**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

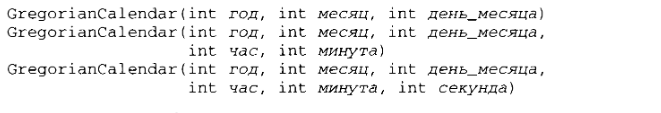
**Calendar –** абстрактный класс, предоставляет ряд методов для преобразования время в миллисекундах в разных составляющих (год, месяц, день, часы, минуты, секунды). Реализацией является **GregorianCalendar**.

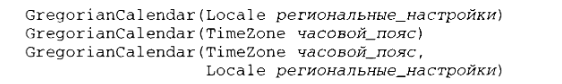
Содержит внутри себя массив целых чисел, каждая ячейка отвечает за год, день и т.д.

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.

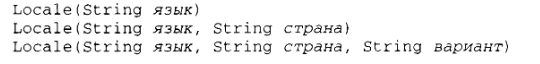
Методы:

* 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)

**LocalDate –** инкапсулирует дату, для предоставления которой используется выбираемый по умолчанию григорианский. 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



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)**

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)**

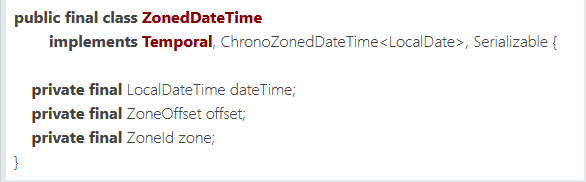
public boolean **isSupported(TemporalField field)** – поддерживается ли указанный TemporalField

**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**. 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()**

public OffsetDateTime **toOffsetDateTime()**

public LocalDateTime **toLocalDateTime()**

public LocalDate **toLocalDate()**

public LocalTime **toLocalTime()**

public ZonedDateTime **withFixedOffsetZone()**

public ZonedDateTime **withEarlierOffsetAtOverlap()**

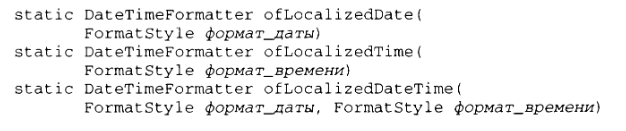
public ZonedDateTime **withLaterOffsetAtOverlap()**

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.

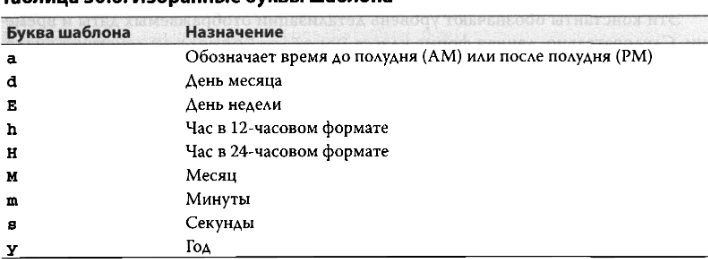
**DateTimeFormatter –** для получения объекта как правило вызывается один из фабричных методов



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 свойствах:

* **String id**: ID уникален.
* **ZoneRules rules**: Правила определения смещения часового пояса (time zone offset) в определенное время на временной шкале.

Класс **ZoneOffset** является подклассом ZoneId. Делится 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 |

**Region-based ZoneId**содержит исторические данные о смещениях часовых поясов.

**Offset-based ZoneId** (**ZoneId** на основе смещения): Эти **ID**часовых поясов начинаются с **"+"** или **"-"**

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