно
* Появились классы для определения длительности и промежутков времени.

**LocalDate –** инкапсулирует дату, для предоставления которой используется выбираемый по умолчанию григорианский календарь. Он не включает информацию о времени и часовом поясе.

Объекты создаются с помощью **фабричных методов**:

static LocalDate **now()**

static LocalDate **now​(Clock clock)**

static LocalDate **now​(ZoneId zone)**

static LocalDate **of​(int year, int month, int dayOfMonth)**

static LocalDate **of​(int year, Month month, int dayOfMonth)**

static LocalDate **ofEpochDay​(long epochDay)**

static LocalDate **ofInstant​(Instant instant, ZoneId zone)**

static LocalDate **ofYearDay​(int year, int dayOfYear)**

static LocalDate **parse​(CharSequence text)**

static LocalDate **parse​(CharSequence text, DateTimeFormatter formatter)**

**Методы:**

public boolean **isBefore(ChronoLocalDate other)**

public boolean **isAfter(ChronoLocalDate other)**

public **int lengthOfMonth()** - количество дней в месяце, представленное объектом LocalDate.

public int **lengthOfYear()** - количество дней в году, представленное этим объектом LocalDate.

public int **getYear()**

public int **getMonthValue()**

public Month **getMonth()**

public int **getDayOfMonth()**

public int **getDayOfYear()**

public **DayOfWeek getDayOfWeek()**

public IsoEra **getEra()**

public IsoChronology **getChronology()**

public int **get(TemporalField field)** – возвращает значение указанного поля (field) этого объекта LocalDate в виде 32-разрядного целого числа. **date.get(ChronoField.DAY\_OF\_MONTH)**

public long **getLong(TemporalField field).** Некоторые поля (field) могут не поддерживаться **LocalDate**, и будет выдано исключение **UnsupportedTemporalTypeException**. Чтобы быть уверенным, используйте метод **isSupported(TemporalField)**

public LocalDate **plusYears(long yearsToAdd)**

public LocalDate **plusMonths(long monthsToAdd)**

public LocalDate **plusWeeks(long weeksToAdd)**

public LocalDate **plusDays(long daysToAdd)**

public LocalDate **plus(TemporalAmount amountToAdd)** - например

  
**Amount это Period либо Duration**

Аналогичные методы **minusX(..)**

public LocalDate **withYear(int year)** - вернуть копию объекта с измененным годом

public LocalDate **withMonth(int month)** - Если день месяца недействителен для года, он будет изменен на последний действительный день месяца.

public LocalDate **withDayOfMonth(int dayOfMonth) -** Если значение дня месяца недопустимо, создается исключение **DateTimeException.**

public LocalDate **withDayOfYear(int dayOfYear) -** Если значение DayOfYear недопустимо, то создается исключение **DateTimeException**

public boolean **isLeapYear()** - является ли год високосным

public boolean **isSupported(TemporalField field)** - поддерживается ли данное ***TemporalField.***

**LocalTime –** представляет местное время (local time). Он не включает информацию о дате и часовом поясе.

Объекты создаются с помощью **фабричных методов**:

public static LocalTime **now()**

public static LocalTime **now(ZoneId zone)**

public static LocalTime **now(Clock clock)**

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 **ofInstant(Instant instant, ZoneId zone)**

public static LocalTime **ofSecondOfDay(long secondOfDay)**

public static LocalTime **ofNanoOfDay(long nanoOfDay)**

public static LocalTime **from(TemporalAccessor temporal)**

public static LocalTime **parse(CharSequence text)**

public static LocalTime **parse(CharSequence text, DateTimeFormatter formatter)**

**Методы:**

public boolean **isAfter(LocalTime other)**

public boolean **isBefore(LocalTime other)**

public String **format(DateTimeFormatter formatter)** - Отформатируйте этот объект **LocalTime** с помощью указанного **DateTimeFormatter**.

public int **getHour()**

public int **getMinute()**

public int **getSecond()**

public int **getNano()**

public int **get(TemporalField field)** - Возвращает значение указанного поля (field) этого объекта LocalTime в виде 32-разрядного целого числа.

public long **getLong(TemporalField field)** - Возвращает значение указанного поля (field) этого объекта LocalTime в виде 64-разрядного целого числа.

public LocalTime **plusHours(long hoursToAdd)**

public LocalTime **plusMinutes(long minutesToAdd)**

public LocalTime **plusSeconds(long secondstoAdd)**

public LocalTime **plusNanos(long nanosToAdd)**

public LocalTime **plus(TemporalAmount amountToAdd)**

Аналогичные методы **minusX(..)**

public LocalTime **withHour(int hour) –** в метод with нельзя передовать значения больше допустимого

public LocalTime **withMinute(int minute)**

public LocalTime **withSecond(int second)**

public LocalTime **withNano(int nanoOfSecond)**

public LocalTime **with(TemporalAdjuster adjuster)** - Возвращает копию этого объекта **LocalTime** с данными о времени, скорректированными объектом ***TemporalAdjuster***.

public **LocalDateTime atDate(LocalDate date)** - используется для объединения этого объекта **LocalTime** с указанным объектом **LocalDate** для создания объекта **LocalDateTime**.

public int **toSecondOfDay()**

public long **toNanoOfDay()**

public long **toEpochSecond(LocalDate date, ZoneOffset offset)**

public LocalTime **truncatedTo(TemporalUnit unit)** - Возвращает копию этого объекта LocalTime с усеченным (truncated) временем в указанной единице измерения.

public ValueRange **range(TemporalField field)**

public boolean **isSupported(TemporalField field)** – поддерживается ли указанный TemporalField

**LocalDateTime –** объект, представляющий собой комбинацию LocalDate и LocalTime.

Методы:

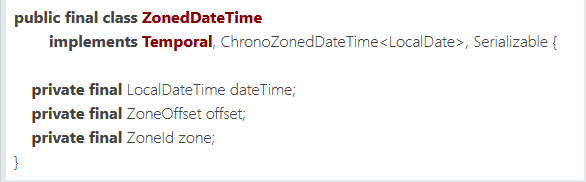
* **toLocalDate()** – получить объект LocalDate
* **toLocalTime()** – получить объект LocalTime

**ZonedDateTime**- это класс, представляющий дату и время с информацией о часовом поясе, который находится в календарной системе **ISO-8601**, такой как

**2007-12-03T10:15:30+01:00 Europe/Paris**,

**2022-10-06T11:21:37.063+03:00[Europe/Minsk]**

Он также поддерживают информацию о летнем времени.



он управляет 3 внутренними объектами, в которых:

1. Объект **LocalDateTime** содержит информацию о дате и времени.
2. Объект **ZoneId**- это имя идентификатора часового пояса, например, **Europe/Paris**.
3. Объект **ZoneOffset** содержит информацию о смещении часового пояса, например **+01:30**.

**ZonedDateTime поддерживает исторические данные.** Например, страна в разные годы использовала разные часовые пояса.

**Фабричные методы для объектов:**

public static ZonedDateTime **now()**

public static ZonedDateTime **now(ZoneId zone)**

public static ZonedDateTime **now(Clock clock)**

public static ZonedDateTime **of(LocalDate date, LocalTime time, ZoneId zone)**

public static ZonedDateTime **of(LocalDateTime localDateTime, ZoneId zone)**

public static ZonedDateTime **of(int year, int month, int dayOfMonth,**

**int hour, int minute, int second,**

**int nanoOfSecond, ZoneId zone)**

public static **ZonedDateTime ofLocal(LocalDateTime localDateTime,**

**ZoneId zone, ZoneOffset preferredOffset)**

public static ZonedDateTime **ofInstant(Instant instant, ZoneId zone)**

public static ZonedDateTime **ofInstant(LocalDateTime localDateTime,**

**ZoneOffset offset, ZoneId zone)**

public static ZonedDateTime **ofStrict(LocalDateTime localDateTime, ZoneOffset offset, ZoneId zone)**

public static ZonedDateTime **from(TemporalAccessor temporal)**

public static ZonedDateTime **parse(CharSequence text)**

public static ZonedDateTime **parse(CharSequence text, DateTimeFormatter formatter)**

Большинство методов класса **ZonedDateTime** аналогичны методам класса **LocalDateTime**

Другие методы:

public ZoneId **getZone()**

public ZoneOffset **getOffset()** - получить смещение

public OffsetDateTime **toOffsetDateTime() –** преобразовать в toOffsetDateTime (у него есть только смещение)

public LocalDateTime **toLocalDateTime()**

public LocalDate **toLocalDate()**

public LocalTime **toLocalTime()**

public ZonedDateTime **withFixedOffsetZone()**

public ZonedDateTime **withEarlierOffsetAtOverlap()**

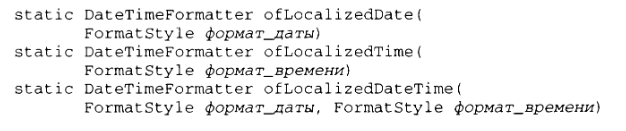
public ZonedDateTime **withLaterOffsetAtOverlap()**

public ZonedDateTime **withZoneSameLocal(ZoneId zone)**

public ZonedDateTime **withZoneSameInstant(ZoneId zone)** – вернуть этот же объект, но с новым часовым поясом

Все 3 класса имеют метод **format(DateTimeFormatter formatter)** для преобразования даты и времени к указанному формату.

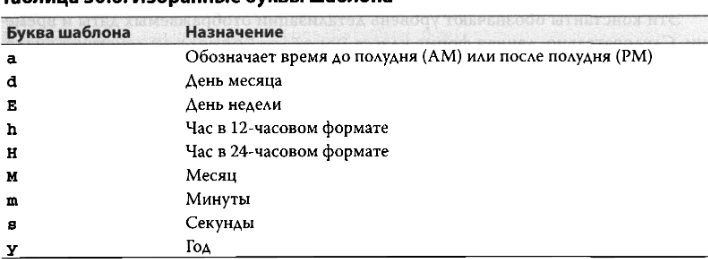
**DateTimeFormatter –** для получения объекта как правило вызывается один из **фабричных методов**



В **FormatStyle** перечисляются следующие константы:

* **FULL**
* **LONG**
* **MEDIUM**
* **SHORT**

Чтобы создать свой формат, можно использовать метод **dtf.ofPattern(String pattern)**. Строка шаблона может содержать следующие символы



От количества повторений буквы зависит выводимый результат.



Распарсить время или дату можно с помощью статического метода **parse(CharSequence строка\_даты\_времени, DateTimeFormatter dtf)**



**ZoneId** абстрактный класс, используется для определения часового пояса и предоставления правил преобразования между **LocalDateTime** и **Instant.**

Два важных **свойства:**

* **String id**: ID уникален.
* **ZoneRules rules**: Правила определения смещения часового пояса (time zone offset) в определенное время(переводы на летнее время и тп).

Делиться на 3 типа:

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | | **Example** | **getId()** |
| **Type1** | **Region-based ZoneId** | **ZoneId**.of("Europe/Paris") | Europe/Paris |
| **Type2** | **Offset-based ZoneId** | **ZoneOffset**.of("-06") | -06 |
| **ZoneOffset**.of("+06:05:20") | +06:05:20 |
| **Type3** | **UTC/GMT/UT ZoneId** | **ZoneId**.ofOffset("UTC", **ZoneOffset**.of("+06")) | UTC+06 |
| **ZoneId**.of("GMT-06:05:20") | GMT-06:05:20 |

Класс **ZoneOffset** является подклассом ZoneId реализует временную зону по простому смещению. ID этих часовых поясов начинаются с + или -.

Класс **ZoneRegion** реализует временную зону по географическому принципу. Содержит исторические данные о смещении часовых поясов.

**UTC, GMT, UT ZoneId**: **ID** часовых поясов этого типа начинаются с **"UTC"**, **"GMT"**  или **"UT"**, за которыми следуют **"+"** или **"-"**.

**INSTANT**

**Instant –** используется для представления мгновения в секундах, которые прошли с 1 января 1970 года. Замена Date, с наносекундной точностью и корректным названием. Внутри хранит long с количеством секунд и int с количеством наносекунд.

Он ничего не знает про таймзону и даже в методе toString() выводит в UTC (В отличии от Date, который использовал локаль и временную зону по умолчанию).

**Методы:**

* **Instant.now() –** текущее время в виде Instant
* **ofEpochSecond(long epochSecond)**
* **from(TemporalAccessor temporal)**
* **parse(final CharSequence text)**
* **isSupported(TemporalField field)**
* **get(TemporalField field)**
* **(TemporalAmount amountToAdd)**

**TEMPORAL**

Интерфейс **Temporal** представляет общую концепцию между понятиями даты, времени и смещения по времени. Он предоставляет основные методы для получения информации от этих объектов.

В нем объявлены следующие **методы**:

* **isSupported(TemporalUnit unit) –** Поддерживается ли указанный TemporalUnit этим классом.
* **with(TemporalAdjuster adjuster)** – возвращает копию этого Temporal, скорректированная указанным adjuster
* **plus(long amountToAdd, TemporalUnit unit) –** возвращает копию Temporal с добавленным значением
* **minus(long, TemporalUnit) - –** возвращает копию Temporal с отнятым значением
* **long until(Temporal endExclusive, TemporalUnit unit) –** вычисляет количество времени между двумя Temporal в указанной единице времени.

**PERIOD**

**Period –** описание календарной длительности в виде кортежа (год, месяц, день)

Значения года, месяца и дня независимы

Статические **фабричные методы**:

* public static Period **ofYears(int years)**
* public static Period **ofMonths(int months)**
* public static Period **ofWeeks(int weeks)**
* public static Period **ofDays(int days)**
* public static Period **of(int years, int months, int days)**
* public static Period **from(TemporalAmount amount)**
* public static Period **parse(CharSequence text) - Period.parse("P5M10D")**
* public static Period **between(LocalDate startDate, LocalDate endDate)**

**методы:**

* **get(TemporalUnit unit) –** вернуть значение указанной единицы
* **public int getYears()**
* **public int getMonths()**
* **public int getDays()**
* **public Period withYears(int years)**
* **public Period withMonths(int months)**
* **public Period withDays(int days)**
* **public Period plus(TemporalAmount amountToAdd)**
* **public Period plusYears(long yearsToAdd)**
* **public Period plusMonths(long monthsToAdd)**
* **public Period plusDays(long daysToAdd)**
* **Аналогичные методы minus**
* **public boolean isZero() –** true если все 3 значения нулевые
* **public boolean isNegative() –** true если одно из значений отрицательное
* **public Duration negated() -** возвращает отрицательную копию этого Period

**DURATION**

**Duration –** описание точной длительности в виде целого количества секунд и долей текущей секунды в виде наносекунд.

Он может быть преобразован в NANOS, MILLIS, SECONDS, MINITES, HOURS или DAYS.

Статические **фабричные методы**:

* public static Duration **ofDays(long days)**
* public static Duration **ofHours(long hours)**
* public static Duration **ofMinutes(long minutes)**
* public static Duration **ofSeconds(long seconds)**
* public static Duration **ofSeconds(long seconds, long nanoAdjustment)**
* public static Duration **ofMillis(long millis)**
* public static Duration **ofNanos(long nanos)**
* public static Duration **of(long amount, TemporalUnit unit)**
* public static Duration **from(TemporalAmount amount)**
* public static Duration **parse(CharSequence text)**
* public static Duration **between(Temporal startInclusive, Temporal endExclusive)**

**методы:**

* **public long toNanos()**
* **public long toMillis()**
* **public long toSeconds()**
* **public long toMinutes() –** Возвращает только целую часть (если внутри лежит 65 секунд, вернет 1 минуту)
* **public long toHours() -** возвращает только целую часть
* **public long toDays() -** возвращает только целую часть
* **public int toNanosPart() -** toNanos() % 10^9
* **public int toMillisPart() -** toMillis() % 10^3
* **public int toSecondsPart() -** toSeconds() % 60
* **public int toMinutesPart() -** toMinutes() % 60
* **public int toHoursPart() -** toHours() % 24
* **public long toDaysPart() -** toDays()
* **public boolean isNegative() -** true если общее число наносекунд в этом Duration меньше 0.
* **public long get(TemporalUnit unit) –** поддерживает только ChronoUnit.SECONDS и ChronoUnit.NANOS
* **public long getSeconds()**
* **public int getNano()**
* **public Duration withSeconds(long seconds)**
* **public Duration withNanos(int nanoOfSecond)**
* **public Duration plus(Duration duration)**
* **public Duration plus(long amountToAdd, TemporalUnit unit) –** поддерживаются следующие ChronoUnit: NANOS, MICROS, MILLIS, SECONDS, MINUTES, HOURS, HALF\_DAYS, DAYS
* **public Duration plusNanos(long nanosToAdd)**
* **public Duration plusMillis(long millisToAdd)**
* **public Duration plusSeconds(long secondsToAdd)**
* **public Duration plusMinutes(long minutesToAdd)**
* **public Duration plusHours(long hoursToAdd)**
* **public Duration plusDays(long daysToAdd)**
* **Аналогичные методы minus**
* **public Duration negated() -** возвращает отрицательную копию этого Duration
* **public Duration abs() -** возвращает абсолютное значение этой Duration, то есть модуль
* **public Temporal addTo(Temporal temporal) –** Добавляет Duration к Temporal
* **public Temporal subtractFrom(Temporal temporal) -** вычитает Duration из указанного Temporal
* **public Duration multipliedBy(long multiplicand) –** умножает эту Duration на указанное значение
* **public Duration dividedBy(long divisor) –** делит Duration на указанное значение.
* **public long dividedBy(Duration divisor) –** эта Duration делится на другую и возвращается число

**TEMPORALUNIT, CHRONOUNIT**

**TemporalUnit –** интерфейс, который представляет собой единицу измерения времени (дни, часы, годы, месяца).

Наболее часто используемые единица определены в реализации этого класса – **ChronoUnit.** Это перечисление, которое содержит следующие значения:

* **NANOS**
* **MICROS**
* **MILLIS**
* **SECONDS**
* **MINUTES**
* **HOURS**
* **HALF\_DAYS** – половина дня, 12 часов
* **DAYS**
* **WEEKS**
* **MONTHS**
* **YEARS**
* **DECADES** - десятилетие
* **CENTURIES** – век (100 лет)
* **MILLENNIA** - тысячелетие
* **ERAS** – эра (Эра связана с периодом правления короля или продолжительностью династии. Концепция эры наиболее очевидна в японской императорской календарной системе.)
* **FOREVER** - Искусственная единица, представляющая концепцию вечности. Максимальная продолжительность Duration

**Методы:**

* **public Duration getDuration() -** расчетная продолжительность единицы измерения (например день-24 часа, месяц – 30 дней (приблизительно))
* **public boolean isDurationEstimated() -** является ли продолжительность единицы измерения точной (например минута всегда равна 60 сек– вернется false, а месяц не всегда 30 дням - true)
* **public boolean isSupportedBy(Temporal temporal) -** поддерживается ли указанная едина указанным объектом temporal.
* **public long between(Temporal temporal1, Temporal temporal) –** промежуток между двумя Temporal в этой единице времени

**CLOCK**

**Clock** в основном используется для тестирования, в качестве поставщика времени который может быть эффективно заглушен. Позволяет нам использовать практически любой ход времени.

Он позволяет нам проверить, что наш код работает корректно с разными часовыми поясами, или при использовании фиксированных часов – что время не влияет на наш код.

Для создания можно использовать следующие **фабричные методы**:

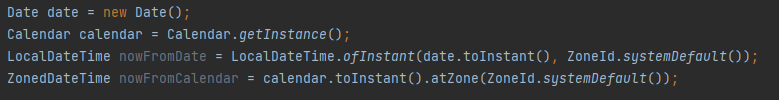
* **offset(Clock, Duration) –** возвращает часы, смещенные на заданную длительность. Основной вариант использования – имитация работы в будущем или в прошлом.
* **systemDefauultZone() –** создает системные часы во временной зоне по умолчанию
* **systemUTC() –** создает системные часы во временной зоне UTC
* **fixed(Instant, ZoneId)**  - фиксированные часы, они не идут а стоят на месте
* **system(ZoneId)** – создает часы в указанной временной зоне
* **tickSeconds(ZoneId)** – создает часы в указанной временной зоне, значение которых округлено до целых секунд
* **tickMinutes(ZoneId)** – создает системные часы в указанной временной зоне, значение которых
* **tick(Clock, Duration)** – округляет значение времени до указанного периода
* **withZone() –** возвращает копию часов с другой таймзоной

Другие методы:

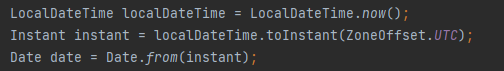
* **getZone() –** получить временную зону, в которой работают часы
* **millis() –** запросить текущее время в миллисекундах по Unix-time
* **instant() –** запросить объект Instant

**Преобразование из старого Api в новое и обратно**

Для преобразования в новый Api удобно использовать метод **toInstant()** который есть в классах Date и Calendar. Instant затем можно преобразовать к нужному объекту даты.



Обратное преобразование выполняется аналогичным образом, используя метод **Date.from(Instant).**



**Задачка найти дату следующего понедельника**

